Advanced Probabilistic Models for Clustering and Projection

Dissertation im Fach Informatik
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

von
Shipeng Yu

Tag der Einreichung: 21 Juni, 2006
Tag der mündlichen Prüfung: 29 September, 2006

Berichterstatter:
Prof. Dr. Hans-Peter Kriegel, Ludwig-Maximilians-Universität München
Prof. Dr. Thomas Hofmann, Technische Universität Darmstadt
Dr. Volker Tresp, Siemens AG, München

To my parents

Abstract

Probabilistic modeling for data mining and machine learning problems is a fundamental research area. The general approach is to assume a generative model underlying the observed data, and estimate model parameters via likelihood maximization. It has the deep probability theory as the mathematical background, and enjoys a large amount of methods from statistical learning, sampling theory and Bayesian statistics. In this thesis we study several advanced probabilistic models for data clustering and feature projection, which are the two important unsupervised learning problems.

The goal of clustering is to group similar data points together to uncover the data clusters. While numerous methods exist for various clustering tasks, one important question still remains, i.e., how to automatically determine the number of clusters. The first part of the thesis answers this question from a mixture modeling perspective. A finite mixture model is first introduced for clustering, in which each mixture component is assumed to be an exponential family distribution for generality. The model is then extended to an infinite mixture model, and its strong connection to Dirichlet process (DP) is uncovered which is a non-parametric Bayesian framework. A variational Bayesian algorithm called VBDMA is derived from this new insight to learn the number of clusters automatically, and empirical studies on some 2D data sets and an image data set verify the effectiveness of this algorithm.

In feature projection, we are interested in dimensionality reduction and aim to find a low-dimensional feature representation for the data. We first review the well-known principal component analysis (PCA) and its probabilistic interpretation (PPCA), and then generalize PPCA to a novel probabilistic model which is able to handle non-linear projection known as kernel PCA. An expectation-maximization (EM) algorithm is derived for kernel PCA such that it is fast and applicable to large data sets. Then we propose a novel supervised projection method called MORP, which can take the output information into account in a supervised learning context. Empirical studies on various data sets show much better results compared to unsupervised projection and other supervised projection methods. At the end we generalize MORP probabilistically to propose SPPCA for supervised projection, and we can also naturally extend the model to S^2PPCA which is a semi-supervised projection method. This allows us to incorporate both the label information and the unlabeled data into the projection process.

In the third part of the thesis, we introduce a unified probabilistic model which can

handle data clustering and feature projection jointly. The model can be viewed as a clustering model with projected features, and a projection model with structured documents. A variational Bayesian learning algorithm can be derived, and it turns out to iterate the clustering operations and projection operations until convergence. Superior performance can be obtained for both clustering and projection.

Acknowledgements

This dissertation is based on my research work that I carried out as a Ph.D. student in a joint Ph.D. program between the KDD group at University of Munich (LMU) and the Learning Systems Department (CT IC 4, previously named Neural Computation) of Siemens AG. During the past three years, I have been extremely fortunate to have the guidance, support, and friendship of a number of people who helped me grow academically and personally.

First, I would like to thank Prof. Hans-Peter Kriegel, my supervisor, for his encouragements, constructive suggestions and constant support during this research. His door is always open to me whenever I need his help. I was impressed by his open-mindedness and academic guidance that make the KDD group at LMU so successful.

I am also greatly thankful to Prof. Thomas Hofmann, who kindly agreed to allocate his time on supervising my thesis, despite his extremely busy research and teaching work. I would also like to thank Prof. Martin Wirsing and Prof. Martin Hofmann, for their very patient instructions on my oral examination.

My co-supervisor at Siemens AG, Dr. Volker Tresp, is the person who had the greatest role in my intellectual development. He introduced me into the field of statistical machine learning. His enthusiasm, sharp thoughts, open-mindedness, and humor made my research a really memorable and joyful journey.

I am indebted to the friendship and fellowship with Dr. Kai Yu, who really helped me a lot in both academic research and personal life. I was amazed by not only his solid knowledge in machine learning, but also his way of thinking. He can always find the critical point of the problem, and we had a very effective and memorable cooperation during my Ph.D. work.

I feel grateful to Prof. Bernd Schürmann, the leader of the Learning Systems Department at Siemens AG, for his constant support to my research. I appreciate his emphasis on both scientific research and real-world applications, which greatly influences my commitment to my career plan.

Finishing a thesis is not a one-person thing. Here I wish to thank the following people: Prof. Christian Böhm, Prof. Zoubin Ghahramani, Zhao Xu, Yi Huang, Mrs. Susanne Grienberger, Franz Krojer, Dr. Matthias Schubert, Dr. Peer Kröger, Mrs. Christa

Singer, Mrs. Christine Herzog, Dr. Kai Heesche, Dr. Christoph Tietz, Dr. Dengyong Zhou, Dr. Mingrui Wu, Karsten Borgwardt, Arthur Zimek, Markus Bundschus, Stefan Reckow, Christof Störmann, Huaien Gao, Dr. Hugo Zaragoza, Dr. Ralf Herbrich, Dr. Thore Graepel, Dr. Ji-Rong Wen, Dr. Wei-Ying Ma, ...

Of course, I am grateful to my parents, for their patience and love. Without them this work would never have come into existence.

Shipeng Yu
Munich, Germany
May, 2006

Table of Contents

List of Algorithms

List of Figures

List of Tables

Prefix

Data clustering and feature projection are two fundamental problems for data mining and machine learning. They both try to uncover the intrinsic structure of the underlying data, and are in general referred to as *unsupervised learning* problems. This is in contrast to *supervised learning*, where the target is to learn a supervised task like classification and regression.

Probabilistic modeling for unsupervised learning is an important research area. The basic assumption of probabilistic modeling is that there is a generative model underlying the observed data, which reflects the intrinsic structure of the data. The generative model is determined by some model parameters, and one task of probabilistic modeling is *learning* which aims to identify these parameters given the model assumption and the observations. The other task is *inference*, by which we can predict how likely a new data point can be generated from this underlying model.

In this thesis we are interested in probabilistic modeling for two important unsupervised learning problems, namely, clustering and projection. For clustering we focus on mixture models, and for projection we consider the so-called latent variable models. These basic models are both well-known in statistics, but in this thesis we study the advanced models and consider, for instance, infinite mixture models and non-linear latent variable models. We then propose a unified framework at the end and solve the two problems simultaneously.

The organization of this thesis is as follows. We study probabilistic models for clustering in Part I, which covers the following chapters:

- Chapter 1 provides an overview of clustering models. The well-known flat clustering and hierarchical clustering algorithms are briefly discussed, which can be categorized into model-based clustering and similarity-based clustering. The following chapters mainly discuss model-based clustering algorithms.

- Chapter 2 studies the well-known finite mixture models for clustering. To allow enough flexibility of the model, we introduce mixture of exponential family distributions and the conjugate family, and study finite mixture models from a Bayesian perspective. A Gibbs sampling method as well as the variational Bayesian approximation are used for inference and learning, and mixture of Gaussians and mixture of Multinomials are illustrated as examples.

- Chapter 3 goes beyond finite mixture models and consider the asymptotic case where the number of mixture components goes to infinity. This chapter builds the connection to the Dirichlet process, a non-parametric Bayesian prior for random measures. Both Gibbs sampling methods and efficient variational mean-field algorithms are described for inference in the model, and empirical studies show the effectiveness of the model.

Then in Part II we turn to probabilistic projection models and discuss the following topics:

- Chapter 4 gives an overview of different projection methods like PCA, kernel PCA and projection models for discrete data.
- Chapter 5 reviews the probabilistic model for PCA and generalizes the model for non-linear projection such as kernel PCA. An efficient EM learning algorithm is derived for kernel PCA in this context, and the incremental and online learning versions are briefly discussed for extensions.
- Chapter 6 considers supervised projection problem where we have both input features and some output labels like classification labels or regression values. A multi-output regularized projection (MORP) algorithm is introduced to solve this problem which is motivated from the latent variable view of PCA. The algorithm turns out to be a generalized eigenvalue problem, and can also be generalized to non-linear projections.
- Chapter 7 provides a probabilistic explanation to MORP and furthermore considers the semi-supervised projection problem where we have unlabeled data as well. An efficient EM algorithm is derived for inference and learning, and various experiments show that the proposed method outperforms the competitors.

The last part discusses a unified model for clustering and projection. After a brief overview of joint clustering-projection models in Chapter 8, we discuss in detail the probabilistic clustering-projection (PCP) model for discrete data in Chapter 9. The models defines a generative process for both clustering and projection, and the standard variational Bayesian learning algorithm corresponds to an iterative process of performing clustering and projection operations. We show that we can obtain both better clustering structure for data points, and better projection model for features.

Part I

Probabilistic Clustering Models

Chapter 1

Overview of Clustering Models

Clustering aims to group similar data points together, and is a fundamental problem for data mining, machine learning and statistics. Apart from the clear intuition which is just data grouping, it is well-known that clustering is not a well-defined problem, and there exists a large amount of work on both theoretical analysis of clustering algorithms and their usage in different application domains. In this part we are more interested in the algorithm level, and the derived models can be applied in principal to any suitable domains.

The most well-known clustering algorithm is probably the k-means, which partitions the data into k disjoint parts (see, e.g., [8]). Mathematically it can be shown that k-means is minimizing an information criterion of the data, and it implicitly assumes that the data are distributed as a mixture of isotropic Gaussians. The easy implementation and good scalability makes it popular in almost all clustering applications. It belongs to the category of *flat clustering*, in which all the clusters are at the same level and have no overlap. In Section 1.1 we review the two different types of flat clustering algorithms, i.e., *model-based clustering* and *similarity-based clustering*. k-means can however be viewed from both perspectives.

A different way of organizing clusters is to build a cluster hierarchy, where upper-level clusters contain lower-level clusters. Clustering algorithms which can generate such a tree structure are called *hierarchical clustering* algorithms [34], and each level of the hierarchy can be viewed as a flat clustering of the data. To generate such a hierarchy, we can gradually merge nearby data points until we obtain one cluster for all the data (agglomerative methods), or start from the top of the tree and each time split one big cluster into several smaller ones (partitioning methods). The well-known linkage methods such as single-linkage and complete-linkage belong to the former type and are widely used for hierarchy generation. In Section 1.2 we briefly review these algorithms. Finally in Section 1.3 we point out the road-map of our contributions in the following chapters of this part.

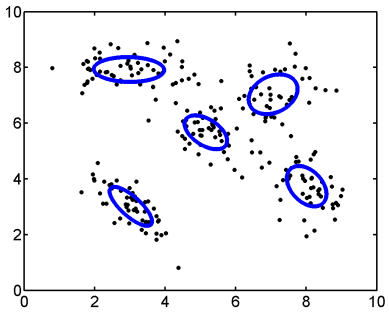


Figure 1.1: Illustration of a Gaussian mixture model on a 2D toy data.

1.1 Flat Clustering

Flat clustering groups the data points into a small number of clusters. It can in general be divided into two categories: model-based clustering and similarity-based clustering. Some algorithms like k-means can be viewed from both perspectives.

1.1.1 Model-based Clustering

In model-based clustering, we need to assign a generative model to the data and train a mixture model with some distributional assumptions [26]. Then each mixture component corresponds to one cluster of the data. One popular choice is to make Gaussian assumptions to the data and train a Gaussian mixture model (GMM) for clustering (see, e.g., [4]). Figure 1.1 models a 2D toy data with 5 Gaussians, in which each Gaussian is denoted as a blue ellipse to visulize the mean (the center of the ellipse) and covariance (the shape of the ellipse). Clearly, we can see that there are 5 clusters, and each Gaussian denotes one cluster.

As the output of model-based clustering, we obtain *soft* cluster assignments for every data point, as well as the soft weight for each cluster. That is, model-based clustering is a *soft clustering*. Sometimes this property is desired because we can model the uncertainty of the whole corpus, and the model is generalizable to new data points as long as they follow the same distributional assumptions. However, these assumptions can sometimes be restrictive, and the model performs not very well if any of the assumptions is violated. For data with high dimensionality like images and texts, model-based clustering may lead to overfitting, and this is why some researchers project the data into some low-dimensional space first (using, e.g., principal component analysis [46]), and then train a clustering model on that space.

The learning algorithm for model-based clustering is just standard parameter estimation for probabilistic models. The expectation-maximization (EM) [18] is one popular approach since it is easily understood and reasonable fast. In EM we randomly initialize

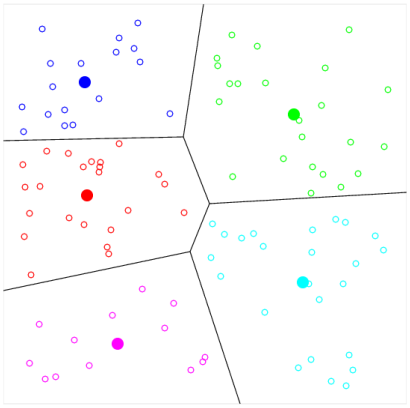


Figure 1.2: Illustration of a k-means clustering on a 2D toy data.

the cluster-specific parameters (e.g., mean and covariance for each component in a GMM), and then iterate the E-step in which soft data-cluster memberships are estimated, and M-step in which cluster-specific parameters are optimized. The data likelihood is guaranteed to increase along EM iterations until convergence.

1.1.2 Similarity-based Clustering

While model-based clustering is soft and theoretically sound, similarity-based approaches are widely applied due to its simplicity. Here one defines a similarity function (or reversely, a dissimilarity or distance function) for every pair of data points, and then groups similar data points together or splits the whole corpus into dissimilar subsets to form clusters. The different algorithms differ in similarity definitions or the way of grouping and splitting.

The k-means algorithm originally belongs to similarity-based methods, where Euclidean distance is used as the distance function. The number of clusters K is pre-defined, and each data point is assigned to the cluster whose center is the closest to this data point. Then the cluster centers are re-calculated, and the memberships are re-assigned until these two steps do not change the current status. After the convergence we have a *partition* of the whole corpus, as can be seen from the cluster boundaries in Figure 1.2. Each cluster is represented through the center point which is the sample mean of the data points belonging to this cluster.

Euclidean distance can only lead to isotropic clusters. If local distance functions are used, e.g., geodesic distance which is calculated along the surface of the data manifold, clustering can be formulated as finding a partitioning or a *cut* of the adjacency graph. This technique is called *spectral clustering* and can lead to arbitrarily shaped clusters [66]. Different cut criteria lead to different spectral clustering algorithms, and all of them turn out to solve a certain (generalized) eigenvalue problem after relaxing the constraints.

Another way of considering local similarity is to use the notion of *density*, which

measures the local connectivity and counts the number of neighbors surrounding each data point with a certain distance. One good example is the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm [23] which explicitly explores the densities and defines one cluster as a maximal set of density-connected points. Two parameters, the maximal radius of the neighborhood and the minimum number of points in the neighborhood, help to define the density connectivity. The algorithm is very efficient and can also learn arbitrarily shaped clusters, but the drawback is that it is not applicable to high dimensional data due to curse of dimensionality.

A common problem of similarity-based methods is that we can only obtain a *hard* cluster assignment for each document. No uncertainty of the data is modeled, and there is no principled way to calculate the weight of obtained clusters.

Remark 1.1.1. *It is shown that the popular k-means algorithm implicitly assumes a Gaussian mixture model for the data, where all the mixture components have the same isotropic covariance matrices [8]. The iterative learning algorithm for k-means is actually an EM algorithm, except that hard memberships (instead of soft ones) are calculated in the E-step, via a* winner-take-all *strategy.*

1.2 Hierarchical Clustering

Both model-based and similarity-based flat clustering algorithms can be extended to hierarchical clustering, but similarity-based methods are more popular. According to the way in which the hierarchy is built, i.e., top-down or bottom-up, hierarchical clustering algorithms can be divided into *partitioning methods* and *agglomerative methods*.

A flat partitioning method relies on some pre-defined criteria functions and partitions the corpus into a pre-given number of subsets. For top-down hierarchical clustering, one or several big clusters are chosen at each level of the hierarchy, and a flat partitioning algorithm such as k-means or spectral clustering is run to partition each of these clusters. Since we can stop at a certain level of the hierarchy, partitioning methods are usually fast and need less space. But it has to choose a number of clusters at each level.

Unlike partitioning methods which are top-down, agglomerative methods are bottom-up and each time merge two nearby clusters until one big cluster is obtained at the top of the hierarchy. The similarity-based agglomerative methods are also called *linkage* methods and have many variants depending on how to define a similarity function $\mathrm{sim}(c_i, c_j)$ for two clusters. In *single-linkage* this is the maximal point-wise similarity of the two clusters, and in *complete-linkage* it is the minimal one:

$$\mathrm{sim}_s(c_i, c_j) = \max_{x \in c_i, y \in c_j} \{\mathrm{sim}(x, y)\}, \tag{1.1}$$

$$\mathrm{sim}_c(c_i, c_j) = \min_{x \in c_i, y \in c_j} \{\mathrm{sim}(x, y)\}. \tag{1.2}$$

A compromise of these two is *average-linkage* and defines the similarity to be the average

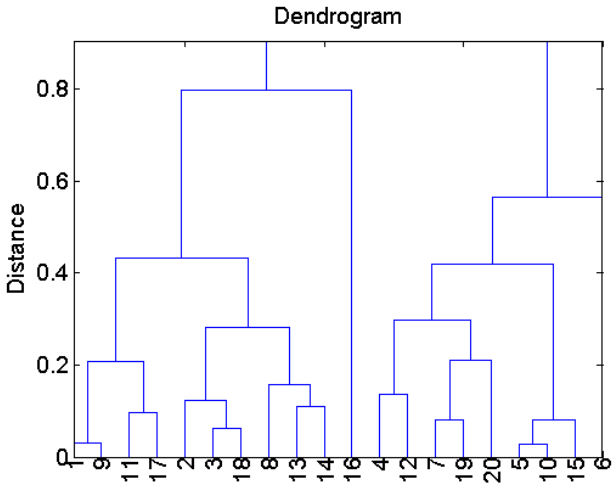


Figure 1.3: Illustration of a dendrogram for hierarchical clustering.

element-wise similarities of the two clusters:

$$\mathrm{sim}_a(c_i, c_j) = \frac{\sum_{x \in c_i, y \in c_j} \mathrm{sim}(x, y)}{|c_i| \cdot |c_j|}, \tag{1.3}$$

where $|c|$ denotes the number of data points that belong to cluster c. Other similarity measures are also available [48]. Agglomerative methods have the advantage that we do not need to pre-define any cluster numbers, but they have time complexity $\mathcal{O}(N^2 \log N)$ and space complexity $\mathcal{O}(N^2)$ for a corpus with N data points, which are very high and make these methods hard to apply for large-scale problems.

One way of visualizing the results of hierarchical clustering algorithms is to organize the clusters hierarchically as a tree. This kind of diagram is sometimes called a *dendrogram*, and in Figure 1.3 we show a dendrogram for complete-linkage on 20 randomly sampled 2D data points. The indices of the data are illustrated horizontally, and the distance of each merge is shown vertically. This provides a very sensible way for cluster analysis.

Other hierarchical clustering models include, e.g., [1, 83]. There also exists some work on extending model-based flat clustering to hierarchical clustering [31, 38], but one has to make assumptions for splitting or merging, and the scalability of these methods is not very good.

1.3 Organization of the Following Chapters

In the following two chapters we investigate the model-based clustering approach and propose a general mixture modeling framework from Bayesian perspective. Unlike most of the research works which only focus on mixture of Gaussian distributions, in Chapter 2

we consider more general settings and give solutions to mixture of exponential family distributions. A variational Bayesian algorithm is introduced for learning which fits each component distribution and the soft data-to-component memberships iteratively, in the same manner as the k-means algorithm.

Identifying the number of clusters is an important problem in clustering analysis. In Chapter 3 we extend the finite mixture models and build connections to non-parametric Bayesian models with Dirichlet process (DP) priors. This allows us to use the same variational solution as in Chapter 2 to obtain a sparse mixture model, which indicates that we can automatically learn this number from the data. A single hyperparameter is seen to control the sparsity of the mixture modeling.

Chapter 2

Finite Mixture Models for Clustering

In this chapter we introduce the well-known finite mixture modeling framework for data clustering. As discussed in Chapter 1, one cluster denotes a group of similar data points, and different similarities lead to different clustering structures. From Bayesian perspective, one cluster is interpreted as one *component distribution* of a mixture model, and the data points which belong to this cluster are assumed to be sampled from this same component distribution. This indicates that finite mixture model defines a *generative process* for data with clustering structure. Under this interpretation, learning clustering structure reduces to the following two steps in mixture model learning:

- **Parameter Estimation**: Learn the parameters for each component distribution. This is to learn the structure (e.g., position, shape, size) of each cluster.
- **Inference**: Predict for each data point which component it belongs to. By this means we can cluster newly observed data points.

It can be shown that the similarity measure used for clustering is implicitly defined in a mixture model, which mainly depends on the distributional assumption of each component. The clustering structure induced by a mixture model is *soft*, which means one data point belongs to each cluster with a certain weight. This is useful in many applications where, for instance, we need to sort the data points within each cluster.

We consider finite mixture modeling in this chapter, i.e., there are a fixed number of components in the mixture. This corresponds to learning a fixed number of clusters for the given data. In Chapter 3 we will relax this limitation and let the system automatically adapt this number. To show the flexibility of mixture models, we assume each component distribution belongs to the very general exponential family (Section 2.1), and derive parameter estimation and inference algorithms for mixture of exponential family distributions (Section 2.2). Special models of mixture of Gaussians and mixture of Multinomials are illustrated for continuous data and discrete data, respectively.

2.1 Mixture of Exponential Family Distributions

We start with the definitions of exponential family distributions and the conjugate family, and then introduce a Bayesian framework for mixture of exponential family distributions. Mixture of Gaussians and mixture of Multinomials are illustrated as two examples.

2.1.1 Exponential Family Distributions and the Conjugate Family

A probability distribution of $\mathbf{x} \in \mathcal{X}$ given parameters $\boldsymbol{\theta}$ is in the *exponential family* if it takes the form

$$P(\mathbf{x}|\boldsymbol{\theta}) = h(\mathbf{x}) \exp\left\{\boldsymbol{\theta}^\top \phi(\mathbf{x}) - A(\boldsymbol{\theta})\right\}, \tag{2.1}$$

where $\boldsymbol{\theta} \in \Theta$ are called the *natural parameters*, and $\phi(\mathbf{x})$ the *sufficient statistics*. The quantity $A(\boldsymbol{\theta})$, known as the *log partition function*, is defined as a normalization factor independent of $\mathbf{x}$:

$$A(\boldsymbol{\theta}) = \log \int_{\mathcal{X}} \exp\left\{\boldsymbol{\theta}^\top \phi(\mathbf{x})\right\} h(\mathbf{x})\, d\mathbf{x}.$$

It is well-known that $A(\boldsymbol{\theta})$ plays an important role for exponential family distributions. In particular, it can be identified as the *cumulant generating function* of $\phi(\mathbf{x})$. For instance, its first derivative gives the expectation (i.e., the first moment) of the sufficient statistics:

$$\frac{\partial A(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbb{E}_{\boldsymbol{\theta}}[\phi(\mathbf{x})] := \int_{\mathcal{X}} \phi(\mathbf{x}) P(\mathbf{x}|\boldsymbol{\theta})\, d\mathbf{x}, \tag{2.2}$$

where notation $\mathbb{E}_{\boldsymbol{\theta}}[\phi(\mathbf{x})]$ denotes the expectation of $\phi(\mathbf{x})$ with respect to distribution $P(\mathbf{x}|\boldsymbol{\theta})$. This can be easily verified from the definition of $A(\boldsymbol{\theta})$. Exponential family covers many well-known distributions such as Bernoulli, Poisson and Gaussian. Table 2.1 lists some of them with domains $\mathcal{X}$ and Θ.

From the Bayesian perspective, we need to assign a *prior distribution* to the natural parameter $\boldsymbol{\theta}$. One such prior is defined as follows and is called the *conjugate family* in the literature:

$$P(\boldsymbol{\theta}|\mu, \nu) = \frac{1}{Z(\boldsymbol{\theta}, \mu, \nu)} \exp\left\{\boldsymbol{\theta}^\top \mu - \nu A(\boldsymbol{\theta})\right\}. \tag{2.3}$$

Here (μ, ν) are called the *hyperparameters*, i.e., the parameters for the prior distribution, with μ having dimensionality $\texttt{dim}(\boldsymbol{\theta})$ and ν a scalar. $Z(\boldsymbol{\theta}, \mu, \nu)$ is a normalization factor such that the distribution is well-defined, i.e., $\int P(\boldsymbol{\theta}|\mu, \nu)\, d\boldsymbol{\theta} = 1$. This distribution is *conjugate* to exponential family in the sense that, after we observe a set of *i.i.d.* data points $\{\mathbf{x}_i\}_{i=1}^N$, the *a posteriori* distribution of $\boldsymbol{\theta}$ takes the same parametric form as the prior. This can be shown by applying Bayes' rule as:

$$P(\boldsymbol{\theta}|\{\mathbf{x}_i\}_{i=1}^N, \mu, \nu) \propto \prod_{i=1}^N P(\mathbf{x}_i|\boldsymbol{\theta}) P(\boldsymbol{\theta}|\mu, \nu) \propto \exp\left\{\boldsymbol{\theta}^\top \left(\mu + \sum_{i=1}^N \phi(\mathbf{x}_i)\right) - (\nu + N) A(\boldsymbol{\theta})\right\}, \tag{2.4}$$

Table 2.1: Examples of exponential family distributions. For some distributions we use canonical notations instead of $\mathbf{x}$ and $\boldsymbol{\theta}$.

Family	Definition $P(\mathbf{x}\|\boldsymbol{\theta})$	$\mathcal{X}$	Θ
Bernoulli	$\theta^x(1-\theta)^{1-x}$	$\{0,1\}$	$[0,1]$
Binomial	$\binom{n}{x}\theta^x(1-\theta)^{n-x}$	$\{0,1,\ldots,n\}$	$[0,1]$
Poisson	$\frac{1}{x!}\lambda^x \exp(-\lambda)$	$\{0,1,2,\ldots\}$	$\lambda > 0$
Exponential	$\beta \exp(-\beta x)$	$(0,+\infty)$	$\beta > 0$
Beta	$\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}$	$[0,1]$	$\alpha > 0, \beta > 0$
Gamma	$\frac{\beta^\alpha}{\Gamma(\alpha)}x^{\alpha-1}\exp(-\beta x)$	$(0,+\infty)$	$\alpha > 0, \beta > 0$
Multinomial $\mathbf{x} \sim \texttt{Mult}(\boldsymbol{\theta})$	$\frac{n!}{\prod_i x_i!}\prod_i \theta_i^{x_i}$	$x_i \in \{0,1,\ldots,n\}$, $\sum_i x_i = n$	$\theta_i \in (0,1)$, $\sum_i \theta_i = 1$
Dirichlet $\mathbf{x} \sim \texttt{Dir}(\boldsymbol{\alpha})$	$\frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)}\prod_i x_i^{\alpha_i - 1}$	$x_i \in (0,1)$, $\sum_i x_i = 1$	$\alpha_i > 0$
Gaussian $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	$(2\pi)^{-d/2}\lvert\boldsymbol{\Sigma}\rvert^{-1/2}$ $\times \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$	$\mathbb{R}^d$	$\boldsymbol{\mu} \in \mathbb{R}^d$, $\mathbf{0} \prec \boldsymbol{\Sigma} \in \mathbb{R}^{d\times d}$
Wishart $\mathbf{W} \sim \mathcal{W}(\mathbf{S}, \nu)$	$\left(2^{\nu k/2}\pi^{k(k-1)/4}\prod_{i=1}^k \Gamma(\frac{\nu+1-i}{2})\right)^{-1}\lvert\mathbf{S}\rvert^{-\nu/2}$ $\times\lvert\mathbf{W}\rvert^{(\nu-k-1)/2}\exp\left(-\frac{1}{2}\texttt{trace}\left(\mathbf{S}^{-1}\mathbf{W}\right)\right)$	$\mathbf{0} \prec \mathbf{W} \in \mathbb{R}^{k\times k}$	$\nu > 0$, $\mathbf{0} \prec \mathbf{S} \in \mathbb{R}^{k\times k}$

where in the posterior we have the new parameters $\hat{\mu} = \mu + \sum_{i=1}^N \phi(\mathbf{x}_i)$ and $\hat{\nu} = \nu + N$. Therefore, μ and ν are sometimes viewed as the *pseudo-counts* (i.e., pseudo-observations without any samples) of sufficient statistics for exponential family distributions.

The conjugate family defined in (2.3) is *minimal* in the sense that it requires the minimal number of hyperparameters. In this chapter we make a further assumption and write the conjugate family as

$$P(\boldsymbol{\theta}|\mu,\nu) = g(\boldsymbol{\theta}) \exp\left\{(\mu^\top, \nu)\begin{pmatrix}\boldsymbol{\theta}\\ -A(\boldsymbol{\theta})\end{pmatrix} - B(\mu,\nu)\right\}, \tag{2.5}$$

which basically means the normalization term in (2.3) contains a factorized form for $\boldsymbol{\theta}$ and the hyperparameters. Now the conjugate family is also in the exponential family, with sufficient statistics $\binom{\boldsymbol{\theta}}{-A(\boldsymbol{\theta})}$ and natural parameter $\binom{\mu}{\nu}$. Applying (2.2) in this case yields

$$\frac{\partial B(\mu,\nu)}{\partial \mu} = \mathbb{E}_{\mu,\nu}[\boldsymbol{\theta}], \qquad \frac{\partial B(\mu,\nu)}{\partial \nu} = \mathbb{E}_{\mu,\nu}[-A(\boldsymbol{\theta})]. \tag{2.6}$$

These results turn out to be useful for subsequent calculations. This assumption is not strong and turns out to be true for many well-known exponential-conjugate family pairs. We list some of them in Table 2.2 for clarity.

Table 2.2: The conjugate family for some exponential family distributions.

Exponential Family	Notation	Conjugate Family	Notation
Binomial	$x \sim \text{Bi}(\theta)$	Beta	$\theta \sim \texttt{Beta}(\alpha, \beta)$
Multinomial	$\mathbf{x} \sim \texttt{Mult}(\boldsymbol{\theta})$	Dirichlet	$\boldsymbol{\theta} \sim \texttt{Dir}(\boldsymbol{\alpha})$
Gaussian (fix $\boldsymbol{\Sigma}$)	$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_0)$	Gaussian	$\boldsymbol{\mu} \sim \mathcal{N}(\mathbf{m}, \mathbf{C})$
Gaussian (fix $\boldsymbol{\mu}$)	$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma})$	(Inverse) Wishart	$\boldsymbol{\Sigma}^{-1} \sim \mathcal{W}(\mathbf{S}, \nu)$
Gaussian	$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian-Wishart	$(\boldsymbol{\mu}, \boldsymbol{\Sigma}^{-1}) \sim \mathcal{N}_{\boldsymbol{\mu}}(\mathbf{m}, \frac{1}{\beta}\boldsymbol{\Sigma})\, \mathcal{W}_{\boldsymbol{\Sigma}^{-1}}(\mathbf{S}, \nu)$

2.1.2 Mixture of Exponential Family Distributions

In mixture modeling, each data point is sampled from a fixed but unknown *component distribution*, which we assume belongs to exponential family in this chapter. For finite mixture modeling we fix the number of components to be K, a finite positive integer. We also focus on the case that all the component distributions take the same form, for instance, Gaussian. Then the likelihood of N *i.i.d.* data points $\mathcal{D} := \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ is formally written as

$$P(\mathcal{D}|\boldsymbol{\pi}, \boldsymbol{\Theta}) = \prod_{i=1}^{N} \sum_{k=1}^{K} P(c_i = k|\boldsymbol{\pi}) P(\mathbf{x}_i|\boldsymbol{\theta}_k), \tag{2.7}$$

where $P(c_i = k|\boldsymbol{\pi})$ is a Multinomial distribution with parameters $\boldsymbol{\pi}$, and $P(\mathbf{x}_i|\boldsymbol{\theta}_k)$ takes the general form (2.1). The K-dimensional vector $\boldsymbol{\pi} := \{\pi_k\}_{k=1}^K$ gives the weights for the component distributions and sums to 1, i.e., $\sum_{k=1}^K \pi_k = 1$. The other parameter $\boldsymbol{\Theta} := \{\boldsymbol{\theta}_k\}_{k=1}^K$ contain the natural parameters of all component distributions. c_i is seen as a random variable of *indicator* for data $\mathbf{x}_i$, saying which of the K components $\mathbf{x}_i$ is sampled from. From the definition of Multinomial distribution we know that $P(c_i = k|\boldsymbol{\pi})$ is simply π_k. Finite mixture model as in (2.7) is in general not a member of exponential family.

To complete the model from Bayesian perspective, we assign priors to all the parameters. For $\boldsymbol{\Theta}$ we assign conjugate prior (2.5) to each $\boldsymbol{\theta}_k$ independently, with the same hyperparameters (μ_0, ν_0):

$$P(\boldsymbol{\Theta}|\mu_0, \nu_0) = \prod_{k=1}^{K} \left\{ g(\boldsymbol{\theta}_k) \exp\left\{ \boldsymbol{\theta}_k^\top \mu_0 - \nu_0 A(\boldsymbol{\theta}_k) - B(\mu_0, \nu_0) \right\} \right\}. \tag{2.8}$$

From the perspective of hierarchical Bayesian modeling, this essentially means that each component in the mixture is *not independent* of other components, but *correlated* in the sense of a common prior. Information from other components can therefore be integrated out for inference in a specific component.

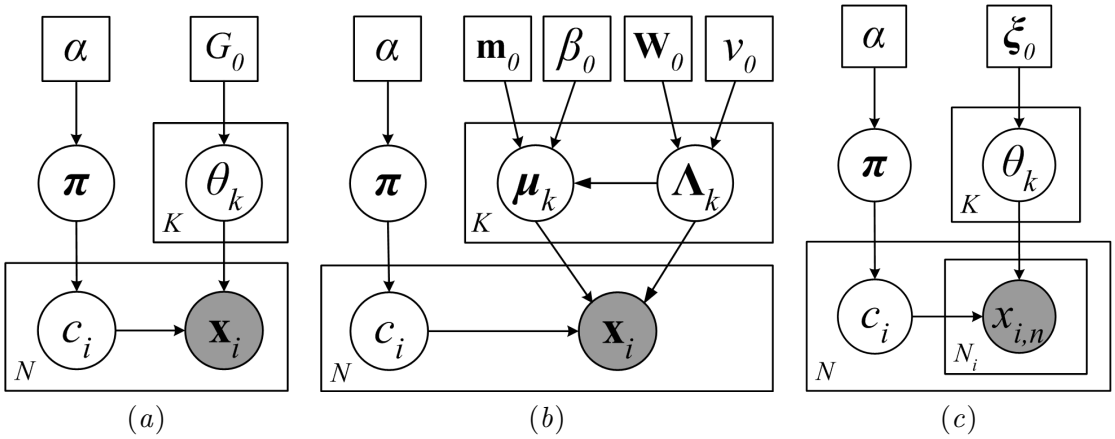


Figure 2.1: Plate models of proposed mixture models. Rectangles (with a number on the bottom-left corner) indicate independent sampling with that specific number of samples, and hidden variables and model parameters are denoted as circles and squares, respectively. Observed quantities are marked in gray. Black arrows denote statistical dependency. (a) shows mixture of exponential family distributions in general, where G_0 denotes the conjugate family distribution (2.5); (b) and (c) show mixture of Gaussians and mixture of Multinomials, respectively.

For the Multinomial parameters $\boldsymbol{\pi}$, we assign a symmetric Dirichlet prior with a (scalar) hyperparameter α:

$$P(\boldsymbol{\pi}|\alpha) = \frac{\Gamma(\alpha)}{[\Gamma(\frac{\alpha}{K})]^K} \prod_{k=1}^{K} \pi_k^{\frac{\alpha}{K}-1}, \tag{2.9}$$

where $\Gamma(\cdot)$ is the Gamma function. That is, we define $\boldsymbol{\pi} \sim \texttt{Dir}(\frac{\alpha}{K}, \ldots, \frac{\alpha}{K})$. Here we make a constraint that all the parameters in this Dirichlet are the same and sum to a scalar that is independent of K, the number of components in the mixture.

With these priors, the final data likelihood can be obtained by integrating out the *latent variables* $\boldsymbol{\pi}$ and $\boldsymbol{\Theta}$:

$$P(\mathcal{D}|\alpha, \mu_0, \nu_0) = \int_{\boldsymbol{\pi}} P(\boldsymbol{\pi}|\alpha) \int_{\boldsymbol{\Theta}} P(\boldsymbol{\Theta}|\mu_0, \nu_0) \left\{ \prod_{i=1}^{N} \sum_{k=1}^{K} P(c_i = k|\boldsymbol{\pi}) P(\mathbf{x}_i|\boldsymbol{\theta}_k) \right\} d\boldsymbol{\Theta}\, d\boldsymbol{\pi}. \tag{2.10}$$

The model has two parameters: α is a positive scalar, and (μ_0, ν_0) has dimensionality $\texttt{dim}(\phi(\mathbf{x})) + 1$. The plate model is illustrated in Figure 2.1(a), where G_0 denotes the conjugate prior distribution (2.5).

Remark 2.1.1. *The mixture model defined above takes a Bayesian perspective: The mixture parameters $\boldsymbol{\pi}$ and $\boldsymbol{\Theta}$ are assigned hyperpriors and are integrated out in the likelihood model. This is in contrast to the* frequentist *view of mixture modeling, in which the model*

depends on parameters $\boldsymbol{\pi}$ and $\boldsymbol{\Theta}$ only. We will mention their difference in terms of learning and inference in subsequent sections.

Remark 2.1.2. *The Dirichlet hyperprior for $\boldsymbol{\pi}$ has parameter at each dimension $\frac{\alpha}{K}$ instead of α. This will make the model easily generalize to infinite mixture models and connect to Dirichlet process mixture models. This point will be clarified in Chapter 3.*

2.1.3 Examples

Here we give two concrete examples of the proposed mixture model. The first one is mixture of Gaussians for continuous data, and the second one is mixture of Multinomials for discrete data.

Mixture of Gaussians

Mixture of Gaussians is perhaps the most studied mixture model in the literature. Different Gaussian mixture models exist depending on different modeling complexity. Here we illustrate the most flexible Gaussian mixture model and assume a complete conjugate prior to both the means and covariances. This model is studied, e.g., in [3, 16]. For easy understanding, we do not stick to the canonical form given in (2.1) and (2.5) for Gaussian distributions.

Suppose we have K Gaussians in the mixture, and the k-th Gaussian is associated with mean vector $\boldsymbol{\mu}_k$ and precision matrix $\boldsymbol{\Lambda}_k$, where the precision matrix is the matrix inverse of the covariance matrix $\boldsymbol{\Sigma}_k$.[1] Let the dimensionality of the feature space be d, i.e., $\mathcal{X} \subset \mathbb{R}^d$, then $\boldsymbol{\mu}_k$ is a column vector of length d, and $\boldsymbol{\Lambda}_k$ is a $d \times d$ positive definite symmetric matrix. Following the notations in Table 2.1 we denote the k-th Gaussian as $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})$. Then the complete generative model for mixture of Gaussians is shown as follows:

$$
\begin{aligned}
\mathbf{x}_i | c_i, \boldsymbol{\Theta} &\overset{\text{ind}}{\sim} \mathcal{N}(\boldsymbol{\mu}_{c_i}, \boldsymbol{\Lambda}_{c_i}^{-1}), \quad i = 1, \ldots, N \\
c_i | \boldsymbol{\pi} &\overset{\text{iid}}{\sim} \texttt{Mult}(\boldsymbol{\pi}), \quad c_i = 1, \ldots, K \\
\boldsymbol{\mu}_k | \boldsymbol{\Lambda}_k, \mathbf{m}_0, \beta_0 &\overset{\text{ind}}{\sim} \mathcal{N}(\mathbf{m}_0, \beta_0^{-1} \boldsymbol{\Lambda}_k^{-1}), \quad k = 1, \ldots, K \\
\boldsymbol{\Lambda}_k | \mathbf{W}_0, \nu_0 &\overset{\text{iid}}{\sim} \mathcal{W}(\mathbf{W}_0, \nu_0), \\
\boldsymbol{\pi} | \alpha &\sim \texttt{Dir}\left(\frac{\alpha}{K}, \ldots, \frac{\alpha}{K}\right),
\end{aligned}
$$

where $\texttt{Mult}$, $\mathcal{W}$ and $\texttt{Dir}$ follow the distribution notations in Table 2.1. We follow statistics conventions and use $\overset{\text{ind}}{\sim}$ and $\overset{\text{iid}}{\sim}$ for independently sampling and *i.i.d.* sampling, respectively. The natural parameters $\boldsymbol{\Theta}$ include all $\boldsymbol{\mu}_k$ and $\boldsymbol{\Lambda}_k$, $k = 1, \ldots, K$. All the hyperparameters

[1] Note that we follow the convention and use bold face $\boldsymbol{\mu}$ to denote the mean of a Gaussian. This should be distinguished from the normal form μ which is used along with ν as the hyperparameters in the conjugate family.

for the model include $\alpha, \mathbf{m}_0, \beta_0, \mathbf{W}_0$ and ν_0. Here $\mathbf{m}_0$ is a length-d column vector, $\mathbf{W}_0$ is a $d \times d$ positive definite symmetric matrix, and α, β_0, ν_0 are positive scalars. The plate model is shown in Figure 2.1(b).

As is well-known for Gaussians, the mean vector and covariance matrix have clear geometrical interpretations. In terms of Gaussian mixture model for clustering, the mean vector of each Gaussian component denotes the center of each cluster, and the covariance matrix determines the (elliptical) shape and coverage of each cluster. The Gaussian mixture model proposed above allows an arbitrarily shaped Gaussian for each cluster, and is thus very flexible.

In Figure 2.2 we show a toy data in two-dimensional space from a mixture of 5 Gaussians. 50 data points are sampled from each Gaussian, so the true weights of all the Gaussians are equal, i.e., $\boldsymbol{\pi} = \{0.2, 0.2, 0.2, 0.2, 0.2\}$. The mean vectors and covariance matrices of these Gaussians are

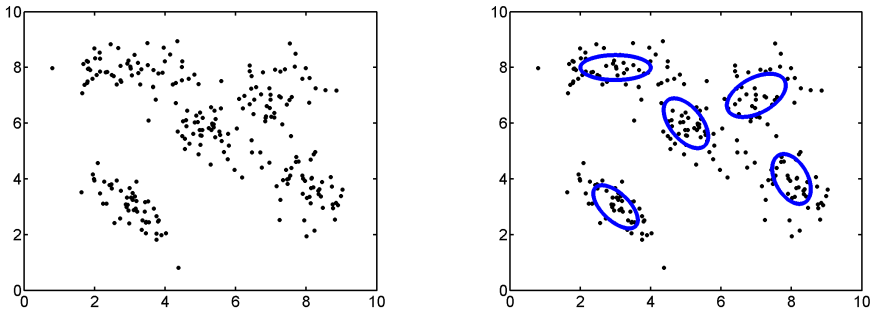
$$\boldsymbol{\mu}_1 = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \boldsymbol{\Sigma}_1 = \begin{pmatrix} 0.4 & -0.3 \\ -0.3 & 0.6 \end{pmatrix}, \qquad \boldsymbol{\mu}_2 = \begin{pmatrix} 3 \\ 8 \end{pmatrix}, \boldsymbol{\Sigma}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 0.2 \end{pmatrix},$$
$$\boldsymbol{\mu}_3 = \begin{pmatrix} 5 \\ 6 \end{pmatrix}, \boldsymbol{\Sigma}_3 = \begin{pmatrix} 0.4 & -0.3 \\ -0.3 & 0.8 \end{pmatrix}, \qquad \boldsymbol{\mu}_4 = \begin{pmatrix} 7 \\ 7 \end{pmatrix}, \boldsymbol{\Sigma}_4 = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.6 \end{pmatrix},$$
$$\boldsymbol{\mu}_5 = \begin{pmatrix} 8 \\ 4 \end{pmatrix}, \boldsymbol{\Sigma}_5 = \begin{pmatrix} 0.3 & -0.2 \\ -0.2 & 0.8 \end{pmatrix}.$$

To show these Gaussians graphically, we draw each of them in Figure 2.2(b) with a blue ellipse which shows the mean vector and standard deviations. It is seen that these Gaussians are somehow mixed together in the boundaries, which makes the problem a bit harder.

Remark 2.1.3. *One can also define simpler Gaussian mixture models to reduce modeling complexity. For instance, one can fix a same covariance matrix for all the Gaussians. This corresponds to a clustering model in which all the clusters share the same shape and coverage but have different centers. Another popular choice is to assume that the covariance matrix takes a diagonal form. This constrains each axis of the cluster ellipsoid to be parallel to one of the coordinate axes.*

Mixture of Multinomials

Mixture of Multinomials has been widely applied to model discrete data. For instance in language modeling, it is used to model a text corpus of documents with discrete word occurrences. This is also known as mixture of unigrams, and each mixture component is called a *topic*. In this model, the k-th topic takes a Multinomial distribution over all the words, and the natural parameter $\boldsymbol{\theta}_k$ denotes these word-generating probabilities. Let V be the number of words in the vocabulary set $\mathcal{V}$, then $\boldsymbol{\theta}_k$ is a column vector of length V and satisfies $\sum_{v=1}^{V} \theta_{k,v} = 1$, for all $k = 1, \ldots, K$. Denote N_i the number of words in



(a) (b)

Figure 2.2: A toy 2D data with a mixture of 5 Gaussians. 50 data points are sampled from each Gaussian. Each Gaussian is drawn with a blue ellipse in (b) to show the truth.

document $\mathbf{x}_i$, the generative model for mixture of Multinomials is as follows:

$$\begin{aligned}
\mathbf{x}_i(n)|c_i, \boldsymbol{\Theta} &\overset{\text{ind}}{\sim} \texttt{Mult}(\boldsymbol{\theta}_{c_i}), \quad n = 1, \ldots, N_i \\
c_i|\boldsymbol{\pi} &\overset{\text{iid}}{\sim} \texttt{Mult}(\boldsymbol{\pi}), \quad c_i = 1, \ldots, K \\
\boldsymbol{\theta}_k|\boldsymbol{\xi}_0 &\overset{\text{iid}}{\sim} \texttt{Dir}(\boldsymbol{\xi}_0), \quad k = 1, \ldots, K \\
\boldsymbol{\pi}|\alpha &\sim \texttt{Dir}\left(\frac{\alpha}{K}, \ldots, \frac{\alpha}{K}\right),
\end{aligned}$$

where the first line is sometimes written more concisely as $\mathbf{x}_i|c_i, \boldsymbol{\Theta} \sim \texttt{Mult}(\boldsymbol{\theta}_{c_i}, N_i)$. Here the hyperparameters are the positive scalar α and the Dirichlet parameter $\boldsymbol{\xi}_0$ of length V. The plate model for Multinomial mixtures is illustrated in Figure 2.1(c).

Remark 2.1.4. *In this model we do not explicitly model N_i for each document $\mathbf{x}_i$ and instead assume it is known and fixed. One can extend this model by assigning a same Poisson prior to all these N_i's.*

2.2 Model Inference and Parameter Estimation

For inference in the proposed mixture model, we need to calculate the *a posteriori* distribution of all the latent variables given observations and model parameters. This can be done via Bayes' rule as

$$\begin{aligned}
P(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c}|\mathcal{D}, \alpha, \mu_0, \nu_0) &= \frac{P(\mathcal{D}|\boldsymbol{\Theta}, \mathbf{c}) \cdot P(\boldsymbol{\Theta}|\mu_0, \nu_0) \cdot P(\mathbf{c}|\boldsymbol{\pi}) \cdot P(\boldsymbol{\pi}|\alpha)}{P(\mathcal{D}|\alpha, \mu_0, \nu_0)} \\
&= \frac{\prod_{i=1}^{N} P(\mathbf{x}_i|\boldsymbol{\theta}_{c_i}) \cdot \prod_{k=1}^{K} P(\boldsymbol{\theta}_k|\mu_0, \nu_0) \cdot \prod_{i=1}^{N} \pi_{c_i} \cdot P(\boldsymbol{\pi}|\alpha)}{P(\mathcal{D}|\alpha, \mu_0, \nu_0)},
\end{aligned}$$

where $\mathbf{c}$ denote the indicator variables $\{c_1, \ldots, c_N\}$, one for each data point. This requires us to calculate the big integral $P(\mathcal{D}|\alpha, \mu_0, \nu_0)$ (as given in (2.10)), which is however

intractable. If one would like to optimize the model parameters α and (μ_0, ν_0), he/she would have to deal with this big integral as well. This is a common problem for Bayesian analysis, and one popular solution in statistics is to perform Markov chain Monte Carlo (MCMC) sampling like Gibbs sampling to approximate the integral. The nice property of MCMC is that the approximated posterior is guaranteed to converge to the true one if we have enough samples, but in practice this could be very slow due to model complexity or bad starting points.

For the proposed mixture model, we first describe a straightforward Gibbs sampling method for model inference, and then focus on a fast approximation method called the *variational Bayesian* method. It will be seen that the variational Bayesian solution is like a deterministic version of the corresponding Gibbs sampler. We then discuss the connection of this learning algorithm to clustering, and give the solutions to mixture of Gaussians and mixture of Multinomials.

2.2.1 Gibbs Sampling

The basic idea of Gibbs sampling is to sequentially sample one latent variable at a time to collect a set of samples from the joint distribution [28]. This normally leads to much easier sampling process for each variable. Therefore, all we should do is to derive the *a posteriori* distribution of each latent variable given the data and all the other variables. This can be easily done via Bayes' rule for the proposed model. Since we also need to learn the hyperparameters, for Gibbs sampling we should also assign priors to α and (μ_0, ν_0) and sample them as well. The current joint distribution of the posterior is therefore $P(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c}, \alpha, \mu_0, \nu_0 | \mathcal{D})$. In Gibbs sampling we choose a starting point for all these variables, and then repeat the following sampling process until a stable situation is achieved:

1. $(\boldsymbol{\Theta}|\mu_0, \nu_0, \mathbf{c}, \mathcal{D})$: For $k = 1, \ldots, K$, draw $(\boldsymbol{\theta}_k|\mu_0, \nu_0, \mathbf{c}, \mathcal{D})$ from the density

$$\hat{P}(\boldsymbol{\theta}_k|\mu_0, \nu_0, \mathbf{c}, \mathcal{D}) \propto P(\boldsymbol{\theta}_k|\mu_0, \nu_0) \prod_{\{j:c_j=k\}} P(\mathbf{x}_j|\boldsymbol{\theta}_k). \tag{2.11}$$

2. $(\mathbf{c}|\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathcal{D})$: For $i = 1, \ldots, N$, draw values

$$\hat{P}(c_i|\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathcal{D}) \propto P(c_i|\boldsymbol{\pi}) P(\mathbf{x}_i|\boldsymbol{\theta}_{c_i}), \qquad c_i = 1, \ldots, K. \tag{2.12}$$

3. $(\boldsymbol{\pi}|\alpha, \mathbf{c})$: The posterior is calculated as $\hat{P}(\boldsymbol{\pi}|\alpha, \mathbf{c}) \propto P(\boldsymbol{\pi}|\alpha) P(\mathbf{c}|\boldsymbol{\pi})$. By the conjugacy of Dirichlet distribution to Multinomial, the posterior is now

$$\hat{P}(\boldsymbol{\pi}|\alpha, \mathbf{c}) \sim \texttt{Dir}\left(\frac{\alpha}{K} + \sum_{i=1}^{N} \delta_1(c_i), \ldots, \frac{\alpha}{K} + \sum_{i=1}^{N} \delta_K(c_i)\right), \tag{2.13}$$

where

$$\delta_k(c_i) = \begin{cases} 1 & \text{if } c_i = k, \\ 0 & \text{otherwise.} \end{cases}$$

4. $(\alpha|\boldsymbol{\pi})$: Assign a prior to α, and draw

$$\hat{P}(\alpha|\boldsymbol{\pi}) \propto P(\alpha)P(\boldsymbol{\pi}|\alpha).$$

5. $(\mu_0, \nu_0|\boldsymbol{\Theta})$: Assign a prior to (μ_0, ν_0), and draw

$$\hat{P}(\mu_0, \nu_0|\boldsymbol{\Theta}) \propto P(\mu_0, \nu_0) \prod_{k=1}^{K} P(\boldsymbol{\theta}_k|\mu_0, \nu_0).$$

All the distributions in this sampling process are known, and there exist efficient ways to obtain random samples from them. In (2.11) the posterior is still in the conjugate family, as seen from (2.4). In (2.12) we normalize such that $\sum_{c_i=1}^{K} \hat{P}(c_i|\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathcal{D}) = 1$. In Steps 4 and 5, the new priors $P(\alpha)$ and $P(\mu_0, \nu_0)$ really depend on prior knowledge and can in principal take any form.

2.2.2 Variational Bayesian Solution

The variational Bayesian algorithm is motivated by approximating the *a posteriori* distribution of latent variables with a tractable family, and then maximizing a lower-bound of the log data likelihood with respect to some *variational parameters* [47, 29]. For model (2.10), by applying Jensen's inequality we have the following lower-bound of the log likelihood $\log P(\mathcal{D}|\alpha, \mu_0, \nu_0)$:

$$\mathcal{L}(\mathcal{D}) \equiv \int_{\boldsymbol{\pi}} \int_{\boldsymbol{\Theta}} \sum_{c_1=1}^{K} \cdots \sum_{c_N=1}^{K} Q(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c}) \log \frac{P(\boldsymbol{\pi}|\alpha)P(\boldsymbol{\Theta}|\mu_0, \nu_0) \left\{\prod_{i=1}^{N} \pi_{c_i} P(\mathbf{x}_i|\boldsymbol{\theta}_{c_i})\right\}}{Q(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c})} \, d\boldsymbol{\Theta} \, d\boldsymbol{\pi}.$$

Here $Q(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c})$ can be any function of $\boldsymbol{\pi}$, $\boldsymbol{\Theta}$ and $\mathbf{c}$, and $\mathcal{L}(\mathcal{D})$ remains a lower bound for $\log P(\mathcal{D}|\alpha, \mu_0, \nu_0)$ regardless of any form for Q. To make the integral tractable, in variational Bayesian method we can assume a simpler family for Q, which is called a *variational distribution*. The most popular form is the *factorized* form, which constrains *independency* for all latent variables: each latent variable is independent of the others in this approximated *a posteriori* distribution. As will be seen shortly, this leads to tractable updates for inference. We therefore use the following variational distribution

$$Q(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c}|\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{\Psi}) = Q(\boldsymbol{\pi}|\boldsymbol{\lambda}) \prod_{k=1}^{K} Q(\boldsymbol{\theta}_k|\mu_k, \nu_k) \prod_{i=1}^{N} Q(c_i|\boldsymbol{\psi}_i)$$

to approximate the true posterior $P(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c}|\mathcal{D}, \alpha, \mu_0, \nu_0)$, with *variational parameters* $\boldsymbol{\lambda}$, $\boldsymbol{\mu} := \{\mu_k\}_{k=1}^{K}$, $\boldsymbol{\nu} := \{\nu_k\}_{k=1}^{K}$ and $\boldsymbol{\Psi} := \{\boldsymbol{\psi}_i\}_{i=1}^{N}$.[2] In principal we can use any distribution for each of these latent variables, but to facilitate learning later on we also use a conjugate distribution for each function Q on the right hand side. Here $Q(\boldsymbol{\pi}|\boldsymbol{\lambda})$ is K-dimensional

[2]Note that we use (μ_k, ν_k) to denote variational parameters for each $\boldsymbol{\theta}_k$ $(k > 0)$, and use (μ_0, ν_0) as the hyperparameters.

Dirichlet, $Q(\boldsymbol{\theta}_k|\mu_k,\nu_k)$ takes the conjugate form as in (2.5), and $Q(c_i|\boldsymbol{\psi}_i)$ is N-dimensional Multinomial. Due to the factorized form for $Q(\boldsymbol{\pi},\boldsymbol{\Theta},\mathbf{c})$, the lower-bound can now be written as:

$$\begin{aligned}\mathcal{L}(\mathcal{D}) = \mathbb{E}_Q[\log P(\boldsymbol{\pi}|\alpha)] + \sum_{k=1}^{K}\mathbb{E}_Q[\log P(\boldsymbol{\theta}_k|\mu_0,\nu_0)] + \sum_{i=1}^{N}\mathbb{E}_Q[\log P(c_i|\boldsymbol{\pi})] \\ + \sum_{i=1}^{N}\mathbb{E}_Q[\log P(\mathbf{x}_i|\boldsymbol{\Theta},c_i)] - \mathbb{E}_Q[\log Q],\end{aligned} \tag{2.14}$$

where all the expectations are with respect to variational distribution $Q(\boldsymbol{\pi},\boldsymbol{\Theta},\mathbf{c}|\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu},\boldsymbol{\psi})$. In the literature, variational Bayesian methods maximize this lower bound *only* with respect to variational parameters $\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu},\boldsymbol{\Psi}$, and thus fix the model parameters α,μ_0,ν_0 (see, e.g., [3, 29]). In this chapter we will however treat it as the E-step of the algorithm, and estimate the model parameters in the M-step.

Calculation of the Lower Bound

The lower bound (2.14) can be calculated directly using the properties of the exponential family distributions. We discuss them in detail in this subsection.

In the first term, the distribution $P(\boldsymbol{\pi}|\alpha)$ is given in (2.9). A directly calculation yields

$$\begin{aligned}\mathbb{E}_Q[\log P(\boldsymbol{\pi}|\alpha)] &= \mathbb{E}_Q\left[\log\Gamma(\alpha) - K\log\Gamma\left(\frac{\alpha}{K}\right)\right] + \left(\frac{\alpha}{K}-1\right)\sum_{k=1}^{K}\mathbb{E}_Q[\log\pi_k] \\ &= \log\Gamma(\alpha) - K\log\Gamma\left(\frac{\alpha}{K}\right) + \left(\frac{\alpha}{K}-1\right)\sum_{k=1}^{K}\mathbb{E}_{\mathtt{Dir}(\boldsymbol{\pi}|\boldsymbol{\lambda})}[\log\pi_k] \\ &= \log\Gamma(\alpha) - K\log\Gamma\left(\frac{\alpha}{K}\right) + \left(\frac{\alpha}{K}-1\right)\sum_{k=1}^{K}\left[\Psi(\lambda_k) - \Psi\left(\sum_{k'=1}^{K}\lambda_{k'}\right)\right],\end{aligned}$$

where $\Psi(\cdot)$ is the *digamma* function, the first derivative of the log Gamma function. In the second equality, the first expectation is simply the constant inside, and the second expectation turns out to need only the variational distribution $Q(\boldsymbol{\pi}|\boldsymbol{\lambda})$ which is a Dirichlet. The third equality comes from a direct calculation of the expectation using (2.2) for Dirichlet.

For the second term, we have, for each $k = 1,\ldots,K$,

$$\begin{aligned}\mathbb{E}_Q[\log P(\boldsymbol{\theta}_k|\mu_0,\nu_0)] &= \mathbb{E}_Q[\log g(\boldsymbol{\theta}_k)] + \mu_0^\top\mathbb{E}_Q[\boldsymbol{\theta}_k] + \nu_0\mathbb{E}_Q[-A(\boldsymbol{\theta}_k)] - B(\mu_0,\nu_0) \\ &= \mathbb{E}_Q[\log g(\boldsymbol{\theta}_k)] + \mu_0^\top\mathbb{E}_{\mu_k,\nu_k}[\boldsymbol{\theta}_k] + \nu_0\mathbb{E}_{\mu_k,\nu_k}[-A(\boldsymbol{\theta}_k)] - B(\mu_0,\nu_0),\end{aligned}$$

where in the second equality the expectation is with respect to the variational distribution $P(\boldsymbol{\theta}_k|\mu_k,\nu_k)$ and can be calculated using (2.6).

The third term uses the properties of Multinomial and Dirichlet distributions:

$$\begin{aligned}\sum_{i=1}^{N}\mathbb{E}_Q[\log P(c_i|\boldsymbol{\pi})] &= \sum_{i=1}^{N}\mathbb{E}_{\texttt{Mult}(c_i|\boldsymbol{\psi}_i)}\Big\{\mathbb{E}_{\texttt{Dir}(\boldsymbol{\pi}|\boldsymbol{\lambda})}[\log P(c_i|\boldsymbol{\pi})]\Big\}\\ &= \sum_{i=1}^{N}\sum_{k=1}^{K} Q(c_i=k|\boldsymbol{\psi}_i)\mathbb{E}_{\texttt{Dir}(\boldsymbol{\pi}|\boldsymbol{\lambda})}[\log \pi_k]\\ &= \sum_{i=1}^{N}\sum_{k=1}^{K}\psi_{i,k}\left[\Psi(\lambda_k)-\Psi\left(\sum_{k'=1}^{K}\lambda_{k'}\right)\right].\end{aligned}$$

There are two expectations to calculate here and we need to do it one by one.

For each $i = 1, \ldots, N$ in the fourth term, we also have two twisted expectations:

$$\begin{aligned}\mathbb{E}_Q[\log P(\mathbf{x}_i|\boldsymbol{\Theta}, c_i)] &= \mathbb{E}_{\texttt{Mult}(c_i|\boldsymbol{\psi}_i)}\Big\{\mathbb{E}_Q[\log P(\mathbf{x}_i|\boldsymbol{\Theta}, c_i)]\Big\}\\ &= \sum_{k=1}^{K} Q(c_i=k|\boldsymbol{\psi}_i)\mathbb{E}_{\mu_k,\nu_k}[\log P(\mathbf{x}_i|\boldsymbol{\theta}_k)]\\ &= \sum_{k=1}^{K}\psi_{i,k}\Big\{\log h(\mathbf{x}_i)+\mathbb{E}_{\mu_k,\nu_k}[\boldsymbol{\theta}_k]^\top\phi(\mathbf{x}_i)+\mathbb{E}_{\mu_k,\nu_k}[-A(\boldsymbol{\theta}_k)]\Big\}.\end{aligned}$$

The fifth term is a bit more complex since it is itself sum of three terms:

$$\begin{aligned}\mathbb{E}_Q[\log Q] &= \mathbb{E}_Q[\log Q(\boldsymbol{\pi}|\boldsymbol{\lambda})]+\sum_{k=1}^{K}\mathbb{E}_Q[\log Q(\boldsymbol{\theta}_k|\mu_k,\nu_k)]+\sum_{i=1}^{N}\mathbb{E}_Q[\log Q(c_i|\boldsymbol{\psi}_i)]\\ &= \log\Gamma\left(\sum_{k=1}^{K}\lambda_k\right)-\sum_{k=1}^{K}\log\Gamma(\lambda_k)+\sum_{k=1}^{K}(\lambda_k-1)\left[\Psi(\lambda_k)-\Psi\left(\sum_{k'=1}^{K}\lambda_{k'}\right)\right]\\ &\quad+\sum_{k=1}^{K}\Big\{\mathbb{E}_Q[\log g(\boldsymbol{\theta}_k)]+\mu_k^\top\mathbb{E}_{\mu_k,\nu_k}[\boldsymbol{\theta}_k]+\nu_k\mathbb{E}_{\mu_k,\nu_k}[-A(\boldsymbol{\theta}_k)]-B(\mu_k,\nu_k)\Big\}\\ &\quad+\sum_{i=1}^{N}\sum_{k=1}^{K}\psi_{i,k}\log\psi_{i,k},\end{aligned}$$

where in the last equality each row is the direct calculation of each of the three terms.

E-step

In the E-step, we can obtain the update equations for variational parameters $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$, $\boldsymbol{\nu}$ and $\boldsymbol{\Psi}$ by setting the partial derivatives of the lower bound with respect to each of them to be zero. For $\boldsymbol{\Psi}$, we should also consider the constraints $\sum_{k=1}^{K}\psi_{i,k}=1$, for all $i = 1, \ldots, N$.

This can be easily accomplished by introducing a Lagrange parameter τ_i and change the lower bound to

$$\mathcal{L}_{\boldsymbol{\psi}_i}(\mathcal{D}) = \sum_{k=1}^{K} \psi_{i,k} \left[\Psi(\lambda_k) - \Psi\left(\sum_{k'=1}^{K} \lambda_{k'} \right) + \log h(\mathbf{x}_i) + \mathbb{E}_{\mu_k,\nu_k}[\boldsymbol{\theta}_k]^\top \phi(\mathbf{x}_i) + \mathbb{E}_{\mu_k,\nu_k}[-A(\boldsymbol{\theta}_k)] \right] - \sum_{k=1}^{K} \psi_{i,k} \log \psi_{i,k} - \tau_i \left(\sum_{k=1}^{K} \psi_{i,k} - 1 \right) + C,$$

where C does not depend on the interested parameter $\boldsymbol{\psi}_i$. Setting the derivative with respect to $\psi_{i,k}$ to be zero yields

$$\hat{\psi}_{i,k} \propto \exp\left\{ \left[\Psi(\lambda_k) - \Psi\left(\sum_{k'=1}^{K} \lambda_{k'} \right) \right] + \mathbb{E}_{\mu_k,\nu_k}[\boldsymbol{\theta}_k]^\top \phi(\mathbf{x}_i) + \mathbb{E}_{\mu_k,\nu_k}[-A(\boldsymbol{\theta}_k)] \right\}, \tag{2.15}$$

where we normalize such that $\sum_{k=1}^{K} \hat{\psi}_{i,k} = 1$, for all $i = 1, \ldots, N$. Using results in (2.6), we can easily calculate the expectations in the equation.

The update equations for the other variational parameters can be similarly derived, and we omit the details and only give the results here:

$$\hat{\mu}_k = \mu_0 + \sum_{i=1}^{N} \psi_{i,k} \phi(\mathbf{x}_i), \tag{2.16}$$

$$\hat{\nu}_k = \nu_0 + \sum_{i=1}^{N} \psi_{i,k}, \tag{2.17}$$

$$\hat{\lambda}_k = \frac{\alpha}{K} + \sum_{i=1}^{N} \psi_{i,k}. \tag{2.18}$$

All these equations recover the theorem in [29] in terms of the exponential family mixture models, and turn out to be very intuitive and explainable. They are like a *deterministic* version of the Gibbs sampling process in the previous subsection.

- $\hat{\psi}_{i,k}$ denote the posterior probabilities $\hat{P}(c_i = k)$ in the Multinomial. In (2.15) they are updated as a product of two factors P_1 and P_2, where

$$P_1 \propto \exp\left\{ \Psi(\lambda_k) - \Psi\left(\sum_{k'=1}^{K} \lambda_{k'} \right) \right\},$$
$$P_2 \propto \exp\left\{ \mathbb{E}_{\mu_k,\nu_k}[\boldsymbol{\theta}_k^\top \phi(\mathbf{x}_i) - A(\boldsymbol{\theta}_k)] \right\},$$

  with other parameters fixed. Since in the variational form $\boldsymbol{\lambda}$ is the parameters of a Dirichlet, it follows from (2.2) that for Dirichlet distribution, we have

$$P_1 \propto \exp\left\{ \mathbb{E}_{\boldsymbol{\lambda}}[\log \pi_k] \right\};$$

because we have $\log P(\mathbf{x}_i|\boldsymbol{\theta}_k) \propto \boldsymbol{\theta}_k^\top \phi(\mathbf{x}_i) - A(\boldsymbol{\theta}_k)$, we have

$$P_2 \propto \exp\left\{\mathbb{E}_{\mu_k,\nu_k}[\log P(\mathbf{x}_i|\boldsymbol{\theta}_k)]\right\}.$$

Therefore, equation (2.15) can be written as

$$\hat{\psi}_{i,k} = \hat{P}(c_i = k) \propto \exp\left\{\mathbb{E}_{\boldsymbol{\lambda}}[\log P(c_i = k|\pi)]\right\} \cdot \exp\left\{\mathbb{E}_{\mu_k,\nu_k}[\log P(\mathbf{x}_i|\boldsymbol{\theta}_k)]\right\}.$$

Compare this equation with (2.12), P_1 is like a *prior* for $c_i = k$, and P_2 measures the *likelihood* of $c_i = k$. The differences are that we calculate the *expectation* of these quantities with other parameters fixed, and that we approximate the expectation as the exponential of expected log values.

- $(\hat{\mu}_k, \hat{\nu}_k)$ are the posterior natural parameters for the k-th component exponential family distribution in the mixture. In (2.16) and (2.17), it turns out that they are both the sum of two factors: the *priors* or *pseudo-counts* in the first term, and the *empirical observations* in the second term. Note that they are very similar to the posterior illustrated in (2.4), except that the empirical observations are weighted by the *belongingness* of each data point to the specific mixture component. They are also explicit updates as given in (2.11) in Gibbs sampling, with $\psi_{i,k}$ fixed.

- $\hat{\lambda}_k$ are the posterior parameters of the Dirichlet for $\boldsymbol{\pi}$, and take similar form as in (2.13). The hard assignments in Gibbs sampling in (2.13) are replaced by soft ones with fixed belongingness $\psi_{i,k}$.

Since these equations are coupled, they should be updated iteratively until convergence. In variational Bayes, (2.15) is called *variational E-step*, and (2.16)∼(2.18) are called *variational M-step*. This yields the algorithm given in [3] for mixture of Gaussians.

M-step

In the M-step, we fix the variational parameters and maximize the lower bound with respect to the model parameters. All the derivations are similar to those for the E-step. Here we find a point estimate for each hyperparameter.

For α we maximize the following part of the lower bound:

$$\mathcal{L}_\alpha(\mathcal{D}) = \log\Gamma(\alpha) - K\log\Gamma\left(\frac{\alpha}{K}\right) + \left(\frac{\alpha}{K} - 1\right)\sum_{k=1}^{K}\left[\Psi(\lambda_k) - \Psi\left(\sum_{k'=1}^{K}\lambda_{k'}\right)\right].$$

Setting the derivative with respect to α to be zero, we obtain the following equation for the update $\hat{\alpha}$:

$$K\left[\Psi\left(\frac{\hat{\alpha}}{K}\right) - \Psi(\hat{\alpha})\right] = \sum_{k=1}^{K}\left[\Psi(\lambda_k) - \Psi\left(\sum_{k'=1}^{K}\lambda_{k'}\right)\right],$$

which is equivalent to

$$\sum_{k=1}^{K} \mathbb{E}_{\hat{\alpha}}[\log \pi_k] = \sum_{k=1}^{K} \mathbb{E}_{\boldsymbol{\lambda}}[\log \pi_k].$$

That is, we are seeking an optimal $\hat{\alpha}$ to match the expectation of $\log \pi_k$ for the corresponding hidden variable $\boldsymbol{\pi}$. This turns out to be the *sufficient statistics* of the Dirichlet distribution $P(\boldsymbol{\pi}|\alpha)$.

Similar results hold for μ_0 and ν_0. Setting the corresponding derivatives to zero yields

$$K \frac{\partial B(\mu, \nu)}{\partial \mu}\bigg|_{\mu=\hat{\mu}_0, \nu=\hat{\nu}_0} = \sum_{k=1}^{K} \frac{\partial B(\mu_k, \nu_k)}{\partial \mu_k},$$

$$K \frac{\partial B(\mu, \nu)}{\partial \nu}\bigg|_{\mu=\hat{\mu}_0, \nu=\hat{\nu}_0} = \sum_{k=1}^{K} \frac{\partial B(\mu_k, \nu_k)}{\partial \nu_k},$$

which are just matching the sufficient statistics:

$$\sum_{k=1}^{K} \mathbb{E}_{\hat{\mu}_0, \hat{\nu}_0}[\boldsymbol{\theta}_k] = \sum_{k=1}^{K} \mathbb{E}_{\mu_k, \nu_k}[\boldsymbol{\theta}_k], \tag{2.19}$$

$$\sum_{k=1}^{K} \mathbb{E}_{\hat{\mu}_0, \hat{\nu}_0}[-A(\boldsymbol{\theta}_k)] = \sum_{k=1}^{K} \mathbb{E}_{\mu_k, \nu_k}[-A(\boldsymbol{\theta}_k)]. \tag{2.20}$$

This means the partial derivatives of the normalization factor with respect to the true prior parameters, (μ_0, ν_0), should be the average of that with respect to the variational parameters of each component, (μ_k, ν_k)'s. We are therefore seeking the best *trade-off* between all these mixture components, in the sense of matching the expectations of parameters of all the components.

Analytical solutions to these parameters are, however, normally unobtainable. We have to use computational methods such as Newton-Raphson method to solve the problems. After we obtain the new hyperparameters, we then run the E-step again and repeat the whole process until convergence.

Remark 2.2.1. *The variational Bayesian method in the machine learning literature is often only referred to as the E-step in the proposed learning process. Part of the reason is that when the observations are insufficient, it could be very dangerous to obtain point estimates for the hyperparameters. In this case to obtain the hyperparameters, one has to do MCMC to integrate them out, or cross-validation to select the best parameter values. However when data are sufficient, updating the hyperparameters in this way is reasonable and formally referred to as the ML-II method [53].*

Remark 2.2.2. *The difference between the proposed learning algorithm and the maximal likelihood (ML) algorithm is clear. In ML algorithm, there are no hyperpriors with parameters α and (μ_0, ν_0), and we simply directly find the best estimates for $\boldsymbol{\pi}$ and $\boldsymbol{\theta}_k$'s given the data, normally using an expectation-maximization (EM) algorithm. In some sense the Bayesian solution given above can be viewed as a smoothed version of the ML algorithm.*

2.2.3 Model Fitting as Clustering

The whole variational Bayesian learning algorithm can be naturally viewed as a clustering algorithm. We can interpret the E-step using the clustering language as follows:

1. Randomly initialize all the cluster-specific information (μ_k, ν_k)'s and cluster weights λ_k's;
2. Repeat the following two steps until convergence:
   (a) Based on the current cluster information, assign each data point to all the clusters softly (update as (2.15));
   (b) Update the cluster weights using (2.18), and cluster-specific information using (2.16) and (2.17);
3. After convergence, the soft cluster membership that data point $\mathbf{x}_i$ belongs to cluster k is given in $\psi_{i,k}$, the k-th cluster is determined via (μ_k, ν_k), and the soft cluster weights are given in the normalized vector $[\lambda_1/\tilde{\lambda}, \ldots, \lambda_K/\tilde{\lambda}]$, with $\tilde{\lambda} := \sum_{k=1}^{K} \lambda_k$.

The whole algorithm looks like a probabilistic version of the well-known k-means algorithm, except that we do not have a hard cluster assignment for each data point, and that the clusters now take much more general form. It is seen from Step 3 that all the clustering-specific information is given in the variational parameters, and the hyperparameters α and (μ_0, ν_0) correspond to smooth terms in the E-step updates. In particular, the soft cluster weights are determined here as the expectations of $\boldsymbol{\pi}$ with respect to the variational distribution $Q(\boldsymbol{\pi}|\boldsymbol{\lambda})$. It will be seen in the next chapter that the hyperparameters will lead to a sparse clustering structure if K is large.

Clustering for New Data

When there is a new data point $\mathbf{x}_*$ available, we can calculate its soft membership to each cluster k using Bayes' rule as

$$P(c_* = k|\mathbf{x}_*, \boldsymbol{\pi}, \boldsymbol{\Theta}) = \frac{P(\mathbf{x}_*|\boldsymbol{\theta}_k)P(c_* = k|\boldsymbol{\pi})}{P(\mathbf{x}_*|\boldsymbol{\pi}, \boldsymbol{\Theta})} = \frac{\pi_k P(\mathbf{x}_*|\boldsymbol{\theta}_k)}{\sum_{k'=1}^{K} \pi_{k'} P(\mathbf{x}_*|\boldsymbol{\theta}_{k'})}.$$

That is, this soft weight depends on the weight of this cluster (π_k), and also the *explainability* of this cluster to this data point (i.e., likelihood $P(\mathbf{x}_*|\boldsymbol{\theta}_k)$). If one would like to calculate the likelihood of the new data, one has to deal with integral like (2.10), and an alternative is to calculate the lower bound using the variational distribution Q.

2.2.4 Variational Bayes for Example Models

As two concrete examples, we apply the variational Bayesian solution to mixture of Gaussians and mixture of Multinomials. All the update equations can be directly derived form the general solution, and we only give the main results here.

Variational Bayes for Mixture of Gaussians

Following the general learning framework in the previous subsection, for mixture of Gaussians we take the following factorized posterior distribution for all the variables

$$Q(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Lambda}, \mathbf{c}) = Q_D(\boldsymbol{\pi}|\boldsymbol{\lambda}) \prod_{k=1}^{K} \left[Q_N(\boldsymbol{\mu}_k|\mathbf{m}_k, \beta_k^{-1}\boldsymbol{\Lambda}_k^{-1}) Q_W(\boldsymbol{\Lambda}_k|\mathbf{W}_k, \nu_k) \right] \prod_{i=1}^{N} Q_M(c_i|\boldsymbol{\psi}_i),$$

where Q_D, Q_N, Q_W and Q_M denote Dirichlet, Gaussian, Wishart and Multinomial, respectively. All the variational parameters are $\boldsymbol{\lambda}$, $\boldsymbol{\Psi}$ and $\{\mathbf{m}_k, \mathbf{W}_k, \beta_k, \nu_k\}_{k=1}^K$. To simplify the notation and build connections to our general solution above, we apply (2.2) to Dirichlet and Wishart distributions and obtain the following expectation equations for the sufficient statistics:

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{\lambda}}[\log \pi_k] := \mathbb{E}_{\mathtt{Dir}(\boldsymbol{\pi}|\boldsymbol{\lambda})}[\log \pi_k] = \Psi(\lambda_k) - \Psi\left(\sum_{k'=1}^{K} \lambda_{k'}\right), \\
&\mathbb{E}_{\mathbf{W}_k,\nu_k}[\boldsymbol{\Lambda}_k] := \mathbb{E}_{\mathcal{W}(\boldsymbol{\Lambda}_k|\mathbf{W}_k,\nu_k)}[\boldsymbol{\Lambda}_k] = \nu_k \mathbf{W}_k, \\
&\mathbb{E}_{\mathbf{W}_k,\nu_k}[\log|\boldsymbol{\Lambda}_k|] := \mathbb{E}_{\mathcal{W}(\boldsymbol{\Lambda}_k|\mathbf{W}_k,\nu_k)}[\log|\boldsymbol{\Lambda}_k|] = \sum_{\ell=1}^{d} \Psi\left(\frac{\nu_k + 1 - \ell}{2}\right) + d\log 2 + \log|\mathbf{W}_k|.
\end{aligned}$$

Using short-hand notations

$$N_k := \sum_{i=1}^{N} \psi_{i,k}, \qquad \bar{\mathbf{x}}_k := \frac{1}{N_k} \sum_{i=1}^{N} \psi_{i,k} \mathbf{x}_i,$$

we have the E-step as follows:

$$\begin{aligned}
\hat{\psi}_{i,k} &\propto \exp\left\{ \mathbb{E}_{\boldsymbol{\lambda}}[\log \pi_k] - \frac{d}{2\beta_k} + \frac{1}{2}\mathbb{E}_{\mathbf{W}_k,\nu_k}[\log|\boldsymbol{\Lambda}_k|] - \frac{1}{2}(\mathbf{x}_i - \mathbf{m}_k)^\top \mathbb{E}_{\mathbf{W}_k,\nu_k}[\boldsymbol{\Lambda}_k](\mathbf{x}_i - \mathbf{m}_k) \right\}, \\
\hat{\mathbf{m}}_k &= \frac{1}{\beta_0 + N_k}(\beta_0 \mathbf{m}_0 + N_k \bar{\mathbf{x}}_k), \\
\hat{\mathbf{W}}_k^{-1} &= \mathbf{W}_0^{-1} + \sum_{i=1}^{N} \psi_{i,k}(\mathbf{x}_i - \bar{\mathbf{x}}_k)(\mathbf{x}_i - \bar{\mathbf{x}}_k)^\top + \frac{\beta_0 N_k}{\beta_0 + N_k}(\bar{\mathbf{x}}_k - \mathbf{m}_0)(\bar{\mathbf{x}}_k - \mathbf{m}_0)^\top, \\
\hat{\beta}_k &= \beta_0 + N_k, \qquad \hat{\nu}_k = \nu_0 + N_k, \qquad \hat{\lambda}_k = \frac{\alpha}{K} + N_k,
\end{aligned}$$

which are quite understandable. For instance, the mean vector of each cluster $\mathbf{m}_k$ is the weighted average of the prior mean $\mathbf{m}_0$ and all the data points, where the weights for each data point $\mathbf{x}_i$ is just the soft cluster membership $\psi_{i,k}$.

The M-step turns out to be

$$\hat{\mathbf{m}}_0 = \left(\sum_{k=1}^{K} \nu_k \mathbf{W}_k\right)^{-1} \sum_{k=1}^{K} \nu_k \mathbf{W}_k \mathbf{m}_k, \qquad \hat{\mathbf{W}}_0 = \frac{\sum_{k=1}^{K} \nu_k \mathbf{W}_k}{K\nu_0},$$

$$\frac{1}{\hat{\beta}_0} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{\beta_k} + \frac{1}{dK}\sum_{k=1}^{K} \nu_k (\mathbf{m}_k - \mathbf{m}_0)^\top \mathbf{W}_k (\mathbf{m}_k - \mathbf{m}_0),$$

$$\sum_{\ell=1}^{d} \Psi\left(\frac{\hat{\nu}_0 + 1 - \ell}{2}\right) = \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}_{\mathbf{W}_k, \nu_k}[\log|\mathbf{\Lambda}_k|] - d\log 2 - \log|\mathbf{W}_0|,$$

where the last equation needs numerical method to solve. It is easily seen that all of the M-step equations are just matching the expectations of certain quantities.

In Figure 2.3 we show the true Gaussian mixtures in (a) and the learned Gaussians in (b) and (c). For learning we set $K = 5$, $\alpha = 1$, $\beta_0 = 0.01$, $\nu_0 = 2$, $\mathbf{m}_0$ the sample mean, and $\mathbf{W}_0$ the inverse of the sample covariance divided by β_0. For computational simplicity we fix ν_0 in the whole learning process. The lower bound of the log likelihood (2.14) after each iteration in the fist E-step (i.e., iterating updates for latent variables $\psi_{i,k}$, $\mathbf{m}_k$, $\mathbf{W}_k$, β_k, ν_k, λ_k with pre-chosen parameters) is shown in (g), and it is seen that it is always increasing, and it converges after about 40 steps. Here we run the iterations until the sum of element-wise changes of membership matrix $\mathbf{\Psi}$ is less than 0.0001. The lower bound at this convergence is -949.73. Running more EM iterations (until convergence after 3 steps) does not improve the lower bound much and yields the final number -922.14. The learned 5 Gaussians and the cluster memberships (see (e) and (f)) are also almost the same. For clarity we give the learned results for the mean vectors and the covariance matrices of these Gaussians as follows:

$$\hat{\boldsymbol{\mu}}_1 = \begin{pmatrix} 3.03 \\ 3.02 \end{pmatrix}, \hat{\mathbf{\Sigma}}_1 = \begin{pmatrix} 0.38 & -0.33 \\ -0.33 & 0.51 \end{pmatrix}, \qquad \hat{\boldsymbol{\mu}}_2 = \begin{pmatrix} 2.96 \\ 7.93 \end{pmatrix}, \hat{\mathbf{\Sigma}}_2 = \begin{pmatrix} 0.99 & -0.02 \\ -0.02 & 0.21 \end{pmatrix},$$

$$\hat{\boldsymbol{\mu}}_3 = \begin{pmatrix} 5.14 \\ 5.69 \end{pmatrix}, \hat{\mathbf{\Sigma}}_3 = \begin{pmatrix} 0.37 & -0.19 \\ -0.19 & 0.39 \end{pmatrix}, \qquad \hat{\boldsymbol{\mu}}_4 = \begin{pmatrix} 7.05 \\ 7.03 \end{pmatrix}, \hat{\mathbf{\Sigma}}_4 = \begin{pmatrix} 0.51 & 0.13 \\ 0.13 & 0.53 \end{pmatrix},$$

$$\hat{\boldsymbol{\mu}}_5 = \begin{pmatrix} 8.00 \\ 3.75 \end{pmatrix}, \hat{\mathbf{\Sigma}}_5 = \begin{pmatrix} 0.35 & -0.18 \\ -0.18 & 0.54 \end{pmatrix}.$$

To calculate these quantities we have the mean vector $\boldsymbol{\mu}_k = \mathbf{m}_k$, and covariance matrix $\mathbf{\Sigma}_k = (\nu_k \mathbf{W}_k)^{-1}$.

Variational Bayes for Mixture of Multinomials

For mixture of Multinomials we take the following form for Q:

$$Q(\boldsymbol{\pi}, \mathbf{\Theta}, \mathbf{c}) = Q_D(\boldsymbol{\pi}|\boldsymbol{\lambda}) \prod_{k=1}^{K} Q_D(\boldsymbol{\theta}_k|\boldsymbol{\xi}_k) \prod_{i=1}^{N} Q_M(c_i|\boldsymbol{\psi}_i),$$

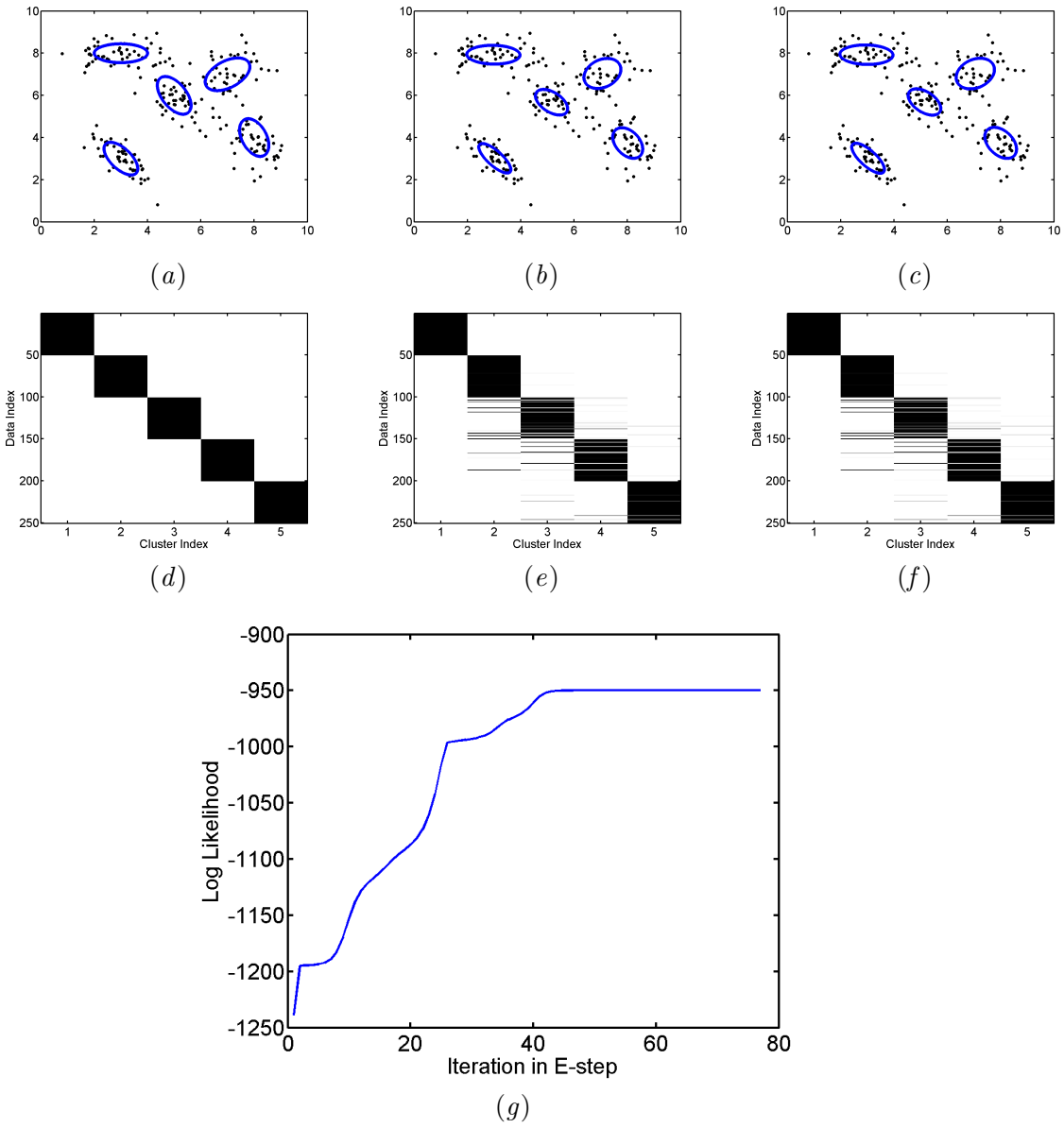


Figure 2.3: Learning for the toy Gaussian mixture data. (a) and (d) show the true Gaussians and the true (hard) cluster membership for each data point. (b) and (e) show the learned Gaussians and soft cluster membership of variational Bayesian learning after 1 EM iteration (with the E-step itself 77 iterations). (c) and (f) show the final results (after 3 EM steps) of variational Bayesian learning. (g) shows the log likelihood after each iteration in the first E-step.

where Q_D and Q_M denote Dirichlet and Multinomial distributions, respectively. Now all the variational parameters are $\boldsymbol{\lambda}$, $\boldsymbol{\Psi}$ and $\{\boldsymbol{\xi}_k\}_{k=1}^K$. As before we apply (2.2) to Dirichlet

and obtain the following expectation equations for sufficient statistics:

$$\mathbb{E}_{\boldsymbol{\lambda}}[\log \pi_k] := \mathbb{E}_{\mathtt{Dir}(\boldsymbol{\pi}|\boldsymbol{\lambda})}[\log \pi_k] = \Psi(\lambda_k) - \Psi\left(\sum_{k'=1}^{K} \lambda_{k'}\right),$$

$$\mathbb{E}_{\boldsymbol{\xi}_k}[\log \theta_{k,v}] := \mathbb{E}_{\mathtt{Dir}(\boldsymbol{\theta}_k|\boldsymbol{\xi}_k)}[\log \theta_{k,v}] = \Psi(\xi_{k,v}) - \Psi\left(\sum_{v'=1}^{V} \xi_{k,v'}\right).$$

The E-step is as follows:

$$\hat{\psi}_{i,k} \propto \exp\left\{\mathbb{E}_{\boldsymbol{\lambda}}[\log \pi_k] + \sum_{n=1}^{N_i}\sum_{v=1}^{V} \delta_v(w_n)\mathbb{E}_{\boldsymbol{\xi}_k}[\log \theta_{k,v}]\right\},$$

$$\hat{\xi}_{k,v} = \xi_{0,v} + \sum_{i=1}^{N}\sum_{n=1}^{N_i}\sum_{v=1}^{V} \delta_v(w_n)\psi_{i,k},$$

$$\hat{\lambda}_k = \frac{\alpha}{K} + \sum_{i=1}^{N} \psi_{i,k},$$

where w_n denote the index of the n-th word ($n = 1, \ldots, N_i$ for data point $\mathbf{x}_i$), and $\delta_v(w_n)$ takes value 1 if $w_n = v$ and 0 otherwise. For the M-step we have to solve the following equation numerically for $\boldsymbol{\xi}_0$:

$$\Psi(\hat{\xi}_{0,v}) - \Psi\left(\sum_{v'=1}^{V} \hat{\xi}_{0,v'}\right) = \frac{1}{K}\sum_{k=1}^{K} \mathbb{E}_{\boldsymbol{\xi}_k}[\log \theta_{k,v}].$$

2.3 Summary

In this chapter we introduce the mixture of exponential family distributions for clustering, and illustrate mixture of Gaussians and mixture of Multinomials as two concrete examples. The mixture model is proposed from a Bayesian perspective, with hyperparameters (μ_0, ν_0) identifying the conjugate prior for the natural parameters of each mixture component, and α constraining the mixing weights of these component distributions. For inference and learning in the proposed model, a straightforward Gibbs sampler is shown for clarity, and more efforts are given to a variational Bayesian solution which is an approximated but fast algorithm. Popular clustering algorithms like k-means are special cases of this method. Some toy problems are used to evaluate the algorithm, and more empirical studies will be performed in the next chapter in which we will answer an important question in finite mixture modeling: How should we choose the number of mixture components K?

Chapter 3

Infinite Mixture Models

A common problem in finite mixture modeling is how to choose the number of mixture components K. In the language of clustering, this corresponds to choosing the number of clusters beforehand. This is a highly non-trivial problem, because one can in principal assign to K any integer from 1 to the number of data points N, and the system can always learn a mixture model for this K. One way of measuring the quality of each K is to compute the *generalization performance* of the model with a specified K on some held-out data, i.e., data that are not used for training. It is also known that this number has strong connections to the model complexity in Bayesian theory.

There has been a lot of research on this issue, but the solution is still open if one does not do expensive MCMC sampling. The most straightforward idea is to train the model with different K's (with independent repeats on different sets of training data), and then pick the K with the highest (average) likelihood on some held-out data. This is in general referred to as *cross-validation* and is a standard technique for model selection. However, this leads to high computational labor because we need to train the model many times with different K's. Another solution is to assign a prior to K, i.e., $P(K)$, and then calculate the *a posteriori* distribution of K to determine this number [14]. But it remains unclear which parametric form to use for $P(K)$. There also exist many information criteria, e.g., Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC) and Minimum Description Length (MDL) (see, e.g., [21]). The basic idea of these criteria is to penalize complicated models (i.e., models with large K), and thus we come up with an appropriate K to trade-off data likelihood and model complexity [26].

In this chapter we start from the finite mixture model and extend it to infinite mixtures in the same framework (Section 3.1).[1] It is known in statistics that this leads to a Dirichlet process (DP) mixture model [25]. The new model not only replaces the parametric prior (i.e., the conjugate family) with a more flexible one, but also have a nice way to determine the number of mixture components automatically (Section 3.2). While DP mixture model normally relies on a MCMC sampling scheme, in Section 3.3 we introduce a variational

[1] This chapter is an extension of the paper [86].

Bayesian solution called the *variational Bayesian Dirichlet-Multinomial allocation* (VB-DMA) for model learning and inference. Apart from the other variational solution, the proposed method is naturally connected to learning in finite mixture models, and it is shown that the hyperparameter α controls the sparsity of the mixtures (Section 3.3.4). Therefore, the variational solution leads to an easy way of determining the component number in the mixture. Some empirical studies are given in Section 3.4 to verify this approach.

3.1 Infinite Mixture Models

Let us start from the finite mixture model with K components. Using the notation of Chapter 2, the likelihood of a given data point $\mathbf{x}_i \in \mathcal{X}$ is written as

$$P(\mathbf{x}_i|\boldsymbol{\pi}, \boldsymbol{\Theta}) = \sum_{k=1}^{K} P(c_i = k|\boldsymbol{\pi}) P(\mathbf{x}_i|\boldsymbol{\theta}_k) = \sum_{k=1}^{K} \pi_k P(\mathbf{x}_i|\boldsymbol{\theta}_k), \tag{3.1}$$

with mixture weights $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_K\}$ satisfying $\sum_{k=1}^{K} \pi_k = 1$, and the set of K natural parameters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_k\}_{k=1}^{K}$. We can equivalently write (3.1) as

$$P(\mathbf{x}_i|\boldsymbol{\pi}, \boldsymbol{\Theta}) = \int_{\boldsymbol{\theta}} P(\mathbf{x}_i|\boldsymbol{\theta}) G_K(\boldsymbol{\theta})\, d\boldsymbol{\theta}, \tag{3.2}$$

with notation

$$G_K(\boldsymbol{\theta}) := P(\boldsymbol{\theta}|\boldsymbol{\pi}, \boldsymbol{\Theta}) = \sum_{k=1}^{K} \pi_k \delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}) \tag{3.3}$$

where $\delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta})$ is the point mass distribution and takes value 1 for $\boldsymbol{\theta} = \boldsymbol{\theta}_k$ and 0 otherwise. It is seen here that distribution $G_K(\boldsymbol{\theta})$ defines a *discrete prior* for $\boldsymbol{\theta}$, which means $\boldsymbol{\theta}$ can only take one of the K values from $\boldsymbol{\Theta}$, with weights defined by $\boldsymbol{\pi}$. The final likelihood for *i.i.d.* data $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, with $\boldsymbol{\pi}$ and $\boldsymbol{\Theta}$ integrated out, can be written with this new notation as

$$P(\mathcal{D}|\alpha, G_0) = \int_{G_K} P(G_K|\alpha, G_0) \left(\prod_{i=1}^{N} \int_{\boldsymbol{\theta}_i} P(\mathbf{x}_i|\boldsymbol{\theta}_i) G_K(\boldsymbol{\theta}_i)\, d\boldsymbol{\theta}_i \right) dG_K, \tag{3.4}$$

in which K is only kept in the discrete distribution G_K. Note that here we use $\boldsymbol{\theta}_i$ to denote the natural parameter for the i-th data point, which is one of the K parameters in $\boldsymbol{\Theta}$. In this notation, model parameters α and G_0 now take the role of tuning the discrete but *unknown* distribution $G_K(\boldsymbol{\theta})$. The plate model is shown in Figure 3.1(a), with dashed rectangle G_K denoting the discrete prior $G_K(\boldsymbol{\theta})$.

When we let $K \to \infty$, it is known in statistics that the unknown distribution G_K tends to be a sample from a *Dirichlet process* (DP), which is a distribution over random

measures. Following the convention for Dirichlet processes, we have the following sampling process for data $\mathcal{D}$:

$$\begin{aligned}\mathbf{x}_i|\boldsymbol{\theta}_i &\overset{\text{ind}}{\sim} P(\cdot|\boldsymbol{\theta}_i), \quad \text{for } i = 1, \ldots, N\\ \boldsymbol{\theta}_i &\overset{\text{iid}}{\sim} G,\\ G &\sim \mathcal{DP}(\alpha, G_0).\end{aligned}$$

As will be formally defined in the next section, a DP has two parameters: the *concentration parameter* (a positive scalar) and a *base distribution*. In our model, the concentration parameter is just α, and the base distribution is $G_0 := P(\boldsymbol{\theta}|\mu_0, \nu_0)$. This model is illustrated in Figure 3.1(b).

Dirichlet process is well-known for the property of obtaining a nonparametric and discrete prior, and thus is widely applied for mixture modeling (see, e.g., [44]). When K is finite, however, the model is not equivalent as defining a Dirichlet process prior for $\boldsymbol{\theta}_i$'s, but is shown to be a good approximation if K is sufficiently large. This finite approximation is sometimes referred to as *Dirichlet-Multinomial allocation* (DMA), and is used for approximated sampling for Dirichlet processes [33]. In both DP and DMA, model selection can be done automatically via sampling methods, and the concentration parameter α is known to control the flexibility of generating new mixture components.

We give the formal definition for DP in the next section, and then turn to inference and learning for DP models. We will give a Gibbs sampling algorithm and several variational Bayesian solutions. We will show that the proposed VBDMA algorithm can detect an appropriate mixture number K under the control of parameter α.

3.2 Dirichlet Processes

Dirichlet process is a nonparametric Bayesian framework for mixture modeling. In this section we give the formal definition, and discuss the various views of sampling from a DP such as Pólya urn sampling, Chinese restaurant process and stick-breaking process. For more details about DP please refer to [25, 2, 64, 22, 44].

Definition 3.2.1. *Let $(\Omega, \mathcal{B})$ be a measurable space, with G_0 a probability measure on the space, and let α be a positive real number. A* Dirichlet process *is the distribution of a random probability measure G over $(\Omega, \mathcal{B})$ such that, for any positive integer r and any finite partition $(A_1, \ldots, A_r)$ of Ω, the random vector $(G(A_1), \ldots, G(A_r))$ is distributed as a finite-dimensional Dirichlet distribution:*

$$(G(A_1), \ldots, G(A_r)) \sim \texttt{Dir}\big(\alpha G_0(A_1), \ldots, \alpha G_0(A_r)\big).$$

We write $G \sim \mathcal{DP}(\alpha, G_0)$ if G is a random probability measure distributed according to the Dirichlet process. G_0 is called the base distribution *of G, and α is called* concentration parameter.

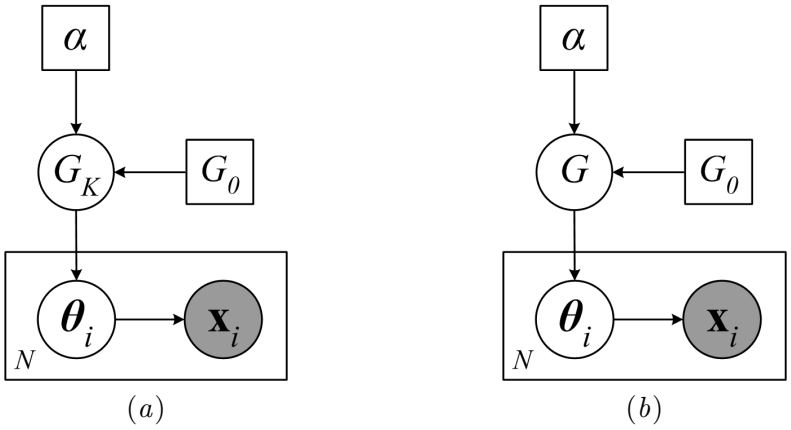


Figure 3.1: Plate models from finite mixture to infinite mixture. (a) shows the finite mixture of exponential family distributions, with G_K denoting the discrete prior for θ_i's; (b) shows Dirichlet process mixture model. See text for the different prior distributions to G_K and G in the two plate models.

When the random measure $G \sim \mathcal{DP}(\alpha, G_0)$, sampling $\boldsymbol{\theta}$ from G can be done analytically with G integrated out, using the *Pólya urn sampling*. Suppose we have observed $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N$ from G, then the $(N+1)$-th sample $\boldsymbol{\theta}_{N+1}$ takes distribution

$$P(\boldsymbol{\theta}_{N+1}|\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N, \alpha, G_0) = \sum_{i=1}^{N} \frac{1}{N+\alpha}\delta_{\boldsymbol{\theta}_i}(\cdot) + \frac{\alpha}{N+\alpha}G_0(\cdot), \tag{3.5}$$

which means the new sample takes each old sample $\boldsymbol{\theta}_i$ with probability proportional to 1, and takes a new one from base distribution G_0 with probability proportional to α. This is like we have a fixed color distribution (G_0) of all the balls in an urn, and each time we randomly pick up one ($\boldsymbol{\theta}_i$) from the urn, and put this ball and a new one with the same color into the urn. α in this ball-picking game denotes the initial number of balls in the urn.

A grouped version of Pólya urn sampling is sometimes called the *Chinese restaurant process* (CRP). Let $\boldsymbol{\Theta}^* := \{\boldsymbol{\theta}_1^*, \ldots, \boldsymbol{\theta}_M^*\}$ denote the M distinct samples out of the total N samples, and N_j the number of samples which take $\boldsymbol{\theta}_j^*$. Note that by definition we have $\sum_{j=1}^{M} N_j = N$. In Chinese restaurant process the $(N+1)$-th sample takes distribution

$$P(\boldsymbol{\theta}_{N+1}|\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N, \alpha, G_0) = \sum_{j=1}^{M} \frac{N_j}{N+\alpha}\delta_{\boldsymbol{\theta}_j^*}(\cdot) + \frac{\alpha}{N+\alpha}G_0(\cdot). \tag{3.6}$$

To get a flavor of the name of the process, imagine there is a Chinese restaurant with infinite number of tables. The first customer comes and randomly choose a table ($\boldsymbol{\theta}_1^*$) and sit down. Then a second customer comes, with probability $\frac{1}{1+\alpha}$ sit at the same table ($\boldsymbol{\theta}_1^*$), and with probability $\frac{\alpha}{1+\alpha}$ choose a new table ($\boldsymbol{\theta}_2^*$). When the $(N+1)$-th customer

comes, suppose there are M table occupied, with N_j customers sitting at table j. Then the new customer sit at table j with probability proportional to N_j, and choose a new table with probability proportional to α. More popular tables get even more popular as new customers keep coming.

Previous two sampling processes integrate out the distribution G and directly sample from the posterior. There also exists an explicit form for G, which can be obtained through *stick-breaking process*. This allows us to directly sample a distribution G from $\mathcal{DP}(\alpha, G_0)$. Let V_j, $j = 1, 2, \ldots$, be *i.i.d.* samples from Beta distribution $\texttt{Beta}(1, \alpha)$, and define (infinite number of) positive scalars

$$\pi_1 = V_1, \qquad \pi_j = V_j \prod_{\ell=1}^{j-1} (1 - V_\ell), \quad j > 1. \tag{3.7}$$

Then a sample from the DP, i.e., G, can be written as an infinite sum:

$$G(\cdot) = \sum_{j=1}^{\infty} \pi_j \delta_{\boldsymbol{\theta}_j^*}(\cdot), \tag{3.8}$$

where $\boldsymbol{\theta}_j^*$ are *i.i.d.* sampled from G_0. The sampling for all the weights π_j's is called a stick-breaking process because it is like breaking a unit-length stick successively independently. It is shown that the finite discrete distribution (3.3) converges to the infinite sum (3.8) under an appropriate definition of convergence.

All these three sampling processes can be used to obtain samples from the following generative model:

$$\begin{aligned} \boldsymbol{\theta}_i &\overset{\text{iid}}{\sim} G, \\ G &\sim \mathcal{DP}(\alpha, G_0). \end{aligned}$$

Pólya urn sampling and Chinese restaurant process obtain samples without directly sampling distribution G, and stick-breaking process sample G first using (3.8), and then sample all the $\boldsymbol{\theta}_i$'s from the infinite mixture G. As can be seen from all the three sampling processes, base distribution G_0 is used to obtain *locations* of all the distinct samples $\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, \ldots$, and concentration parameter α controls the *grouping effect* or the *sparsity* of all the obtained samples. If α is small, new samples will have high probabilities to pick up a previously chosen location, and grouping happens frequently. On the other hand if α is large, each time a new location will be chosen at high probability, and we do not have a sparse structure of the samples. Figure 3.2 shows a sampling of 10,000 $\boldsymbol{\theta}$'s from a Dirichlet process $\mathcal{DP}(\alpha, G_0)$. In this toy illustration each $\boldsymbol{\theta}$ is just a one-dimensional real number. The top figure shows the base distribution G_0 which is a Gaussian $\mathcal{N}(0, 1)$, and the middle and bottom figures show the number of samples at the distinct locations with $\alpha = 10$ and $\alpha = 10000$, respectively. When α is relatively small, all the samples occur at a small number of locations, and we see a clear grouping effect in the middle figure; on the other hand if α is large, all the locations are occupied with the counts proportional to their likelihood values in the base distribution G_0, as shown in the bottom figure.

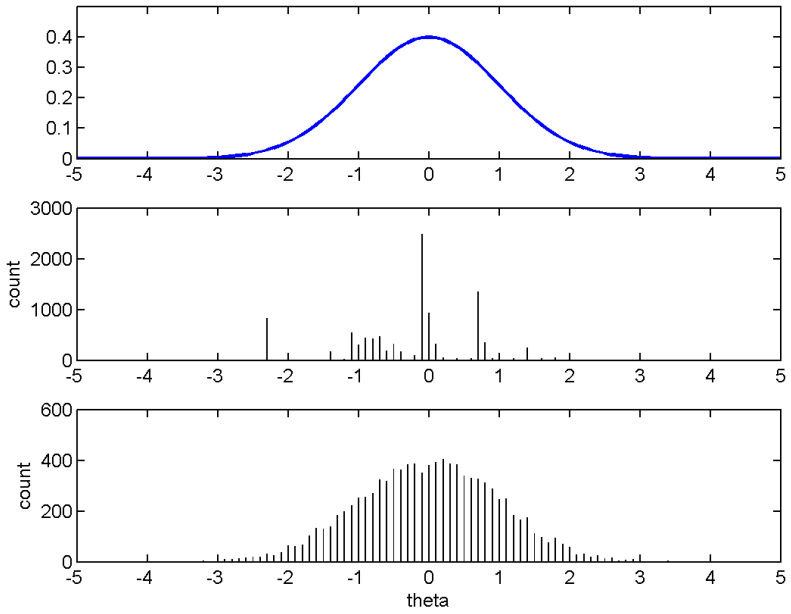


Figure 3.2: Sampling of 10,000 $\boldsymbol{\theta}$'s from a DP with base distribution G_0 a Gaussian $\mathcal{N}(0,1)$ (top), and concentration parameter $\alpha = 10$ (middle) and $\alpha = 10000$ (bottom).

As mentioned in the very first paper about Dirichlet processes, one natural application of DP is mixture modeling, which is called *Dirichlet process mixture model*. In this model, the DP prior is assigned to the parameter of the likelihood, i.e., θ_i in our notation, and observations $\mathbf{x}_i$ are sampled given these parameters. The generative model for DP mixture models is as follows (which we have mentioned in the previous section):

$$\begin{aligned} \mathbf{x}_i|\boldsymbol{\theta}_i &\overset{\text{ind}}{\sim} P(\cdot|\boldsymbol{\theta}_i), \quad \text{for } i = 1, \ldots, N \\ \boldsymbol{\theta}_i &\overset{\text{iid}}{\sim} G, \\ G &\sim \mathcal{DP}(\alpha, G_0). \end{aligned}$$

The plate model is shown in Figure 3.1(b). The key difference here is that we do not directly observe the samples from a DP, i.e., $\boldsymbol{\theta}_i$'s, but treat them as latent variables. The likelihood of observations $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ is now

$$P(\mathcal{D}|\alpha, G_0) = \int_G P(G|\alpha, G_0) \left\{ \prod_{i=1}^{N} \int_{\boldsymbol{\theta}_i} P(\mathbf{x}_i|\boldsymbol{\theta}_i) G(\boldsymbol{\theta}_i)\, d\boldsymbol{\theta}_i \right\} dG,$$

similar to the finite mixture model (3.4). To facilitate sampling and variational inference, we choose G_0 such that it is conjugate to the likelihood model $P(\mathbf{x}|\boldsymbol{\theta})$. In the case of exponential family distribution, we choose G_0 to be the conjugate family $P(\boldsymbol{\theta}|\mu_0, \nu_0)$ with hyperparameter μ_0 and ν_0.

This directly leads to a clustering interpretation of DP mixture model. Since DP prior induces a grouping structure on the set of all $\boldsymbol{\theta}_i$'s, two data points which share the same parameter can be thought to belong to the same cluster. The nice property of DP mixture model is that there is no need to pre-choose a number of clusters, since the sparsity automatically occurs in DP sampling. In another word, we can infer this number automatically from the data. The hyperparameter α is now controlling the sparsity of the mixture, i.e., the number of clusters in the data.

Remark 3.2.1. *In parametric Bayesian analysis, one puts a (conjugate) parametric prior on the likelihood parameters. Examples of this include Gaussian prior for the mean of a Gaussian likelihood, and Dirichlet prior for the parameters of Multinomial observations. In terms of an exponential family likelihood with parameter $\boldsymbol{\theta}$, we assign the conjugate family, i.e., $G_0 = P(\boldsymbol{\theta}|\mu_0, \nu_0)$, as the parametric prior for $\boldsymbol{\theta}$. However, it is often argued that this parametric prior is too strong to maintain enough flexibility of Bayesian modeling. Non-parametric Bayes emerges in this context and replaces the parametric prior G_0 with a non-parametric one, G, and put G_0 in a higher level of the hierarchy. The prior G for $\boldsymbol{\theta}$ is now flexible enough, but still controlled by G_0 and a sparsity parameter α. This change from parametric models to non-parametric models is sometimes called* Dirichlet enhancement*, and the new model is equivalent to an infinite mixture of the underlying parametric models.*

In the following sections we review the Gibbs sampling methods for DP mixture models, and then introduce truncation to DP and two variational Bayesian methods.

3.3 Inference and Parameter Estimation

Since we cannot directly sample a distribution G from a DP, the inference in the DP mixture model remains intractable. There exist various Markov chain Monte Carlo (MCMC) methods to this problem [55], but in this section we are more interested in variational methods. In the following we briefly review certain Gibbs sampling methods, and then investigate corresponding variational methods.

3.3.1 Gibbs Sampling with Pólya Urn Process

The Gibbs sampling for DP mixture model is a natural extension of the Pólya urn sampling for DP. The key point here is that sampling of a parameter $\boldsymbol{\theta}_i$ depends not only on all the other samples $\boldsymbol{\Theta}_{-i} := \{\boldsymbol{\theta}_\ell\}_{\ell=1}^N \setminus \boldsymbol{\theta}_i$, but also on the observation $\mathbf{x}_i$. Based on the exchangeability, we iteratively sample one $\boldsymbol{\theta}_i$ conditioned on the other parameters $\boldsymbol{\Theta}_{-i}$, hyperpriors α, G_0 and data $\mathcal{D}$ as

$$\begin{aligned}\hat{P}(\boldsymbol{\theta}_i|\boldsymbol{\Theta}_{-i}, \alpha, G_0, \mathcal{D}) &\propto P(\mathbf{x}_i|\cdot)P(\cdot|\boldsymbol{\Theta}_{-i}, \alpha, G_0) \\ &\propto \sum_{\ell \neq i} P(\mathbf{x}_i|\boldsymbol{\theta}_\ell)\delta_{\boldsymbol{\theta}_\ell}(\cdot) + \alpha P(\mathbf{x}_i|\cdot)G_0(\cdot), \end{aligned} \tag{3.9}$$

where the final normalization factor can be calculated as

$$Z = \sum_{\ell \neq i} P(\mathbf{x}_i|\boldsymbol{\theta}_\ell) + \alpha \int_{\boldsymbol{\theta}} P(\mathbf{x}_i|\boldsymbol{\theta}) G_0(\boldsymbol{\theta})\, d\boldsymbol{\theta}.$$

With a conjugate prior G_0, the integral here is tractable. Now a sample of $\boldsymbol{\theta}_i$ takes a previously known $\boldsymbol{\theta}_\ell$ with probability proportional to its "interpretability" to data $\mathbf{x}_i$, i.e., $P(\mathbf{x}_i|\boldsymbol{\theta}_\ell)$, and takes a new sample from G_0 with probability proportional to α times the interpretability of this new sample to $\mathbf{x}_i$. Equation (3.9) can also be written using Chinese restaurant process with grouped parameters. By assigning priors to α and G_0, we can also Gibbs-sample these hyperparameters. Then iterating the sampling process we can solve the problem.

3.3.2 Truncated Dirichlet Process

The Pólya urn process defines a clear sampling structure for the posterior of $\boldsymbol{\theta}$ and avoids the difficult problem of sampling the unknown distribution G. However, the sampling process denoted in (3.9) could be very slow to converge because we update the $\boldsymbol{\theta}_i$'s one by one. There are methods in which G is considered explicitly using stick-breaking process, and the updates for Gibbs sampling can be done *in block*. This is applicable if we truncate the stick-breaking process, which is called *truncated Dirichlet process* (TDP) [45].

In TDP we choose a positive integer M which is large enough, and force $V_M = 1$ in the stick-breaking process (3.7). Then it is easily seen that $\pi_j = 0$ for all $j > M$, and distribution G is now a finite sum with M point-mass functions:

$$G(\cdot) = \sum_{j=1}^{M} \pi_j \delta_{\boldsymbol{\theta}_j^*}(\cdot), \tag{3.10}$$

with the weights $\boldsymbol{\pi} = \{\pi_j\}_{j=1}^M$ satisfying $\sum_{j=1}^M \pi_j = 1$. It can be shown that with a sufficiently large M, TDP approaches a true DP.

Under the approximation of TDP, the DP mixture model is now similar to a finite mixture model, with a stick-breaking prior on $\boldsymbol{\pi}$. With notation $\boldsymbol{\Theta}^* = \{\boldsymbol{\theta}_j^*\}_{j=1}^M$, the likelihood of the data $\mathcal{D}$ is now

$$P(\mathcal{D}|\alpha, G_0) = \int_{\boldsymbol{\pi}} P(\boldsymbol{\pi}|\alpha) \int_{\boldsymbol{\Theta}^*} P(\boldsymbol{\Theta}^*|G_0) \left\{ \prod_{i=1}^{N} \sum_{c_i=1}^{M} P(c_i|\boldsymbol{\pi}) P(\mathbf{x}_i|c_i, \boldsymbol{\Theta}^*) \right\} d\boldsymbol{\Theta}^*\, d\boldsymbol{\pi}, \tag{3.11}$$

with M the truncation number. This is to say in the beginning we have at most M mixture components. Here $P(\boldsymbol{\pi}|\alpha)$ is the stick-breaking process defined in (3.7) with truncation number M, and $P(\boldsymbol{\Theta}^*|G_0) = \prod_{j=1}^M P(\boldsymbol{\theta}_j^*|G_0) = \prod_{j=1}^M P(\boldsymbol{\theta}_j^*|\mu_0, \nu_0)$ in which all $\boldsymbol{\theta}_j^*$'s are *i.i.d.* sampled from G_0. In the curly brackets, $P(c_i|\boldsymbol{\pi})$ is an M-dimensional Multinomial, and $P(\mathbf{x}_i|c_i, \boldsymbol{\Theta}^*) = P(\mathbf{x}_i|\boldsymbol{\theta}_{c_i}^*)$, which means c_i is the indicator variable saying which component data point $\mathbf{x}_i$ takes out of the M mixture components.

Blocked Gibbs Sampling by TDP

Under TDP, a *blocked Gibbs sampling* method exists to solve the inference problem [45]. The joint distribution of the posterior is now $P(\boldsymbol{\pi}, \boldsymbol{\Theta}^*, \mathbf{c}, \alpha, \mu_0, \nu_0 | \mathcal{D})$.

1. $(\boldsymbol{\Theta}^* | \mu_0, \nu_0, \mathbf{c}, \mathcal{D})$: Assume $\{c_1^*, \ldots, c_m^*\}$ are the unique values of the indicator vector $\mathbf{c}$, then draw $(\boldsymbol{\theta}^*_{c_k^*} | \mu_0, \nu_0, \mathbf{c}, \mathcal{D})$ from the density

$$\hat{P}(\boldsymbol{\theta}^*_{c_k^*} | \mu_0, \nu_0, \mathbf{c}, \mathcal{D}) \propto P(\boldsymbol{\theta}^*_{c_k^*} | \mu_0, \nu_0) \prod_{\{i: c_i = c_k^*\}} P(\mathbf{x}_i | \boldsymbol{\theta}^*_{c_k^*});$$

   For $\tilde{\boldsymbol{\theta}} \in \boldsymbol{\Theta}^* \backslash \{\boldsymbol{\theta}^*_{c_1}, \ldots, \boldsymbol{\theta}^*_{c_m}\}$, draw $\hat{P}(\tilde{\boldsymbol{\theta}} | \mu_0, \nu_0, \mathbf{c}, \mathcal{D}) = P(\tilde{\boldsymbol{\theta}} | \mu_0, \nu_0)$.

2. $(\mathbf{c} | \boldsymbol{\pi}, \boldsymbol{\Theta}^*, \mathcal{D})$: For $i = 1, \ldots, N$, draw values

$$\hat{P}(c_i | \boldsymbol{\pi}, \boldsymbol{\Theta}^*, \mathcal{D}) \propto P(c_i | \boldsymbol{\pi}) P(\mathbf{x}_i | \boldsymbol{\theta}^*_{c_i}), \qquad c_i = 1, \ldots, M.$$

3. $(\boldsymbol{\pi} | \alpha, \mathbf{c})$: The posterior is calculated as $\hat{P}(\boldsymbol{\pi} | \alpha, \mathbf{c}) \propto P(\boldsymbol{\pi} | \alpha) P(\mathbf{c} | \boldsymbol{\pi})$. It is known that after the stick-breaking process, the likelihood of $\boldsymbol{\pi}$ given α has a generalized Dirichlet distribution which is conjugate to Multinomial [15]. Therefore, we can draw values from the posterior

$$\pi_1 = V_1^*, \qquad \pi_j = V_j^* \prod_{\ell=1}^{j-1} (1 - V_\ell^*) \quad \text{for } j = 2, \ldots, M,$$

   where V_M^* is fixed as 1, and

$$V_j^* \sim \text{Beta}\left(1 + \sum_{i=1}^{N} \delta_j(c_i),\ \alpha + \sum_{\ell=j+1}^{M} \sum_{i=1}^{N} \delta_\ell(c_i)\right), \quad \text{for } j = 1, \ldots, M-1.$$

4. $(\alpha | \boldsymbol{\pi})$: Assign a prior to α, and draw

$$\hat{P}(\alpha | \boldsymbol{\pi}) \propto P(\alpha) P(\boldsymbol{\pi} | \alpha).$$

5. $(\mu_0, \nu_0 | \boldsymbol{\Theta}^*)$: Assign a prior to (μ_0, ν_0), and draw

$$\hat{P}(\mu_0, \nu_0 | \boldsymbol{\Theta}^*) \propto P(\mu_0, \nu_0) \prod_{j=1}^{M} P(\boldsymbol{\theta}_j^* | \mu_0, \nu_0).$$

This Gibbs sampler is said to be *blocked* because sampling of each distinct $\boldsymbol{\theta}_j^*$ is done once for a group of data points (which share this parameter). This yields faster sampling because the number of distinct parameters is much smaller than the number of data points.

Variational Bayesian TDP

Based on TDP, Blei and Jordan proposed a variational Bayesian method in [9] to directly approximate the *a posteriori* distribution of all the latent variables $\mathbf{V}, \boldsymbol{\Theta}^*, \mathbf{c}$, with $\mathbf{V} := \{V_j\}_{j=1}^M$. In this chapter we call this method *variational Bayesian TDP* (VBTDP). They do not model $\boldsymbol{\pi}$ because $\boldsymbol{\pi}$ is completely determined by $\mathbf{V}$. A fully factorized variational distribution is imposed on the posterior as follows:

$$Q(\mathbf{V}, \boldsymbol{\Theta}^*, \mathbf{c}|\boldsymbol{\Lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{\Psi}) = \prod_{j=1}^{M} Q(V_j|\lambda_j) \prod_{j=1}^{M} Q(\boldsymbol{\theta}_j^*|\mu_j, \nu_j) \prod_{i=1}^{N} Q(c_i|\boldsymbol{\psi}_i),$$

where $Q(V_j|\lambda_j)$ is a Beta distribution $\texttt{Beta}(\lambda_{j,1}, \lambda_{j,2})$, $Q(\boldsymbol{\theta}_j^*|\mu_j, \nu_j)$ takes the conjugate form (2.5), and $Q(c_i|\boldsymbol{\psi}_i)$ is N-dimensional Multinomial. The Jensen's inequality is applied to (3.11), and the following lower bound can be derived based on the variational distribution Q:

$$\begin{aligned}\mathcal{L}_{\text{TDP}} = &\sum_{j=1}^{M} \mathbb{E}_Q[\log P(V_j|\alpha)] + \sum_{j=1}^{M} \mathbb{E}_Q[\log P(\boldsymbol{\theta}_j^*|\mu_0, \nu_0)] + \sum_{i=1}^{N} \mathbb{E}_Q[\log P(c_i|\mathbf{V})] \\ &+ \sum_{i=1}^{N} \mathbb{E}_Q[\log P(\mathbf{x}_i|\boldsymbol{\Theta}^*, c_i)] - \mathbb{E}_Q[\log Q(\mathbf{V}, \boldsymbol{\Theta}^*, \mathbf{c})].\end{aligned}$$

Calculation of this lower bound is very similar to the case of finite mixture model in the last chapter. Here we just give the exact equations for the first and third terms. Applying (2.6) to Beta distribution leads to

$$\begin{aligned}\sum_{j=1}^{M} \mathbb{E}_Q[\log P(V_j|\alpha)] &= \sum_{j=1}^{M} \Big\{ \log\alpha + (\alpha - 1)\mathbb{E}_{Q(V_j|\lambda_j)}[\log(1 - V_j)] \Big\} \\ &= M \log\alpha + (\alpha - 1) \sum_{j=1}^{M} [\Psi(\lambda_{j,2}) - \Psi(\lambda_{j,1} + \lambda_{j,2})].\end{aligned}$$

The third term can be similarly calculated as that for each $i = 1, \ldots, N$,

$$\begin{aligned}\mathbb{E}_Q[\log P(c_i|\mathbf{V})] &= \sum_{j=1}^{M} Q(c_i = j|\boldsymbol{\psi}_i) \mathbb{E}_{Q(\mathbf{V}|\boldsymbol{\Lambda})} \left[\log \left(V_j \prod_{\ell=1}^{j-1} (1 - V_\ell) \right) \right] \\ &= \sum_{j=1}^{M} \psi_{i,j} \left\{ \mathbb{E}_{Q(V_j|\lambda_j)}[\log V_j] + \sum_{\ell=1}^{j-1} \mathbb{E}_{Q(V_\ell|\lambda_\ell)}[\log(1 - V_\ell)] \right\} \\ &= \sum_{j=1}^{M} \psi_{i,j} \left\{ \Big[\Psi(\lambda_{j,1}) - \Psi(\lambda_{j,1} + \lambda_{j,2}) \Big] + \sum_{\ell=1}^{j-1} \Big[\Psi(\lambda_{\ell,2}) - \Psi(\lambda_{\ell,1} + \lambda_{\ell,2}) \Big] \right\}.\end{aligned}$$

Following the recipe of variational Bayesian method which we introduce in the previous chapter, in E-step we maximize the lower bound with respect to the variational parameters $(\boldsymbol{\Lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{\Psi})$. It is not hard to obtain the following update equations:

$$\hat{\psi}_{i,j} \propto \exp\Bigg\{\Big[\Psi(\lambda_{j,1}) - \Psi(\lambda_{j,1}+\lambda_{j,2})\Big] + \sum_{\ell=1}^{j-1}\Big[\Psi(\lambda_{\ell,2}) - \Psi(\lambda_{\ell,1}+\lambda_{\ell,2})\Big] + \mathbb{E}_{\mu_j,\nu_j}[\boldsymbol{\theta}_j^*]^\top \phi(\mathbf{x}_i) + \mathbb{E}_{\mu_j,\nu_j}[-A(\boldsymbol{\theta}_j^*)]\Bigg\}, \tag{3.12}$$

$$\hat{\mu}_j = \mu_0 + \sum_{i=1}^{N}\psi_{i,j}\phi(\mathbf{x}_i), \qquad \hat{\nu}_j = \nu_0 + \sum_{i=1}^{N}\psi_{i,j}, \tag{3.13}$$

$$\hat{\lambda}_{j,1} = 1 + \sum_{i=1}^{N}\psi_{i,j}, \qquad \hat{\lambda}_{j,2} = \alpha + \sum_{i=1}^{N}\sum_{\ell=j+1}^{M}\psi_{i,\ell}, \tag{3.14}$$

where in the first equation we normalize such that $\sum_{j=1}^{M}\hat{\psi}_{i,j} = 1$, for all $i = 1, \ldots, N$.

If a direct update of the hyperparameter is desired, we can do it in the M-step and maximize the lower bound with respect to the model parameters α and (μ_0, ν_0). For α the update is analytical:

$$\hat{\alpha} = \frac{M}{\sum_{j=1}^{M}\Big[\Psi(\lambda_{j,1}+\lambda_{j,2}) - \Psi(\lambda_{j,2})\Big]}, \tag{3.15}$$

and for (μ_0, ν_0) we have

$$M\frac{\partial B(\mu,\nu)}{\partial \mu}\bigg|_{\mu=\hat{\mu}_0,\nu=\hat{\nu}_0} = \sum_{j=1}^{M}\frac{\partial B(\mu_j,\nu_j)}{\partial \mu_j}, \tag{3.16}$$

$$M\frac{\partial B(\mu,\nu)}{\partial \nu}\bigg|_{\mu=\hat{\mu}_0,\nu=\hat{\nu}_0} = \sum_{j=1}^{M}\frac{\partial B(\mu_j,\nu_j)}{\partial \nu_j}, \tag{3.17}$$

which are the same as equations (2.19) and (2.20). We need to use computational methods such as Newton-Raphson to find the optimal values for μ_0 and ν_0. In the paper of Blei and Jordan, the M-step is not derived. Model parameters α and G_0 remains the same in their method. The whole algorithm is summarized in Algorithm 3.1 for clarity.

3.3.3 Dirichlet-Multinomial Allocation

TDP is not the only approximation to DP. Another approximation is called the *Dirichlet-Multinomial allocation* (DMA) [33]. In DMA we also have a large truncation number M, and have similar truncation form as TDP in (3.11). The only difference is that the prior $P(\boldsymbol{\pi}|\alpha)$ now takes an exchangeable M-dimensional Dirichlet distribution

Algorithm 3.1 VBTDP Algorithm for Dirichlet Process Mixture Model

Require: N observed data points $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$.
Require: Initial model parameters α, μ_0 and ν_0.
1: Choose a large enough mixture number M (e.g., set $M = N$). Initialize $\mathbf{\Psi} = \{\psi_{i,j}\}$ randomly and normalize such that $\sum_{j=1}^{M} \psi_{i,j} = 1$, for all $i = 1, \ldots, N$.
2: **repeat**
3: {E-step}
4: **repeat**
5: Update component-specific variables μ_j, ν_j, $\lambda_{j,1}$, $\lambda_{j,2}$ via (3.13) and (3.14);
6: Update membership variables $\psi_{i,j}$ via (3.12);
7: **until** the improvement of log-likelihood is smaller than a threshold.
8: {M-step} (optional)
9: Update concentration parameter α via (3.15);
10: Update hyperparameters μ_0 and ν_0 via (3.16) and (3.17);
11: **until** the improvement of log-likelihood is smaller than a threshold.
Output: Mixture component parameters (μ_j, ν_j)'s, mixing weights calculated in (3.21), and data-component membership $\psi_{i,j}$'s.

$\texttt{Dir}(\alpha/M, \ldots, \alpha/M)$, not a stick-breaking prior as in TDP. It is shown that as $M \to \infty$, DMA approaches a true DP.

It is not hard to see that this approximation to DP is equivalent to the finite mixture model we have discussed in detail in the last chapter. This provides a very nice connection between finite mixture models and DP mixture models, and allows us to achieve similar sparsity in finite mixtures. In the following we will first review the Gibbs sampling algorithm under DMA, and then introduce the *variational Bayesian Dirichlet-Multinomial allocation* (VBDMA) method which is novel for approximated inference in DP mixture models. A qualitative analysis of the functionality of α in VBDMA is given after that, which shows that VBDMA can also learn the mixture numbers automatically for finite mixture models.

Blocked Gibbs Sampling by DMA

Since the key difference between DMA and TDP is the prior for $\boldsymbol{\pi}$, the blocked Gibbs sampling under DMA approximation is the same as that for TDP except Step 3, the sampling of $\boldsymbol{\pi}$ given α and $\mathbf{c}$. By the conjugacy of Dirichlet distribution to Multinomial, the posterior is now

$$\hat{P}(\boldsymbol{\pi}|\alpha, \mathbf{c}) \sim \texttt{Dir}\left(\frac{\alpha}{M} + \sum_{i=1}^{N} \delta_1(c_i), \ldots, \frac{\alpha}{M} + \sum_{i=1}^{N} \delta_M(c_i)\right),$$

without any sampling of V_j's from the Beta distribution. It shares the same property of blocked sampling as that for TDP.

Variational Bayesian DMA

Different from VBTDP, the proposed VBDMA algorithm assigns a variational distribution to latent variables $\boldsymbol{\pi}$, $\boldsymbol{\Theta}^*$ and $\mathbf{c}$, motivated by the DMA approximation to DP. The variational form is now

$$Q(\boldsymbol{\pi}, \boldsymbol{\Theta}^*, \mathbf{c}|\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{\Psi}) = Q(\boldsymbol{\pi}|\boldsymbol{\lambda}) \prod_{j=1}^{M} Q(\boldsymbol{\theta}_j^*|\mu_j, \nu_j) \prod_{i=1}^{N} Q(c_i|\boldsymbol{\psi}_i),$$

in which we directly assign a variational distribution on $\boldsymbol{\pi}$ instead of V_j's. $Q(\boldsymbol{\pi}|\boldsymbol{\lambda})$ is now M-dimensional Dirichlet, with parameter $\boldsymbol{\lambda}$ of length M. This is in contrast to VBTDP where the variational parameter $\boldsymbol{\lambda}$ has length $2M$. Using less variational parameters will be shown to be more robust. Other variational distributions remain the same as those for VBTDP. The lower bound of the log likelihood is now

$$\begin{aligned}\mathcal{L}_{\text{DMA}} = \mathbb{E}_Q[\log P(\boldsymbol{\pi}|\alpha)] + \sum_{j=1}^{M} \mathbb{E}_Q[\log P(\boldsymbol{\theta}_j^*|\mu_0, \nu_0)] + \sum_{i=1}^{N} \mathbb{E}_Q[\log P(c_i|\boldsymbol{\pi})] \\ + \sum_{i=1}^{N} \mathbb{E}_Q[\log P(\mathbf{x}_i|\boldsymbol{\Theta}^*, c_i)] - \mathbb{E}_Q[\log Q(\boldsymbol{\pi}, \boldsymbol{\Theta}^*, \mathbf{c})],\end{aligned}$$

the same as (2.14) in Chapter 2 with slightly different notations. The E-step, which we repeat here with new notations, is given as follows:

$$\hat{\psi}_{i,j} \propto \exp\left\{\left[\Psi(\lambda_j) - \Psi\left(\sum_{\ell=1}^{M} \lambda_\ell\right)\right] + \mathbb{E}_{\mu_j,\nu_j}[\boldsymbol{\theta}_j^*]^\top \phi(\mathbf{x}_i) + \mathbb{E}_{\mu_j,\nu_j}[-A(\boldsymbol{\theta}_j^*)]\right\}, \tag{3.18}$$

$$\hat{\mu}_j = \mu_0 + \sum_{i=1}^{N} \psi_{i,j}\phi(\mathbf{x}_i), \qquad \hat{\nu}_j = \nu_0 + \sum_{i=1}^{N} \psi_{i,j}, \qquad \hat{\lambda}_j = \frac{\alpha}{M} + \sum_{i=1}^{N} \psi_{i,j}. \tag{3.19}$$

As can be seen clearly, the main differences of this E-step compared to that for VBTDP are the updates to mixture memberships $\psi_{i,j}$'s and cluster weights λ_j's. In the M-step we maximize the lower bound with respect to the model parameters α and (μ_0, ν_0). The updates for (μ_0, ν_0) are the same as those for TDP, and for α we need to solve

$$K\left[\Psi\left(\frac{\hat{\alpha}}{M}\right) - \Psi(\hat{\alpha})\right] = \sum_{j=1}^{M}\left[\Psi(\lambda_j) - \Psi\left(\sum_{j'=1}^{M} \lambda_{j'}\right)\right]. \tag{3.20}$$

The algorithm is summarized in Algorithm 3.2.

3.3.4 Comparison of Variational Bayesian Methods for DP

In this section we compare the two variational Bayesian methods for DP mixture models. They are based on two different finite approximations to a true DP, and assume different

Algorithm 3.2 VBDMA Algorithm for Dirichlet Process Mixture Model

Require: N observed data points $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$.
Require: Initial model parameters α, μ_0 and ν_0.
Choose a large enough mixture number M (e.g., set $M = N$). Initialize $\mathbf{\Psi} = \{\psi_{i,j}\}$ randomly and normalize such that $\sum_{j=1}^{M} \psi_{i,j} = 1$, for all $i = 1, \ldots, N$.
repeat
  {E-step}
  repeat
    Update component-specific variables μ_j, ν_j, λ_j via (3.19);
    Update membership variables $\psi_{i,j}$ via (3.18);
  until the improvement of log-likelihood is smaller than a threshold.
  {M-step} (optional)
  Update concentration parameter α via (3.20);
  Update hyperparameters μ_0 and ν_0 via (3.16) and (3.17);
until the improvement of log-likelihood is smaller than a threshold.
Output: Mixture component parameters (μ_j, ν_j)'s, mixing weights calculated as $\pi_j = \lambda_j / \sum_{j'=1}^{M} \lambda_{j'}$, and data-component membership $\psi_{i,j}$'s.

variational priors for the mixing weight $\boldsymbol{\pi}$. Starting from the finite mixture modeling perspective, VBDMA is more attractive because it yields the same variational updates as that for finite mixture model. It also explains the functionality of α in finite mixture modeling which we analyze qualitatively in the following.

Sparsity of Infinite Mixture under VBDMA

Let us first fix α and focus on the VBDMA solution (3.18)$\sim$(3.19) with a large enough M (by setting, e.g., $M = N$). With an uninformative initialization of all the variational parameters (e.g., we set $\mu_j = \mu_0$, $\nu_j = \nu_0$ and $\lambda_j = \alpha/M$ for all $j = 1, \ldots, M$), we first fit the mixture membership $\psi_{i,j}$ using (3.18), and then update the other parameters using (3.19). Since all the components have the same prior terms $\mathbb{E}_{\boldsymbol{\lambda}}[\log \pi_j] = \Psi(\lambda_j) - \Psi\left(\sum_{j'=1}^{N} \lambda_{j'}\right)$ initially, in (3.18) the assignment probabilities $\psi_{i,j}$ will solely depend on the *empirical explanation* of $\mathbf{x}_i$ given component parameter $\boldsymbol{\theta}_j$, i.e., $\mathbb{E}_{\mu_j,\nu_j}[\boldsymbol{\theta}_j^{*\top} \phi(\mathbf{x}_i) - A(\boldsymbol{\theta}_j^*)]$. This will make the updated $\psi_{i,j}$ unevenly distributed, and after normalizing such that $\sum_j \psi_{i,j} = 1$ for each i, we can observe small assignment probabilities $\sum_{i=1}^{N} \psi_{i,j}$ for those "unlikely" components. When these values are fed into (3.19), these components will get smaller values for λ_j, and thus the prior term $\mathbb{E}_{\boldsymbol{\lambda}}[\log \pi_j]$ in (3.18) will also get smaller, which makes $\psi_{i,j}$ more sharply distributed. Eventually, these components will get $\psi_{i,j} = 0$, for all data points $\mathbf{x}_i$. This in turn leads to $\mu_j = \mu_0$, $\nu_j = \nu_0$ and $\lambda_j = \alpha/M$, all equal to the hyperparameters. When M is very large, α/M is very small, and these components almost have no chance to get bigger $\psi_{i,j}$ in the future for some data $\mathbf{x}_i$, as seen from (3.18). Finally when the algorithm converges, we obtain only a small number of *effective*

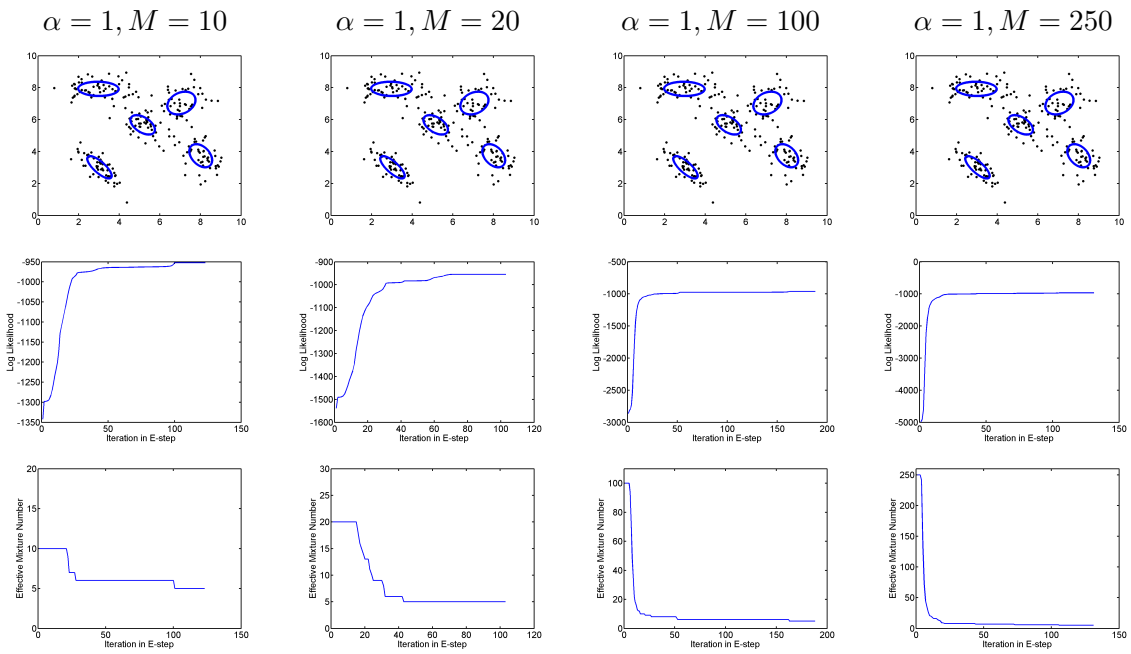


Figure 3.3: Fitting the toy Gaussian mixture data using VBDMA with fixed α but different M values. The middle row shows the log likelihood with respect to each iteration in VBDMA, and the bottom row shows the effective numbers of mixtures along all iterations. Initialization is done with $\beta_0 = 0.01$, $\nu_0 = 2$ and random initialization for $\mathbf{\Psi}$.

components.

This phenomenon is illustrated in Figure 3.3 for the mixture of 5 Gaussians we studied in the last chapter. When we fix $\alpha = 1$, sparsity is obtained for all the M's in the figure, ranging from 10 to 250 which is the total number of data points. When M becomes larger, the fitted number of mixtures does not vary too much, but tends to be stable (upper row). The middle row shows the curve of log likelihood with respect to each iteration, and we also show the effective number of mixtures for each iteration in the bottom row for each M. It can be seen that each time the effective number decreases, the log likelihood is improved. Therefore, VBDMA can be used to find this effective number. As will be seen next, the strength of sparsity is not random, but depends strongly on the parameter α.

Functionality of α in VBDMA

Now we investigate the situation in which M is fixed, but α is allowed to change. When α is small, updates for λ_j will mostly depend on the empirical assignments $\sum_{i=1}^{N} \psi_{i,j}$ in (3.19), and thus quickly get unbalanced. Then similar to the previous discussion, $\psi_{i,j}$ will get an even sharper distribution in the next update, and the algorithm quickly converges to a small number of components that fit the data best. In the limiting case that $\alpha = 0$,

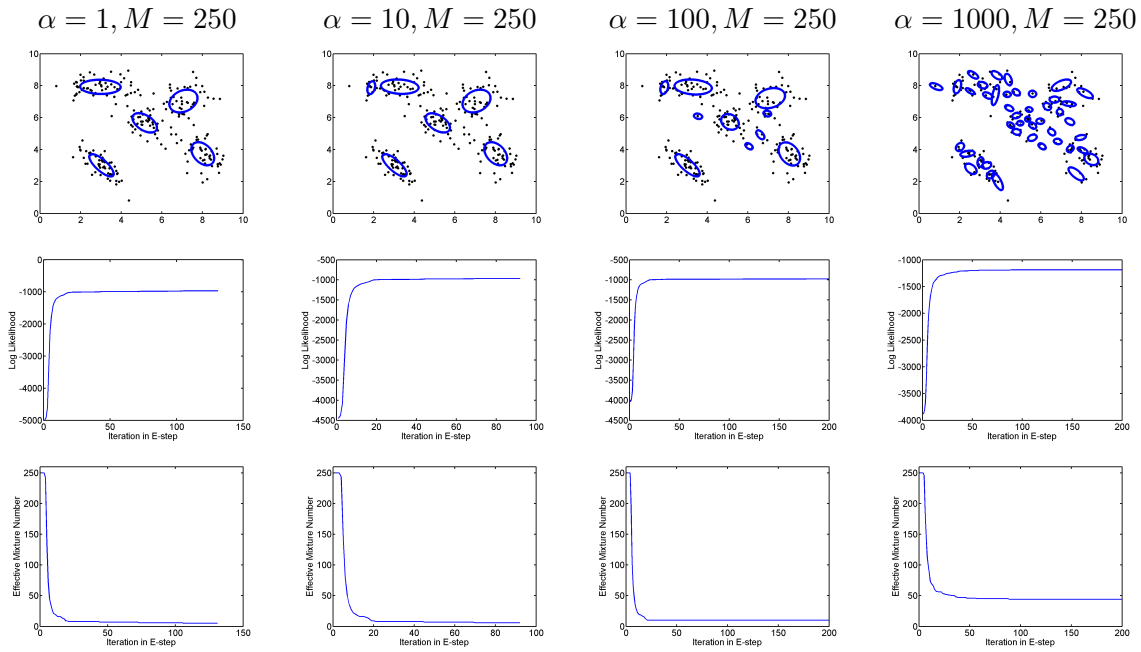


Figure 3.4: Fitting the toy Gaussian mixture data using VBDMA with fixed K but different α values. The middle row shows the log likelihood with respect to each iteration in VBDMA, and the bottom row shows the effective numbers of mixtures along all iterations. Initialization is done with $\beta_0 = 0.01$, $\nu_0 = 2$ and random initialization for $\mathbf{\Psi}$.

λ_j's are purely determined by empirical updates, and we are making a *maximum likelihood* estimate for the mixing weights $\boldsymbol{\pi}$.

On the other hand when α is relatively large, the prior term α/M will dominate the update equation (3.19), and thus λ_j will not be very unbalanced in one step. This will in turn make the update equation (3.18) smooth for $\psi_{i,j}$, and more components will survive than that with small α. As the iteration continues, certainly some components will be "dead" because of their poor fit to the data, but the death rate is much slower and we could expect more components left after convergence. A limiting case for this is to let $\alpha \to \infty$, which corresponds to fix the $\boldsymbol{\pi}$ *a priori* to be $\{\frac{1}{M}, \ldots, \frac{1}{M}\}$, and does not change it in the whole learning process. This leads to non-sparsity of the learned model.

Figure 3.4 shows the results with different α values. With M fixed as 250 which is the total number of data points, smaller α (e.g., 1 or 10) leads to higher sparsity, and larger α (e.g., 1000) results in more components. Therefore, choosing a suitable α means choosing a desirable number of mixtures.

Our final experiment on this toy data is to randomize the learning algorithm 20 times with different α's and M's, and investigate the number of mixtures after learning. The results are given in the top table in Table 3.1, with means and standard deviations. Sparsity is obtained for almost all the M's with small α values, but for large α's sparsity is not

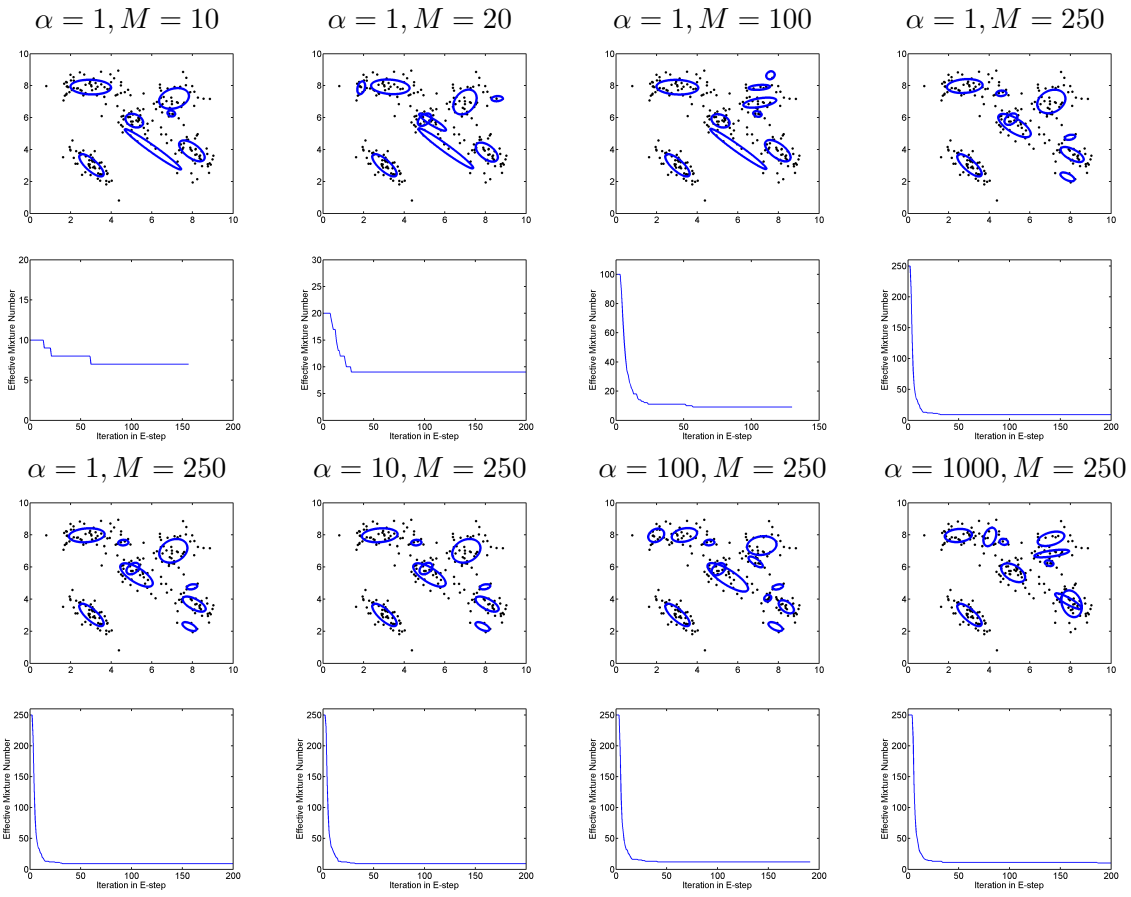


Figure 3.5: Fitting the toy Gaussian mixture data using VBTDP with different α and M values. The second and fourth rows show the effective numbers of mixtures along all iterations. Initialization is done with $\beta_0 = 0.01$, $\nu_0 = 2$ and random initialization for $\mathbf{\Psi}$.

always achieved. This is because the large α value induces a strong prior and almost all the components survive after learning. When α becomes larger and larger, the curve will approach the line $y = x$ with small deviations, which indicates that we are exactly fitting M components without model selection.

Discussion

We also tested the VBTDP on this Gaussian mixture data, and the results are shown in Figure 3.5 and the bottom table of Table 3.1. Sparsity is also achieved for VBTDP, but its behavior is different from VBDMA. With fixed $\alpha = 1$, improving M will lead to a slightly more number of mixtures, as seen in the figure and the table. When α is increased with fixed $M = 250$, we cannot obtain a non-sparse mixture model as in VBDMA. From our experience, the model fitting of VBTDP is not as good as VBDMA (as can be seen

Table 3.1: The number of learned mixture components (means and standard deviations) in VBDMA (top) and VBTDP (bottom) for the toy Gaussian data with different initial K and α values. The experiments are repeated 20 times randomly.

	$K = 5$	$K = 10$	$K = 20$	$K = 50$	$K = 100$	$K = 250$
$\alpha = 1$	4.45 ± 0.60	6.00 ± 1.03	6.70 ± 0.86	7.15 ± 1.27	6.85 ± 1.42	6.25 ± 1.16
$\alpha = 10$	4.95 ± 0.22	7.80 ± 1.01	8.65 ± 1.14	7.35 ± 1.04	7.10 ± 1.37	6.45 ± 1.10
$\alpha = 100$	5.00 ± 0.00	10.00 ± 0.00	19.90 ± 0.31	21.20 ± 1.58	11.40 ± 1.76	7.80 ± 1.40
$\alpha = 1000$	5.00 ± 0.00	10.00 ± 0.00	20.00 ± 0.00	49.65 ± 0.49	69.05 ± 2.19	45.05 ± 2.06
$\alpha = 10000$	5.00 ± 0.00	10.00 ± 0.00	20.00 ± 0.00	49.90 ± 0.31	85.10 ± 2.47	87.75 ± 2.07

	$K = 5$	$K = 10$	$K = 20$	$K = 50$	$K = 100$	$K = 250$
$\alpha = 1$	4.50 ± 0.61	6.30 ± 1.03	7.35 ± 1.46	8.15 ± 1.39	8.55 ± 1.23	9.00 ± 1.62
$\alpha = 10$	4.65 ± 0.49	6.75 ± 0.91	7.85 ± 1.14	8.50 ± 1.24	8.80 ± 1.32	9.15 ± 1.09
$\alpha = 100$	4.60 ± 0.60	7.55 ± 1.15	8.95 ± 1.79	9.60 ± 1.70	9.90 ± 1.21	10.10 ± 1.33
$\alpha = 1000$	4.65 ± 0.49	7.80 ± 1.01	10.45 ± 1.47	10.80 ± 2.07	11.15 ± 2.06	11.10 ± 2.31
$\alpha = 10000$	4.60 ± 0.50	7.75 ± 1.02	10.20 ± 1.32	11.05 ± 2.01	11.50 ± 1.82	11.40 ± 2.19

from Figure 3.3, 3.4 and 3.5), but due to the local minima problem and sensitivity to the initialization, it is difficult to make a general conclusion.

It is known that the stick-breaking variational form in VBTDP induces a *generalized Dirichlet distribution* to the weight $\boldsymbol{\pi}$, and uses twice as many parameters as a Dirichlet distribution [15]. Let us rewrite $\alpha_j = \lambda_{j,1}$ and $\beta_j = \lambda_{j,2}$ to replace the variational parameter $\boldsymbol{\Lambda}$ in VBTDP, the density for $\boldsymbol{\pi}$ can now be written as

$$P(\boldsymbol{\pi}|\boldsymbol{\Lambda}) = \pi_M^{\beta_{M-1}-1} \prod_{j=1}^{M-1} \frac{\Gamma(\alpha_j + \beta_j)}{\Gamma(\alpha_j)\Gamma(\beta_j)} \pi_j^{\alpha_j - 1} (1 - \pi_1 - \cdots - \pi_j)^{\beta_j - \alpha_{j+1} - \beta_{j+1}}.$$

The generalized Dirichlet distribution is also conjugate to Multinomial distribution, and it reduces to the Dirichlet distribution if $\beta_j = \alpha_{j+1} + \beta_{j+1}$, for $j = 1, \ldots, M-1$. After convergence in VBTDP, the weight vector $\boldsymbol{\pi}$, which is the soft weights of the clustering structure, can be obtained as the expectation with respect to the generalized Dirichlet distribution, calculated as

$$\mathbb{E}[\pi_1] = \frac{\alpha_1}{\alpha_1 + \beta_1}, \qquad \mathbb{E}[\pi_j] = \frac{\alpha_j}{\alpha_j + \beta_j} \prod_{\ell=1}^{j-1} \frac{\beta_\ell}{\alpha_\ell + \beta_\ell}, \quad \text{for } j = 2, \ldots, M. \tag{3.21}$$

Given this result, in the E-step VBDMA can be viewed as doing the same approximation as VBTDP, except that extra constraints are maintained such that $\beta_j = \alpha_{j+1} + \beta_{j+1}$, for $j = 1, \ldots, M-1$. Although the general Dirichlet distribution is more flexible than Dirichlet distribution, using it as the variational distribution to approximate the true posterior is more likely to overfit. Other properties of generalized Dirichlet distribution include that each dimension of $\boldsymbol{\pi}$ is not always *negatively correlated* to other dimensions (i.e,. observing

a sample from one dimension will surely increase the expected value of the parameter for this dimension, but decrease those for the other dimensions) as in Dirichlet distribution, and that the order of these dimensions is important for sampling and learning [76]. Both properties are unnecessary for mixture modeling, and the latter is even contradictory to Bayesian exchangeability in this context. In the next section we will compare these two methods using some real data to show the difference.

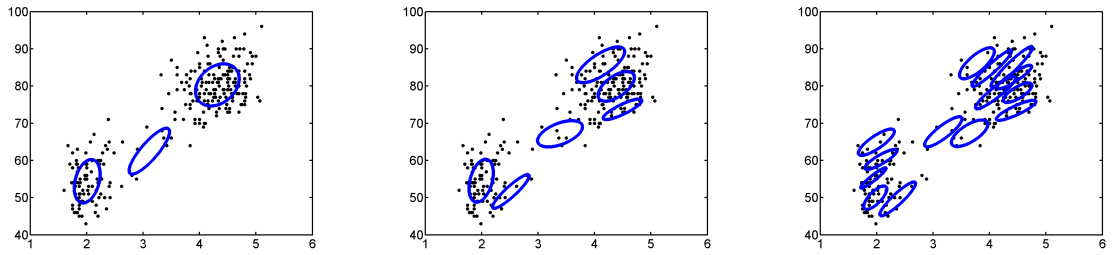
3.4 Empirical Studies

In this section we study the proposed model on some real data with Gaussian mixture models. We use the following two data sets. One is the "Old Faithful" data that contains 272 observations from the Old Faithful Geyser in the Yellowstone National Park.[2] The data is two-dimensional, and each observation consists of the duration of the eruption and the waiting time to the next eruption. The other data set is a subset of Corel image database and contains 1021 images.[3] Color histogram features for each image are extracted, and then mapped to a five-dimensional space (using standard PCA) for mixture modeling.

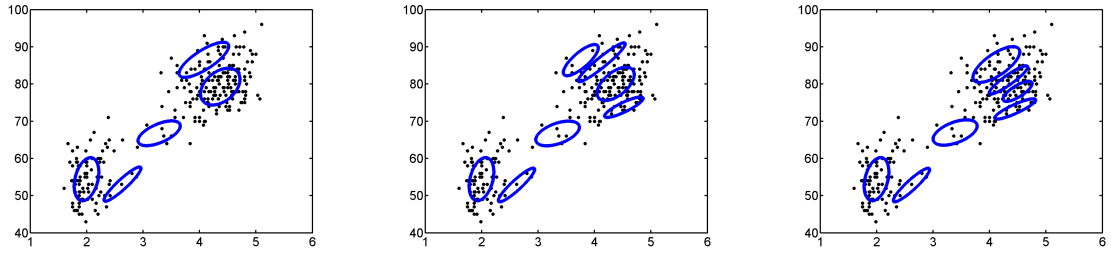
3.4.1 Old Faithful Data

We fit a mixture of Gaussians to the "Old Faithful" data, with K set to 272 initially, identical to the number of data points. Other parameters are initialized as $\beta_0 = 0.1$, $\nu_0 = 2$, $\mathbf{m}_0$ the sample mean, and $\mathbf{W}_0$ the inverse of the sample covariance divided by β_0. The results for some α's are shown in top row of Figure 3.6. It can be seen that small α leads to small number of Gaussians, and larger α leads to more. Choosing α to be 100, 500 and 1000 results in 3, 6 and 15 Gaussians, respectively. All of them fit the data very well, but in different granularities. The final log likelihoods of the three model fitting are -1174.55, -1187.75 and -1253.17, respectively. One can do a model selection using this likelihood value and prefer the first one, but now there is no need to choose K *a priori* because this number is automatically determined by the VBDMA algorithm with a suitable α. In the bottom figure we also illustrate the curves of log likelihood and the effective number of components with respect to each iteration for $\alpha = 100$, and it can be seen that each time the effective mixture number decreases, the likelihood has a noticeable increase (we mark three of them using dashed lines). Model fitting based on VBTDP is shown in the middle row with the same initial parameters, and it is seen that the α values does not affect the number of learned Gaussians that much. As discussed before, we lose the sparsity control of the mixture modeling.

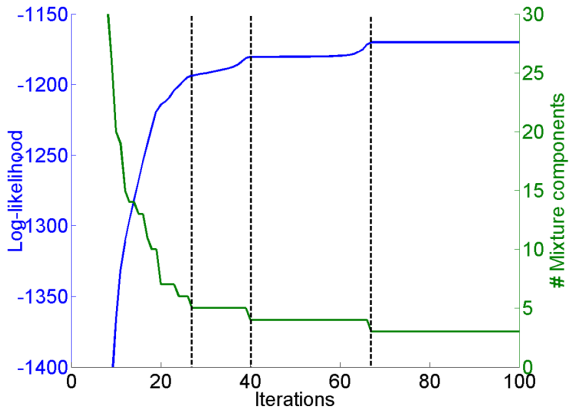
[2]The data is available at `http://www.uvm.edu/~dhowell/StatPages/More_Stuff/geyser2a.dat`.
[3]The Corel images are originally from `http://elib.cs.berkeley.edu/corel/`.



$\alpha = 100, K = 272$ $\alpha = 500, K = 272$ $\alpha = 1000, K = 272$



$\alpha = 100, K = 272$ $\alpha = 500, K = 272$ $\alpha = 1000, K = 272$



Likelihood and mixture numbers of all iterations for $\alpha = 100$

Figure 3.6: Fitting a mixture of Gaussians on the "Old Faithful" data set using VBDMA (top row) and VBTDP (middle row) with fixed $K = 272$ and different α values. The bottom plot shows the log-likelihood and the number of effective mixture components with respect to each iteration in E-step of VBDMA.

3.4.2 Image Data

The images are fitted with Gaussian mixtures with K pre-defined to be 1021, the number of images in the whole data set. Model parameters are set as $\beta_0 = 0.1$, $\nu_0 = 5$, with $\mathbf{m}_0$ and $\mathbf{W}_0$ defined as above. Using VBDMA method, we obtain respectively 18 and

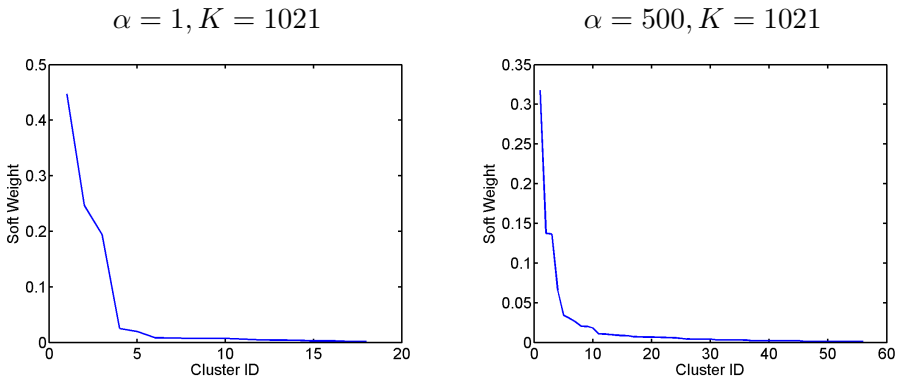


Figure 3.7: Component weights after learning with VBDMA on the Corel image data set.

56 mixture components with $\alpha = 1$ and $\alpha = 500$, for which the soft weights are shown in descending order in Figure 3.7. It can be read that for $\alpha = 1$, only 5 out of the 18 components have weights larger than 0.01. For $\alpha = 500$ this number is 13. To show the characteristic of each component, we illustrate the 10 images which have the highest weight for each component, one row per component, in Figure 3.8 and Figure 3.9, respectively for $\alpha = 1$ and $\alpha = 500$. It can be seen that images of different color histograms can be grouped reasonably, especially for $\alpha = 1$. Setting $\alpha = 500$ will lead to more components, among which, for instance, castles are split into two components 1 and 3 due to different colors. In general, larger α leads to more but still meaningful components in the mixture modeling. There are noises in this mixture modeling, since fitting such a flexible Gaussian mixture model in image data with a reasonable number of dimensionality is not an easy task.

The same parameter settings are used for the VBTDP method, and the results are shown in Figure 3.10. For the two different α values, VBTDP obtains 16 and 20 components, respectively, and among them 10 and 11 have weights larger than 0.01. We also show the images of each component in Figure 3.10, and it can be seen that...

3.5 Summary

This chapter answers the important question of how to identify the number of mixture components, and builds connections to infinite mixture modeling with Dirichlet processes. It turns out that the symmetric Dirichlet prior proposed in the last chapter for the mixing weights is the key point to approximate a DP in the limiting case of $K \to \infty$. While this prior has been used in several previous works [30, 16], its connection to non-parametric Bayesian framework with Dirichlet process is never uncovered. Several researchers have observed the phenomena that their models (which use this prior) can automatically detect the number of mixtures, but they either provide no explanation for this [16], or give the wrong one [30].

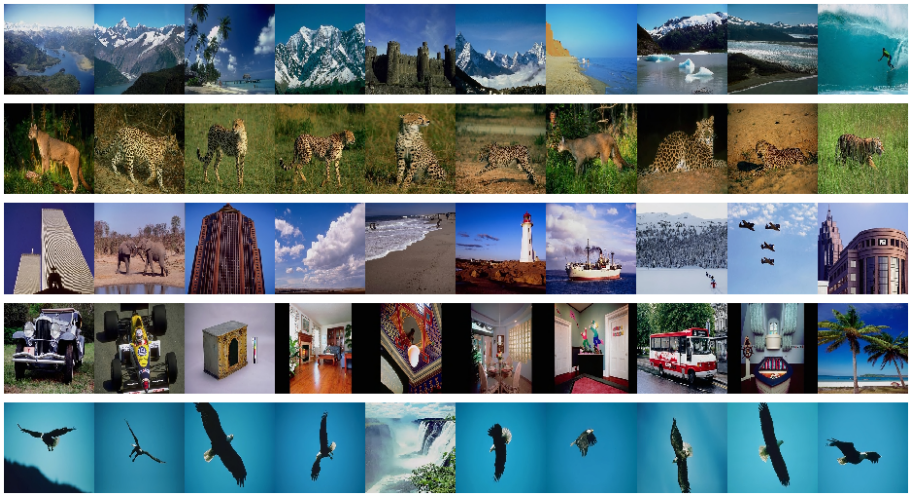


Figure 3.8: The 5 clusters of largest weights using VBDMA with $\alpha = 1$, one per row with 10 images of largest membership weights.

We emphasize that this connection to Dirichlet process is important, in that:

- This explains why parametric Bayesian modeling of finite mixtures obtains similar results as non-parametric modeling. Automatically detecting number of mixtures is one important result among others.
- This makes it possible to borrow known results about Dirichlet process from statistics community and apply them on finite parametric modeling. For example, the concentration parameter α in DP is known for controlling the flexibility of generating new mixtures. It is also the case in finite modeling, as seen in this chapter.
- Variational methods originally developed for finite mixture models can (in general) be applied for model inference and learning with DP prior. This makes it possible to propose complicated models with possibly infinite number of mixtures, and then use the known variational methods to solve it with the approximation of finite mixtures. This can be applied, for instance, for infinite mixtures of probabilistic PCA [69] and infinite mixtures of latent Dirichlet allocation (LDA) [10]. These are however left as future works.

Another variational Bayesian method for infinite mixture model is VBTDP which is based on TDP as a finite approximation to DP. Some empirical studies are performed in this chapter to compare these two variational solutions. VBDMA is shown to be superior to VBTDP, and has a nice property of controlling the sparsity of mixture modeling with a single hyperparameter α.

For future work one can apply the proposed model to more complex exponential family distributions for various applications, e.g., infinite mixture of probabilistic PCA models

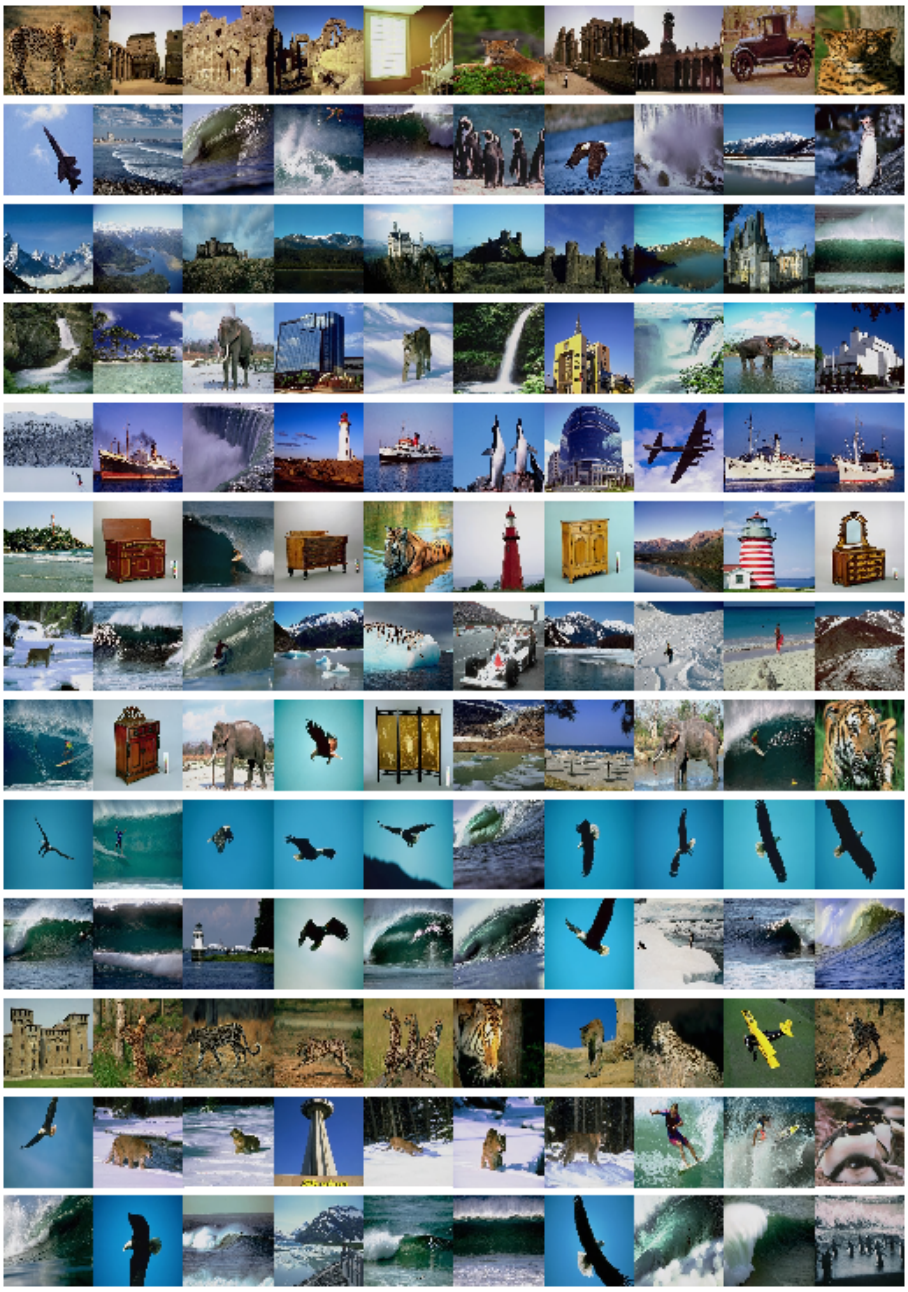


Figure 3.9: The 13 clusters of largest weights using VBDMA with $\alpha = 500$, one per row with 10 images of largest membership weights.

and infinite mixture of latent Dirichlet allocation models. The usage of Dirichlet processes in relational learning is also worth investigating [78].

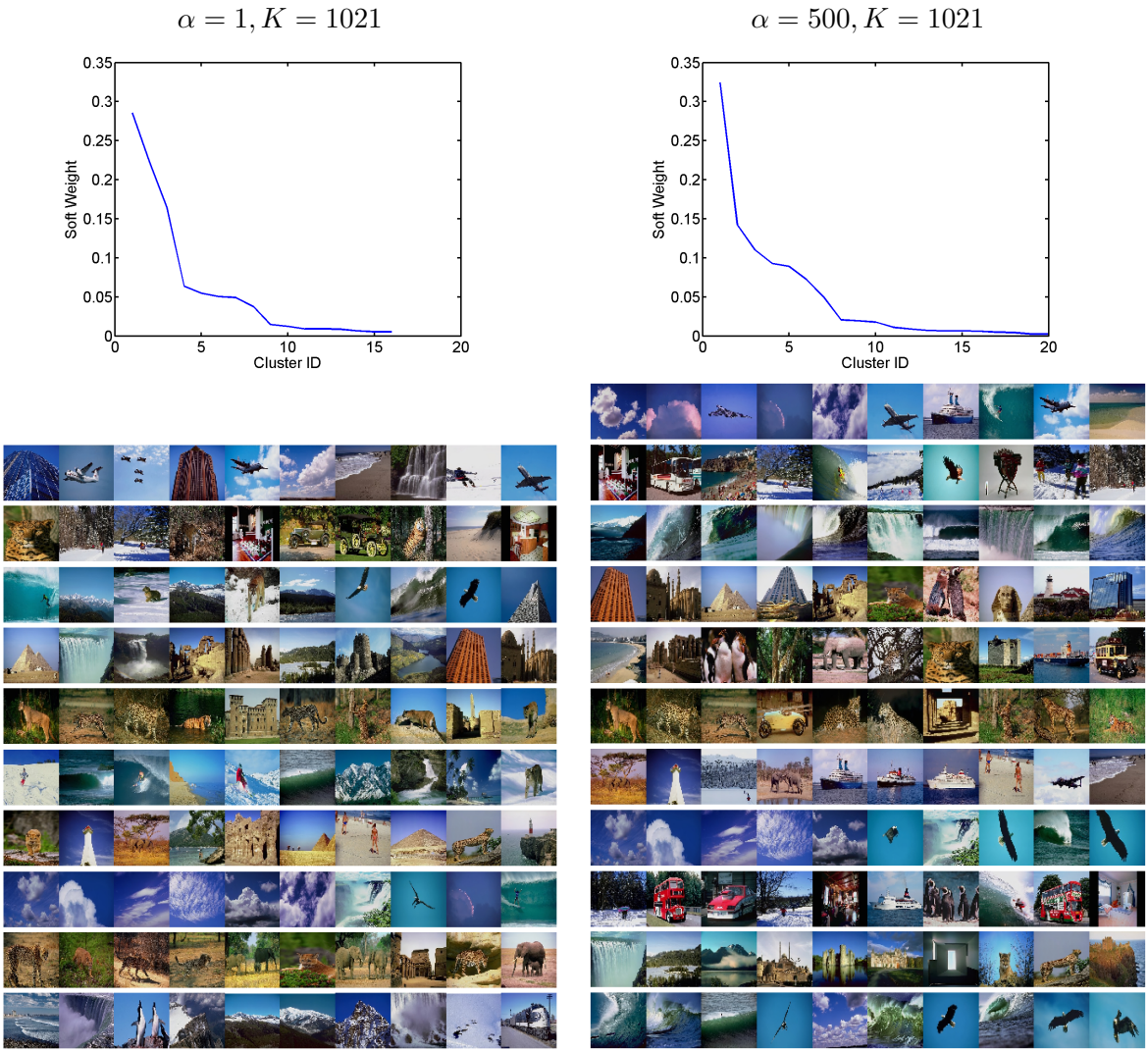


Figure 3.10: Results of VBTDP on the Corel image data set.

Part II

Probabilistic Projection Models

Chapter 4

Overview of Projection Models

Dimensionality reduction aims to find a low-dimensional representation of the data. This is important for many machine learning and data mining problems, either because the original dimensionality is too high to apply certain algorithms, or because there are noisy dimensions which we want to remove. Other motivations include improving the efficiency of learning, and accelerating certain algorithms. One typical example of high-dimensional data is the text document corpus for information retrieval. In the vector space model (VSM), each document is represented as a vector of terms, and each term defines one dimension in the space. Hence the dimensionality of the space is the number of distinct terms in the corpus, which could easily reach ten thousands for a middle-sized corpus. Another example is in face recognition, where each face image is cropped to 32×32 pixels and has at least one feature for each pixel. This also generates a very high dimensional space for the face images.

One way of reducing the number of features is to select a subset of features which are informative enough or sufficient for the learning algorithms. This is in general called *feature selection* and has been extensively studied in the data mining community. For information retrieval, removing the stop-words (which are the common words like *the* and *of*) is one popular feature selection technique. Feature selection is in general fast and easy to implement, but in many applications it is not the individual features that are important, but a (weighted) *combination* of them. Extracting these features is not only important for the learning algorithms, but also useful for understanding the data. This so-called *feature transformation* or *projection* is the main topic of this overview.

Most of the research on projection methods works on *continuous data*, i.e., each feature dimension can take a real number in some domain. The face image data belongs to this category. In this chapter, however, we consider both continuous data and discrete data, where for *discrete data* the domain of each feature dimension is the set of (non-negative) integers. One typical example is the *bag-of-words* representation of the document-word matrix, where each entry denotes the occurrences of the corresponding word in the document. In the rest of this chapter we summarize different projection methods for these two

types of data (in Section 4.1 and Section 4.2 respectively), and in Section 4.3 we point out the road-map of our contributions in the following chapters.

We summarize our notations here for clarity. In what follows we consider a set of N data (e.g., documents or images), and each data i is described by an M-dimensional feature vector $\mathbf{x}_i \in \mathcal{X}$. For continuous data $\mathcal{X} \subset \mathbb{R}^M$, and for discrete data $\mathcal{X} \subset \mathbb{Z}_+^M$. For projection we aim to derive a mapping $\Psi : \mathcal{X} \mapsto \mathcal{Z}$ which maps the input features into a K-dimensional space $\mathcal{Z}$.

4.1 Projection Models for Continuous Data

In this section we give an overview of projection models for continuous data, mostly from the perspective of global projection. Some local projection methods are summarized at the end of this section.

4.1.1 Principal Component Analysis (PCA)

Probably the most popular projection method for continuous data is the *principal component analysis* (PCA), which performs a *singular value decomposition* (SVD) to the data matrix and obtains the sub-eigenspace with large singular values [46]. Denote $\mathbf{X} := [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$ the input matrix after centralization, i.e., we subtract the sample mean $\bar{\mathbf{x}} := \frac{1}{N}\sum_{i=1}^N \mathbf{x}_i$ from each input. Let $\mathbf{X} = \mathbf{V}\mathbf{D}\mathbf{U}^\top$ be the SVD of $\mathbf{X}$, where $\mathbf{V}$ and $\mathbf{U}$ are $N \times N$ and $M \times M$ column orthogonal matrices, respectively, and $\mathbf{D}$ is $N \times M$ diagonal matrix with singular values sorted in descending order along the diagonal. Then it is known that the first K columns of $\mathbf{U}$, which we denote $\mathbf{U}_K$, defines the mapping Ψ for PCA. This means that for any data point $\mathbf{x}_*$, its projection onto the principal component space is calculated as $\mathbf{z}_* = \mathbf{U}_K^\top \mathbf{x}_*$. It is easy to verify that the projections of input data $\mathbf{X}$ onto the principal component space can be calculated as $\mathbf{Z} := \mathbf{V}_K\mathbf{D}_K \in \mathbb{R}^{N\times K}$, where $\mathbf{V}_K$ contains the first K columns of $\mathbf{V}$, and $\mathbf{D}_K$ is the top left $K \times K$ sub matrix of $\mathbf{D}$.

PCA has many important properties. The most well-known property of PCA, which also motivates the name *principal components*, is that the K *principal axes* $\mathbf{u}_k$, $k \in \{1, \ldots, K\}$, are those orthonormal axes onto which the retained variance under projection is maximal [41]. This can be done by eigen-decomposing the sample covariance matrix $\mathbf{S} := \frac{1}{N}\mathbf{X}^\top\mathbf{X}$, and taking the K dominant eigenvectors (i.e., those with the largest associated eigenvalues) as the projection axes. It is easy to verify that this is equivalent to the SVD interpretation we illustrated above. Figure 4.1 shows the two principal axes for a two-dimensional toy data.

Another important property of PCA is regarding to the reconstruction error of each data point from its projection [56]. Let $\mathbf{z}_i \in \mathbb{R}^K$ be the projection of data point $\mathbf{x}_i$ onto a K-dimensional subspace, and $\mathbf{W} \in \mathbb{R}^{M\times K}$ be the orthonormal *loading matrix* which maps from the subspace back to the original feature space. Then the *reconstruction error*

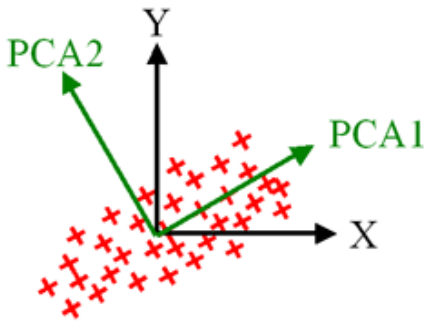


Figure 4.1: Illustration of PCA on a 2D toy data.

of this projection can be calculated using vector 2-norm as

$$\sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{W}^\top \mathbf{z}_i\|^2 = \|\mathbf{X} - \mathbf{Z}\mathbf{W}^\top\|_F^2,$$

where $\|\cdot\|_F$ denotes the Frobenius norm for matrix, and $\mathbf{Z} := [\mathbf{z}_1, \ldots, \mathbf{z}_N]^\top$. It can be proved that the following optimization problem

$$\min_{\mathbf{W},\mathbf{Z}} \quad \|\mathbf{X} - \mathbf{Z}\mathbf{W}^\top\|_F^2 \tag{4.1}$$

$$\text{subject to:} \quad \mathbf{W} \in \mathbb{R}^{M\times K}, \mathbf{W}^\top\mathbf{W} = \mathbf{I} \tag{4.2}$$

has solution $\mathbf{W} = \mathbf{U}_K$ and $\mathbf{Z} = \mathbf{V}_K\mathbf{D}_K$, which recovers the PCA solutions. This indicates that the subspace obtained in PCA projection, which is called the *principal subspace*, is *optimal* under the criteria of minimizing the reconstruction error in vector 2-norm.

Due to the clean mathematical formulation and the nice properties, PCA has been used in various data mining applications. For instance, the principal components are known as *eigenfaces* in face recognition [71], and *latent semantics* in information retrieval [17]. Both of them provide essential information about the underlying data. In the latter case, the PCA formulation is in general called *latent semantic indexing* (LSI) or *latent semantic analysis* (LSA).

4.1.2 Kernel Principal Component Analysis (Kernel PCA)

As can be seen from the SVD formulation, PCA is looking for a *linear projection* of the data, where each principal axis is a linear combination of the original feature dimensions. In some applications like image retrieval and bioinformatics, it makes sense to consider *non-linear projections*, because linear projections are too restrictive. One straightforward idea of extending linear PCA to non-linear PCA is to first map the original data points into a new feature space $\mathcal{F}$, i.e., define $\boldsymbol{\phi} : \mathbf{x} \in \mathcal{X} \mapsto \boldsymbol{\phi}(\mathbf{x}) \in \mathcal{F}$, and then perform linear

PCA to the new data matrix $\boldsymbol{\phi}(\mathbf{X}) := [\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_N)]^\top$ in the new space. One can, for example, define a set of *basis functions* on the original feature dimensions to obtain the mapping $\boldsymbol{\phi}(\cdot)$, but there exists a technology called *kernel PCA* in which we do not need to know the explicit mapping $\boldsymbol{\phi}(\cdot)$. This is based on a *dual* formulation of the SVD solution to linear PCA [61].

Let $\mathbf{K} := \mathbf{X}\mathbf{X}^\top$ be the $N \times N$ matrix with the (i, j)-th entry $\mathbf{K}_{ij} = \mathbf{x}_i^\top \mathbf{x}_j = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$, the inner-product of $\mathbf{x}_i$ and $\mathbf{x}_j$ in $\mathcal{X}$. This matrix is sometimes called the *Gram matrix*. Then if the SVD of data matrix $\mathbf{X}$ is $\mathbf{V}\mathbf{D}\mathbf{U}^\top$, the eigen-decomposition of $\mathbf{K}$ would be

$$\mathbf{K} = \mathbf{V}\mathbf{D}^2\mathbf{V}^\top,$$

because $\mathbf{U}$ is an orthonormal matrix satisfying $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$. From this starting point, we can solve PCA in the dual form as follows:

1. Construct the Gram matrix $\mathbf{K}$ using the inner-product in $\mathcal{X}$;
2. Calculate the eigen-decomposition of $\mathbf{K}$ to get the eigenvectors in $\mathbf{V}$ and square roots of eigenvalues in $\mathbf{D}$, sorted in descending order;
3. The projections onto the principal subspace for $\mathbf{X}$ are $\mathbf{Z} = \mathbf{V}_K\mathbf{D}_K$, where $\mathbf{V}_K$ contains the first K columns of $\mathbf{V}$, and $\mathbf{D}_K$ the top left $K \times K$ sub matrix of $\mathbf{D}$.

To make projection for a new data point $\mathbf{x}_*$, we can calculate the projection as

$$\mathbf{z}_* = \mathbf{U}^\top\mathbf{x}_* = \mathbf{D}^{-1}\mathbf{V}^\top(\mathbf{V}\mathbf{D}\mathbf{U}^\top)\mathbf{x}_* = \mathbf{D}^{-1}\mathbf{V}^\top\mathbf{X}\mathbf{x}_* = \mathbf{D}^{-1}\mathbf{V}^\top\mathbf{k}(\mathbf{X}, \mathbf{x}_*),$$

where the vector $\mathbf{k}(\mathbf{X}, \mathbf{x}_*) := [\langle \mathbf{x}_1, \mathbf{x}_* \rangle, \ldots, \langle \mathbf{x}_N, \mathbf{x}_* \rangle]^\top$ also depends only on the inner-product. Then we truncate $\mathbf{z}_*$ at length K to obtain the projection onto the K-dimensional principal subspace.

The key observation from this dual formulation is that the whole PCA algorithm can be derived using *only* the inner-product in the feature space $\mathcal{X}$. If we have a mapping $\boldsymbol{\phi}(\cdot)$ which maps $\mathbf{x} \in \mathcal{X}$ into $\boldsymbol{\phi}(\mathbf{x}) \in \mathcal{F}$, and define an inner-product $\langle \cdot, \cdot \rangle_\mathcal{F}$ in the new space $\mathcal{F}$, then PCA can be performed with Gram matrix $\mathbf{K}$ defined as $\mathbf{K}_{ij} = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle_\mathcal{F}$. Furthermore, the reproducing kernel Hilbert space (RKHS) theory tells us that we can directly define this inner-product using a *kernel function* $\kappa(\cdot, \cdot)$, i.e., $\langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle_\mathcal{F} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, if $\kappa(\cdot, \cdot)$ satisfies the Mercer conditions [62]. Therefore, for non-linear PCA we just need to choose a kernel function (e.g., the popular RBF kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\alpha\|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$), calculate the *kernel matrix* $\mathbf{K}$ as $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, and perform the dual form PCA. Since $\mathbf{K}$ is $N \times N$, we can in principal project the data up to N dimensions.

Since we are now working in the new space $\mathcal{F}$, we need to make all the data points *centered* in this space. We can also do this without knowing the explicit mapping $\boldsymbol{\phi}(\cdot)$, by modifying the kernel matrix $\mathbf{K}$ as

$$\hat{\mathbf{K}} = \mathbf{K} - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^\top\mathbf{K} - \frac{1}{N}\mathbf{K}\mathbf{1}_N\mathbf{1}_N^\top + \frac{1}{N^2}\mathbf{1}_N\mathbf{1}_N^\top\mathbf{K}\mathbf{1}_N\mathbf{1}_N^\top, \tag{4.3}$$

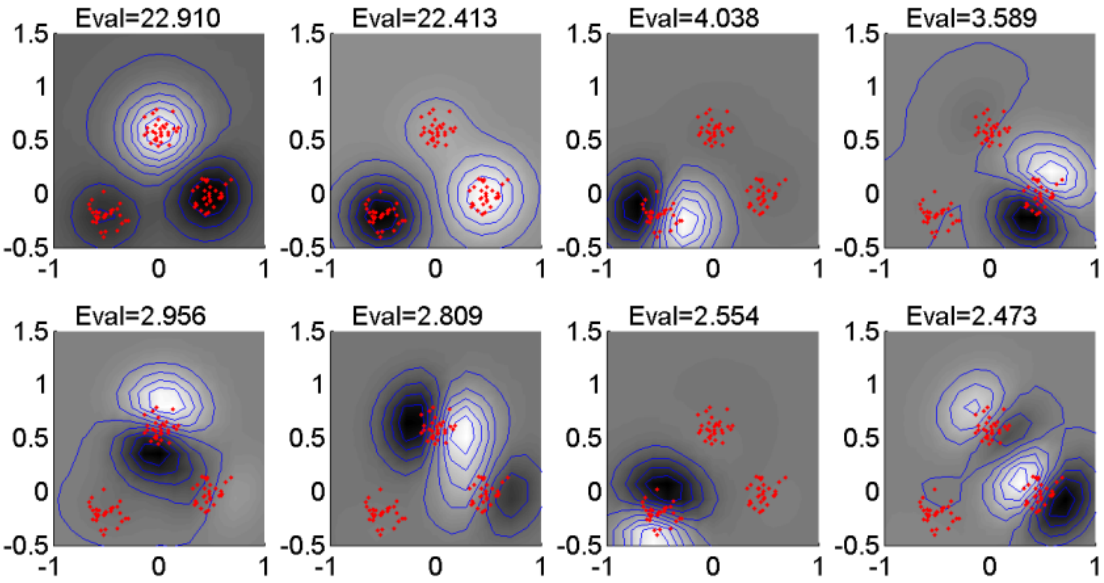


Figure 4.2: Illustration of kernel PCA on a 2D toy data. RBF kernel function is used with $\alpha = 10$. The first 8 principal components are shown here, with eigenvalues shown on top of each figure. Blue lines are contours, with white and dark color indicating high and low values, respectively. This toy example for kernel PCA is provided by Bernhard Schölkopf with the MATLAB code available at `http://www.kernel-machines.org/code/kpca_toy.m`.

where $\mathbf{1}_N$ denotes the all one column vector $[1, \ldots, 1]^\top$ of length N. For the kernel vector $\mathbf{k}(\mathbf{X}, \mathbf{x}_*)$ given test data $\mathbf{x}_*$, it can also be centered by

$$\hat{\mathbf{k}} = \mathbf{k} - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^\top\mathbf{k} - \frac{1}{N}\mathbf{K}\mathbf{1}_N + \frac{1}{N^2}\mathbf{1}_N\mathbf{1}_N^\top\mathbf{K}\mathbf{1}_N. \tag{4.4}$$

Figure 4.2 illustrates the first eight kernel PCA components for a 2D toy data, generated from three symmetric Gaussian distributions with standard deviation 0.1 and means $(-0.5, -0.2)$, $(0, 0.6)$ and $(0.5, 0)$. Since the projection is non-linear, we show the contours of each principal component on the 2D surface, with white and dark regions indicating high and low values, respectively. The RBF kernel function with $\alpha = 10$ is used here. It can be seen that the data points in different clusters can be detected using the first two components, and more detailed structures are shown in the other components. The data itself have only two dimensions, but in kernel PCA we can obtain up to N projection dimensions.

Remark 4.1.1. *The non-linear PCA reduces to linear one if we choose the linear kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top\mathbf{x}_j$, i.e., the normal inner-product in Euclidean space. In the case of $N > M$, we can still calculate the eigen-decomposition of $\mathbf{K}$, but the $N - M$ smallest eigenvalues will be zero, leaving the effective projection dimensions to be M. In the case*

of $M > N$, this dual form with linear kernel function leads to a more efficient solution for linear PCA because we only need to solve a $N \times N$ eigenvalue problem.

4.1.3 Local Projection Methods

PCA is known to be a *global* projection method because each principal axis maximizes the global variance of the data. There also exist *local* projection methods, in which the reconstruction is done locally instead of globally. For instance, the locally linear embedding (LLE) [60] looks for a projection that preserves the local geometry in the neighborhood of each data point. Isomap [67] generalizes the multi-dimensional scaling (MDS) [50] to consider an approximation of the geodesic distance for projection construction. Laplacian eigenmap [7] considers the Laplacian operator on the affinity graph to build the low-dimensional projection. All of these embedding methods take into account the local manifold structure of the data, instead of the global structure with Euclidean distance.

4.2 Projection Methods for Discrete Data

Discrete data can be viewed as a special case of continuous data, and all the projection methods can be applied here. But the intrinsic structure of discrete data makes *projection* a more interesting problem. Without loss of generality, we consider non-negative discrete data here, where each feature value is a positive integer or zero.

Let us take the document-word matrix in text information retrieval as an example, where its (i, j)-th entry denotes the number of occurrences of word j in document i. Compared to the general projection (e.g., PCA) of the word features into a subspace, it is more natural to constrain that each dimension of the subspace is a *positive* combination of the word features. This means that each dimension of the subspace, which we call a *topic*, should only have positive associations with (a subset of) all words.

This extra constraint can be added to the optimization view of PCA to obtain the following problem:

$$\min_{\mathbf{W},\mathbf{Z}} \quad \ell(\mathbf{X}, \mathbf{Z}\mathbf{W}^\top) \tag{4.5}$$

$$\text{subject to:} \quad \mathbf{W} \in \mathbb{R}_+^{M\times K}, \mathbf{Z} \in \mathbb{R}_+^{N\times K}, \tag{4.6}$$

where $\ell(\cdot, \cdot)$ denotes a *divergence function*, and the only constraint we have is that both the projection matrix $\mathbf{Z}$ and the mapping matrix $\mathbf{W}$ have only non-negative entries. We do not need orthogonality for $\mathbf{W}$, because it is not possible for an non-negative matrix to be orthogonal. The divergence function has two input matrices with the same size, is non-negative, and equals zero if and only if the two matrices are identical. A family of such divergence functions is called the *Bregman divergence* [11], which include popular

choices like the matrix Frobenius norm, matrix 2-norm, and the following *I-Divergence*:

$$\ell_{\mathrm{I}}(\mathbf{A}, \mathbf{B}) = \sum_{ij} \left(\mathbf{A}_{ij} \log \frac{\mathbf{A}_{ij}}{\mathbf{B}_{ij}} - \mathbf{A}_{ij} + \mathbf{B}_{ij} \right).$$

I-Divergence is a generalization of the Kullback-Leibler divergence for probabilistic distributions, and is not a distance function since it is not symmetric.

Problem (4.5) is in general called the *non-negative matrix factorization* (NMF) and is studied in [51]. An EM-like iterative algorithm exists to solve this optimization problem for some divergence functions like Frobenius norm and I-Divergence, and unlike the PCA solution, NMF solutions are only local minima. For I-Divergence, the update equations can be derived as

$$\hat{\mathbf{Z}}_{ik} \propto \mathbf{Z}_{ik} \sum_{j=1}^{M} \frac{\mathbf{X}_{ij}}{(\mathbf{Z}\mathbf{W}^\top)_{ij}} \mathbf{W}_{jk}, \tag{4.7}$$

$$\hat{\mathbf{W}}_{jk} = \mathbf{W}_{jk} \sum_{i=1}^{N} \frac{\mathbf{X}_{ij}}{(\mathbf{Z}\mathbf{W}^\top)_{ij}} \mathbf{Z}_{ik}, \tag{4.8}$$

where to remove the ambiguity we normalize in (4.7) such that $\sum_{i=1}^{N} \mathbf{Z}_{ik} = 1$, for all $k = 1, \ldots, K$. A random initialization of $\mathbf{Z}$ and $\mathbf{W}$ will lead to a reasonable solution.

When we choose $K < M$, we can view the columns of $\mathbf{W}$ as the component axes or the new *bases*, and each row of $\mathbf{Z}$ denotes the projection or the *weights* on these bases for each data point. This view of projection is different from PCA and is more like a clustering model for features, where each data point takes a weighted combination on these feature clusters. Put in the context of document-word data, each column of $\mathbf{W}$ defines a projection basis which maps all the words onto a topic; in this projected space, each row of $\mathbf{Z}$ gives the coordinate for each document.

4.2.1 Probabilistic Latent Semantic Indexing (pLSI)

Independently of [51], Hofmann [39] proposed the probabilistic latent semantic indexing (pLSI) which is applied for document modeling. In pLSI, a generative model for the documents and their word occurrences is introduced as follows. For a document-word pair (d, w),

1. A latent class variable z is sampled from $P(z)$;
2. A document is sampled from distribution $P(d|z)$;
3. A word is sampled from distribution $P(w|z)$.

Therefore, the joint probability of the pair (d, w) can be calculated as

$$P(d, w) = \sum_{z} P(z) P(d, w|z) = \sum_{z} P(z) P(d|z) P(w|z), \tag{4.9}$$

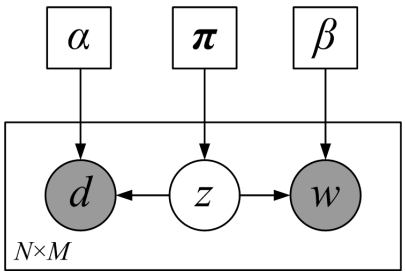


Figure 4.3: Plate model for pLSI model

which means the document d and the word w are independent given the latent semantic class label z. The graphical model of pLSI is shown in Figure 4.3.

In pLSI, the latent variable z identifies the joint probability of a given document-word pair. Let us assume the number of documents N and words M are both fixed, thus z have a Multinomial distribution with dimension $N \times M$. When we limit this latent variable to be K-dimensional distributed, the latent variable actually identifies a clustering process in the joint space of documents and words. In [39] Hofmann interprets z as the *topics* for words, which correspond to projection space for words.

EM Learning Algorithm for pLSI

Treating z as the latent variable, we can easily learn parameters for pLSI model with an EM algorithm. The log-likelihood for maximization can be written as

$$\mathcal{L}_{\text{pLSI}} = \sum_d \sum_w n(d,w) \log P(d,w) = \sum_d \sum_w n(d,w) \log \sum_z P(z)P(d|z)P(w|z),$$

where $n(d,w)$ denotes the number of occurrences of w in d.

In the E-step, the latent class membership for the given document-word pair is calculated using Bayes' rule with the current estimated parameters:

$$P(z|d,w) = \frac{P(z)P(d|z)P(w|z)}{\sum_{z'} P(z')P(d|z')P(w|z')}. \tag{4.10}$$

In the M-step, all the parameters are updated using the estimated class membership for each document-word pair. That is,

$$P(z) = \frac{\sum_{d,w} n(d,w)P(z|d,w)}{\sum_{d,w} n(d,w)}, \tag{4.11}$$

$$P(d|z) = \frac{\sum_{w} n(d,w)P(z|d,w)}{\sum_{d',w} n(d',w)P(z|d',w)}, \tag{4.12}$$

$$P(w|z) = \frac{\sum_{d} n(d,w)P(z|d,w)}{\sum_{d,w'} n(d,w')P(z|d,w')}. \tag{4.13}$$

Finally we repeat the EM steps until convergence. In [39] Hofmann shows better performance of pLSI over PCA for document modeling.

Equivalence of NMF and pLSI

While both NMF and pLSI are applied for positive observations, the relationships between them are not discovered. In this subsection we show that they are equivalent when I-Divergence is used for NMF, and that the iterative updating algorithm for NMF (4.7)∼(4.8) is equivalent to the EM learning for pLSI.

To simulate PLSA in NMF, we make a preprocessing to the data matrix $\mathbf{X}$ and divide each entry by the element-wise sum of $\mathbf{X}$, i.e., $\mathbf{X}_{ij} \leftarrow \mathbf{X}_{ij} / \sum_{i'j'} \mathbf{X}_{i'j'}$, which corresponds to the assumption in PLSA that pairs (d, w) are independently sampled. Then if we add a more constraint to the NMF problem,

$$\sum_{ij} (\mathbf{Z}\mathbf{W}^\top)_{ij} = 1, \tag{4.14}$$

minimizing I-Divergence is now identical to maximizing the *likelihood*

$$\ell_{\mathrm{I}}(\mathbf{X}, \mathbf{Z}\mathbf{W}^\top) = \sum_{ij} \mathbf{X}_{ij} \log(\mathbf{Z}\mathbf{W}^\top)_{ij}.$$

The probabilistic interpretation to this NMF is to define

$$P(d_i, w_j) = \mathbf{X}_{ij}, \quad P(w_j|z_k) = \mathbf{W}_{jk}, \quad P(d_i, z_k) = \mathbf{Z}_{ik},$$

then the NMF is now

$$P(d_i, w_j) = \sum_k P(d_i, z_k) P(w_j|z_k) = \sum_k P(d_i|z_k) P(w_j|z_k) P(z_k),$$

identical to PLSA model. To borrow the optimization problem from PLSA, we rewrite the NMF problem as:

$$\begin{aligned}
&\text{factorize} && \mathbf{X}_{N\times M} = \mathbf{Z}_{N\times K} \cdot \underbrace{\mathtt{diag}(z_1, \ldots, z_k)}_{\mathbf{D}_{K\times K}} \cdot \mathbf{W}^\top_{M\times K} \\
&\text{maximize} && \sum_{ij} \mathbf{X}_{ij} \log(\mathbf{Z}\mathbf{D}\mathbf{W}^\top)_{ij} \\
&\text{subjectto} && \sum_i \mathbf{Z}_{ik} = 1, \sum_j \mathbf{W}_{jk} = 1, \sum_k z_k = 1.
\end{aligned}$$

where the constraints here are identical to constraints (4.14). Standard EM algorithm in

PLSA corresponds to the following iterative algorithm for NMF:

$$\hat{\mathbf{Z}}_{ik} \propto \mathbf{Z}_{ik} \sum_j \frac{\mathbf{X}_{ij}}{(\mathbf{ZDW}^\top)_{ij}} \mathbf{D}_{kk} \mathbf{W}_{jk}, \tag{4.15}$$

$$\hat{\mathbf{W}}_{jk} \propto \mathbf{W}_{jk} \sum_i \frac{\mathbf{X}_{ij}}{(\mathbf{ZDW}^\top)_{ij}} \mathbf{D}_{kk} \mathbf{Z}_{ik}, \tag{4.16}$$

$$\hat{\mathbf{D}}_{kk} = \mathbf{D}_{kk} \sum_{ij} \frac{\mathbf{X}_{ij}}{(\mathbf{ZDW}^\top)_{ij}} \mathbf{Z}_{ik} \mathbf{W}_{jk}, \tag{4.17}$$

where in (4.15) and (4.16) we normalized such that $\sum_{i=1}^N \mathbf{Z}_{ik} = 1$ and $\sum_{j=1}^M \mathbf{W}_{jk} = 1$, respectively, for all $k = 1, \ldots, K$. Note that here we directly substitute the E-step into the M-step updates. Compared to the standard algorithm (4.7)$\sim$(4.8) for NMF, these updates are the same except an extra diagonal matrix $\mathbf{D}$. One advantage of this modified algorithm is that the weight of each topic can be directly read from the matrix $\mathbf{D}$.

From above we can see that PLSA is actually calculating a non-negative decomposition of the word count matrix $\mathbf{X}$ after we make the element-wise normalization. This is not a limitation of the algorithm since it is only a factor of constant and does not change the factorization structure.

4.2.2 Latent Dirichlet Allocation (LDA)

It is argued that pLSI is not a *well-defined* model, since it treats each document as an index and thus is not generalizable to new documents. Another problem of pLSI is that longer documents get higher weights in the model, which also indicates that the documents are not independently sampled. To solve the problem, Blei et al. [10] introduced the latent Dirichlet allocation (LDA) model which shows better performance than pLSI.

The plate model for LDA is shown in Figure 4.4. Given the Dirichlet prior α, document i is sampled independently through a *topic mixture* $\boldsymbol{\theta}_i$ which defines the mixing weights for this document. Then all the N_i words within this document are sampled independently by first choosing a topic z given the topic mixture $\boldsymbol{\theta}_i$, and then sampling the word based on the projection matrix β. Since not document indices but the contents are modeled directly, the model is generalizable to new documents.

Since there is a coupling between latent variables $\boldsymbol{\theta}_i$ and $z_{i,n}$, learning and inference are not tractable for LDA model. Blei et al. [10] adopts a variational EM learning algorithm which updates the posterior of $\boldsymbol{\theta}_i$ and $z_{i,n}$ iteratively through a variational distribution.

Remark 4.2.1. *The LDA-type models are "correct" for document modeling since it has the correct independence assumptions compared to pLSI model. This is called the* discrete PCA *in [12] because it can be viewed as using Multinomial distribution instead of Gaussian in a PCA-type representation, as clarified in the beginning of this section.*

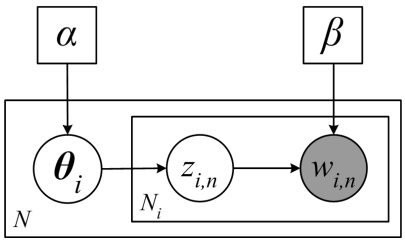


Figure 4.4: Plate model for LDA model

4.3 Organization of the Following Chapters

In Chapter 5 we review a probabilistic explanation of PCA which is known as PPCA, and then extend it to kernel PCA which can handle non-linear projections. An EM learning algorithm is derived for this model which is faster and has potential to apply to large data sets. An incremental kernel PCA can also be straightforwardly derived.

Then in Chapter 6 we go beyond unsupervised projection and introduce the MORP algorithm for *supervised projection*. Here we assume there are not only features associated with each data point, but also some output labels. MORP is motivated from a latent variable model and can handle both linear projection and non-linear projection. Experiments show that the model outperforms other supervised projection methods in various data sets.

Finally in Chapter 7 we consider a probabilistic version of MORP and introduce the SPPCA and S^{2}PPCA models for supervised and semi-supervised projections, respectively. An efficient EM algorithm can be derived which can handle large data sets. The semi-supervised effect of S^{2}PPCA model makes it applicable to many applications like face recognition and text classification.

Projection models for discrete data will be considered in Part III where we jointly perform clustering and projection for discrete data.

Chapter 5

Probabilistic Kernel Principal Component Analysis

Various projection methods exist for continuous data, among which the principal component analysis (PCA) is a linear projection method and becomes very popular in the last several decades. To relax the linearity of PCA, kernel PCA is introduced to generalize linear PCA to non-linear mappings via non-linear kernel functions. Both of the two methods turn out to solve an eigenvalue problem and have clear mathematical formulations, but the following questions occur for some real-world problems:

- How to apply PCA if the input data have missing entries?
- How to apply these methods if the dimensionality M or the number of data points N is too large?
- Is it possible to perform PCA locally instead of globally?

For deterministic methods like PCA and kernel PCA, it is difficult to deal with these questions directly. A probabilistic model, on the other hand, can handle them easily: the missing entries can be integrated out for learning; the EM algorithm can be used to solve the problem iteratively and possibly incrementally; localized PCA can be done via a mixture model.

At the end of the last century, several authors proposed a probabilistic version of PCA, which in this thesis we call the *probabilistic PCA* (PPCA) [70, 59]. They show that their models achieve the canonical linear PCA in the asymptotic case, and that PPCA provides these many benefits that canonical PCA does not have. However, this probabilistic interpretation is only for linear PCA, and there is no similar model which gives the same benefits for kernel PCA. This is exactly the goal of this chapter. In Section 5.1 we first review the PPCA framework and discuss the EM algorithm for learning with PPCA model. This algorithm is extended in Section 5.2 to non-linear cases and is proved to solve a kernel PCA problem. Some discussions are given thereafter, along with the benefits of

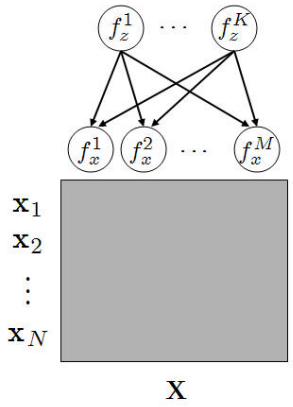


Figure 5.1: Illustration of the PPCA model. $\mathbf{X}$ denotes the input matrix, where each row is one data point. $f_x^1, \ldots, f_x^M$ are the M input features. On the top $f_z^1, \ldots, f_z^K$ are the K latent variables. They are all in circles because they are variables in the probabilistic models. The arrows denote probability dependency.

the proposed framework. To show one of these benefits, we discuss an incremental version of the algorithm in Section 5.3 and illustrate its usage on some toy data.

5.1 Probabilistic Principal Component Analysis (PPCA)

While PCA originates from the analysis of data variances, the PPCA model emerges from the statistics community and acts as a probabilistic explanation for PCA [70, 59]. PPCA is a latent variable model and defines a generative process for each data point $\mathbf{x}$ as (see Figure 5.1 for an illustration)

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon},$$

where $\mathbf{z} \in \mathbb{R}^K$ are called the *latent variables*, and $\mathbf{W}$ is a $M \times K$ matrix called *factor loadings*. In this probabilistic model, latent variables $\mathbf{z}$ are conventionally assumed as a Gaussian distribution with zero mean and unit variance, i.e., $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\boldsymbol{\epsilon}$ defines a noise process which also takes an isotropic Gaussian form as $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, with σ^2 the *noise level*. Additionally, we have parameters $\boldsymbol{\mu} \in \mathbb{R}^M$ which allow non-zero means for the data.

The PPCA model indicates that given the latent variable $\mathbf{z}$, $\mathbf{x}$ is Gaussian distributed:

$$\mathbf{x}|\mathbf{z} \sim \mathcal{N}(\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I}).$$

With $\mathbf{z}$ integrated out, it turns out that observation $\mathbf{x}$ is also Gaussian distributed:

$$\mathbf{x} \sim \mathcal{N}\left(\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I}\right). \tag{5.1}$$

Based on the Bayes' rule, the *a posteriori* distribution of $\mathbf{z}$ given observation $\mathbf{x}$ is also a Gaussian:

$$\mathbf{z}|\mathbf{x} \sim \mathcal{N}\left((\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I})^{-1}\mathbf{W}^\top(\mathbf{x} - \boldsymbol{\mu}),\ \sigma^2(\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I})^{-1}\right). \tag{5.2}$$

Remark 5.1.1. *The generative model for PPCA is similar to the factor analysis [5]. The only difference is the noise process: In factor analysis the noise levels for different dimensions can be different, leading to a noise process* $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ *with* $\boldsymbol{\Sigma} = \mathtt{diag}(\sigma_1^2, \ldots, \sigma_M^2)$. *Both models assume that in the noise process every two dimensions are independent. For a detailed comparison of these two models please refer to [70].*

It is shown that PPCA has strong connections to PCA. We summarize the related results in the following proposition without proof, since this is a corollary of Theorem 7.3.1 in the later Chapter 7. A detailed proof can also be found in the Appendix of [70].

Proposition 5.1.1. *Let* $\mathbf{S} = \frac{1}{N}\sum_{i=1}^N(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^\top$ *be the sample covariance matrix for data* $\{\mathbf{x}_i\}_{i=1}^N$, *and* $\lambda_1 \geq \ldots \geq \lambda_M$ *be its eigenvalues with eigenvectors* $\mathbf{u}_1, \ldots, \mathbf{u}_M$, *then if the latent space in PPCA model is* K*-dimensional,*

(i) *The maximum likelihood estimates of the mean* $\boldsymbol{\mu}$ *and the noise level* σ^2 *are respectively*

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^N \mathbf{x}_i, \qquad \sigma^2 = \frac{1}{M-K}\sum_{j=K+1}^M \lambda_j. \tag{5.3}$$

(ii) *The maximal likelihood estimate of* $\mathbf{W}$ *is given as*

$$\mathbf{W} = \mathbf{U}_K(\boldsymbol{\Lambda}_K - \sigma^2\mathbf{I})^{\frac{1}{2}}\mathbf{R}, \tag{5.4}$$

where $\boldsymbol{\Lambda}_K = \mathtt{diag}(\lambda_1, \ldots, \lambda_K)$, $\mathbf{U}_K = [\mathbf{u}_1, \ldots, \mathbf{u}_K]$, *and* $\mathbf{R}$ *is an arbitrary* $K \times K$ *orthogonal matrix.*

(iii) *The mean projections* $\mathbf{z}_*$ *for new input* $\mathbf{x}_*$ *is given as*

$$\mathbf{z}_* = \mathbf{R}^\top\left(\boldsymbol{\Lambda}_K - \sigma^2\mathbf{I}\right)^{\frac{1}{2}}\boldsymbol{\Lambda}_K^{-1}\mathbf{U}_K^\top(\mathbf{x}_* - \boldsymbol{\mu}).$$

It is seen that the mean vector $\boldsymbol{\mu}$ is simply the sample mean, and the noise level σ^2 is the average of the minor $M - K$ eigenvalues. The loading matrix $\mathbf{W}$ has an arbitrary factor $\mathbf{R}$ which is an orthogonal matrix. This indicates that the latent space is invariant under an arbitrary rotation. One can perform an SVD to $\mathbf{W}^\top\mathbf{W}$ to recover $\mathbf{R}$ if necessary.

5.1.1 Connection to PCA

When $\sigma^2 \to 0$, projection (5.2) is seen to become a singular at the mean

$$\left(\mathbf{W}^\top \mathbf{W}\right)^{-1} \mathbf{W}^\top (\mathbf{x} - \boldsymbol{\mu}),$$

which can be further simplified by plugging in the ML estimate of $\mathbf{W}$ to obtain

$$\mathbf{z} = \mathbf{R}^\top \boldsymbol{\Lambda}_K^{-\frac{1}{2}} \mathbf{U}_K^\top (\mathbf{x} - \boldsymbol{\mu}).$$

The result can also be directly obtained from Theorem 5.2.1(iii), and it is seen to be equivalent to PCA up to a scaling factor $\boldsymbol{\Lambda}_K^{-\frac{1}{2}}$ and a rotation factor $\mathbf{R}^\top$. This indicates that the principal subspace obtained in PCA is the same as that in PPCA, and the projections of data $\mathbf{x}$ onto the K-dimensional principal subspace in PCA are identical to the latent variables $\mathbf{z}$ up to a scaling and rotation factor.

5.1.2 EM Learning for PPCA model

Learning in probabilistic models reduces to maximizing the data (log) likelihood with respect to all the model parameters. In the case of PPCA model, the log likelihood of the whole observation $\mathcal{D} := \{\mathbf{x}_i\}_{i=1}^N$ is

$$\begin{aligned}\mathcal{L}(\mathcal{D}) &= \sum_{i=1}^N \log P(\mathbf{x}_i) = \sum_{i=1}^N \log \int P(\mathbf{x}_i|\mathbf{z}_i) P(\mathbf{z}_i)\, d\mathbf{z}_i \\ &= -\frac{N}{2}\left\{M \log(2\pi) + \log|\mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I}| + \mathtt{trace}\left((\mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I})^{-1}\mathbf{S}\right)\right\}. \qquad (5.5)\end{aligned}$$

In this section we describe an *expectation-maximization* (EM) learning procedure for PPCA model. EM iterates the two steps *expectation* (E-step) and *maximization* (M-step) until convergence, and it is guaranteed to find a local minima of the data likelihood. In the E-step, we fix the model parameters ($\mathbf{W}$, $\boldsymbol{\mu}$ and σ^2 for PPCA model) and calculate the expected distributions of latent variables (all the $\mathbf{z}_i$'s for PPCA model), and in the M-step we fix this distribution and maximize the *complete data likelihood* with respect to the model parameters. As will be discussed later, EM learning for PPCA models is important because it can deal with very large data sets, and it has, in particular when there are no missing entries, no local minima problem up to a rotation factor. For simplicity we only outline the update equations in the following and omit details (see [70] for all the derivations).

In the E-step, for each data point i, we estimate the distribution of $\mathbf{z}_i$ given observation $\mathbf{x}_i$. This is done using (5.2), and we calculate the sufficient statistics as

$$\langle \mathbf{z}_i \rangle = \left(\mathbf{W}^\top \mathbf{W} + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{W}^\top (\mathbf{x}_i - \boldsymbol{\mu}), \qquad (5.6)$$

$$\langle \mathbf{z}_i \mathbf{z}_i^\top \rangle = \sigma^2 \left(\mathbf{W}^\top \mathbf{W} + \sigma^2 \mathbf{I}\right)^{-1} + \langle \mathbf{z}_i \rangle \langle \mathbf{z}_i \rangle^\top, \qquad (5.7)$$

where $\langle\cdot\rangle$ denotes the expectation under the posterior distribution $P(\mathbf{z}_i|\mathbf{x}_i)$ given in (5.2).

In the M-step, we maximize the complete log-likelihood

$$\tilde{\mathcal{L}}(\mathcal{D}) = \sum_{i=1}^{N} \int P(\mathbf{z}_i|\mathbf{x}_i) \log \Big(P(\mathbf{x}_i|\mathbf{z}_i) P(\mathbf{z}_i) \Big) \, d\mathbf{z}_i$$

with respect to the model parameters, holding $P(\mathbf{z}_i|\mathbf{x}_i)$ fixed from the E-step. This can be done by setting the partial derivative with respect to each parameter to be zero. For the mean vector $\boldsymbol{\mu}$ we have

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i, \tag{5.8}$$

which is simply the sample mean of the data. For the other two parameters $\mathbf{W}$ and σ^2, we can obtain the following updates:

$$\hat{\mathbf{W}} = \left[\sum_{i=1}^{N} (\mathbf{x}_i - \boldsymbol{\mu}) \langle \mathbf{z}_i \rangle^\top \right] \left[\sum_{i=1}^{N} \langle \mathbf{z}_i \mathbf{z}_i^\top \rangle \right]^{-1}, \tag{5.9}$$

$$\hat{\sigma}^2 = \frac{1}{MN} \sum_{i=1}^{N} \left[\|\mathbf{x}_i - \boldsymbol{\mu}\|^2 + \texttt{trace} \left(\langle \mathbf{z}_i \mathbf{z}_i^\top \rangle \hat{\mathbf{W}}^\top \hat{\mathbf{W}} \right) - 2 \langle \mathbf{z}_i \rangle^\top \hat{\mathbf{W}}^\top (\mathbf{x}_i - \boldsymbol{\mu}) \right]. \tag{5.10}$$

Iterating these EM steps yields in convergence the ML estimates given in (5.4) and (5.3). One version of the proof for Theorem 5.2.1 is to look for the stationary point of the iterative algorithm. It is well-known that EM algorithm often finds a local minima, and different initializations lead to different solutions. But from Theorem 5.2.1 we know that the EM algorithm for PPCA will lead to the same solution up to a rotation factor, with different matrix $\mathbf{R}$ from different starting points. As mentioned before one can easily recover $\mathbf{R}$ from the obtained $\mathbf{W}$.

Computational Issues

The time complexity for the EM algorithm is $\mathcal{O}(mMNK)$, with m the number of EM iterations.[1] It is linear in the number of data points N and the input dimension M. The space complexity is $\mathcal{O}(MN)$, which is also linear both in N and M. The projection for a test data point is just a linear operation and costs $\mathcal{O}(MK)$ time.

5.1.3 Discussion

The equivalence of PPCA and PCA provides an important insight to PCA projections: PCA implicitly assumes that the data are distributed as a Gaussian in the original feature

[1] Note that we only need to calculate the diagonal entries for matrix trace when updating the noise level.

space. In another word, if the data are not Gaussian distributed, PCA is not guaranteed to provide a meaningful solution. Therefore, one should check the Gaussianity before applying PCA. Sometimes a pre-transformation or re-scaling of some feature dimensions are necessary to approximate Gaussianity.

As a probabilistic framework, PPCA also provides additional benefits over PCA. The EM learning procedure is one key benefit among others. There is no need to calculate and store the $M \times M$ sample covariance matrix $\mathbf{S}$, so the EM algorithm can be used for PCA projections of very high dimensional data (e.g., $M > 10000$). The EM algorithm also provides a principled way of handling missing entries in $\mathbf{X}$ if any, since the missing entries can be estimated in the E-step and are integrated out for the M-step. It is also possible to consider mixture of PCA models and achieve a local PCA modeling [69]. In the next section we will bring these advantages to kernel PCA as well.

5.2 Probabilistic Kernel PCA

PPCA can be viewed as a probabilistic model for linear PCA, but however up to now there is no probabilistic interpretation for kernel PCA. In this section we start from the PPCA model and extend it for kernel PCA. We call this model the *probabilistic kernel PCA* (PKPCA). The key point is to derive the dual form of the EM algorithm for PPCA.

5.2.1 Dual Form of EM Learning

To get a non-linear extension of the EM learning algorithm, we first rewrite the EM learner in its dual form. For this we need some notations. Since $\boldsymbol{\mu}$ is directly obtainable from (5.8), we can subtract this sample mean from each data point before we launch the EM algorithm. Therefore for simplicity, in the following we denote $\mathbf{x}$ the *centered* data vector $\mathbf{x} - \boldsymbol{\mu}$. Let $\mathbf{X}$ denote the $N \times M$ centered input matrix $[\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$, and $\mathbf{Z}$ the $N \times K$ latent variable matrix $[\langle \mathbf{z}_1 \rangle, \ldots, \langle \mathbf{z}_N \rangle]^\top$. We further define $\mathbf{C} := \sum_{i=1}^{N} \langle \mathbf{z}_i \mathbf{z}_i^\top \rangle$, then we can rewrite (5.9) in matrix form as

$$\hat{\mathbf{W}} = \mathbf{X}^\top \mathbf{Z} \mathbf{C}^{-1}. \tag{5.11}$$

From this we can calculate

$$\hat{\mathbf{W}}^\top \hat{\mathbf{W}} = \mathbf{C}^{-1} \mathbf{Z}^\top \mathbf{X} \mathbf{X}^\top \mathbf{Z} \mathbf{C}^{-1} = \mathbf{C}^{-1} \mathbf{Z}^\top \mathbf{K} \mathbf{Z} \mathbf{C}^{-1}, \tag{5.12}$$

$$\hat{\mathbf{W}}^\top \mathbf{x} = \mathbf{C}^{-1} \mathbf{Z}^\top \mathbf{X} \mathbf{x} = \mathbf{C}^{-1} \mathbf{Z}^\top \mathbf{k}(\mathbf{X}, \mathbf{x}), \tag{5.13}$$

which are the building blocks for the non-linear extension. The matrix $\mathbf{K} := \mathbf{X}\mathbf{X}^\top$ has each entry the inner-product of data points of corresponding row and column,

$$\mathbf{K}_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle = \mathbf{x}_i^\top \mathbf{x}_j,$$

and is once again the Gram matrix of input data. As defined for kernel PCA, $\mathbf{k}(\mathbf{X}, \mathbf{x}) = \mathbf{X}\mathbf{x}$ is a N-dimensional column vector with the i-th entry $\langle \mathbf{x}_i, \mathbf{x} \rangle$. Therefore these quantities are related to input $\mathbf{X}$ and $\mathbf{x}$ *only* via the inner-product $\langle \cdot, \cdot \rangle$.

Applying (5.12) and (5.13) into sufficient statistics (5.6), we get

$$\langle \mathbf{z}_i \rangle = \left(\mathbf{C}^{-1}\mathbf{Z}^\top \mathbf{K}\mathbf{Z}\mathbf{C}^{-1} + \sigma^2 \mathbf{I} \right)^{-1} \mathbf{C}^{-1}\mathbf{Z}^\top \mathbf{k}(\mathbf{X}, \mathbf{x}_i), \tag{5.14}$$

or equivalently,

$$\hat{\mathbf{Z}} = \mathbf{K}\mathbf{Z}\mathbf{C}^{-1} \left(\mathbf{C}^{-1}\mathbf{Z}^\top \mathbf{K}\mathbf{Z}\mathbf{C}^{-1} + \sigma^2 \mathbf{I} \right)^{-1}, \tag{5.15}$$

by collecting $\langle \mathbf{z}_i \rangle$ in columns and transposing it. Sufficient statistics (5.7) can be written in terms of $\mathbf{C}$:

$$\hat{\mathbf{C}} = \sum_{i=1}^{N} \langle \mathbf{z}_i \mathbf{z}_i^\top \rangle = N\sigma^2 \left(\mathbf{C}^{-1}\mathbf{Z}^\top \mathbf{K}\mathbf{Z}\mathbf{C}^{-1} + \sigma^2 \mathbf{I} \right)^{-1} + \hat{\mathbf{Z}}^\top \hat{\mathbf{Z}}. \tag{5.16}$$

Updates for variances σ^2 can be written in terms of the updated $\mathbf{Z}$ and $\mathbf{C}$ as

$$\hat{\sigma}^2 = \frac{1}{MN} \left[\mathtt{trace}\left(\mathbf{K}\right) - \mathtt{trace}\left(\hat{\mathbf{Z}}^\top \mathbf{K} \hat{\mathbf{Z}} \hat{\mathbf{C}}^{-1} \right) \right], \tag{5.17}$$

which can be easily verified from (5.10). Therefore, it can be seen that all the interested terms in the EM algorithm take input data into account *only* via the Gram matrix $\mathbf{K}$, except for the mapping matrix $\mathbf{W}$. This is not a problem, because the inference task (5.2) can be written in terms of Gram matrix $\mathbf{K}$ instead of mapping $\mathbf{W}$, if we apply equations (5.12) and (5.13).

With this dual form, we can pre-define some basis functions for inputs, or directly work in the reproducing kernel Hilbert space (RKHS) induced by a kernel function $\kappa(\cdot, \cdot)$, like what we did to extend linear PCA to kernel PCA. This implicitly defines a mapping $\boldsymbol{\phi} : \mathbf{x} \in \mathcal{X} \mapsto \boldsymbol{\phi}(\mathbf{x}) \in \mathcal{F}$, which maps $\mathbf{x}$ into a high-dimensional feature space $\mathcal{F}$. Then the kernel function is basically

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle_{\mathcal{F}} =: \mathbf{K}_{ij},$$

and we can directly work with kernel matrix $\mathbf{K}$ and kernel function $\kappa(\cdot, \cdot)$, without knowing $\boldsymbol{\phi}(\cdot)$ explicitly. This results in a non-linear mapping from input $\mathbf{x}$ to latent variable $\mathbf{z}$.

As in the case of kernel PCA, we need to center the data points in the new feature space $\mathcal{F}$. We can do this by modifying the kernel matrix $\mathbf{K}$ and vector $\mathbf{k}(\mathbf{X}, \mathbf{x})$ without knowing the explicit mapping $\boldsymbol{\phi}(\cdot)$, as in (4.3) and (4.4).

5.2.2 Analytical Solution of Dual Form

We can derive the analytical solution for PKPCA from the stationary point of the dual E-step (5.15), (5.16) and M-step (5.17). The main result is given in the following theorem.

Theorem 5.2.1. *Let* $\mathbf{K}$ *be the centered kernel matrix, and* $\lambda_1 \geq \ldots \geq \lambda_N$ *be its eigenvalues with eigenvectors* $\mathbf{v}_1, \ldots, \mathbf{v}_N$*. If the latent space is* K*-dimensional, in PKPCA the dual EM algorithm leads to the following solution:*

$$\mathbf{Z} = \sqrt{N}\mathbf{V}_K \left(\mathbf{I} - N\sigma^2 \mathbf{\Lambda}_K^{-1}\right)^{\frac{1}{2}} \mathbf{R}, \tag{5.18}$$

$$\sigma^2 = \frac{1}{N(M-K)} \sum_{j=K+1}^{N} \lambda_j, \tag{5.19}$$

where $\mathbf{\Lambda}_K = \mathtt{diag}(\lambda_1, \ldots, \lambda_K)$*,* $\mathbf{V}_K = [\mathbf{v}_1, \ldots, \mathbf{v}_K]$*, and* $\mathbf{R}$ *is an arbitrary* $K \times K$ *orthogonal matrix.*

Proof. We first find the stationary point of the dual learning algorithm, at which the updates (5.15) and (5.16) will not change the sufficient statistics $\mathbf{Z}$ and $\mathbf{C}$. Using the notation $\mathbf{B} := \mathbf{Z}\mathbf{C}^{-1}$, we rewrite these two updates as

$$\mathbf{Z} = \mathbf{K}\mathbf{B}\left(\mathbf{B}^\top \mathbf{K}\mathbf{B} + \sigma^2 \mathbf{I}\right)^{-1},$$
$$\mathbf{C} = N\sigma^2 \left(\mathbf{B}^\top \mathbf{K}\mathbf{B} + \sigma^2 \mathbf{I}\right)^{-1} + \left(\mathbf{B}^\top \mathbf{K}\mathbf{B} + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{B}^\top \mathbf{K}^2 \mathbf{B} \left(\mathbf{B}^\top \mathbf{K}\mathbf{B} + \sigma^2 \mathbf{I}\right)^{-1}.$$

Putting them into the definition $\mathbf{B} = \mathbf{Z}\mathbf{C}^{-1}$, we obtain the following equation for $\mathbf{B}$ after some mathematics:

$$N\mathbf{B}\left(\sigma^2 \mathbf{I} + \mathbf{B}^\top \mathbf{K}\mathbf{B}\right) = \mathbf{K}\mathbf{B}. \tag{5.20}$$

In the following we solve this equation for the $N \times K$ matrix $\mathbf{B}$. Denote the eigen-decomposition of $K \times K$ square matrix $\mathbf{B}^\top \mathbf{K}\mathbf{B}$ as $\mathbf{R}^\top \mathbf{L}\mathbf{R}$, where $\mathbf{L} = \mathtt{diag}(l_1, \ldots, l_K)$ is a diagonal matrix, and $\mathbf{R} = [\mathbf{r}_1, \ldots, \mathbf{r}_K]^\top$ a $K \times K$ orthonormal matrix. Then left multiplying $\mathbf{B}^\top$ and right multiplying $\mathbf{R}^\top$ on both sides of (5.20) yield

$$N\mathbf{B}^\top \mathbf{B}\mathbf{R}^\top \left(\sigma^2 \mathbf{I} + \mathbf{L}\right) = \mathbf{R}^\top \mathbf{L},$$

which simplifies to

$$\mathbf{B}^\top \mathbf{B}\mathbf{r}_k = \frac{l_k}{N(\sigma^2 + l_k)} \mathbf{r}_k, \quad k = 1, \ldots, K,$$

if we write in columns. This clearly indicates that the $\mathbf{r}_k$'s are also eigenvectors of $\mathbf{B}^\top \mathbf{B}$, with corresponding eigenvalues given in the diagonal matrix $\frac{1}{N}\mathbf{L}\left(\sigma^2 \mathbf{I} + \mathbf{L}\right)^{-1}$. This means we can write the eigen-decomposition of $\mathbf{B}^\top \mathbf{B}$ as

$$\mathbf{B}^\top \mathbf{B} = \frac{1}{N} \mathbf{R}^\top \mathbf{L}\left(\sigma^2 \mathbf{I} + \mathbf{L}\right)^{-1} \mathbf{R}. \tag{5.21}$$

Recall that by definition,

$$\mathbf{B}^\top \mathbf{K}\mathbf{B} = \mathbf{R}^\top \mathbf{L}\mathbf{R}, \tag{5.22}$$

and use the new notation $\mathbf{V} := \sqrt{N}\mathbf{B}\mathbf{R}^\top\mathbf{L}^{-\frac{1}{2}}\left(\sigma^2\mathbf{I}+\mathbf{L}\right)^{\frac{1}{2}}$, from (5.21) and (5.22) we have:

$$\begin{aligned}\mathbf{V}^\top\mathbf{V} &= \mathbf{I},\\ \mathbf{V}^\top\mathbf{K}\mathbf{V} &= N\left(\sigma^2\mathbf{I}+\mathbf{L}\right),\end{aligned}$$

by simple matrix multiplication. It is clear now that the $N \times K$ matrix $\mathbf{V}$ is column orthonormal and contains the eigenvectors of $\mathbf{K}$ in columns. The corresponding eigenvalues are contained in the diagonal matrix $\mathbf{\Lambda} := N\left(\sigma^2\mathbf{I}+\mathbf{L}\right)$. Then solving for $\mathbf{B}$ from $\mathbf{V}$ and $\mathbf{\Lambda}$ yields

$$\mathbf{B} = \frac{1}{\sqrt{N}}\mathbf{V}\left(\mathbf{I}-N\sigma^2\mathbf{\Lambda}^{-1}\right)^{\frac{1}{2}}\mathbf{R}.$$

As a consequence of this result, the stationary point for the latent variable matrix $\mathbf{Z}$ can be obtained as

$$\begin{aligned}\mathbf{Z} = \mathbf{K}\mathbf{B}\left(\mathbf{B}^\top\mathbf{K}\mathbf{B}+\sigma^2\mathbf{I}\right)^{-1} &= \sqrt{N}\mathbf{K}\mathbf{V}\left(\mathbf{I}-N\sigma^2\mathbf{\Lambda}^{-1}\right)^{\frac{1}{2}}\mathbf{\Lambda}^{-1}\mathbf{R}\\ &= \sqrt{N}\mathbf{V}\left(\mathbf{I}-N\sigma^2\mathbf{\Lambda}^{-1}\right)^{\frac{1}{2}}\mathbf{R},\end{aligned}$$

where the second equality holds because $\mathbf{K}\mathbf{V} = \mathbf{V}\mathbf{\Lambda}$. To calculate the noise level, first note that left multiplying $\mathbf{B}^\top\mathbf{K}$ on both sides of (5.20) yields

$$\mathbf{B}^\top\mathbf{K}^2\mathbf{B} = N\mathbf{B}^\top\mathbf{K}\mathbf{B}(\sigma^2\mathbf{I}+\mathbf{B}^\top\mathbf{K}\mathbf{B}) = N\mathbf{R}^\top\mathbf{L}(\sigma^2\mathbf{I}+\mathbf{L})\mathbf{R},$$

where the second equality is obtained with definition (5.22). Then a direct calculation (using the definition of $\mathbf{V}$) leads to

$$\begin{aligned}\mathbf{V}^\top\mathbf{K}^2\mathbf{V} &= \left(\sigma^2\mathbf{I}+\mathbf{L}\right)^{\frac{1}{2}}\mathbf{L}^{-\frac{1}{2}}\sqrt{N}\mathbf{R}\left[\mathbf{B}^\top\mathbf{K}^2\mathbf{B}\right]\mathbf{R}^\top\sqrt{N}\mathbf{L}^{-\frac{1}{2}}\left(\sigma^2\mathbf{I}+\mathbf{L}\right)^{\frac{1}{2}}\\ &= N\left(\sigma^2\mathbf{I}+\mathbf{L}\right)^{\frac{1}{2}}\mathbf{L}^{-\frac{1}{2}}\mathbf{R}\left[N\mathbf{R}^\top\mathbf{L}(\sigma^2\mathbf{I}+\mathbf{L})\mathbf{R}\right]\mathbf{R}^\top\mathbf{L}^{-\frac{1}{2}}\left(\sigma^2\mathbf{I}+\mathbf{L}\right)^{\frac{1}{2}}\\ &= N^2(\sigma^2\mathbf{I}+\mathbf{L})^2 = \mathbf{\Lambda}^2.\end{aligned}$$

Plugging this into (5.17), we obtain

$$\begin{aligned}\sigma^2 &= \frac{1}{MN}\left[\texttt{trace}\left(\mathbf{K}\right)-\texttt{trace}\left(\mathbf{Z}^\top\mathbf{K}\mathbf{B}\right)\right]\\ &= \frac{1}{MN}\left[\texttt{trace}\left(\mathbf{K}\right)-\texttt{trace}\left(\mathbf{V}^\top\mathbf{K}^2\mathbf{V}\left(\mathbf{\Lambda}^{-1}-N\sigma^2\mathbf{\Lambda}^{-2}\right)\right)\right]\\ &= \frac{1}{MN}\left[\texttt{trace}\left(\mathbf{K}\right)-\texttt{trace}\left(\mathbf{\Lambda}-N\sigma^2\right)\right] = \frac{1}{MN}\left[\texttt{trace}\left(\mathbf{K}\right)-\texttt{trace}\left(\mathbf{\Lambda}\right)+NK\sigma^2\right].\end{aligned}$$

This leads to the stationary point for the noise level as

$$\sigma^2 = \frac{1}{N(M-K)}\sum_{j=K+1}^{N}\lambda_j, \tag{5.23}$$

where λ_j are now the remaining eigenvalues of the kernel matrix $\mathbf{K}$.

The last step is to prove that the K eigenvectors denoted in the columns of $\mathbf{V}$ are the leading eigenvectors. For this we need to directly calculate the log-likelihood (5.5) in dual form. First notice that

$$\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I} = \mathbf{B}^\top\mathbf{K}\mathbf{B} + \sigma^2\mathbf{I} = \mathbf{R}^\top\mathbf{L}\mathbf{R} + \sigma^2\mathbf{I} = \mathbf{R}^\top(\mathbf{L} + \sigma^2\mathbf{I})\mathbf{R} = \frac{1}{N}\mathbf{R}^\top\mathbf{\Lambda}\mathbf{R}.$$

Then for the second term in (5.5) we have

$$\log|\mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I}| = \log\left(\sigma^{2(M-K)}|\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I}|\right) = (M-K)\log\sigma^2 + \sum_{j=1}^{K}\log\lambda_k - K\log N.$$

The last trace term can be calculated using $\mathbf{S} = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$ as

$$\begin{aligned}
&\texttt{trace}\left((\mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I})^{-1}\mathbf{S}\right) = \frac{1}{N\sigma^2}\texttt{trace}\left((\mathbf{I} - \mathbf{W}(\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I})^{-1}\mathbf{W}^\top)\mathbf{X}^\top\mathbf{X}\right) \\
&= \frac{1}{N\sigma^2}\left\{\texttt{trace}\left(\mathbf{X}\mathbf{X}^\top\right) - \texttt{trace}\left(\mathbf{X}\mathbf{W}(\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I})^{-1}\mathbf{W}^\top\mathbf{X}^\top\right)\right\} \\
&= \frac{1}{N\sigma^2}\left\{\texttt{trace}\left(\mathbf{K}\right) - \texttt{trace}\left(\mathbf{K}\mathbf{B}(\mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I})^{-1}\mathbf{B}^\top\mathbf{K}\right)\right\} \\
&= \frac{1}{N\sigma^2}\left\{\texttt{trace}\left(\mathbf{K}\right) - \texttt{trace}\left(\mathbf{\Lambda} - N\sigma^2\mathbf{I}\right)\right\} = \frac{1}{N\sigma^2}\sum_{j=K+1}^{N}\lambda_j + K.
\end{aligned}$$

Therefore the data log-likelihood is now

$$\mathcal{L}(\mathcal{D}) = -\frac{N}{2}\left\{M\log(2\pi) + \sum_{j=1}^{K}\log\lambda_j + (M-K)\log\sigma^2 - K\log N + \frac{1}{N\sigma^2}\sum_{j=K+1}^{N}\lambda_j + K\right\}.$$

Plug in (5.23) for σ^2, maximization of the data log-likelihood is equivalent to the following minimization problem

$$\min_{\lambda_{K+1},\ldots,\lambda_N}\left\{\log\sum_{j=K+1}^{N}\lambda_j - \frac{1}{M-K}\sum_{j=K+1}^{N}\log\lambda_j\right\}.$$

This is the equivalent problem as discussed in Appendix A.2 and A.3 of [70]. It is shown that this quantity is minimized only if the "discarded" eigenvalues $\lambda_{K+1},\ldots,\lambda_N$ are the smallest eigenvalues of $\mathbf{K}$. This means $\mathbf{\Lambda}$ and $\mathbf{V}$ contain the leading eigenvalues and eigenvectors, i.e., $\mathbf{\Lambda} = \mathbf{\Lambda}_K$ and $\mathbf{V} = \mathbf{V}_K$. Other stationary points are saddle points on the likelihood surface. This completes the proof. □

Note that for notation simplicity, in Theorem 5.2.1 we also use $\mathbf{\Lambda}$ and λ_j's to denote the eigenvalues, the same as in Proposition 5.1.1. But their meanings are slightly different. Let $\mathbf{X} = \mathbf{V}\mathbf{D}\mathbf{U}^\top$ be the eigen-decomposition of centered data matrix $\mathbf{X}$, then the sample covariance matrix $\mathbf{S} = \frac{1}{N}\mathbf{X}^\top\mathbf{X} = \mathbf{U}(\frac{1}{N}\mathbf{D}^2)\mathbf{U}^\top$, and kernel matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top = \mathbf{V}\mathbf{D}^2\mathbf{V}^\top$.

Therefore, the two diagonal matrix of eigenvalues have a constant factor (N) difference. This also explains why the two equations for σ^2, (5.19) and (5.3), has a constant difference factor of N.

In Theorem 5.2.1, M is the dimensionality of the input data for PPCA. In the dual form, M should denote the dimensionality of the new feature space $\mathcal{F}$ if a first feature mapping $\boldsymbol{\phi}(\cdot)$ is available. In some reproducing kernel Hilbert spaces this number could be very high, or even infinite (e.g., for RBF kernel). Since the sum of eigenvalues of $\mathbf{K}$ is finite, the ML estimate for σ^2 therefore tends to zero, and this corresponds to kernel PCA as will be seen in the next subsection. For the implementation of the dual form EM algorithm, however, forcing $\sigma^2 = 0$ may lead to numerical instability of matrix inverse in (5.15), so we recommend to treat σ^2 as a tunable parameter instead of a parameter to optimize. From the proof of Theorem 5.2.1 it is seen that any such σ^2 will lead to the stationary point (5.18).

5.2.3 Connection to Kernel PCA

Plugging the matrix $\mathbf{B}$ back into the mean of projection (5.14), for a test data $\mathbf{x}_*$ we can calculate the mean

$$\begin{aligned}\langle \mathbf{z}_* \rangle &= \left(\mathbf{B}^\top \mathbf{K}\mathbf{B} + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{B}^\top \mathbf{k}(\mathbf{X}, \mathbf{x}_*) \\ &= \mathbf{R}^\top \sqrt{N}\boldsymbol{\Lambda}^{-1} \left(\mathbf{I} - N\sigma^2 \boldsymbol{\Lambda}^{-1}\right)^{\frac{1}{2}} \mathbf{V}^\top \mathbf{k}(\mathbf{X}, \mathbf{x}_*).\end{aligned}$$

When $\sigma^2 \to 0$, the projection is seen to be a singular at

$$\mathbf{z}_* = \mathbf{R}^\top \sqrt{N}\boldsymbol{\Lambda}^{-1}\mathbf{V}^\top \mathbf{k}(\mathbf{X}, \mathbf{x}_*),$$

which is equivalent to the mapping given by kernel PCA up to a rotation factor $\mathbf{R}^\top$. The j-th entry in the projected vector is seen to be

$$\mathbf{z}^j(\cdot) = \frac{\sqrt{N}}{\lambda_j} \sum_{i=1}^{N} (\mathbf{R}^\top \mathbf{v}_j)_i \kappa(\mathbf{x}_i, \cdot),$$

which is known as the j-th *eigen-function* of kernel PCA.

5.2.4 EM Algorithm for Kernel PCA

The final EM learning algorithm for kernel PCA is shown in Algorithm 5.1. Since $\sigma^2 > 0$ and $\mathbf{K}$ is positive semi-definite, the matrix inversion in the algorithm is always well-defined and numerically stable. The stopping criteria is the data log-likelihood

$$\hat{\mathcal{L}}(\mathcal{D}) = -\frac{N}{2}\left\{\log|\mathbf{B}^\top \mathbf{K}\mathbf{B} + \sigma^2\mathbf{I}| - \frac{1}{N\sigma^2}\texttt{trace}\left(\mathbf{K}\mathbf{B}(\mathbf{B}^\top \mathbf{K}\mathbf{B} + \sigma^2\mathbf{I})^{-1}\mathbf{B}^\top \mathbf{K}\right)\right\}, \quad (5.24)$$

Algorithm 5.1 EM Learning Algorithm for Kernel PCA

Require: N input data points $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$, with $\mathbf{x}_i \in \mathbb{R}^M$.
Require: A kernel function κ and a noise parameter $\sigma^2 > 0$.
1: Calculate the kernel matrix $\mathbf{K}$ using κ, and make it centralized using (4.3).
2: Initialize $\mathbf{Z} \in \mathbb{R}^{N \times K}$ randomly (e.g., sample N data points from a K-dimensional Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$), and set $\mathbf{C} = \mathbf{Z}^\top \mathbf{Z}$.
3: **repeat**
4: Calculate $\mathbf{B} = \mathbf{Z}\mathbf{C}^{-1}$;
5: Update $\mathbf{Z}$ via (5.15): $\mathbf{Z} = \mathbf{K}\mathbf{B}\left(\mathbf{B}^\top \mathbf{K}\mathbf{B} + \sigma^2 \mathbf{I}\right)^{-1}$;
6: Update $\mathbf{C}$ via (5.16): $\mathbf{C} = N\sigma^2 \left(\mathbf{B}^\top \mathbf{K}\mathbf{B} + \sigma^2 \mathbf{I}\right)^{-1} + \mathbf{Z}^\top \mathbf{Z}$;
7: **until** the improvement of log-likelihood (5.24) is smaller than a threshold.
Output: Up to a rotation factor, projections for input data $\mathbf{X}$ are given in $\mathbf{Z}$.
Output: Up to a rotation factor, projection for a new data point $\mathbf{x}_*$ is calculated as $\mathbf{z}_* = \left(\mathbf{B}^\top \mathbf{K}\mathbf{B} + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{B}^\top \mathbf{k}(\mathbf{X}, \mathbf{x}_*)$, with $\mathbf{k}(\mathbf{X}, \mathbf{x}_*)$ centralized via (4.4).

with constant terms ignored.

If the exact kernel PCA projection is desired, one can perform a singular value decomposition to the obtained projection matrix $\mathbf{Z}$ and recover the rotation matrix $\mathbf{R}$. Let $\mathbf{Z} = \mathbf{U}_Z \mathbf{D}_Z \mathbf{V}_Z^\top$ be the SVD of $\mathbf{Z}$, then $\hat{\mathbf{Z}} := \mathbf{Z}\mathbf{V}_Z$ gives the exact kernel PCA projection. One can also recover the leading eigenvalues of $\mathbf{K}$ as $\mathbf{\Lambda}_K = N^2\sigma^2(N\mathbf{I} - \mathbf{D}_Z^2)^{-1}$.

In Figure 5.2 we show the results of the EM learning algorithm on the same toy data as given in Figure 4.2. The algorithm is initialized randomly as seen in the first column of second to fourth rows. As the EM algorithm is running, the first three eigenvectors approach the true ones quickly, and at 20 iterations they are almost the final results. The algorithm finally converges within 100 steps. It is seen that the learned top eigenvectors match the true ones very well (except a constant -1 factor for the first eigenvector).

Computational Issues

The time complexity of the algorithm is $\mathcal{O}(mN^2K)$ plus $\mathcal{O}(N^2M)$ which is the one-time calculation of the kernel matrix. Here m is the number of EM steps. The space complexity is $\mathcal{O}\left(N^2\right)$. Both of them are now quadratic in the number of data points N. The time for projecting a test data point is now $\mathcal{O}(NM)$. The algorithm in general handles non-linear mappings for input data, and is also applicable to linear PCA if the dimensionality of data M is larger than the number of data N. In this case the algorithm is more efficient than the EM algorithm for PPCA [58] in which the time complexity is $\mathcal{O}(mNMK)$.

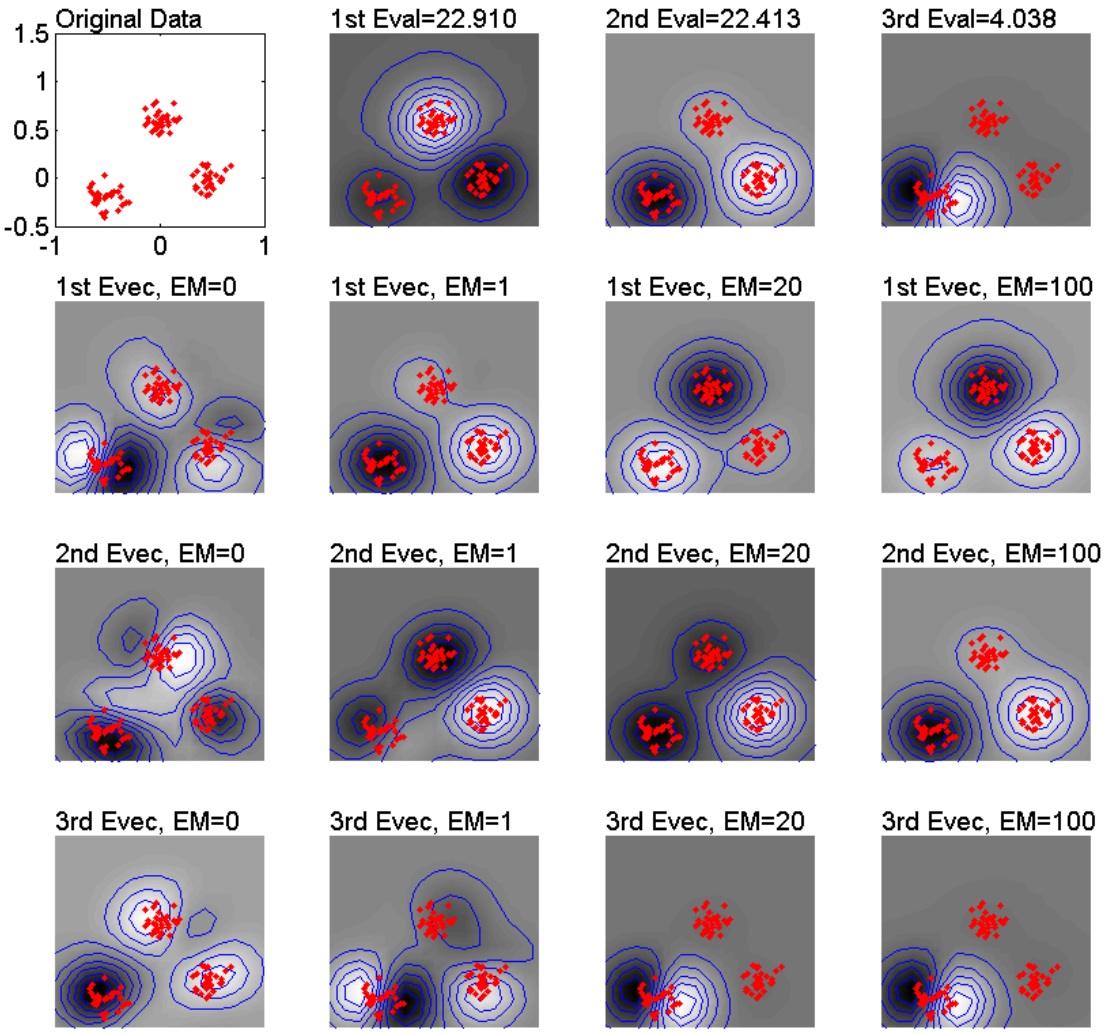


Figure 5.2: Illustration of EM algorithm for kernel PCA on the same 2D toy data as in Figure 4.2 with the same parameters. The noise level σ^2 are set to 0.01. In the first row we show the original data and the top three eigenvectors of kernel PCA with direct calculation. In the second to fourth rows we show the top three eigenvectors along the EM iterations until convergence.

5.2.5 Discussion

The proposed EM algorithm for kernel PCA is important for non-linear projections. On one hand, it can be viewed as an application of *kernel trick* to the EM algorithm; on the other hand, due to the Gaussian assumption of PPCA we are in fact modeling the data in the RKHS as a Gaussian distribution. While this assumption is problematic in some cases, its equivalence to kernel PCA indicates that the obtained projections are reasonable.

Other benefits of the EM algorithm for PKPCA are summarized in the following.

- Instead of solving an eigenvalue problem for a big $N \times N$ matrix, in the EM algorithm we only calculate matrix multiplication and matrix inverse for small matrices. This means we can handle bigger problem for kernel PCA.
- One can control the algorithm by early-stopping without full convergence, which indicates we can solve the problem faster.
- With this iterative learning algorithm, it is possible to consider each data point one by one and achieve an incremental learning for kernel PCA (see Section 5.3).
- With non-zero σ^2, the probabilistic interpretation allows us to consider more complicated models such as mixture of kernel PCA model.

It is also seen here that the iterative algorithm can be used to find the *principal subspace* of a given symmetric matrix. This is useful in many applications such as non-linear dimensionality reduction for classification, and can be applied to other kernel methods like kernel canonical correlation analysis [35].

5.2.6 Empirical Studies

We compare the conventional kernel PCA and the proposed EM algorithm on the Reuters-21578 text data and Corel image data. We denote them as Reuters and Corel in the following, respectively. The Reuters data we use contains 10195 documents with 36030 words, and we extract standard `TF-IDF` features for each document. It is difficult to perform PCA directly because we have too many words. The second data set is a subset of Corel image database and contains 1021 images. We extract and combine *color histogram* (216-dim.), *correlagram* (256-dim.), *first and second color moments* (9-dim.) and *Pyramid wavelet texture* (10-dim.) to form 491-dimensional input features to represent the images. All the features are then centralized and standardized with deviation 1.

For the kernel PCA solutions, we choose linear kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$ for Reuters data, and RBF kernel for the Corel image data with $\alpha = 0.002$. Due to the linear kernel function, the kernel PCA solution is equivalent to linear PCA for Reuters data. For the comparison we did two types of experiments. In the first experiment, we randomly choose a fixed number (3000 for Reuters and 1021 for Corel) of data points N, and project the data into various dimensions K using kernel PCA and the EM algorithm. For a second experiment, we fix the projection dimensionality K to be 50, and choose different N to make the projection. The comparison curve of time complexity is shown in Figure 5.3, with left column showing the first experiment and right column the second one. The results for Reuters data is shown in the top row, and those for Corel is in the bottom row. In both experiments, to solve the kernel PCA eigenvalue problem directly we use the MATLAB function `eigs` with default parameter settings. For the EM algorithm, σ^2 is fixed as 0.01, and the algorithm converges in average within 20 iterations. We repeat

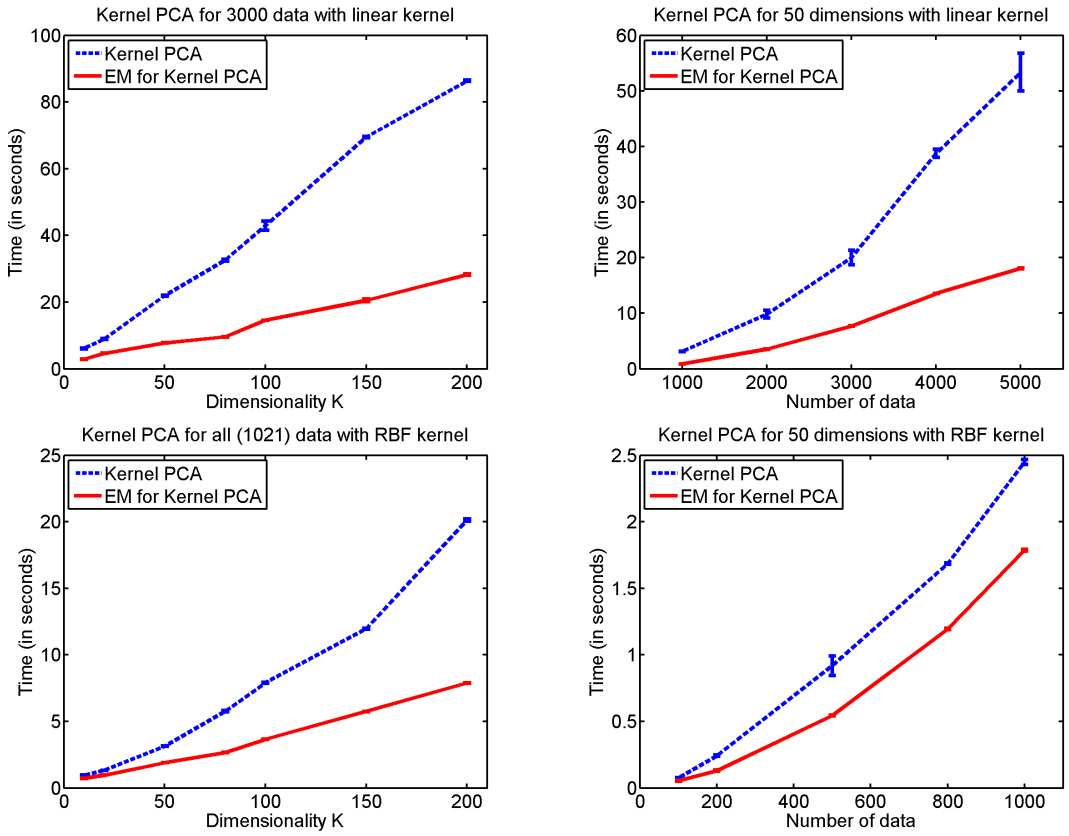


Figure 5.3: Time comparison of kernel PCA and the EM algorithm for kernel PCA for Reuters-21578 (top row) and Corel image data (bottom row). For the left figures we fix the number of data points N and vary the projection dimension K, while for the right figures we fix K and vary N. Linear kernel is used in top row, and RBF kernel is used in bottom row with parameter $\alpha = 0.002$. All the experiments are repeated 10 times independently.

the experiments 10 times independently, and the variances are shown in the figures with error bars.

It can be seen that the EM algorithm gets much less time than solving the eigenvalue problem directly. We also compared the classification accuracy based on the projections, and both versions achieve almost the same performance. This indicates that the EM solution of kernel PCA is a good alternative for real-world applications.

5.3 Incremental Kernel PCA

The eigenvalue solution for kernel PCA requires a full kernel matrix $\mathbf{K}$ be calculated *a priori*. When some additional data are available, we have to, however, re-calculate the new kernel matrix (with more dimensions) and solve a new eigenvalue problem of larger dimension. This is not necessary for the EM algorithm proposed in the previous section. In this section we consider an *incremental learning* for kernel PCA which naturally follows from the EM iterations.

Let

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{12}^\top & \mathbf{K}_{22} \end{bmatrix}, \quad \mathbf{Z} = \begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix}$$

be the block representations of the kernel matrix $\mathbf{K}$ and the latent variable matrix $\mathbf{Z}$, where $\mathbf{K}_{11} \in \mathbb{R}^{N_1 \times N_1}$, $\mathbf{K}_{12} \in \mathbb{R}^{N_1 \times N_2}$, $\mathbf{K}_{22} \in \mathbb{R}^{N_2 \times N_2}$, and $\mathbf{Z}_1 \in \mathbb{R}^{N_1 \times K}$, $\mathbf{Z}_2 \in \mathbb{R}^{N_2 \times K}$. Note that by definition we have $N_1 + N_2 = N$. Then we can write the matrix multiplication in block-wise form as

$$\mathbf{Z}^\top \mathbf{K} \mathbf{Z} = \begin{bmatrix} \mathbf{Z}_1^\top & \mathbf{Z}_2^\top \end{bmatrix} \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{12}^\top & \mathbf{K}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix} = \mathbf{Z}_1^\top \mathbf{K}_{11} \mathbf{Z}_1 + \left(\mathbf{Z}_1^\top \mathbf{K}_{12} \mathbf{Z}_2 + \mathbf{Z}_2^\top \mathbf{K}_{12}^\top \mathbf{Z}_1 + \mathbf{Z}_2^\top \mathbf{K}_{22} \mathbf{Z}_2 \right), \tag{5.25}$$

$$\mathbf{Z}^\top \mathbf{Z} = \begin{bmatrix} \mathbf{Z}_1^\top & \mathbf{Z}_2^\top \end{bmatrix} \begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix} = \mathbf{Z}_1^\top \mathbf{Z}_1 + \mathbf{Z}_2^\top \mathbf{Z}_2^\top, \tag{5.26}$$

both of which equal to a first term for N_1 data points plus some informative terms about the rest N_2 data points. The term $\mathbf{KZ}$ can be calculated in a similar way:

$$\mathbf{K} \mathbf{Z} = \begin{bmatrix} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \mathbf{K}_{12}^\top & \mathbf{K}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{11} \mathbf{Z}_1 + \mathbf{K}_{12} \mathbf{Z}_2 \\ \mathbf{K}_{12}^\top \mathbf{Z}_1 + \mathbf{K}_{22} \mathbf{Z}_2 \end{bmatrix}. \tag{5.27}$$

It is seen from Algorithm 5.1 that these are the terms related to $\mathbf{Z}$ in the whole algorithm. Suppose we observed N_1 data points at first and run Algorithm 5.1 until convergence. Then when the extra N_2 data points are available, we just need to randomly initialize projections $\mathbf{Z}_2$ and use the converged $\mathbf{Z}_1$ as the starting point for previous N_1 points. This normally leads to faster convergence of the whole algorithm. But before the EM iterations we need to center the whole kernel matrix $\mathbf{K}$, which can also be done via *incremental kernel centering*.

5.3.1 Incremental Kernel Centering

For the first N_1 data points, we center the kernel using (4.3):

$$\hat{\mathbf{K}}_1 = \mathbf{K}_{11} - \mathbf{1}_{N_1} \texttt{rowmean}(\mathbf{K}_{11}) - \texttt{rowmean}(\mathbf{K}_{11})^\top \mathbf{1}_{N_1}^\top + \texttt{elemean}(\mathbf{K}_{11}) \mathbf{1}_{N_1} \mathbf{1}_{N_1}^\top,$$

in which $\texttt{rowmean}(\mathbf{K}_{11}) := \frac{1}{N_1} \mathbf{1}_{N_1}^\top \mathbf{K}_{11}$ is *row-wise mean vector* of $\mathbf{K}_{11}$, and $\texttt{elemean}(\mathbf{K}_{11}) := \frac{1}{N_1^2} \mathbf{1}_{N_1}^\top \mathbf{K}_{11} \mathbf{1}_{N_1} = \frac{1}{N_1^2} \sum_{i,j=1}^{N_1} \kappa(\mathbf{x}_i, \mathbf{x}_j)$ is *element-wise mean value* of $\mathbf{K}_{11}$.

Algorithm 5.2 EM Learning Algorithm for Incremental Kernel PCA

Require: Kernel PCA results $\mathbf{Z}_1 \in \mathbb{R}^{N_1 \times K}$, $\mathbf{C} \in \mathbb{R}^{K \times K}$, and kernel $\mathbf{K}_{11} \in \mathbb{R}^{N_1 \times N_1}$ for N_1 input data points.

Require: N_2 new data points, kernel function κ and a noise parameter $\sigma^2 > 0$.

1: Calculate new kernel matrices $\mathbf{K}_{12} \in \mathbb{R}^{N_1 \times N_2}$ and $\mathbf{K}_{22} \in \mathbb{R}^{N_2 \times N_2}$ using κ, and center all the kernels using (5.28)∼(5.30).

2: Initialize $\mathbf{Z}_2 \in \mathbb{R}^{N_2 \times K}$ randomly, and update $\mathbf{C} \leftarrow \mathbf{C} + \mathbf{Z}_2^\top \mathbf{Z}_2$.

3: **repeat**

4: Calculate $\mathbf{Z}^\top \mathbf{K} \mathbf{Z}$, $\mathbf{Z}^\top \mathbf{Z}$ and $\mathbf{K}\mathbf{Z}$ using (5.25), (5.26) and (5.27), respectively;

5: Update $\mathbf{Z} = [\mathbf{Z}_1^\top, \mathbf{Z}_2^\top]^\top$ as: $\mathbf{Z} \leftarrow \mathbf{K}\mathbf{Z} \left(\mathbf{Z}^\top \mathbf{K} \mathbf{Z} + \sigma^2 \mathbf{C}^2\right)^{-1} \mathbf{C}$;

6: Update $\mathbf{C}$ as: $\mathbf{C} \leftarrow N\sigma^2 \mathbf{C} \left(\mathbf{Z}^\top \mathbf{K} \mathbf{Z} + \sigma^2 \mathbf{C}^2\right)^{-1} \mathbf{C} + \mathbf{Z}^\top \mathbf{Z}$;

7: **until** the improvement of log-likelihood (5.24) is smaller than a threshold.

Output: Up to a rotation factor, projections are given in $\mathbf{Z}_1$ and $\mathbf{Z}_2$ for the two data sets.

Output: Up to a rotation factor, projection for a new data point $\mathbf{x}_*$ is calculated as $\mathbf{z}_* = \mathbf{C} \left(\mathbf{Z}^\top \mathbf{K} \mathbf{Z} + \sigma^2 \mathbf{C}^2\right)^{-1} \mathbf{Z}^\top \mathbf{k}(\mathbf{X}, \mathbf{x}_*)$, with $\mathbf{k}(\mathbf{X}, \mathbf{x}_*)$ centralized via (4.4).

When the extra N_2 data points are available, kernel centering should be performed to the whole kernel matrix $\mathbf{K}$. This can be written in block-wise form as

$$\hat{\mathbf{K}}_{11} = \mathbf{K}_{11} - \mathbf{1}_{N_1} \mathtt{rowmean}_1(\mathbf{K}) - \mathtt{rowmean}_1(\mathbf{K})^\top \mathbf{1}_{N_1}^\top + \mathtt{elemean}(\mathbf{K}) \mathbf{1}_{N_1} \mathbf{1}_{N_1}^\top, \tag{5.28}$$

$$\hat{\mathbf{K}}_{12} = \mathbf{K}_{12} - \mathbf{1}_{N_1} \mathtt{rowmean}_2(\mathbf{K}) - \mathtt{rowmean}_1(\mathbf{K})^\top \mathbf{1}_{N_2}^\top + \mathtt{elemean}(\mathbf{K}) \mathbf{1}_{N_1} \mathbf{1}_{N_2}^\top, \tag{5.29}$$

$$\hat{\mathbf{K}}_{22} = \mathbf{K}_{22} - \mathbf{1}_{N_2} \mathtt{rowmean}_2(\mathbf{K}) - \mathtt{rowmean}_2(\mathbf{K})^\top \mathbf{1}_{N_2}^\top + \mathtt{elemean}(\mathbf{K}) \mathbf{1}_{N_2} \mathbf{1}_{N_2}^\top, \tag{5.30}$$

where $\mathtt{rowmean}_1(\mathbf{K})$ and $\mathtt{rowmean}_2(\mathbf{K})$ contain the first N_1 and last N_2 entries of the length-N row vector $\mathtt{rowmean}(\mathbf{K})$, respectively. These equations indicate that for the "old" kernel $\mathbf{K}_{11}$ we need to re-centralize it using (5.28), and the two "new" kernel matrices are centered using the global mean vector and mean value. Note that

$$\mathtt{rowmean}_1(\mathbf{K}) = \frac{1}{N} \left(N_1 \, \mathtt{rowmean}(\mathbf{K}_1) + N_2 \, \mathtt{rowmean}(\mathbf{K}_{12}^\top) \right)$$

and $\mathtt{elemean}(\mathbf{K})$ is equal to the mean value of vector $\mathtt{rowmean}(\mathbf{K})$. Therefore for the new kernel centering, we can reuse the known vector $\mathtt{rowmean}(\mathbf{K}_{11})$ and do not need to know the old kernel matrix $\mathbf{K}_{11}$.

5.3.2 Online Kernel PCA

The ideal solution for online kernel PCA is to process each data point one by one and neglect the "used" kernel matrices. That is, we only store the $N \times K$ projection matrix $\mathbf{Z}$ for the N data points which have been processed, and neglect the $N \times N$ kernel matrix $\mathbf{K}$. This will save a lot of space, but is however applicable only if the derived projection matrix $\mathbf{Z}$ is fixed for future incremental learning. A trade-off is to additionally store the

$N \times K$ matrix $\mathbf{KZ}$ and fix it in the EM iterations at each incremental step, but this will lead to a local minima and may, due to our experience, converge to some minor principal components.

5.3.3 EM Algorithm for Incremental Kernel PCA

We summarize the incremental kernel PCA algorithm in Algorithm 5.2. In this algorithm both $\mathbf{Z}_1$ and $\mathbf{Z}_2$ are updated, and the storage requirement is still $\mathcal{O}(N^2)$ since we need to store the whole kernel matrix $\mathbf{K}$. If $\mathbf{Z}_1$ is fixed in this procedure, or as a trade-off $\mathbf{K}_{11}\mathbf{Z}_1$ is fixed in (5.27), we do not need $\mathbf{K}_{11}$ any more and the inputs should be $\mathbf{Z}_1$, $\mathbf{K}_{11}\mathbf{Z}_1$, $\mathbf{C}$ and $\texttt{rowmean}(\mathbf{K}_{11})$. In this case the storage requirement is only $\mathcal{O}(NK) + \mathcal{O}(K^2)$.

Figure 5.4 shows an example of incremental kernel PCA on the same 2D toy data. We use two out of the three clusters to do kernel PCA first (first row), and then each time add 10 data points from the third cluster. Incremental kernel PCA is used to learn the top three eigenvectors, and it is seen that it is gradually adapted to the third cluster and finally achieves the same results as kernel PCA (fourth row).

5.4 Summary

In this chapter we proposed a probabilistic interpretation for kernel PCA. This probabilistic version generalizes the idea of probabilistic PCA to non-linear PCA and provides an iterative algorithm for kernel PCA. It also provides other new perspectives to kernel PCA such as an incremental kernel PCA.

PPCA provides a probabilistic interpretation of linear PCA, and the essential assumption is that the input data is distributed as a Gaussian. The success of PCA on many data sets indicates that many data can be assumed Gaussian distributed. In this chapter we generalize this idea to the feature space, and due to the success of kernel PCA this modeling assumption is also reasonable. We illustrate an incremental kernel PCA in this chapter, and other extensions to PPCA is also possible for probabilistic kernel PCA such as a mixture of probabilistic kernel PCA. For some future work, we can consider a novelty detection algorithm in the RKHS space by calculating the likelihood of each data point in this probabilistic model, and optimize the kernel parameters (e.g., α in RBF kernel) and dimensionality K by maximization of (penalized) data likelihood.

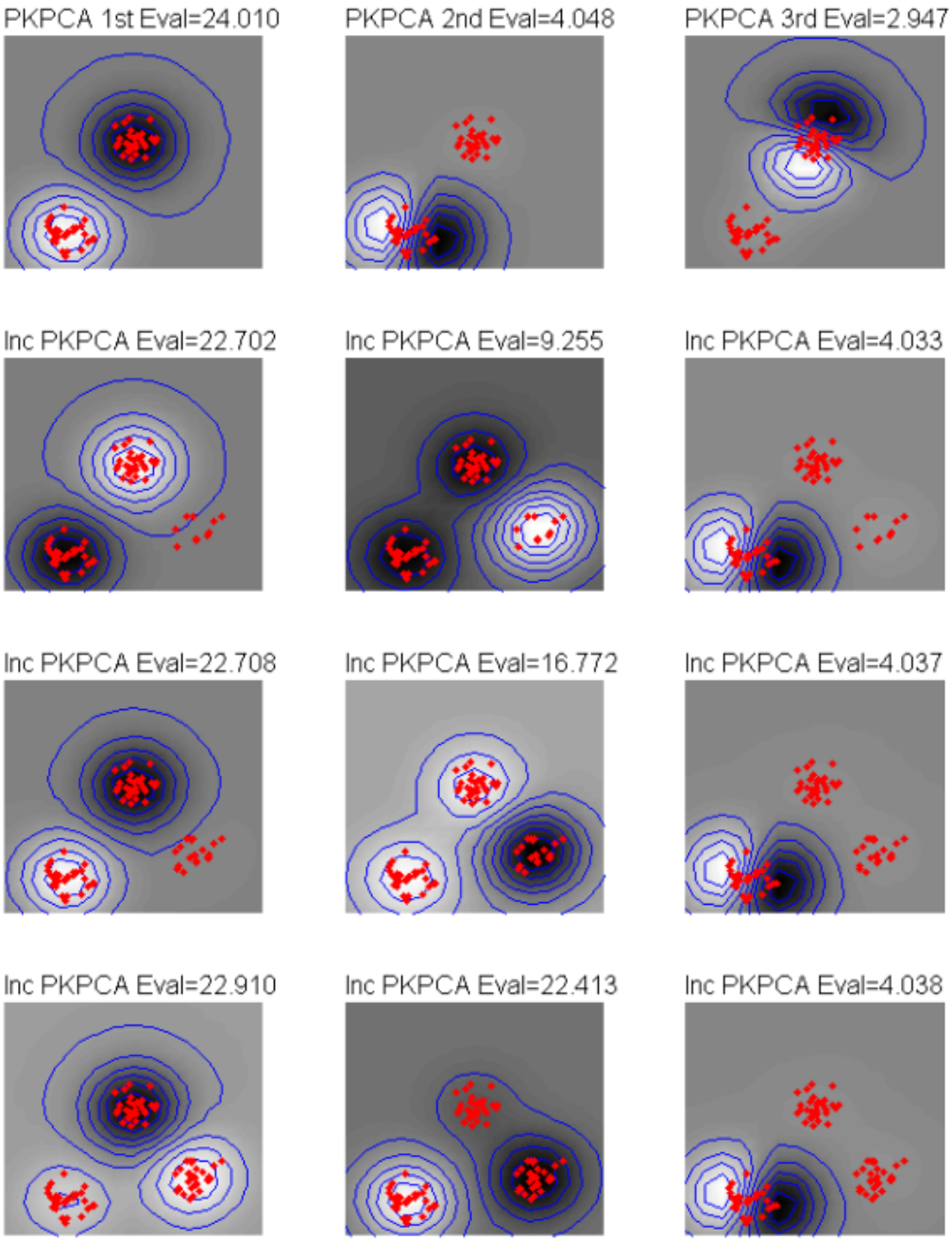


Figure 5.4: Illustration of EM algorithm for incremental kernel PCA on the same 2D toy data. The first row shows the top three eigenvectors of (probabilistic) kernel PCA on two of the three clusters. The second to the fourth rows show the top three eigenvectors of incremental kernel PCA where each time we add 10 data points from the third cluster. The fourth row uses all the 30 data points from the third cluster and achieves the same results as kernel PCA (see Figure 4.2 or Figure 5.2). For each row the EM algorithm converges within 100 steps.

Chapter 6

Supervised Feature Projection

In the previous chapter we only considered *unsupervised projection*, where only input features $\mathbf{X}$ are considered. When there are some outputs available (e.g., regression values or classification results), it is often beneficial to consider *supervised projection*, which is based not only on the inputs, but also on the target values. In general this leads to an *informed* or *biased* feature projection, which will be more relevant to the particular supervised learning problem. In the case that we have only one output dimension, i.e., a regression or classification task, it is ideal to have the projection function informed by the *dependency* between inputs and outputs. More generally if we have multiple output dimensions, i.e., for an input $\mathbf{x}$ the corresponding output is no longer a scalar but a vector $\mathbf{y} = [y_1, \ldots, y_L]^\top$, the *intra-correlation* between different dimensions of output should also be taken into account.

The setting is very general in real-world applications. One working example in this chapter is to predict users' preferences on a set of paintings, which is a typical multi-output problem since for each painting many persons' preferences have to be estimated. One may treat each person separately, but a notable fact is that people's tastes are usually correlated. One technology, referred as *collaborative filtering* [6], explores people's "like-mindedness" to make predictions. Another example is the problem of multi-label text or image categorization, where each document/image is allowed to be associated with more than one category and where categories often have semantic correlations. For example, a document talking about category *car* must also belong to the category *vehicle*; an image in category *ski* is likely to be associated with *snow*.

In this chapter we introduce a supervised feature projection algorithm called the *Multi-Output Regularized Projection* (MORP).[1] We use the term "multi-output" in the name because we want to emphasize its capability of handling multiple outputs, but it is certainly applicable to the single-output case. The algorithm is motivated from a latent variable model which is analogous to PPCA, and in Section 6.1 we derive an analytical solution to the proposed optimization problem. Then we formally introduce the MORP algorithm

[1] This chapter is based on conference papers [82, 81] and the journal paper [85].

in Section 6.2, in which both the primal form (linear projection) and the dual form (non-linear projection) are discussed. In Section 6.3 we point out its connections to kernel PCA and some supervised projection methods, and in Section 6.4 we report some experimental results on preference prediction and multi-label classification. Both experiments show that MORP obtains very good supervised projections.

In this chapter we motivate the algorithm from a probabilistic perspective, but all the solutions are deterministic and exact. In the next chapter we will focus on an EM learning algorithm to this problem and solve *semi-supervised feature projection* as well. For some additional notations, for each data point i, $i = 1, \ldots, N$, we have an M-dimensional feature vector $\mathbf{x}_i \in \mathcal{X}$, and in general an L-dimensional output vector $\mathbf{y}_i \in \mathcal{Y}$. Similarly to the input data matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top \in \mathbb{R}^{N \times M}$, we group the outputs in the matrix $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]^\top \in \mathbb{R}^{N \times L}$. For projection we still aim to derive a mapping $\Psi : \mathcal{X} \mapsto \mathcal{Z}$ that projects the input features into a K-dimensional latent space $\mathcal{Z}$.

6.1 Supervised Latent Variable Model

The supervised latent variable model is motivated from the unsupervised latent variable model in PPCA, which we briefly review at first.

6.1.1 Unsupervised Latent Variable Model

In unsupervised linear projection, we aim at finding a linear mapping from the input space $\mathcal{X}$ to some low-dimensional latent space $\mathcal{Z}$, while most of the structure in the data can be explained and recovered. In this sense we can turn this linear projection problem to an optimization problem, where we are trying to minimize the *reconstruction error*:

$$\begin{aligned} \min_{\mathbf{A},\mathbf{Z}} \quad & \|\mathbf{X} - \mathbf{Z}\mathbf{A}\|^2 \\ \text{subject to:} \quad & \mathbf{Z}^\top \mathbf{Z} = \mathbf{I}, \end{aligned} \tag{6.1}$$

where $\mathbf{Z} \in \mathbb{R}^{N \times K}$ gives the K-dimensional *projections* of objects, and $\mathbf{A} \in \mathbb{R}^{K \times M}$ is the *loading matrix*. By constraining $\mathbf{Z}^\top \mathbf{Z} = \mathbf{I}$, we restrict the K latent variables to be linearly independent, i.e., to have diagonal covariance matrix in latent space.

Since matrix product $\mathbf{Z}\mathbf{A}$ has rank K, in Problem (6.1) we are indeed seeking a low-rank approximation to the data matrix $\mathbf{X}$. Please note that this problem is different from the optimization problem for PCA (see (4.1)), because we are now constraining the projection matrix $\mathbf{Z}$ to be orthonormal, not the loading matrix $\mathbf{A}$. It can be shown that Problem (6.1) is actually equivalent to the PPCA model.

The derived projection explains the covariance of input data, which is however not necessarily relevant to the output quantity. Hence unsupervised feature projections may or may not be beneficial to supervised learning problems. When output information is available, it is more desired to consider the *correlation* between input $\mathbf{X}$ and output $\mathbf{Y}$,

and the *intra-correlation* between the L dimensions of $\mathbf{Y}$ if $L > 1$. Therefore, we turn to *supervised projection* in the next subsection, incorporating both input $\mathbf{X}$ and output $\mathbf{Y}$.

6.1.2 Supervised Latent-Variable Model

The unsupervised projection Problem (6.1) explicitly represents the projections of input data $\mathbf{X}$ in matrix $\mathbf{Z}$. To consider the output information, we can enforce the projections $\mathbf{Z}$ in Problem (6.1) sensitive to $\mathbf{Y}$ as well. Thus in supervised projection we can solve the following optimization problem:

$$\begin{aligned} \min_{\mathbf{A},\mathbf{B},\mathbf{Z}} \quad & (1-\beta)\|\mathbf{X}-\mathbf{Z}\mathbf{A}\|^2 + \beta\|\mathbf{Y}-\mathbf{Z}\mathbf{B}\|^2 \\ \text{subject to:} \quad & \mathbf{Z}^\top\mathbf{Z} = \mathbf{I}, \end{aligned} \tag{6.2}$$

where $\mathbf{Z} \in \mathbb{R}^{N\times K}$ gives the K-dimensional projections of objects, for features of both $\mathbf{X}$ and $\mathbf{Y}$, and $\mathbf{A} \in \mathbb{R}^{K\times M}$, $\mathbf{B} \in \mathbb{R}^{K\times L}$ are the loading matrices. $0 \le \beta \le 1$ is a tuning parameter determining how much the projections should be biased by the outputs. As before, $\mathbf{Z}^\top\mathbf{Z} = \mathbf{I}$ restricts the K latent variables to be linearly independent. Clearly, the cost function is a trade-off between the *reconstruction error* of both $\mathbf{X}$ and $\mathbf{Y}$. We wish to find the optimal projections that give the minimum reconstruction error.

To see the optimization problem more clearly, we rewrite the cost in (6.2) as

$$(1-\beta)\|\mathbf{X}-\mathbf{Z}\mathbf{A}\|^2 + \beta\sum_{\ell=1}^{L}\sum_{i=1}^{N}\left((\mathbf{y}_i)_\ell - \sum_{k=1}^{K}\mathbf{Z}_{ik}\mathbf{B}_{k\ell}\right)^2,$$

where $(\mathbf{y}_i)_\ell$ denote the ℓ-th entry of vector $\mathbf{y}_i$, and $\mathbf{Z}_{ik}$ denote the (i,k)-th entry in matrix $\mathbf{Z}$. Then we have the following observations:

- When $L = 1$, the second part of the cost constrains that the output $\mathbf{Y}$ can be *linearly reconstructed* from the latent projection $\mathbf{Z}$. Therefore, in the whole optimization problem we are minimizing the *correlation* between $\mathbf{X}$ and $\mathbf{Y}$.
- When $L > 1$, all the columns of $\mathbf{Y}$ are constrained to be linearly reconstructable from $\mathbf{Z}$. Therefore they are not considered independently, but *jointly*. In another word, we are minimizing the *intra-correlation* between columns of $\mathbf{Y}$.

We would like to mention here that one can very easily generalize this cost function to have a different weight β_ℓ for the ℓ-th output dimension, while maintaining a proper normalization for all β_ℓ's. In this way one can constrain each output dimension differently, which leads to a more flexible cost with however more free parameters. In the following we stick to the simpler setting since all the learning algorithms can be easily generalized to cover this case.

Remark 6.1.1. *When $L = 1$, finding the mapping from $\mathbf{X}$ to $\mathbf{Y}$ is known as a regression problem if $\mathbf{y}_i \in \mathbb{R}$, or multi-class classification if $\mathbf{y}_i$ is chosen from a finite set of integers.*

In the general case when $L > 1$, the former is called multivariate regression, while the latter is called multi-label classification. Therefore our multi-output setting covers all these cases.[2]

The following proposition states the interdependency between $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{Z}$ at the optimum.

Proposition 6.1.1. *If $\mathbf{Z}, \mathbf{A}$ and $\mathbf{B}$ are the optimal solutions to the problem (6.2), and let $\mathbf{K} = (1-\beta)\mathbf{X}\mathbf{X}^\top + \beta\mathbf{Y}\mathbf{Y}^\top$, then:*

(i) $\mathbf{A} = \mathbf{Z}^\top\mathbf{X}$, $\mathbf{B} = \mathbf{Z}^\top\mathbf{Y}$;

(ii) *At the optimum, the objective function (6.2) equals to* $\texttt{trace}\left(\mathbf{K}\right) - \texttt{trace}\left(\mathbf{Z}^\top\mathbf{K}\mathbf{Z}\right)$.

Proof. Applying the rule $\|\mathbf{C}\|^2 = \texttt{trace}\left(\mathbf{C}\mathbf{C}^\top\right)$ for an arbitrary matrix $\mathbf{C}$, we obtain

$$\begin{aligned} J(\mathbf{A},\mathbf{B},\mathbf{Z}) :&= (1-\beta)\|\mathbf{X}-\mathbf{Z}\mathbf{A}\|^2 + \beta\|\mathbf{Y}-\mathbf{Z}\mathbf{B}\|^2 \\ &= (1-\beta)\texttt{trace}\left(\mathbf{X}\mathbf{X}^\top - 2\mathbf{Z}\mathbf{A}\mathbf{X}^\top + \mathbf{A}^\top\mathbf{Z}^\top\mathbf{Z}\mathbf{A}\right) \\ &\quad + \beta\texttt{trace}\left(\mathbf{Y}\mathbf{Y}^\top - 2\mathbf{Z}\mathbf{B}\mathbf{Y}^\top + \mathbf{B}^\top\mathbf{Z}^\top\mathbf{Z}\mathbf{B}\right). \end{aligned}$$

Let the derivative of J with respect to $\mathbf{A}$ and $\mathbf{B}$ be zero, we have

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{A}} &= 2(1-\beta)(\mathbf{Z}^\top\mathbf{X} - \mathbf{Z}^\top\mathbf{Z}\mathbf{A}) = 0 \Rightarrow \mathbf{A} = \mathbf{Z}^\top\mathbf{X} \\ \frac{\partial J}{\partial \mathbf{B}} &= 2\beta(\mathbf{Z}^\top\mathbf{Y} - \mathbf{Z}^\top\mathbf{Z}\mathbf{B}) = 0 \Rightarrow \mathbf{B} = \mathbf{Z}^\top\mathbf{Y} \end{aligned}$$

which proves (i). Then we use the results (i) to replace $\mathbf{A}$ and $\mathbf{B}$ in J and obtain conclusion (ii). □

Since $\texttt{trace}\left(\mathbf{K}\right)$ is fixed, Proposition 6.1.1 suggests that Problem (6.2) can be considered to be an optimization problem only with respect to $\mathbf{Z}$:

$$\begin{aligned} \max_{\mathbf{Z}\in\mathbb{R}^{N\times K}} \quad & \texttt{trace}\left(\mathbf{Z}^\top\mathbf{K}\mathbf{Z}\right) \\ \text{subject to:} \quad & \mathbf{Z}^\top\mathbf{Z} = \mathbf{I}. \end{aligned} \tag{6.3}$$

Note that an ambiguity arises in (6.2) and (6.3): If $\mathbf{Z}$ is the solution, then $\tilde{\mathbf{Z}} = \mathbf{Z}\mathbf{R}$ is also a solution, given an arbitrary rotation matrix $\mathbf{R}$. The following theorem summarizes the situation.

[2] In this paper we explicitly distinguish between multi-class classification and multi-label classification, both of which classify objects into multiple categories. In "multi-class classification", an object can *only* belong to one category, while in "multi-label classification" one object can belong to several categories simultaneously. Therefore, the former is a special case of the latter. In this paper, multi-class classification is viewed as a *single-output* problem, since we can label the multiple categories as integers and assign one integer to one object. The more general multi-label classification is considered as a *multi-output* setting, where each output could be a binary or multi-class classifier. Our setting covers both cases and is thus very general.

Theorem 6.1.2. *Suppose that* $[\mathbf{z}_1, \ldots, \mathbf{z}_N]$ *are the eigenvectors of matrix* $\mathbf{K}$*, and* $\lambda_1 \geq \ldots \geq \lambda_N$ *are the corresponding eigenvalues. If* $\tilde{\mathbf{Z}}$ *solves* (6.3)*, then*

(i) $\tilde{\mathbf{Z}} = [\mathbf{z}_1, \ldots, \mathbf{z}_K]\mathbf{R}$*, where* $\mathbf{R}$ *is an arbitrary* $K \times K$ *orthogonal rotation matrix;*

(ii) *The maximum of the objective function* (6.3) *is* $\sum_{k=1}^{K} \lambda_k$*.*

Proof. Denote $\tilde{\mathbf{Z}} =: [\tilde{\mathbf{z}}_1, \ldots, \tilde{\mathbf{z}}_K]$. The Lagrange formalism of Problem (6.3) is

$$L(\tilde{\mathbf{Z}}, \tilde{\mathbf{\Lambda}}) = \sum_{k=1}^{K} \tilde{\mathbf{z}}_k^\top \mathbf{K} \tilde{\mathbf{z}}_k - \sum_{k=1}^{K} \tilde{\lambda}_{k,k} (\tilde{\mathbf{z}}_k^\top \tilde{\mathbf{z}}_k - 1) - 2 \sum_{k>j} \tilde{\lambda}_{k,j} \tilde{\mathbf{z}}_k^\top \tilde{\mathbf{z}}_j,$$

where $(\tilde{\mathbf{\Lambda}})_{k,j} = \tilde{\lambda}_{k,j}$ is a symmetric matrix if we define $\tilde{\lambda}_{k,j} = \tilde{\lambda}_{j,k}$ for $k < j$. Setting its derivative with respect to $\tilde{\mathbf{z}}_k$ to be zero, we obtain

$$\frac{\partial L}{\partial \tilde{\mathbf{z}}_k} = 2\mathbf{K}\tilde{\mathbf{z}}_k - 2 \sum_{j=1}^{K} \tilde{\lambda}_{k,j} \tilde{\mathbf{z}}_j = 0, \quad k = 1, \ldots, K,$$

which can be rewritten as $\mathbf{K}\tilde{\mathbf{Z}} = \tilde{\mathbf{Z}}\tilde{\mathbf{\Lambda}}$. Since $\tilde{\mathbf{\Lambda}}$ is a symmetric matrix, we have $\tilde{\mathbf{\Lambda}} = \mathbf{R}^\top \mathbf{\Lambda} \mathbf{R}$ where $\mathbf{\Lambda}$ is a diagonal matrix and $\mathbf{R} \in \mathbb{R}^{K \times K}$ is an orthogonal rotation matrix satisfying $\mathbf{R}\mathbf{R}^\top = \mathbf{R}^\top \mathbf{R} = \mathbf{I}$. Then

$$\mathbf{K}\tilde{\mathbf{Z}} = \tilde{\mathbf{Z}}\mathbf{R}^\top \mathbf{\Lambda} \mathbf{R} \quad \Rightarrow \quad \mathbf{K}\tilde{\mathbf{Z}}\mathbf{R}^\top = \tilde{\mathbf{Z}}\mathbf{R}^\top \mathbf{\Lambda}.$$

Since $\mathbf{\Lambda}$ is diagonal, it is easy to see that the columns of $\mathbf{Z} = \tilde{\mathbf{Z}}\mathbf{R}^\top$ are the eigenvectors of $\mathbf{K}$. Thus the optimal $\tilde{\mathbf{Z}}$ is formed by an arbitrary rotation of $\mathbf{K}$'s eigenvectors, i.e. $\tilde{\mathbf{Z}} = \mathbf{Z}\mathbf{R}$. Inserting $\tilde{\mathbf{Z}}$ back into the objective function, we have the value of objective function as $\texttt{trace}\,(\mathbf{\Lambda})$, i.e., sum of the K corresponding eigenvalues of $\mathbf{K}$. It is easy to see that the maximal $\texttt{trace}\,(\mathbf{\Lambda})$ is the sum of the K largest eigenvalues, which proves (ii). In this case, $\tilde{\mathbf{Z}}$ is an arbitrary rotation of the K largest eigenvectors, thus conclusion (i) holds. □

Theorem 6.1.2 states that the eigenvectors of $\mathbf{K}$ form a solution of Problem (6.2), and any arbitrary rotation does not change the optimum. Therefore to remove the ambiguity, we only focus on the solutions given by the eigenvectors of $\mathbf{K}$, i.e., $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_K]$, and change Problem (6.2) equivalently to:[3]

$$\begin{aligned} \max_{\mathbf{z} \in \mathbb{R}^N} \quad & \mathbf{z}^\top \mathbf{K} \mathbf{z} \\ \text{subject to :} \quad & \mathbf{z}^\top \mathbf{z} = 1. \end{aligned} \tag{6.4}$$

[3]Solving Problem (6.4) itself only gives the first eigenvector $\mathbf{z}_1$ of $\mathbf{K}$. The full optimization problem should be recursively computing $\mathbf{z}_j$ by maximizing $\mathbf{z}^\top \mathbf{K} \mathbf{z}$ with the constraint $\mathbf{z}^\top \mathbf{z} = 1$ and $\mathbf{z} \perp \text{span}\{\mathbf{z}_1, \ldots, \mathbf{z}_{j-1}\}$. Here we state the problem as (6.4) for simplicity and also because its Lagrange formulism directly leads to the eigenvalue problem.

Optimizing the Lagrange form of Problem (6.4) leads to the eigenvalue problem $\mathbf{K}\mathbf{z} = \lambda\mathbf{z}$. Let $\mathbf{z}_1, \ldots, \mathbf{z}_N$ be the eigenvectors of $\mathbf{K}$ with the eigenvalues sorted in a *non-increasing* order. Using the first K eigenvectors, we solve Problem (6.2) as $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_K]$, $\mathbf{A} = \mathbf{Z}^\top\mathbf{X}$, and $\mathbf{B} = \mathbf{Z}^\top\mathbf{Y}$.

6.2 Multi-Output Regularized Projection

The solution to the supervised latent variable model is elegant, but only applicable to training data since for test data we do not have any output information. Therefore to complete the MORP algorithm, we need to refine the original problem.

Linear Constraint

It is easy to see that solving Problem (6.4) only gives the projections for training data with both input features in $\mathbf{X}$ and labels in $\mathbf{Y}$. We wish to construct a mapping $\Psi : \mathcal{X} \mapsto \mathcal{Z}$ such that it can handle new data with *only* the input features. To achieve this we can constrain that the latent variables are *linear mappings* of the input features $\mathbf{X}$:

$$\mathbf{Z} = \mathbf{X}\mathbf{W}.$$

This is one key step for the proposed supervised projection algorithm. With this constraint we turn the original problem to an optimization problem with respect to linear weight matrix $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_K] \in \mathbb{R}^{M\times K}$, and by definition we have $\mathbf{z}_i = \mathbf{X}\mathbf{w}_i$, for $i = 1, \ldots, K$. Plugging $\mathbf{z} = \mathbf{X}\mathbf{w}$ into (6.4), we have a new optimization problem with respect to a length-M vector $\mathbf{w}$:

$$\begin{aligned} \max_{\mathbf{w}\in\mathbb{R}^M} \quad & \mathbf{w}^\top\mathbf{X}^\top\mathbf{K}\mathbf{X}\mathbf{w} \\ \text{subject to :} \quad & \mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w} = 1. \end{aligned} \tag{6.5}$$

Regularization

Similar to other linear systems, the learned mappings can be unstable when the linear space $\mathtt{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ has a lower dimensionality than M, due to the small size of training set or dependence between input features.[4] As a result, a disturbance of $\mathbf{w}$ with an arbitrary $\mathbf{w}^* \perp \mathtt{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ does not change the objective function of optimization since $(\mathbf{w} + \mathbf{w}^*)^\top\mathbf{x}_i = \mathbf{w}^\top\mathbf{x}_i$, but may dramatically change the projections of unseen test points which are not in the spanned space. To improve the stability, we have to constrain $\mathbf{w}$ in some way.

[4] This will be a crucial problem when we kernelize the supervised projection and consider non-linear mapping (cf. Section 6.2.2), since the dimensionality of data point $\mathbf{x}$ in the reproducing kernel Hilbert space (RKHS) could be very high, or even infinite (e.g., in case of RBF kernel). See, e.g., [62].

Suppose $\texttt{rank}(\mathbf{K}) = N$, then maximizing (6.4) is equivalent to minimizing $\mathbf{z}^\top \mathbf{K}^{-1}\mathbf{z}$.[5] We introduce the *Tikhonov regularization* [68] into Problem (6.5) as the following

$$\begin{aligned} \min_{\mathbf{w}\in\mathbb{R}^M} \quad & \mathbf{w}^\top \mathbf{X}^\top \mathbf{K}^{-1}\mathbf{X}\mathbf{w} + \gamma\|\mathbf{w}\|^2 \\ \text{subject to :} \quad & \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} = 1, \end{aligned} \tag{6.6}$$

where $\|\mathbf{w}\|^2 = \mathbf{w}^\top\mathbf{w}$ is a penalty term which has been used in the ridge regression (see [37]), and γ is a tuning parameter. Setting the derivative of its Lagrange formulism with respect to $\mathbf{w}$ to be zero, we reach a generalized eigenvector problem:

$$\left[\mathbf{X}^\top \mathbf{K}^{-1}\mathbf{X} + \gamma\mathbf{I}\right]\mathbf{w} = \tilde{\lambda}\mathbf{X}^\top\mathbf{X}\mathbf{w}, \tag{6.7}$$

which gives generalized eigenvectors $\mathbf{w}_1, \ldots, \mathbf{w}_M$ with eigenvalues $\tilde{\lambda}_1 \leq \ldots \leq \tilde{\lambda}_M$. Note we sort eigenvalues in non-decreasing order, since we take the K eigenvectors with the smallest eigenvalues to form the mapping.

The following theorem shows that the regularization term $\|\mathbf{w}\|^2$ removes the ambiguity of mapping functions by restricting $\mathbf{w}$ in the span of $\mathbf{x}_i$, $i = 1, \ldots, N$, and thus improves the stability of mapping functions.

Theorem 6.2.1. *If* $\mathbf{w}$ *is an eigenvector of the generalized eigenvalue problem* (6.7)*, then* $\mathbf{w}$ *must be a linear combination of* $\mathbf{x}_i, i = 1, \ldots, N$*, i.e., there exists* $\boldsymbol{\alpha} \in \mathbb{R}^N$ *such that*

$$\mathbf{w} = \mathbf{X}^\top\boldsymbol{\alpha} = \sum_{i=1}^{N}(\boldsymbol{\alpha})_i\mathbf{x}_i.$$

Proof. Let $J(\mathbf{w})$ denote the cost function in (6.6), i.e.,

$$J(\mathbf{w}) := \mathbf{w}^\top \mathbf{X}^\top \mathbf{K}^{-1}\mathbf{X}\mathbf{w} + \gamma\|\mathbf{w}\|^2.$$

Obviously $J(\mathbf{w})$ achieves the minimum at the first eigenvector $\mathbf{w}$ of the generalized eigenvalue problem (6.7). Denote $\mathbf{w}_\|$ as the projection of $\mathbf{w}$ on $\texttt{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, then we can write $\mathbf{w} = \mathbf{w}_\| + \mathbf{w}_\perp$, where $\mathbf{w}_\perp$ is orthogonal to the subspace. Now we compare $J(\mathbf{w}_\|)$ with $J(\mathbf{w})$. Since

$$\mathbf{w}^\top\mathbf{x}_i = \mathbf{w}_\|^\top\mathbf{x}_i + \mathbf{w}_\perp^\top\mathbf{x}_i = \mathbf{w}_\|^\top\mathbf{x}_i,$$

we have $\mathbf{X}\mathbf{w}_\| = \mathbf{X}\mathbf{w}$, which means $J(\mathbf{w}_\|)$ and $J(\mathbf{w})$ agree on the first term. Since $\|\mathbf{w}\|^2 = \|\mathbf{w}_\|\|^2 + \|\mathbf{w}_\perp\|^2 \geq \|\mathbf{w}_\|\|^2$, $J(\mathbf{w}) \geq J(\mathbf{w}_\|)$ holds. This however must be an

[5] One can also minimize $\mathbf{z}^\top(-\mathbf{K})\mathbf{z}$ which is also equivalent, but then we lose non-negativity of its eigenvalues which may cause problems later on in solving the generalized eigenvalue problem. For the invertibility of $\mathbf{K}$, it is easy to show that $\mathbf{K}$ is at least positive semi-definite, since we have $\mathbf{u}^\top\mathbf{K}\mathbf{u} = (1-\beta)\mathbf{u}^\top\mathbf{X}\mathbf{X}^\top\mathbf{u} + \beta\mathbf{u}^\top\mathbf{Y}\mathbf{Y}^\top\mathbf{u} = (1-\beta)\|\mathbf{X}^\top\mathbf{u}\|^2 + \beta\|\mathbf{Y}^\top\mathbf{u}\|^2 \geq 0, \forall\mathbf{u} \in \mathbb{R}^N$. In case that $\mathbf{K}$ is not positive definite, it suffices to use pseudo-inverse instead, or makes it so by adding a tiny positive scalar to diagonal elements. In the dual form in Section 6.2.2, $\mathbf{K}$ is in most cases positive definite since $\mathbf{K}_x$ is normally positive definite (e.g., RBF kernel) and $\mathbf{K}_y$ is at least positive semi-definite.

equality since $J(\mathbf{w})$ achieves the minimum, so we have $\|\mathbf{w}_\perp\| = 0$ and hence $\mathbf{w}_\perp = 0$, which means $\mathbf{w}$ is a linear combination of $\mathbf{x}_i, i = 1, \ldots, N$.

So far we have proved the theorem for the first eigenvector (with the smallest eigenvalue). Given eigenvectors $\mathbf{w}_j$, $j = 1, \ldots, n-1$, it is known that the n-th eigenvector is obtained by first deflating the matrix $\mathbf{K}^{-1}$ with $\mathbf{K}^\dagger = \mathbf{K}^{-1} - \sum_{j=1}^{n-1} \lambda_j \mathbf{X}\mathbf{w}_j\mathbf{w}_j^\top\mathbf{X}^\top$, and then solving the following problem

$$\begin{aligned} \min_{\mathbf{w}\in\mathbb{R}^M} \quad & \mathbf{w}^\top\mathbf{X}^\top\mathbf{K}^\dagger\mathbf{X}\mathbf{w} + \gamma\|\mathbf{w}\|^2 \\ \text{subject to :} \quad & \mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w} = 1. \end{aligned}$$

Following the same procedure we can prove the eigenvector $\mathbf{w}_n$ also lies in the span of $\mathbf{x}_i, i = 1, \ldots, N$. This completes the proof. □

6.2.1 Multi-Output Regularized Projection - Primal Form

In Problem (6.7) we are interested in the eigenvectors with the smallest eigenvalues, whose computation is however the most unstable part in solving an eigenvalue problem. Thus we let $\lambda = 1/\tilde{\lambda}$ and turn the problem into an equivalent one as

$$\mathbf{X}^\top\mathbf{X}\mathbf{w} = \lambda\left[\mathbf{X}^\top\mathbf{K}^{-1}\mathbf{X} + \gamma\mathbf{I}\right]\mathbf{w}, \tag{6.8}$$

where we are seeking the K eigenvectors with the *largest* eigenvalues. To solve this problem we note that matrix $\mathbf{Q} = \left[\mathbf{X}^\top\mathbf{K}^{-1}\mathbf{X} + \gamma\mathbf{I}\right]$ is symmetric and positive definite, so there exists a symmetric and positive definite matrix $\mathbf{L}$ such that $\mathbf{Q} = \mathbf{L}^2$. Then we can change the problem to an equivalent one as $\mathbf{L}^{-1}\mathbf{X}^\top\mathbf{X}\mathbf{L}^{-1}\mathbf{L}\mathbf{w} = \lambda\mathbf{L}\mathbf{w}$, in which we can solve an eigenvalue problem for matrix $\mathbf{L}^{-1}\mathbf{X}^\top\mathbf{X}\mathbf{L}^{-1}$ with eigenvectors given as $\mathbf{z} = \mathbf{L}\mathbf{w}$. After that we can recover $\mathbf{w}$ as $\mathbf{w} = \mathbf{L}^{-1}\mathbf{z}$. Note that the solution satisfies $\mathbf{w}^\top\mathbf{Q}\mathbf{w} = 1$. This leads to an additional scaling factor for the mapping.

As can be seen from the optimization problem in (6.2), MORP assumes that the projections of all the data points in $\mathbf{X}$ have $\mathbf{I}$ as the covariance matrix. This means all the scaling factors (i.e., the variance on each projection direction) are not maintained in the projection values. This will cause problems since the pair-wise distances are changed. Therefore, we add these scaling factors back after we find the projection directions, which will recover PCA if no output information is available.

This primal form of the MORP algorithm is summarized in Algorithm 6.1. To only extract the projection dimensions which represent the intrinsic structure of the data, we centralize the data before performing the algorithm, i.e., subtract the sample mean from each data point.

6.2.2 Multi-Output Regularized Projection - Dual Form

So far we have considered linear mappings which project inputs $\mathbf{x}$ into latent space $\mathcal{Z}$. However, Theorem 6.2.1 implies that we can also derive a non-linear mapping.

Algorithm 6.1 Multi-Output Regularized Projection in Primal Form

Require: A set of N data points with input features $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top \in \mathbb{R}^{N \times M}$ and outputs $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]^\top \in \mathbb{R}^{N \times L}$.

Require: Projection dimension $K > 0$, parameters $0 \leq \beta \leq 1$, $\gamma \geq 0$.

1: Centralize data: $\mathbf{x}_i \Leftarrow \mathbf{x}_i - \bar{\mathbf{x}}$, $\mathbf{y}_i \Leftarrow \mathbf{y}_i - \bar{\mathbf{y}}$ where $\bar{\mathbf{x}} = \frac{1}{N}\sum_i \mathbf{x}_i$, $\bar{\mathbf{y}} = \frac{1}{N}\sum_i \mathbf{y}_i$.

2: Calculate $\mathbf{K} = (1-\beta)\mathbf{X}\mathbf{X}^\top + \beta\mathbf{Y}\mathbf{Y}^\top$.

3: Set $\mathbf{P} = \mathbf{X}^\top\mathbf{X}$ and $\mathbf{Q} = \left[\mathbf{X}^\top\mathbf{K}^{-1}\mathbf{X} + \gamma\mathbf{I}\right]$. Solve the generalized eigenvalue problem: $\mathbf{P}\mathbf{w} = \lambda\mathbf{Q}\mathbf{w}$, obtain eigenvectors $\mathbf{w}_1, \ldots, \mathbf{w}_K$ with largest K eigenvalues $\lambda_1 \geq \ldots \geq \lambda_K$ such that $\mathbf{w}^\top\mathbf{Q}\mathbf{w} = 1$.

Output: Projection function for the k-th dimension as $\psi_k(\mathbf{x}) = \mathbf{w}_k^\top\mathbf{x}$, $k = 1, \ldots, K$.

Let $\kappa_x(\cdot,\cdot)$ be the inner product in $\mathcal{X}$, i.e., $\kappa_x(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle = \mathbf{x}_i^\top\mathbf{x}_j$, then from Theorem 6.2.1 we obtain

$$\mathbf{z} = \mathbf{X}\mathbf{w} = \mathbf{X}\mathbf{X}^\top\boldsymbol{\alpha} = \mathbf{K}_x\boldsymbol{\alpha},$$

where $\mathbf{K}_x$ is the $N \times N$ Gram matrix with $(\mathbf{K}_x)_{ij} = \kappa_x(\mathbf{x}_i, \mathbf{x}_j)$. $\|\mathbf{w}\|^2$ can also be calculated using $\mathbf{K}_x$:

$$\|\mathbf{w}\|^2 = \mathbf{w}^\top\mathbf{w} = \boldsymbol{\alpha}^\top\mathbf{X}\mathbf{X}^\top\boldsymbol{\alpha} = \boldsymbol{\alpha}^\top\mathbf{K}_x\boldsymbol{\alpha}.$$

Similarly, we can define $\kappa_y(\cdot,\cdot)$ for inner product in $\mathcal{Y}$ and obtain a Gram matrix $\mathbf{K}_y = \mathbf{Y}\mathbf{Y}^\top$. Then we can calculate the matrix $\mathbf{K}$ using these Gram matrices as:

$$\mathbf{K} = (1-\beta)\mathbf{K}_x + \beta\mathbf{K}_y, \tag{6.9}$$

and express the *dual form* of Problem (6.6) with respect to coefficients $\boldsymbol{\alpha}$ as

$$\begin{aligned} \min_{\boldsymbol{\alpha}\in\mathbb{R}^N} \quad & \boldsymbol{\alpha}^\top\mathbf{K}_x\mathbf{K}^{-1}\mathbf{K}_x\boldsymbol{\alpha} + \gamma\boldsymbol{\alpha}^\top\mathbf{K}_x\boldsymbol{\alpha} \\ \text{subject to :} \quad & \boldsymbol{\alpha}^\top\mathbf{K}_x^2\boldsymbol{\alpha} = 1. \end{aligned} \tag{6.10}$$

The Lagrange solution of this problem leads to a generalized eigenvalue problem

$$\left[\mathbf{K}_x\mathbf{K}^{-1}\mathbf{K}_x + \gamma\mathbf{K}_x\right]\boldsymbol{\alpha} = \tilde{\lambda}\mathbf{K}_x^2\boldsymbol{\alpha}. \tag{6.11}$$

We obtain the generalized eigenvectors $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N$, with $\tilde{\lambda}_1 \leq \ldots \leq \tilde{\lambda}_N$. The first K eigenvectors are applied to form the mappings. Without the scaling factor, the k-th mapping function, $k = 1, \ldots, K$, is given by

$$\psi_k(\mathbf{x}) = \mathbf{w}_k^\top\mathbf{x} = \sum_{i=1}^{N}(\boldsymbol{\alpha}_k)_i\kappa_x(\mathbf{x}_i, \mathbf{x}).$$

As before we define $\lambda = 1/\tilde{\lambda}$ and change (6.11) to the following equivalent form:

$$\mathbf{K}_x^2\boldsymbol{\alpha} = \lambda\left[\mathbf{K}_x\mathbf{K}^{-1}\mathbf{K}_x + \gamma\mathbf{K}_x\right]\boldsymbol{\alpha}, \tag{6.12}$$

and hence we can choose the K eigenvectors with the largest eigenvalues.

Algorithm 6.2 Multi-Output Regularized Projection in Dual Form

Require: A set of N data points with input features $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top \in \mathbb{R}^{N \times M}$ and outputs $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]^\top \in \mathbb{R}^{N \times L}$.

Require: Kernel functions $\kappa_x(\cdot, \cdot)$ and $\kappa_y(\cdot, \cdot)$ for input space $\mathcal{X}$ and output space $\mathcal{Y}$.

Require: Projection dimension $K > 0$, parameters $0 \leq \beta \leq 1$, $\gamma \geq 0$.

1: Calculate two $N \times N$ matrices $(\mathbf{K}_x)_{ij} = \kappa_x(\mathbf{x}_i, \mathbf{x}_j)$, $(\mathbf{K}_y)_{ij} = \kappa_y(\mathbf{y}_i, \mathbf{y}_j)$.

2: Centralize the kernel matrices $\mathbf{K}_x$ and $\mathbf{K}_y$ using (4.3).

3: Calculate $\mathbf{K} = (1 - \beta)\mathbf{K}_x + \beta \mathbf{K}_y$.

4: Set $\mathbf{P} = \mathbf{K}_x^2$ and $\mathbf{Q} = \mathbf{K}_x \mathbf{K}^{-1} \mathbf{K}_x + \gamma \mathbf{K}_x$. Solve the generalized eigenvalue problem: $\mathbf{P}\boldsymbol{\alpha} = \lambda \mathbf{Q}\boldsymbol{\alpha}$, obtain eigenvectors $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_K$ with largest eigenvalues $\lambda_1 \geq \ldots \geq \lambda_K$ such that $\boldsymbol{\alpha}^\top \mathbf{Q} \boldsymbol{\alpha} = 1$.

Output: Projection function for the k-th dimension as $\psi_k(\mathbf{x}) = \mathbf{k}(\mathbf{X}, \mathbf{x})^\top \boldsymbol{\alpha}_k$, $k = 1, \ldots, K$, where $\mathbf{k}(\mathbf{X}, \mathbf{x}) := [\kappa_x(\mathbf{x}_1, \mathbf{x}), \ldots, \kappa_x(\mathbf{x}_N, \mathbf{x})]^\top$ and centralized via (4.4).

The algorithm is ready to deal with *non-linear* mappings. For this we consider a non-linear mapping $\boldsymbol{\phi} : \mathbf{x} \in \mathcal{X} \mapsto \boldsymbol{\phi}(\mathbf{x}) \in \mathcal{F}$, which maps $\mathbf{x}$ into a high-dimensional or even infinite-dimensional feature space $\mathcal{F}$, and change $\mathbf{X}$ to be $[\boldsymbol{\phi}(\mathbf{x}_1), \ldots, \boldsymbol{\phi}(\mathbf{x}_N)]^\top$. Then the kernel function is accordingly defined as

$$\kappa_x(\mathbf{x}_i, \mathbf{x}_j) = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle_{\mathcal{F}},$$

where we still have $\mathbf{K}_x = \mathbf{X}\mathbf{X}^\top$. Therefore, we can directly work with kernels (e.g., RBF kernel $\kappa_x(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\alpha \|\mathbf{x}_i - \mathbf{x}_j\|^2)$), without knowing $\boldsymbol{\phi}(\cdot)$ explicitly.

Similarly, we can define a non-linear mapping for $\mathcal{Y}$ and directly work on the corresponding kernel matrix $\mathbf{K}_y$. Although this paper mainly considers the linear kernel to explore the linear correlation of multivariate outputs, the formulism implies that the method can generally handle more complex outputs by using some other suitable kernels.

For centering of the data, as seen in Chapter 4 we can achieve this in the new feature space $\mathcal{F}$ without knowing the explicit mapping $\boldsymbol{\phi}(\cdot)$. The final dual form of the algorithm is summarized in Algorithm 6.2.

6.2.3 Discussions

MORP defines a general solution for supervised projection, i.e., *minimization of an output-regularized cost function*. In general one can go beyond Frobenius norm and consider more general cost for $\mathbf{X}$ and $\mathbf{Y}$:

$$(1 - \beta) f(\mathbf{X}, \mathbf{Z}) + \beta g(\mathbf{Y}, \mathbf{Z}),$$

where f and g define the *input-specific cost* and *output-specific cost* respectively, with respect to the observation ($\mathbf{X}$ or $\mathbf{Y}$) and the projection $\mathbf{Z}$. There may be some parameters involved (like $\mathbf{A}$ and $\mathbf{B}$ in the Frobenius norm case), and in general there is no analytical solution to this optimization problem. For instance, f could be matrix 1-norm (like the

case for sparse PCA [88]), and g could be hinge-lose for binary classification problem [72]. For simplicity and tractability we stick to the Frobenius norm in this chapter.

As a natural extension of Problem (6.2), we can have different output sets, say $\mathbf{Y}_1$ and $\mathbf{Y}_2$, associated with all input data. In this case we can add the reconstruction error of $\mathbf{Y}_1$ and $\mathbf{Y}_2$ to the cost function, but with different weights β:

$$(1-\beta_1-\beta_2)\|\mathbf{X}-\mathbf{ZA}\|^2+\beta_1\|\mathbf{Y}_1-\mathbf{ZB}_1\|^2+\beta_2\|\mathbf{Y}_2-\mathbf{ZB}_2\|^2,$$

and potentially $\mathbf{Y}_1$ and $\mathbf{Y}_2$ could have different intra-correlations. Both of these two output sets can be incorporated into MORP by defining possibly different kernels for $\mathbf{Y}_1$ and $\mathbf{Y}_2$, and including them into the matrix $\mathbf{K}$. Therefore, MORP introduces an elegant way to take into account various supervised information and allows great flexibility and generalization ability.

It is also possible to regularize the normal PCA projection using other types of supervised information, such as a hierarchy of outputs [43]. This information can be viewed as a different kind of cost in the MORP model.

Computational Issues

MORP solves a generalized eigenvalue problem for $M \times M$ matrices in the primal form, and for $N \times N$ matrices in the dual form, which in computational complexity is similar to unsupervised projection PCA and kernel PCA (see [32] for details of generalized eigenvalue problems). For implementation it is very easy and just takes several lines with Matlab. The calculations of kernels and matrix multiplications are the most time-consuming parts of the algorithm, as well as the matrix inversion in kernel form. But in general, the projection is quite acceptable and takes less than one minute for about 1000 data points with 500 input features (RBF kernel) and 50 output features (linear kernel).[6]

Parameters

MORP has three tuning parameters: K, β and γ. They should be chosen beforehand and probably be determined by cross-validation.

K is the dimensionality of the latent space and controls the reconstruction capability of the projection. In linear MORP, $K \leq M$ holds, while in the kernel case K is only upper bounded by number of the data points. Selection of K depends on the applications built on the learned mapping. Small K is sometimes preferred because it may be sufficient to recover the structure behind input and output data, and it is helpful to cut down the computational burden.

β satisfies $0 \leq \beta \leq 1$ and controls the trade-off between reconstruction errors of $\mathbf{X}$ and $\mathbf{Y}$. Special cases when $\beta = 0$ and $\beta = 1$ will be discussed in Section 6.3. In real

[6] We test the algorithm using Matlab on a PC with 2.0GHz CPU and 512M memory.

Algorithm 6.3 Simplified MORP Algorithm in Dual Form

Require: A set of N data points with input features $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top \in \mathbb{R}^{N \times M}$ and outputs $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]^\top \in \mathbb{R}^{N \times L}$.

Require: Kernel functions $\kappa_x(\cdot, \cdot)$ and $\kappa_y(\cdot, \cdot)$ for input space $\mathcal{X}$ and output space $\mathcal{Y}$.

Require: Projection dimension $K > 0$, parameters $0 \leq \beta \leq 1$.

1: Calculate two $N \times N$ matrices $(\mathbf{K}_x)_{ij} = \kappa_x(\mathbf{x}_i, \mathbf{x}_j)$, $(\mathbf{K}_y)_{ij} = \kappa_y(\mathbf{y}_i, \mathbf{y}_j)$.

2: Centralize the kernel matrices $\mathbf{K}_x$ and $\mathbf{K}_y$ using (4.3).

3: Calculate $\mathbf{K} = (1 - \beta)\mathbf{K}_x + \beta\mathbf{K}_y$.

4: Solve eigenvalue problem: $\mathbf{K}\mathbf{z} = \lambda\mathbf{z}$, obtain eigenvectors $\mathbf{z}_1, \ldots, \mathbf{z}_K$ with largest eigenvalues $\lambda_1 \geq \ldots \geq \lambda_K$.

5: Calculate $\boldsymbol{\alpha}_k = \mathbf{K}_x^{-1}\mathbf{z}_k$, $k = 1, \ldots, K$.

Output: Projection function for the k-th dimension as $\psi_k(\mathbf{x}) = \sqrt{\lambda_k}\mathbf{k}(\mathbf{X}, \mathbf{x})^\top\boldsymbol{\alpha}_k$, $k = 1, \ldots, K$, where $\mathbf{k}(\mathbf{X}, \mathbf{x}) := [\kappa_x(\mathbf{x}_1, \mathbf{x}), \ldots, \kappa_x(\mathbf{x}_N, \mathbf{x})]^\top$ and centralized via (4.4).

world applications, the value of β should depend on the quality of input and output with respect to the learning tasks. If input data are already enough or many of the outputs are missing, β should be relatively small; on the other hand if correlations among the multiple outputs are very strong, β is usually large and we are forcing the mapping to align more with principal components of $\mathbf{Y}$. β should also take into account the balance of traces of $\mathbf{K}_x$ and $\mathbf{K}_y$, since otherwise the matrix with large eigenvalues will dominate the matrix sum. From our experience, $\beta = 0.5$ is normally a good choice after we balance the traces of $\mathbf{K}_x$ and $\mathbf{K}_y$ to be the same. Performance comparison in the next section will give more details.

The non-negative scalar γ is set to prevent overfitting of the mapping functions. We found in our experiments that the quality of mappings is insensitive to γ if it is not too large. This is especially true for the dual form solution, because for positive definite matrices $\mathbf{K}_x$ and $\mathbf{K}$, matrix $\mathbf{Q}$ in Algorithm 6.2 is already very stable. Therefore we fixed it to be 0 for simplicity. In this case the whole learning algorithm can be greatly simplified, and the whole MORP algorithm can be viewed as a slightly modification of the kernel PCA algorithm. It is summarized in Algorithm 6.3 for clarity.

6.3 Connections to Related Works

The proposed algorithm MORP is a *supervised* projection from the input space to the latent space and aims at minimizing the reconstruction errors of both input data $\mathbf{X}$ and output data $\mathbf{Y}$. The algorithm is naturally generalizable to non-linear mappings and can explore the intra-correlation of multiple outputs.

In the literature there are some other well-known supervised projection methods, such as linear discriminant analysis (LDA) (see, e.g., [37, 65]), canonical correlation analysis (CCA) [42, 36], partial least squares (PLS) [74, 57], and kernel dependency estimation (KDE) [73]. In this section we briefly review these methods and point out the substantial

differences as well as possible connections between MORP and these methods. Other recent works include kernel dimensionality reduction [27], multi-task learning (e.g., [24, 79, 63]) and will also be briefly discussed.

6.3.1 Kernel Principal Component Analysis (Kernel PCA)

As discussed in Chapter 4, kernel PCA performs linear PCA in the reproducing kernel Hilbert space [61]. Let $\mathbf{K}_x$ denote the centered kernel matrix, kernel PCA solves the following eigenvalue problem: $\mathbf{K}_x\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}$. After the eigenvectors $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_K$ with largest eigenvalues $\lambda_1 \geq \ldots \geq \lambda_K$ are obtained, the non-linear mappings $\psi_k(\mathbf{x}) = \frac{1}{\sqrt{\lambda_k}} \sum_{i=1}^{N} (\boldsymbol{\alpha}_k)_i \kappa_x(\mathbf{x}_i, \mathbf{x}), k = 1, \ldots, K$ project the input data $\mathbf{x}$ to a K-dimensional latent space.

The proposed method MORP is motivated from optimization problem (6.2) and is also performing an unsupervised projection when $\beta = 0$, which is identical to Problem (6.1). In this case we have $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$, which is just the kernel matrix $\mathbf{K}_x$ for $\mathbf{X}$ in dual form, as revealed by (6.9). Then from Theorem 6.1.2 and remarks after Problem (6.4), it is clear that when $\beta = 0$, MORP is also solving the eigenvalue problem for $\mathbf{K}_x$ and thus is *identical* to kernel PCA. This also clarifies the connection between kernel PCA and Problem (6.1): The optimal solution $\mathbf{Z}$ to Problem (6.1) corresponds to, up to a rotation and scaling factor, the K non-linear principal components of kernel PCA in columns.

When $\beta = 0$, the equivalence of MORP and kernel PCA can also be shown from (6.10) which changes to

$$\begin{aligned} \min_{\boldsymbol{\alpha} \in \mathbb{R}^N} \quad & (1+\gamma)\boldsymbol{\alpha}^\top \mathbf{K}_x \boldsymbol{\alpha} \\ \text{subject to :} \quad & \boldsymbol{\alpha}^\top \mathbf{K}_x^2 \boldsymbol{\alpha} = 1 \end{aligned}$$

since $\mathbf{K} = \mathbf{K}_x$ holds. Under this situation, the regularization term controlled by γ is just a rescaling of the cost function and therefore does not change the cost function at all. Hence γ is just a nuisance parameter. On the other hand, if we let $\gamma \to \infty$, the regularization term in (6.10) dominates the cost function and MORP tends to be kernel PCA, whatever β is.

When $\beta > 0$, MORP actually performs *output regularized* kernel PCA or *supervised* kernel PCA, since it can be viewed as directly modifying the kernel matrix $\mathbf{K}$ with output information. With moderate β, the mapping takes into account the kernel of $\mathbf{Y}$, but is meanwhile restricted to the input space $\mathcal{X}$. No information of $\mathbf{y}$ is required for calculating MORP projection of a new data point $\mathbf{x}$.

6.3.2 Linear Discriminant Analysis (LDA)

LDA or Fisher Discriminant Analysis (FDA) is a canonical supervised projection for input data $\mathbf{X}$ and conceptually can only handle binary classification problems (see [37]).

It chooses a projection direction $\mathbf{w}$ that maximizes *inter-distance* of projected means, and meanwhile minimizes *intra-variances* of both classes. Therefore it is focusing on single classification problem where the output is one-dimensional, while in contrast MORP considers predictions with multivariate outputs and is thus more general.

A recently proposed approach Kernel Dimensionality Reduction (KDR) [27] is also a supervised method and aims to find a low-dimensional effective subspace that retains the statistical relationship between input data and output data. However, it has similar limitations and can only handle one-dimensional output.

6.3.3 Canonical Correlation Analysis (CCA) and Partial Least Square (PLS)

CCA has a long history in the statistics community (back to [42]) and aims at discovering the correlations between two representatives of the same objects (e.g., inputs $\mathbf{X}$ and outputs $\mathbf{Y}$ in our setting). The optimization problem solving CCA can be written as:

$$\begin{aligned} \max_{\mathbf{z}_x, \mathbf{z}_y \in \mathbb{R}^N} \quad & \texttt{Corr}(\mathbf{z}_x, \mathbf{z}_y) \\ \text{subject to :} \quad & \mathbf{z}_x = \mathbf{X}\mathbf{w}_x, \mathbf{z}_y = \mathbf{Y}\mathbf{w}_y, \end{aligned}$$

which is equivalent to minimization of $\|\mathbf{z}_x - \mathbf{z}_y\|^2$ when both $\mathbf{z}_x$ and $\mathbf{z}_y$ have norm 1 (see a recent discussion in [36]). In this sense CCA is a certain kind of supervised projection, but it does not require the projections $\mathbf{z}_x$ and $\mathbf{z}_y$ to guarantee low-reconstruction error of $\mathbf{X}$ and $\mathbf{Y}$. Therefore CCA only considers the *inter*-correlation between $\mathbf{z}_x$ and $\mathbf{z}_y$, but ignores the *intra*-correlation of either (especially $\mathbf{y}$). Instead, MORP takes into account all the inter- and intra-dependencies, since the projections minimize the reconstruction error of inputs and outputs simultaneously.

Another related approach is PLS, which is originally developed for regression problems in chemometrics [74, 75]. PLS aims at finding orthogonal projection directions for inputs $\mathbf{X}$, each of which maximizes the covariance between the outputs $\mathbf{Y}$ and a linear combination of $\mathbf{X}$:

$$\begin{aligned} \max_{\mathbf{z}_x \in \mathbb{R}^N} \quad & \texttt{Cov}(\mathbf{z}_x, \mathbf{Y}) \\ \text{subject to :} \quad & \mathbf{z}_x = \mathbf{X}\mathbf{w}_x, \mathbf{w}_x^\top \mathbf{w}_x = 1. \end{aligned}$$

PLS can be seen as a penalized CCA, since covariance is simply correlation weighted by square root of variance. Fukumizu et al. [68] pointed out that PLS cannot find a space of larger dimensionality than that of $\mathbf{Y}$, thus its generalization performance on new dimensions of outputs is restricted. Instead, our method can find in principle N orthogonal dimensions (if $\mathbf{K}_x$ is positive definite).

6.3.4 Kernel Dependency Estimation (KDE)

MORP is also related to kernel dependency estimation, a two-stage method for discovering dependency between possibly non-linear mappings of inputs and outputs [73]. In the first step, KDE performs kernel PCA on output $\mathbf{Y}$ and obtains some principal components $\mathbf{z}_y$; then a regression model with kernels (kernel ridge regression in [73]) is applied to $\mathbf{X}$, for projections of $\mathbf{Y}$ to each $\mathbf{z}_y$. No explicit mappings for $\mathbf{X}$ are available and a certain cost function has to be defined and minimized to find the output for a text point (so-called "pre-image" problem).

If applied to regression, MORP has similar behavior if $\beta = 1$: We are looking for a projection of $\mathbf{X}$ that seems enforced to entirely explain the dependency of outputs, as can be seen from Problem (6.2). However, it turns out to be not true if we introduce the regularization to prevent overfitting, since the Lagrange formalism of minimizing the regularizer $\boldsymbol{\alpha}^\top \mathbf{K}_x \boldsymbol{\alpha}$, under the constraint $\boldsymbol{\alpha}^\top \mathbf{K}_x^2 \boldsymbol{\alpha} = 1$, tends to a kernel PCA of input features. To see it more clearly, recall that $\mathbf{z} = \mathbf{K}_x \boldsymbol{\alpha}$ and thus we can write the cost function in (6.10) equivalently as

$$\max_{\mathbf{z} \in \mathbb{R}^N} \quad \mathbf{z}^\top \left(\mathbf{K}_y^{-1} + \gamma \mathbf{K}_x^{-1} \right)^{-1} \mathbf{z}, \tag{6.13}$$

where $\mathbf{K} = \mathbf{K}_y$ holds when $\beta = 1$ and we change the minimization to maximization by adding inversion to the matrix sum. Geometrically, (6.13) enforces $\mathbf{z}$ to be close to the eigenvector of $\mathbf{K}_y$ as well as that of $\mathbf{K}_x$, both with the largest eigenvalue. Therefore in this special case, MORP is performing *input regularized kernel PCA* for output $\mathbf{Y}$, while finally obtaining a mapping for $\mathbf{X}$ explicitly. Compared to the two-step approach taken by KDE, MORP can be a *feature mapping* step for regression models and provides a more elegant and direct way for multi-output regression.

6.3.5 Multi-task Learning

The work is also related to the recent research on multi-task learning (e.g., [24, 79, 63]), which learns many related predictive tasks together by exploring their dependency. We can first use the proposed algorithm to map the input features into a new space and then treat each task independently using the new representatives as input features. This two-stage solution can more easily deal with new tasks while multi-task learning has to retrain all the tasks once new tasks need to be handled.

6.4 Empirical Study

In this section we evaluate the proposed MORP algorithm based on two settings. The first is *prediction of user preferences*, in which we predict users' preferences on some data based on both the content features and rankings of other users. If we take each user as one output for all the data, we can think this setting as a natural multi-output problem, where

common interests of users stand for the intra-correlation among outputs. The second is to perform *multi-label classification* on the projected space, taking MORP as a preprocessing or feature transformation step. In this setting, we allow one data object to belong to multiple categories, and therefore different classification problems could have correlations between each other. This information will be utilized in MORP for deriving the mapping.

6.4.1 Prediction of User Preferences

Our first experiment is performed on a painting database which contains 642 paintings from 47 artists. A web-based online survey is built to gather user ratings. For all the paintings, we extract and combine *color histogram* (216-dim.), *correlagram* (256-dim.), *first and second color moments* (9-dim.) and *Pyramid wavelet texture* (10-dim.) to form 491-dimensional input features to represent the images. All the features are then centered and standardized with deviation 1. For the online survey, each user gave ratings, i.e., "like" or "dislike", to a randomly selected subset of paintings. Finally we obtained a total of $L = 190$ users' ratings encoded as $+1$ and -1. On average, each user had rated 89 paintings, and each painting was rated by 30 users.

In the experiment, a set of users are selected as *test users*, and 10-fold cross-validation is preformed for each test user with one fold training and 9 folds testing. A SVM using RBF kernel with all the 491 image features can be trained for each test user, and this is denoted Original Features and serves as the baseline. We will basically compare three projection methods. Kernel PCA performs unsupervised projection and maps the input data into a low-dimensional space. The two supervised methods, MORP and Kernel CCA, additionally make use of the rating information of the other users. All of the three competing methods use the same RBF kernel as in Original Features and same dimensionality. The new features given by these methods are then fed into a linear SVM for classification.

These algorithms are evaluated using two metrics. One is *Top-N accuracy*, i.e., the proportion of truly liked paintings among the N top-ranked paintings. Since normal users only care about the quality of first returned items, this quantity reflects the *subjective* quality of an information filtering system. The other is the *ROC* (receiver operating characteristics) *curve*, which plots *sensitivity* versus 1-*specificity*. Sensitivity is defined as the probability that a good painting is recommended by the system, and specificity is the probability that a disliked painting is rejected by the system. By changing the cut point (e.g. return top 10 or 20 paintings), a curve can be plotted. The area under the curve (AUC) measures the *objective* quality of ranking. A higher AUC indicates a better ranking.

We choose all the parameters for these algorithms as follows. The RBF kernel width $\sigma = 25$, which gives Original Features the best performance and is then fixed for all the projection methods. Different values for dimensionality K yield similar comparison results between these projection methods, and for simplicity we fix $K = 50$. In MORP, β is simply chosen as 0.5 to give equivalent weights to $\mathbf{K}_x$ and $\mathbf{K}_y$, after we scale $\mathbf{K}_x$ and $\mathbf{K}_y$

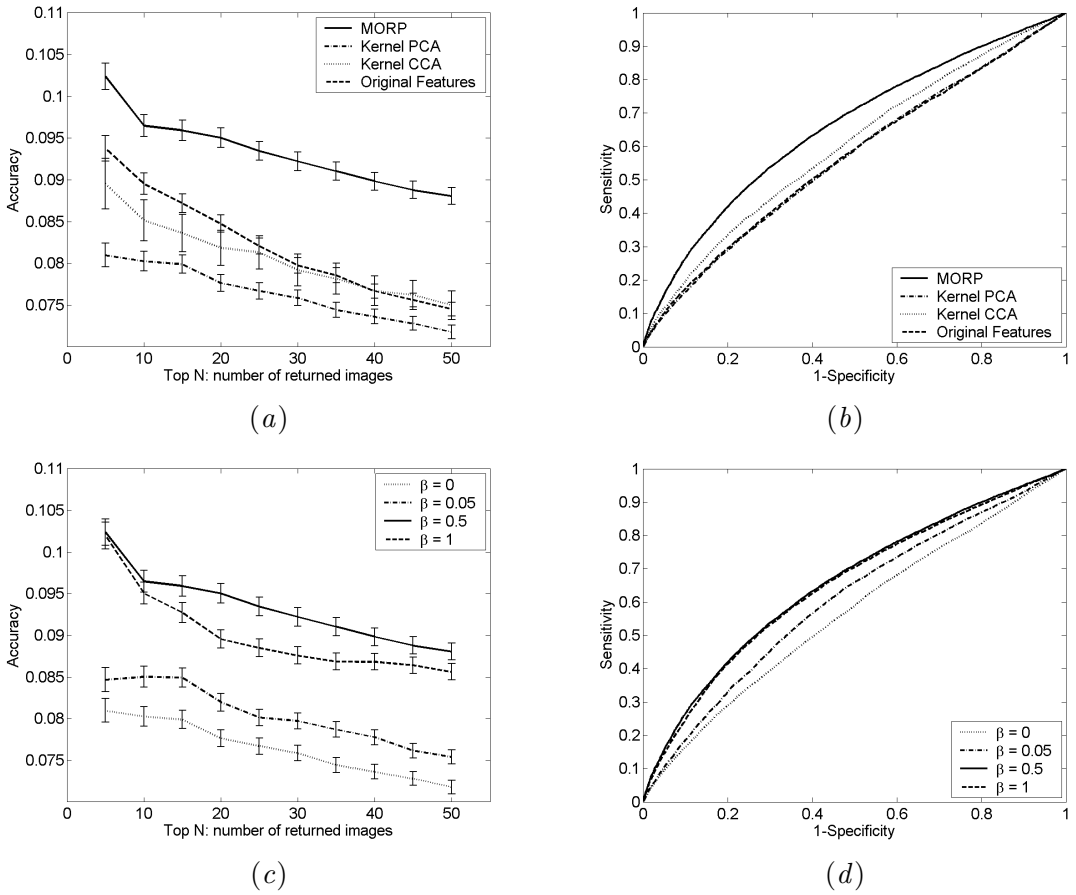


Figure 6.1: Comparison of algorithms for predicting user preferences. Figures on the left ((a),(c)) show the mean and standard deviation of prediction accuracy at different top number of returned images, and figures on the right ((b),(d)) show the corresponding ROC curve, i.e., Sensitivity versus 1-Specificity. The upper row compares four methods: MORP ($\beta = 0.5$, $\gamma = 1$), KERNEL PCA, KERNEL CCA (regularization parameter 0.9) and ORIGINAL FEATURES (with SVM). RBF kernel is used with $\sigma = 25$ for all kernel methods. All the projection methods use latent space with $K = 50$. The lower row compares MORP algorithms with different β values, where we have scaled $\mathbf{K}_x$ and $\mathbf{K}_y$ to ensure they have equal traces for balance. γ and K are chosen the same as in upper row.

to ensure they have equal traces for balance. γ is insensitive to the result and is simply fixed as 0. For KERNEL CCA we tune the regularization parameter for best performance and set it to be 0.9.

The performances of the four algorithms are shown in Figure 6.1(a) and (b), which clearly indicate that MORP significantly outperforms the rest in terms of both Top-N accuracy and ROC curve. The unsupervised methods ORIGINAL FEATURES and KERNEL PCA give unsatisfying results due to their ignorance of the correlation between user

ratings. ORIGINAL FEATURES performs better than KERNEL PCA, because it considers all the features for paintings. The other supervised method KERNEL CCA is unsuccessful in this setting, but obtains slightly better results than unsupervised methods in terms of ROC curve.

Our method can be seen as a way to combine content-based filters and collaborative filters. The two-stage treatment first learns a feature mapping based on many users' ratings, and then uses the new features to feed content-based filters. The parameter β controls the trade-off between the *content-based kernel* $\mathbf{K}_x$ and the *preference kernel* $\mathbf{K}_y$. In the second experiment, we study the impact of β in the performance of preference prediction, as shown in Figure 6.1(c) and (d) (as before, we scale $\mathbf{K}_x$ and $\mathbf{K}_y$ to ensure they have equal traces for balance). With $\beta = 0$, MORP is indeed kernel PCA and making an unsupervised projection, which gives a bad performance. As β increases gradually, the performance improves significantly, as shown here when $\beta = 0.05$. This clearly shows that the quality of projection has been improved by exploring the correlation among users. $\beta = 0.5$ gives the best results, corresponding to an even balance between the eigenspaces of the two matrices. And when β increases further, the performance drops down and overfitting occurs if no information of input is considered.

In the last experiment, we visualize the projections of paintings in the first two dimensions and see if we observe interesting distributions. As shown in Figure 6.2, we visualize four artists' paintings, Dali, Van Gogh, Monet and Asian (an anonymous asian painter). We denote points with different shapes and colors for paintings of different painters, and illustrate some images for clear explanation. The annotation beside each image clarifies the corresponding painter. For KERNEL PCA (Figure 6.2(a)), projections are built based on the low-level features of images, and paintings of different painters are somehow departed from each other (e.g., Dali's on the right, and Asian's on the left). This recovers the unsupervised characteristic of KERNEL PCA. However, for the specific task of user preferences prediction, this is not sufficient. It is more interesting to investigate the patterns found by MORP (Figure 6.2(b)), which forces the projection to reflect user correlations. Let's take a close look at the differences between Figure 6.2(a) and (b). Roughly speaking, there are three groups: left, middle and right, in Figure 6.2(b). First of all, it appears that the paintings of Van Gogh and Monet are frequently staying close (in left group), indicating that people often have similar preferences to these two artists' works, i.e., a user either likes both or dislikes both. Secondly, Van Gogh's self portraits (annotated as Z1, Z2 and Z3) stay very close from users' preferences (middle group), but it is interesting that they seem to be outliers in his paintings, and people's preferences to them are more correlated to the opinions for Dali's works. Furthermore, Dali's paintings in early years (e.g., the two marked out as D1 and D2 painted in 1922) substantially differ from the majority of his works in style. Instead, D1 and D2 stay close to the Asian's paintings, which are mainly about houses and buildings in countryside. Though a rigorous interpretation of the visualized distribution is lacking, we can still conclude that MORP maps paintings into a very meaningful space which will be beneficial for predicting interests for new users.

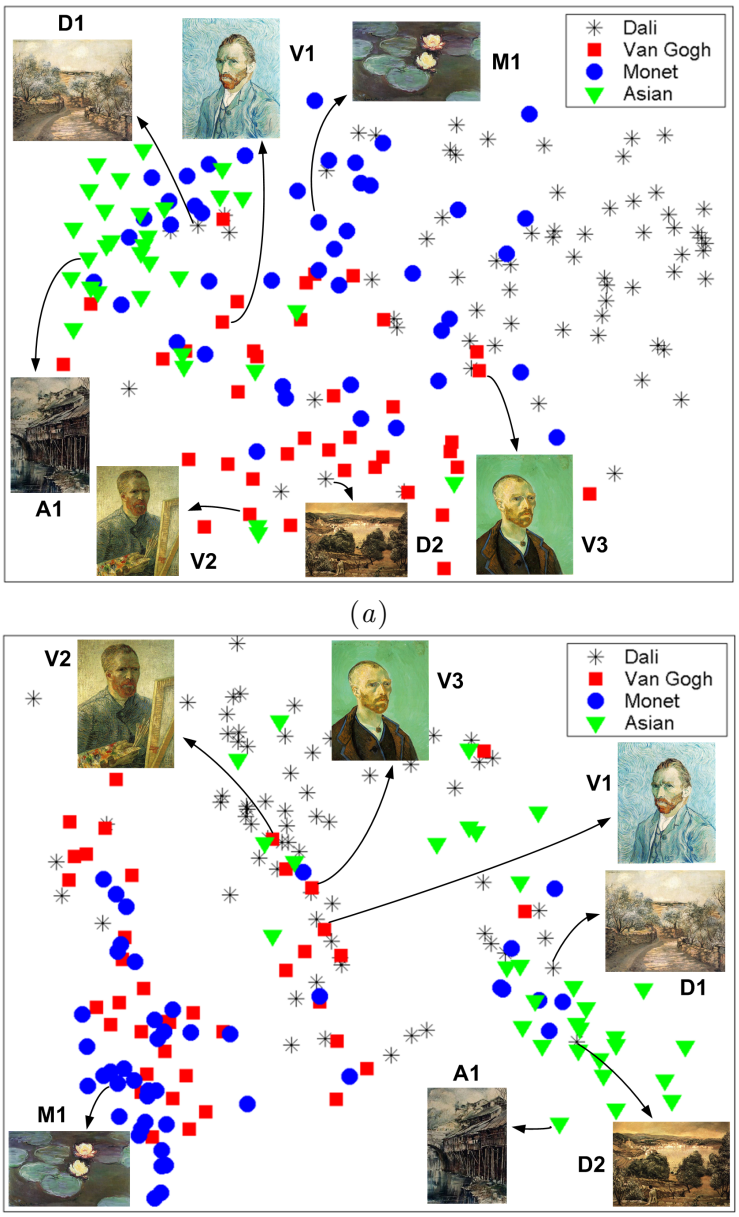


Figure 6.2: Visualization of paintings in the first two projected dimensions for KERNEL PCA (a) and MORP (b). Four painters are compared, and several images are shown with annotations ('D' for Dali, 'Z' for Van Gogh, 'M' for Monet, and 'A' for the Asian painter. Numbers show indexes for each painter). Parameter settings are the same as Figure 6.1(a).

6.4.2 Multi-label Classification

The experiment in this subsection is based on two text and one image data sets. The first text data is taken from Reuters-21578, which contains all the documents with multiple categories. Eliminating those minor categories that contain less then 50 documents, we have 47 categories to work with. Picking up all the words that occur at least in 5 documents, we finally obtain 1600 documents with 6076 words. The second text data is a subset of the RCV1-v2 data set, provided by Reuters and corrected by [52]. Since it is common that one document is assigned to multiple topics, this is an ideal data set for multi-label classification. After the same preprocessing, we finally obtain 9726 documents with 8997 words, and have 79 topics left. Standard TF-IDF features are then computed for these two text data sets. Our last data is a subset of the Corel image database that contains 1021 images. We manually labeled them into 37 categories. In average each image belongs to 3.6 categories and each category contains 98 images. As in the previous painting case, we extract the same 491-dimensional features as the input features for images and centralize them. In the following we denote "Reuters", "RCV1" and "Corel" for these data sets, respectively.

In the first setting (I), we randomly pick up 70% categories for classification and employ 5-fold cross-validation with one fold training and 4 folds testing. This is a standard classification setting, and our goal is to evaluate whether the feature mappings are generalizable to new data points. We will test the four algorithms described in the previous subsection, i.e., Original Features, Kernel PCA, MORP and Kernel CCA. Original Features still serves as the baseline, Kernel PCA defines unsupervised mappings, and the latter two give supervised mappings. Note that this setting is actually a *batch* version of many single-output binary classification tasks, where the performance is averaged over all tasks.

We also have a second setting (II), which aims to test the generalization ability of the projection methods on new categorization tasks. For this we consider the classification problems for the rest 30% categories. To make a fair comparison, we perform 5-fold cross-validation on previous unseen data, using the feature mappings derived from setting (I). We will also compare all the four methods in this setting.

The classification performance is compared using F_1 measure and AUC score. F_1 measure defines a trade-off between precision and recall, and is known to be a good metric for classification evaluation. Alternatively, AUC score measures the quality of ranking for specific classification problems. Both of these scores are averaged over all the output dimensions. We also tried classification accuracy, but didn't get informative comparison because most of the classification problems are very unbalanced (more than 90% of data are negative examples).

We use RBF kernel with width $\sigma = 25$ for the Corel data (which gives Original Features the best performance), and use linear kernels for the text data sets. For MORP we set the parameter β to 0.5 after rescaling $\mathbf{K}_x$ and $\mathbf{K}_y$, and fix γ as 0, same as previous painting experiment. The regularization parameter for Kernel CCA is tuned for

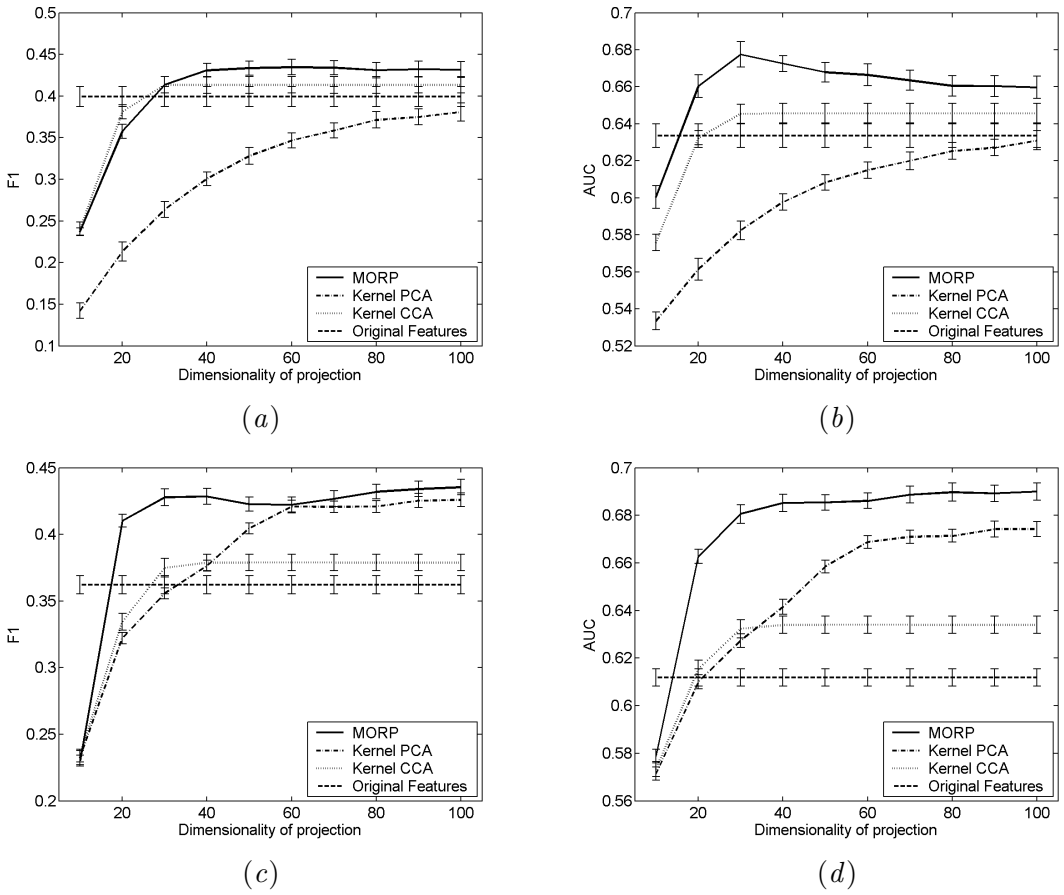


Figure 6.3: Classification performance on Reuters data set. Upper rows ((a),(b)) show results with setting (I), and lower rows ((c),(d)) show results with setting (II).

each data set and set to 0.9, 0.3 and 0.3 for Reuters, RCV1 and Corel, respectively. For both settings we repeat the experiments 10 times with randomization, and the performance versus dimensionality of projection is shown with means and standard deviations in Figure 6.3, Figure 6.4, Figure 6.5 for Reuters, RCV1, Corel, respectively.

The first observation from these figures is that MORP outperforms KERNEL PCA in almost all the cases. This indicates that the mapping functions in MORP are generalizable to new test data for setting (I), and also generalizable to new related prediction tasks, as seen in setting (II). The difference is especially big for setting (I), where the predictions are made for the known categories. By incorporating the output information for the training data, MORP can obtain more informative mappings for these specific tasks.

KERNEL CCA also performs a supervised projection, and in general it obtains worse but comparable results as MORP in setting (I). However, the performance is quite bad for setting (II), and in most cases it is even worse than KERNEL PCA. This indicates

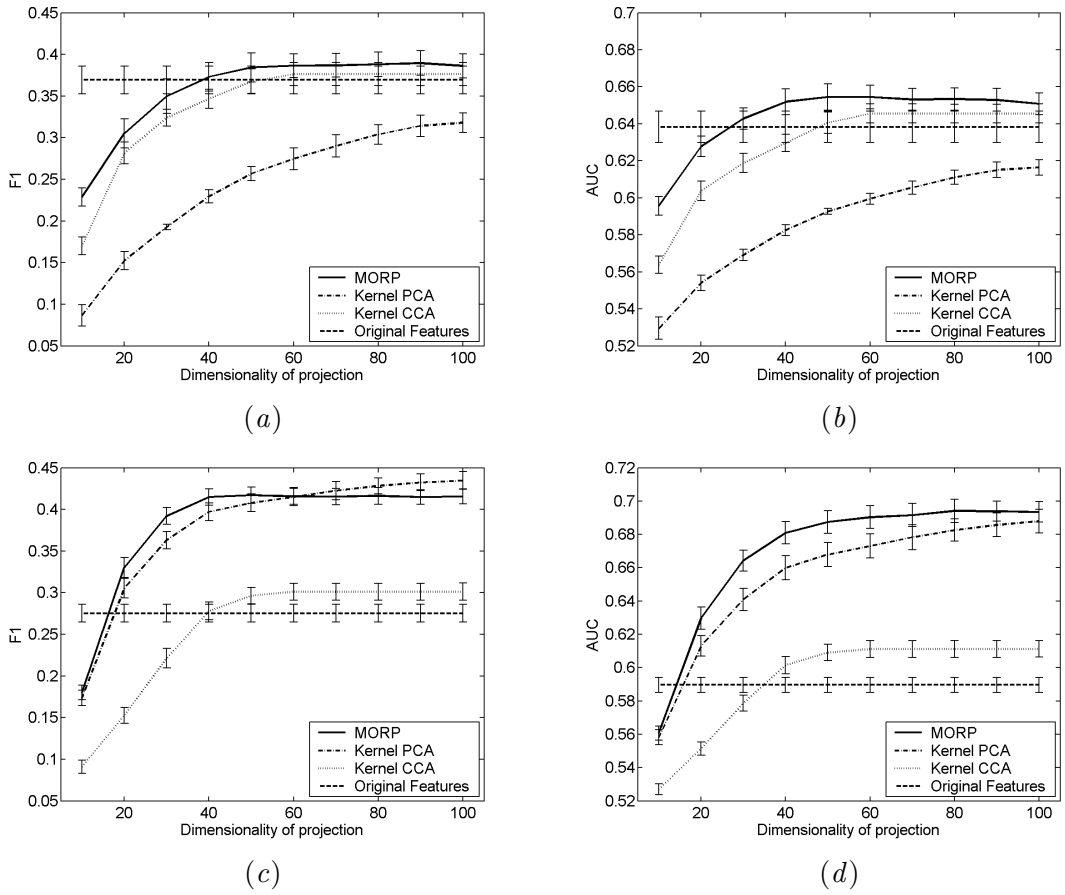


Figure 6.4: Classification performance on RCV1 data set. Upper rows ((a),(b)) show results with setting (I), and lower rows ((c),(d)) show results with setting (II).

that Kernel CCA suffers from overfitting and is not generalizable to new prediction tasks. It can also be seen that Kernel CCA approaches a constant performance after a small number of dimensions. The reason is that Kernel CCA could only extract pairs of mappings (one for $\mathbf{X}$ and the other for $\mathbf{Y}$), and thus could not obtain more dimensions than the number of outputs for training. This is very limited when we want the mappings generalizable to new outputs. In contrast, MORP does not have this problem and in general could extract N directions.

Another observation from these figures is that projected data can lead to better classification performance than Original Features that simply uses all the original features. This is especially the case in setting (II), where a large gap can be observed for all projection methods, even for the unsupervised method Kernel PCA. This suggests that projecting input data into a low-dimensional space can not only accelerate the classification tasks, but also improve the performance. Therefore, it is of great importance to derive a good projection method for supervised learning. MORP is seen to outperform all

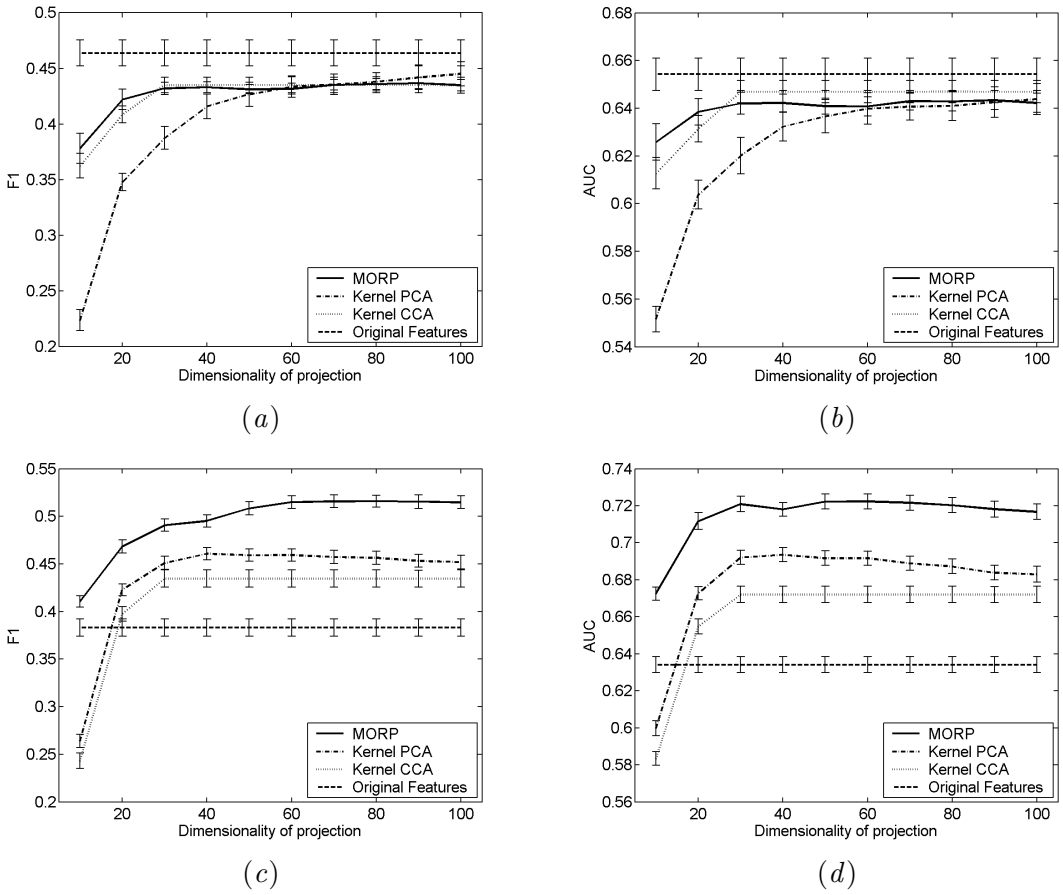


Figure 6.5: Classification performance on Corel data set. Upper rows ((a),(b)) show results with setting (I), and lower rows ((c),(d)) show results with setting (II).

the other methods in setting (II), and thus is a very good choice.

MORP has a tunable parameter β that controls the combination weights of the two kernels $\mathbf{K}_x$ and $\mathbf{K}_y$. For previous figures it is set to be 0.5 after rescaling the two kernels, and in this last experiments we study the classification performance when β is varied. Since we can see similar results for the three data sets, we only show in Figure 6.6 the illustrations for Reuters. The dimensionality K is also insensitive to the results and is simply fixed as 50.

A first impression from Figure 6.6 is that the two metrics have similar trends, and the curves are rather smooth (except when β approaching 1 in setting (II)). This indicates that the performance is not very sensitive to small changes of β value. When β increases from 0 to 1, it is seen that all the curves first increase and then decrease, indicating that a good trade-off should be identified for best performance. When β approaches 0, MORP tends to be kernel PCA and thus unsupervised. Outputs are ignored in this case, and

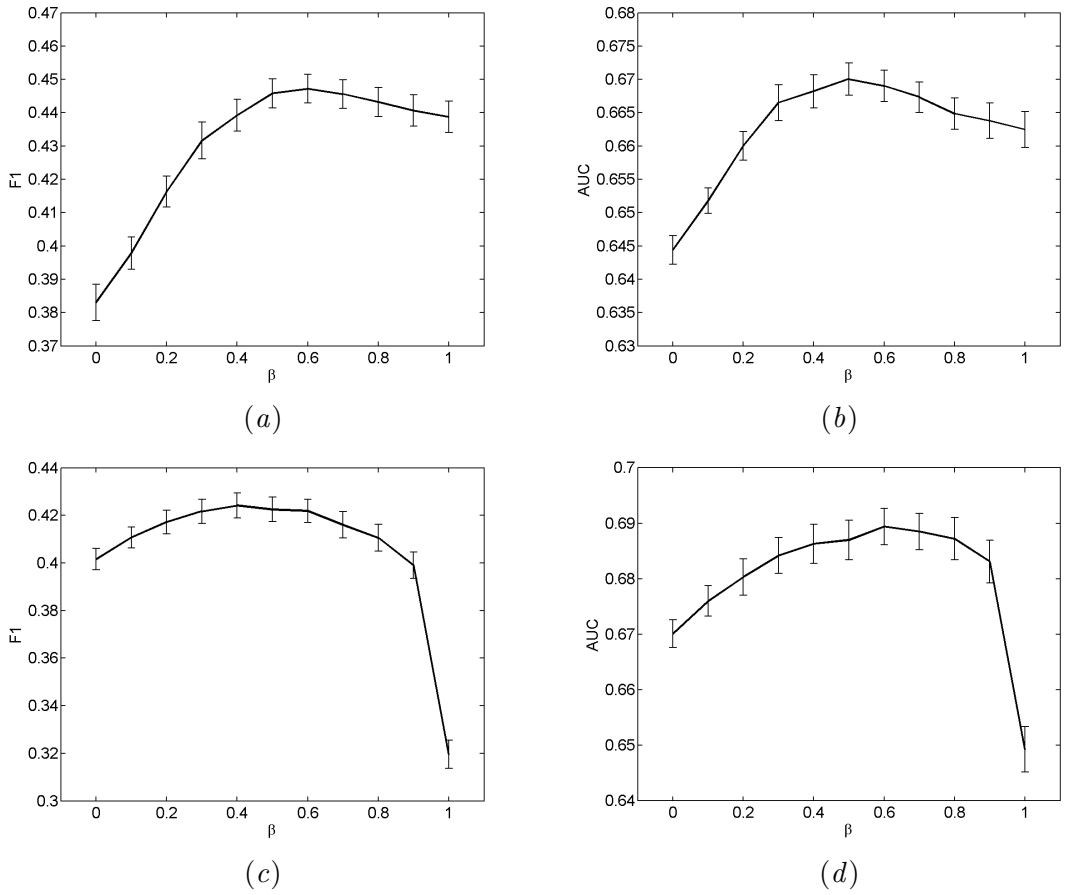


Figure 6.6: Performance of MORP with respect to β for Reuters data set. Upper rows ((a),(b)) show results with setting (I), and lower rows ((c),(d)) show results with setting (II). Dimensionality K is chosen to be 50. All the β values are chosen after we scale $\mathbf{K}_x$ and $\mathbf{K}_y$ to have equal traces.

poor performance is observed for both settings. On the other hand when β approaches 1, the mappings tend to solely explain outputs $\mathbf{Y}$, ignoring the intrinsic structure of inputs $\mathbf{X}$. This also leads to poor performance, especially for setting (II) because the mappings are not good to generalize to new outputs. Overfitting occurs in this case, where a sharp decrease can be observed with even a much worse performance than kernel PCA ($\beta = 0$). Finally, $\beta = 0.5$ is seen to be a good trade-off for both settings. From our experiences, a slightly larger β (e.g., 0.6) is better for setting (I), and a slightly smaller β (e.g., 0.4) is more stable for setting (II). One can also do model selection to select the best β for real world applications, and we hope this observation could be a good guidance.

6.5 Summary

In this chapter we introduce a supervised feature projection method called the MORP for multi-output regularized projection. It is based on a supervised latent variable model, and the projections retain the statistical information of not only input features but also the (possibly multivariate) outputs. We present both the primal and the dual formalisms for the linear mappings such that non-linear mappings can be derived by using reproducing kernels. The final solution ends up as a simple generalized eigenvalue problem that can be easily solved. The great advantage of proposed algorithm is that, given the learned projections, each dimension of outputs can then be modeled independently and multiple predictive problems can be solved simultaneously. The algorithm is applied for user preference prediction and multi-label classification, both with very encouraging results.

The MORP algorithm in this chapter solves a generalized eigenvalue problem. As discussed in Chapter 5, this deterministic algorithm has certain drawbacks and cannot be applied to large-scale problems. In the next chapter we will present a probabilistic solution, and this allows us to elegantly model unlabeled data as well and achieves semi-supervised projection. It also provides a means to learn β from the data. For future work, it is worth trying other loss functions for reconstruction errors, and applying other types of kernels to explore richer structured outputs.

Chapter 7

Supervised Probabilistic Principal Component Analysis

In this chapter we introduce a probabilistic framework for supervised feature projection. This is a probabilistic extension of the MORP algorithm proposed in the previous chapter, and an EM learning algorithm replaces the eigen-decomposition solution for MORP. The motivations of this new framework are as follows:

- Similar to the advantages of PPCA over PCA, the probabilistic framework allows the algorithm to be applicable to large-scale data sets and to data sets with missing data.
- The parameters in MORP can be learned from the inputs and outputs if the training data are sufficient.
- The probabilistic model provides an elegant way of incorporating the unlabeled data into projection learning. That is, we can derive a *semi-supervised feature projection* which is difficult in MORP.

We would emphasize that semi-supervised projection is quite novel in feature projection community. To the best of our knowledge, there is no other elegant solution to this problem. The semi-supervised setting is often true in real world problems, because labeling is expensive or unlabeled data are very easy to obtain. The popular supervised projection algorithms like linear discriminative analysis (LDA) and partial least square (PLS) cannot incorporate the unlabeled data into the mapping, which will cause problems when we have only very few labeled points. An ideal projection method should be able to take into account both the observed labeling information and the unlabeled inputs.

We call the new supervised projection model the *supervised probabilistic principal component analysis* (SPPCA), and the semi-supervised version the *semi-supervised probabilis-*

tic principal component analysis (S^{2}PPCA).[1] Both of them can be viewed as an extension of the PPCA model for unsupervised projection (Section 7.1). For parameter estimation we derive an efficient EM learning algorithm for both models (Section 7.2), and provide some theoretical justifications for the model behaviors (Section 7.3). Experimental results on various learning tasks show promising performance for both SPPCA and S^{2}PPCA models (Section 7.4).

7.1 The SPPCA Model

In this section we first review the PPCA model with some new notations, and then present our supervised models.

7.1.1 Probabilistic PCA (PPCA)

The PPCA model is a latent variable model and defines a generative process for each object $\mathbf{x}$ as

$$\mathbf{x} = \mathbf{W}_x\mathbf{z} + \boldsymbol{\mu}_x + \boldsymbol{\epsilon}_x,$$

where $\mathbf{z} \in \mathbb{R}^K$ are the latent variables, and $\mathbf{W}_x$ is a $M \times K$ factor loading matrix. The latent variables $\mathbf{z}$ are assumed to satisfy $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and the noise process $\boldsymbol{\epsilon}_x$ also takes an isotropic Gaussian form as $\boldsymbol{\epsilon}_x \sim \mathcal{N}(\mathbf{0}, \sigma_x^2\mathbf{I})$, with σ_x^2 the *noise level*. Additionally, $\boldsymbol{\mu}_x \in \mathbb{R}^M$ allows non-zero means for the inputs. Note here that we use subscript x to denote the model parameters with respect to input $\mathbf{x}$. See Figure 7.1(a) for an illustration of PPCA.

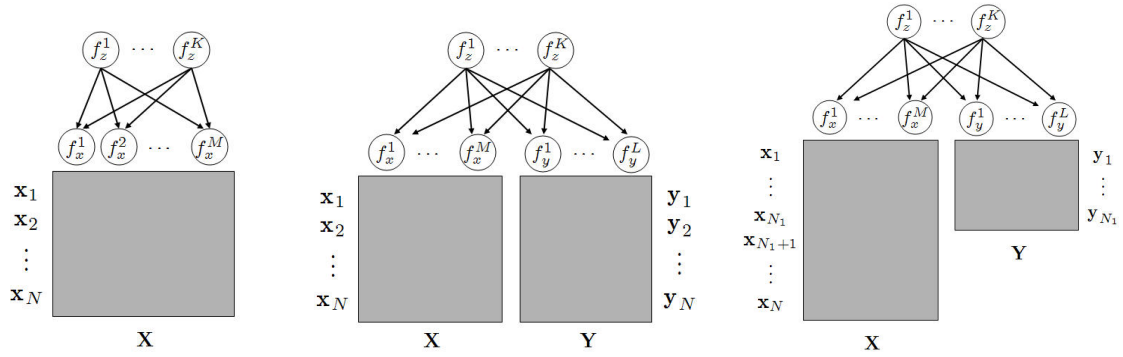
Proposition 5.1.1 gives the analytical solution of PPCA, and it is shown that when $\sigma_x^2 \to 0$, the projections of data $\mathbf{x}$ onto the K-dimensional principal subspace in PCA are identical to the latent variables $\mathbf{z}$ up to a rotation and scaling factor [70]. This result can be seen as a special case of the more general Theorem 7.3.1 in Section 7.3.

7.1.2 Supervised PPCA (SPPCA)

The key point of PPCA model is that all the M dimensions of $\mathbf{x}$ are *conditionally independent* given the latent variables $\mathbf{z}$, due to the isotropic property of the noise process. This indicates that the principal components in PPCA are *the K latent variables which best explain the data covariance*.

When supervised information is available, each object $\mathbf{x}$ is associated with an output value $y \in \mathcal{Y}$, e.g., $y \in \mathbb{R}$ for regression task and $y \in \{+1, -1\}$ for classification task. In general we believe there are *covariances* between input space $\mathcal{X}$ and output space $\mathcal{Y}$ (since otherwise the supervised learning task is not learnable), and it is reasonable to extend PPCA to model this covariance as well. Furthermore, when there are more than one

[1] This chapter is based on [87].



(a) PPCA (b) SPPCA (c) S^{2}PPCA

Figure 7.1: Illustrations of the three models PPCA, SPPCA and S^{2}PPCA. $\mathbf{X}$ and $\mathbf{Y}$ denote respectively the input and output matrices, where each row is one data point. $f_x^1, \ldots, f_x^M$ are the M input features, and $f_y^1, \ldots, f_y^L$ are the L outputs. On the top $f_z^1, \ldots, f_z^K$ are the K latent variables in each model. They are all in circles because they are variables in the probabilistic models. The arrows denote probabilistic dependency.

learning tasks (i.e., in a multi-task learning setting [24]), the *covariances between different tasks* can also be modeled by latent variables.

We now formally describe our proposed model family which we call the *supervised probabilistic principal component analysis* (SPPCA). Let the number of outputs be L, and each object $\mathbf{x}$ be associated with an output vector $\mathbf{y} = [y_1, \ldots, y_L]^\top \in \mathcal{Y} \subset \mathbb{R}^L$. In SPPCA the observed data $(\mathbf{x}, \mathbf{y})$ is generated from a latent variable model as

$$\begin{aligned} \mathbf{x} &= \mathbf{W}_x \mathbf{z} + \boldsymbol{\mu}_x + \boldsymbol{\epsilon}_x, \\ \mathbf{y} &= \mathbf{f}(\mathbf{z}, \boldsymbol{\Theta}) + \boldsymbol{\epsilon}_y, \end{aligned}$$

where $\mathbf{f}(\mathbf{z}, \boldsymbol{\Theta}) = [f_1(\mathbf{z}, \theta_1), \ldots, f_L(\mathbf{z}, \theta_L)]^\top$ encode the values of L deterministic functions $f_1, \ldots, f_L$ with parameters $\boldsymbol{\Theta} = \{\theta_1, \ldots, \theta_L\}$. Here $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ are the latent variables *shared by both inputs* $\mathbf{x}$ *and outputs* $\mathbf{y}$, and the two noise models are independent to each other and both defined as isotropic Gaussians: $\boldsymbol{\epsilon}_x \sim \mathcal{N}(\mathbf{0}, \sigma_x^2 \mathbf{I}), \boldsymbol{\epsilon}_y \sim \mathcal{N}(\mathbf{0}, \sigma_y^2 \mathbf{I})$. We use two noise levels σ_x^2 and σ_y^2 for inputs and outputs, respectively, and it is also straightforward to define different noise levels for different output dimensions if desired. See Figure 7.1(b) for an illustration of the model.

In SPPCA model we keep the nice property of *conditional independence*, i.e., all the input and output dimensions are conditionally independent to each other given the latent variables. If we integrate out the latent variables $\mathbf{z}$, the likelihood of observation $(\mathbf{x}, \mathbf{y})$ is obtained as

$$P(\mathbf{x}, \mathbf{y}) = \int P(\mathbf{x}, \mathbf{y}|\mathbf{z}) P(\mathbf{z}) \, d\mathbf{z} = \int P(\mathbf{x}|\mathbf{z}) P(\mathbf{y}|\mathbf{z}) P(\mathbf{z}) \, d\mathbf{z},$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and from the latent variable model,

$$\mathbf{x}|\mathbf{z} \sim \mathcal{N}(\mathbf{W}_x\mathbf{z} + \boldsymbol{\mu}_x, \sigma_x^2\mathbf{I}), \quad \mathbf{y}|\mathbf{z} \sim \mathcal{N}(\mathbf{f}(\mathbf{z}, \boldsymbol{\Theta}), \sigma_y^2\mathbf{I}). \tag{7.1}$$

After observing N pairs, the likelihood of all the observations $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$, with *i.i.d.* assumption, is simply $P(\mathcal{D}) = \prod_{i=1}^N P(\mathbf{x}_i, \mathbf{y}_i)$.

In the following we consider the simplest model in this family, i.e., we assume each function f_ℓ, $\ell = 1, \ldots, L$, is linear in $\mathbf{z}$:

$$f_\ell(\mathbf{z}, \theta_\ell) = {\mathbf{w}_y^\ell}^\top \mathbf{z} + \boldsymbol{\mu}_y^\ell,$$

where the parameters $\theta_\ell = \{\mathbf{w}_y^\ell, \mu_y^\ell\}$ include the linear coefficients and intercepts. Then we can group all the f_ℓ's and write

$$\mathbf{f}(\mathbf{z}, \boldsymbol{\Theta}) = \mathbf{W}_y\mathbf{z} + \boldsymbol{\mu}_y,$$

a similar form as the generative model for $\mathbf{x}$ where $\mathbf{W}_y = [\mathbf{w}_y^1, \ldots, \mathbf{w}_y^L]^\top$ and $\boldsymbol{\mu}_y = [\mu_y^1, \ldots, \mu_y^L]^\top$. The reason why we choose this form for $\mathbf{f}$ is that the EM learning is simple (see the next section), and we have closed form solution (see Section 7.3). We will discuss other forms of $\mathbf{f}$ in Section 7.3.3 which may need special approximation techniques.

Let us denote

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_x \\ \mathbf{W}_y \end{pmatrix}, \qquad \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{pmatrix}, \qquad \boldsymbol{\Phi} = \begin{pmatrix} \sigma_x^2\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \sigma_y^2\mathbf{I} \end{pmatrix},$$

then based on the model assumption, it is easily seen that $(\mathbf{x}, \mathbf{y})$ are jointly Gaussian distributed, with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Phi} + \mathbf{W}\mathbf{W}^\top$. All the parameters for the SPPCA model are $\boldsymbol{\Omega} = \{\mathbf{W}_x, \mathbf{W}_y, \boldsymbol{\mu}_x, \boldsymbol{\mu}_y, \sigma_x^2, \sigma_y^2\}$.

7.1.3 Semi-Supervised PPCA (S^{2}PPCA)

In SPPCA model, we assume we observe both the inputs $\mathbf{x}$ and outputs $\mathbf{y}$ for every data point. In many real world problems, however, we may only observe the outputs for a small portion of data, and have many *unlabeled data* in which only inputs $\mathbf{x}$ are known. This may be because some measures are unobservable, the labeling cost is too high, or simply we have too many unlabeled data available. Learning in this situation is in general called *semi-supervised learning*. For learning a projection, an ideal model would incorporate both the unlabeled inputs and the partially labeled outputs to define the mapping.

This can be easily done under the SPPCA framework. Let the number of labeled and unlabeled data points be N_1 and N_2, respectively, with $N = N_1 + N_2$. The whole observation is now $\mathcal{D} = \mathcal{D}_1 \bigcup \mathcal{D}_2 = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N_1} \bigcup \{\mathbf{x}_{i'}\}_{i'=N_1+1}^{N}$. The likelihood, with the independence assumption of all the data points, is calculated as

$$P(\mathcal{D}) = P(\mathcal{D}_1)P(\mathcal{D}_2) = \prod_{i=1}^{N_1} P(\mathbf{x}_i, \mathbf{y}_i) \prod_{i'=N_1+1}^{N} P(\mathbf{x}_{i'}),$$

where $P(\mathbf{x}_i, \mathbf{y}_i)$ is calculated as in SPPCA model, and $P(\mathbf{x}_{i'}) = \int P(\mathbf{x}_{i'}|\mathbf{z}_{i'})P(\mathbf{z}_{i'})\, d\mathbf{z}_{i'}$. Due to its applicability to semi-supervised projection, we call it *semi-supervised PPCA* or *S^2 PPCA* in this paper. Figure 7.1(c) illustrates this model.

Under the additional assumptions that all the f_ℓ's are linear, it can be easily checked that all the likelihood terms in this product are Gaussians. This makes the model easy to learn. Other forms of $\mathbf{f}$ will be discussed in Section 7.3.3.

When $N_2 = 0$, S^{2}PPCA degrades to SPPCA which is purely supervised. This means one can view SPPCA as a special case of S^{2}PPCA model with no unlabeled data. From the perspective of probabilistic modeling, S^{2}PPCA can also be viewed as an SPPCA model where all the $\mathbf{y}$'s for the N_2 unlabeled points are missing. Due to this close relationship, in the following we use SPPCA to denote both models unless clearly specified.

7.1.4 Projections in SPPCA Models

Analogous to the PPCA model, in SPPCA models the projection of data point $\mathbf{x}$ is directly given in the latent variables $\mathbf{z}$. If we know all the parameters $\mathbf{\Omega}$, calculating this projection is simply an *inference* problem. To do this we can apply Bayes' rule and calculate the posterior distribution of $\mathbf{z}$. Therefore we can obtain not only the mean projection vector, but also the *uncertainty* of the projection.

Projection for Fully Observed Data

When both inputs $\mathbf{x}$ and outputs $\mathbf{y}$ are observed, we can calculate the posterior distribution of $\mathbf{z}$ given $(\mathbf{x}, \mathbf{y})$ as

$$P(\mathbf{z}|\mathbf{x}, \mathbf{y}) \propto P(\mathbf{x}, \mathbf{y}|\mathbf{z})P(\mathbf{z}) = P(\mathbf{x}|\mathbf{z})P(\mathbf{y}|\mathbf{z})P(\mathbf{z}). \tag{7.2}$$

Since all the three terms on the right hand side are Gaussians, this distribution is also Gaussian $\mathcal{N}(\boldsymbol{\mu}_\mathbf{z}, \mathbf{\Sigma}_\mathbf{z})$ with

$$\mu_\mathbf{z} = \mathbf{A}^{-1}\left[\frac{1}{\sigma_x^2}\mathbf{W}_x^\top(\mathbf{x} - \boldsymbol{\mu}_x) + \frac{1}{\sigma_y^2}\mathbf{W}_y^\top(\mathbf{y} - \boldsymbol{\mu}_y)\right],\ \mathbf{\Sigma}_\mathbf{z} = \mathbf{A}^{-1},$$

where $\mathbf{A}$ is a $K \times K$ matrix defined as

$$\mathbf{A} = \frac{1}{\sigma_x^2}\mathbf{W}_x^\top\mathbf{W}_x + \frac{1}{\sigma_y^2}\mathbf{W}_y^\top\mathbf{W}_y + \mathbf{I}. \tag{7.3}$$

This means that the projection is $\boldsymbol{\mu}_\mathbf{z}$ with uncertainty $\mathbf{\Sigma}_\mathbf{z}$.

Projection for Pure Input Data

For a test data $\mathbf{x}_*$ that has no output information, what are the most likely latent variables $\mathbf{z}_*$? This is our ultimate goal in projection, and can also be done using Bayes' rule

$$P(\mathbf{z}_*|\mathbf{x}_*) \propto P(\mathbf{x}_*|\mathbf{z}_*)P(\mathbf{z}_*). \tag{7.4}$$

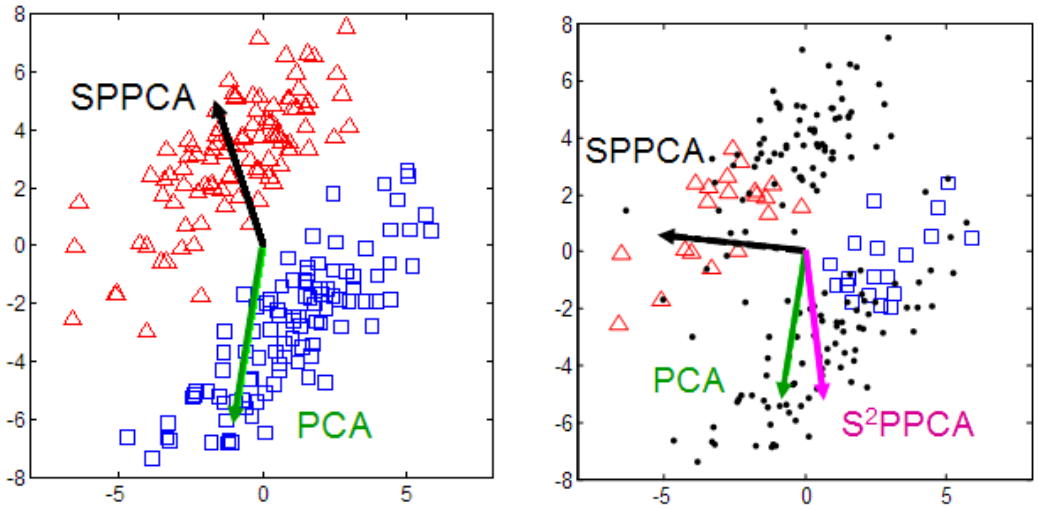


Figure 7.2: Projection directions for a 2D toy data. In the left figure, the data are fully labeled as $+1$ and -1, with red and blue colors, respectively. In the right figure, only part of the data are labeled, and unlabeled data are marked as black points. The first projection directions of various methods are shown with different colors.

This turns out again to be a Gaussian $\mathcal{N}(\boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}}, \boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}})$, with

$$\boldsymbol{\mu}_{\mathbf{z}|\mathbf{x}} = (\mathbf{W}_x^\top \mathbf{W}_x + \sigma_x^2 \mathbf{I})^{-1} \mathbf{W}_x^\top (\mathbf{x}_* - \boldsymbol{\mu}_x),$$
$$\boldsymbol{\Sigma}_{\mathbf{z}|\mathbf{x}} = \sigma_x^2 (\mathbf{W}_x^\top \mathbf{W}_x + \sigma_x^2 \mathbf{I})^{-1}.$$

This result looks similar as that in PPCA model, but the projection now is *supervised* because the learning of $\mathbf{W}_x$ is influenced by those observed outputs. This is clarified in the next section and will be theoretically proven in Section 7.3.

In Figure 7.2 we show the PCA, SPPCA and S^{2}PPCA projection directions for a 2D toy data. It is seen that PCA only finds the direction with maximal data variance, and SPPCA incorporates the label information and finds a direction which can separate the two classes well. When there are unlabeled data (right figure), S^{2}PPCA will also take the structure of unlabeled data into account and find a trade-off between PCA and SPPCA.

7.2 Learning in SPPCA Model

Learning in probabilistic models reduces to maximizing the data (log) likelihood with respect to all the model parameters. In the case of SPPCA model, the log likelihood of the whole observation $\mathcal{D}$ is

$$\mathcal{L}(\mathcal{D}) = \sum_{i=1}^{N} \log \int P(\mathbf{x}_i|\mathbf{z}_i) P(\mathbf{y}_i|\mathbf{z}_i) P(\mathbf{z}_i)\, d\mathbf{z}_i.$$

For SPPCA analytical solution exists, and we summarize it later in Theorem 7.3.1. For S^{2}PPCA model, since all the outputs for the unlabeled data are missing, there is no analytical solution. Fortunately we can derive an EM algorithm which is applicable to both models.

As shown for PPCA model, the EM algorithm iterates the two steps *expectation* (E-step) and *maximization* (M-step) until convergence, and it is guaranteed to find a local minima of the data likelihood. In the E-step, we fix the model parameters ($\mathbf{\Omega}$ for SPPCA models) and calculate the expected distributions of latent variables (all the $\mathbf{z}_i$'s for SPPCA models), and in the M-step we fix this distribution and maximize the *complete data likelihood* with respect to the model parameters. As will be discussed later, EM learning for SPPCA models is important because it can deal with very large data sets, and it has, in particular for SPPCA model with no unlabeled points, no local minima problem up to a rotation factor (see Section 7.3). For simplicity we only outline the update equations in the following and omit details which are similar to that of PPCA model.

7.2.1 EM Learning for SPPCA

In the E-step, for each data point i, we estimate the distribution of $\mathbf{z}_i$ given observation $(\mathbf{x}_i, \mathbf{y}_i)$. This is done using (7.2), and we calculate the sufficient statistics as

$$\langle \mathbf{z}_i \rangle = \mathbf{A}^{-1} \left[\frac{1}{\sigma_x^2} \mathbf{W}_x^\top (\mathbf{x}_i - \boldsymbol{\mu}_x) + \frac{1}{\sigma_y^2} \mathbf{W}_y^\top (\mathbf{y}_i - \boldsymbol{\mu}_y) \right], \tag{7.5}$$

$$\langle \mathbf{z}_i \mathbf{z}_i^\top \rangle = \mathbf{A}^{-1} + \langle \mathbf{z}_i \rangle \langle \mathbf{z}_i \rangle^\top, \tag{7.6}$$

where $\langle \cdot \rangle$ denotes the expectation under the posterior distribution $P(\mathbf{z}_i | \mathbf{x}_i, \mathbf{y}_i)$ given in (7.2).

In the M-step, we maximize the complete log-likelihood

$$\hat{\mathcal{L}}(\mathcal{D}) = \sum_{i=1}^{N} \int P(\mathbf{z}_i | \mathbf{x}_i, \mathbf{y}_i) \log \Big(P(\mathbf{x}_i | \mathbf{z}_i) P(\mathbf{y}_i | \mathbf{z}_i) P(\mathbf{z}_i) \Big) \, d\mathbf{z}_i$$

with respect to the model parameters, holding $P(\mathbf{z}_i | \mathbf{x}_i, \mathbf{y}_i)$ fixed from the E-step. This can be done by setting the partial derivatives with respect to each parameter to be zero. For means of $\mathbf{x}$ and $\mathbf{y}$ we have

$$\hat{\boldsymbol{\mu}}_x = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i, \qquad \hat{\boldsymbol{\mu}}_y = \frac{1}{N} \sum_{i=1}^{N} \mathbf{y}_i, \tag{7.7}$$

which are just sample means. Since they are always the same in all EM iterations, we can *center* the data by subtracting these means in the beginning and ignore these parameters in the learning process. So for simplicity we change the notations $\mathbf{x}_i$ and $\mathbf{y}_i$ to be the centered vectors in the following.

Algorithm 7.1 Learning in SPPCA Model - Primal Form

Require: N data points $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ with inputs $\mathbf{x}_i \in \mathbb{R}^M$ and outputs $\mathbf{y}_i \in \mathbb{R}^L$.
Require: A desired dimension $K < M$.
1: Calculate the sample means (7.7) and center the data by $\mathbf{x}_i \Leftarrow \mathbf{x}_i - \boldsymbol{\mu}_x$, $\mathbf{y}_i \Leftarrow \mathbf{y}_i - \boldsymbol{\mu}_y$.
2: Initialize model parameters $\boldsymbol{\Omega}$ randomly.
3: **repeat**
4: {E-step}
5: **for** $i = 1$ to N **do**
6: Calculate sufficient statistics (7.5) and (7.6);
7: **end for**
8: {M-step}
9: Update $\mathbf{W}_x$ and $\mathbf{W}_y$ via (7.8);
10: Update σ_x^2 and σ_y^2 via (7.9) and (7.10);
11: **until** the change of $\boldsymbol{\Omega}$ is smaller than a threshold.
Output: Parameters $\boldsymbol{\Omega}$ and projection vectors $\{\mathbf{z}_i\}_{i=1}^N$ which are obtained from E-step. For test data $\mathbf{x}_*$, the mean projection $\mathbf{z}_* = (\mathbf{W}_x^\top \mathbf{W}_x + \sigma_x^2 \mathbf{I})^{-1} \mathbf{W}_x^\top (\mathbf{x}_* - \boldsymbol{\mu}_x)$.

The mapping matrices $\mathbf{W}_x$ and $\mathbf{W}_y$ are updated as

$$\hat{\mathbf{W}}_x = \mathbf{X}^\top \mathbf{Z} \mathbf{C}^{-1}, \qquad \hat{\mathbf{W}}_y = \mathbf{Y}^\top \mathbf{Z} \mathbf{C}^{-1}, \tag{7.8}$$

where for clarity we use matrix notations $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$, $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]^\top$ and $\mathbf{Z} = [\langle \mathbf{z}_1 \rangle, \ldots, \langle \mathbf{z}_N \rangle]^\top$. Matrix $\mathbf{C}$ is defined to be a summation of all second sufficient statistics of the data, i.e., $\mathbf{C} = \sum_{i=1}^N \langle \mathbf{z}_i \mathbf{z}_i^\top \rangle$. Finally the noise levels are updated as

$$\hat{\sigma}_x^2 = \frac{1}{MN} \left[\sum_{i=1}^N \|\mathbf{x}_i\|^2 + \mathtt{trace}\left(\hat{\mathbf{W}}_x^\top \hat{\mathbf{W}}_x \mathbf{C} \right) - 2\,\mathtt{trace}\left(\mathbf{X} \hat{\mathbf{W}}_x \mathbf{Z}^\top \right) \right], \tag{7.9}$$

$$\hat{\sigma}_y^2 = \frac{1}{LN} \left[\sum_{i=1}^N \|\mathbf{y}_i\|^2 + \mathtt{trace}\left(\hat{\mathbf{W}}_y^\top \hat{\mathbf{W}}_y \mathbf{C} \right) - 2\,\mathtt{trace}\left(\mathbf{Y} \hat{\mathbf{W}}_y \mathbf{Z}^\top \right) \right], \tag{7.10}$$

where $\|\cdot\|$ denotes vector 2-norm. The whole algorithm is summarized in Algorithm 7.1 for clarity.

7.2.2 EM Learning for S^{2}PPCA

The log likelihood of the observations in S^{2}PPCA model is a sum of two parts: $\mathcal{L}_1 = \log P(\mathcal{D}_1)$ which contains all the labeled points, and $\mathcal{L}_2 = \log P(\mathcal{D}_2)$ which includes all unlabeled points. Therefore in E-step we need to deal with them differently. For a labeled points $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}_1$, the latent variables $\mathbf{z}_i$ are estimated as (7.5) and (7.6), the same as in SPPCA model. For an unlabeled point $\mathbf{x}_{i'} \in \mathcal{D}_2$, the distribution of $\mathbf{z}_{i'}$ is only conditioned

Algorithm 7.2 Learning in S^{2}PPCA Model - Primal Form

Require: N_1 labeled data points $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N_1}$ and N_2 unlabeled points $\{\mathbf{x}_{i'}\}_{i'=N_1+1}^{N}$, with inputs $\mathbf{x} \in \mathbb{R}^M$ and observed outputs $\mathbf{y} \in \mathbb{R}^L$. A desired dimension $K < M$.

Calculate the sample means (7.7) and center the data by $\mathbf{x}_i \Leftarrow \mathbf{x}_i - \boldsymbol{\mu}_x$, $\mathbf{y}_i \Leftarrow \mathbf{y}_i - \boldsymbol{\mu}_y$, $\mathbf{x}_{i'} \Leftarrow \mathbf{x}_{i'} - \boldsymbol{\mu}_x$.
Initialize model parameters $\boldsymbol{\Omega}$ randomly.
repeat
  {E-step}
  for $i = 1$ to N_1 **do**
    Calculate (7.5) and (7.6) for labeled data i;
  end for
  for $i' = N_1 + 1$ to N **do**
    Calculate (7.11) and (7.12) for unlabeled data i';
  end for
  {M-step}
  Update $\mathbf{W}_x$ and $\mathbf{W}_y$ via (7.13) and (7.14);
  Update σ_x^2 and σ_y^2 via (7.15) and (7.16);
until the change of $\boldsymbol{\Omega}$ is smaller than a threshold.

Output: Parameters $\boldsymbol{\Omega}$ and projection vectors $\{\mathbf{z}_i\}_{i=1}^{N}$ which are obtained from E-step. For test data $\mathbf{x}_*$, the mean projection $\mathbf{z}_* = (\mathbf{W}_x^\top \mathbf{W}_x + \sigma_x^2 \mathbf{I})^{-1} \mathbf{W}_x^\top (\mathbf{x}_* - \boldsymbol{\mu}_x)$.

on input $\mathbf{x}_{i'}$, which can be calculated via (7.4), with sufficient statistics (the data are assumed centered already):

$$\langle \mathbf{z}_{i'} \rangle = (\mathbf{W}_x^\top \mathbf{W}_x + \sigma_x^2 \mathbf{I})^{-1} \mathbf{W}_x^\top \mathbf{x}_{i'}, \tag{7.11}$$

$$\langle \mathbf{z}_{i'} \mathbf{z}_{i'}^\top \rangle = (\mathbf{W}_x^\top \mathbf{W}_x + \sigma_x^2 \mathbf{I})^{-1} + \langle \mathbf{z}_{i'} \rangle \langle \mathbf{z}_{i'} \rangle^\top, \tag{7.12}$$

where here $\langle \cdot \rangle$ denotes the expectation under the posterior distribution $P(\mathbf{z}_{i'} | \mathbf{x}_{i'})$ given in (7.4).

The M-step is similarly obtained by setting the partial derivatives of the complete log likelihood with respect to each parameter to zero. For the two mapping matrices, we have the updates

$$\hat{\mathbf{W}}_x = (\mathbf{X}_1^\top \mathbf{Z}_1 + \mathbf{X}_2^\top \mathbf{Z}_2)(\mathbf{C}_1 + \mathbf{C}_2)^{-1}, \tag{7.13}$$

$$\hat{\mathbf{W}}_y = \mathbf{Y}^\top \mathbf{Z}_1 \mathbf{C}_1^{-1}, \tag{7.14}$$

where $\mathbf{X}_1$, $\mathbf{Z}_1$, $\mathbf{C}_1$ are defined for labeled data, i.e., $\mathbf{X}_1 = [\mathbf{x}_1, \ldots, \mathbf{x}_{N_1}]^\top$, $\mathbf{Z}_1 = [\langle \mathbf{z}_1 \rangle, \ldots, \langle \mathbf{z}_{N_1} \rangle]^\top$, $\mathbf{C}_1 = \sum_{i=1}^{N_1} \langle \mathbf{z}_i \mathbf{z}_i^\top \rangle$, and $\mathbf{X}_2$, $\mathbf{Z}_2$, $\mathbf{C}_2$ are similarly defined for unlabeled data. It is seen that the update for $\mathbf{W}_x$ depends on both labeled data and unlabeled data, while $\mathbf{W}_y$ only depends on the labeled data. Updates for the noise levels are similar to those in SPPCA

model, except that for σ_x^2 we need to consider both labeled data and unlabeled data:

$$\hat{\sigma}_x^2 = \frac{1}{MN}\left[\sum_{i=1}^{N}\|\mathbf{x}_i\|^2 + \mathtt{trace}\left(\hat{\mathbf{W}}_x^\top \hat{\mathbf{W}}_x(\mathbf{C}_1+\mathbf{C}_2)\right) - 2\,\mathtt{trace}\left(\hat{\mathbf{W}}_x(\mathbf{Z}_1^\top\mathbf{X}_1 + \mathbf{Z}_2^\top\mathbf{X}_2)\right)\right], \tag{7.15}$$

$$\hat{\sigma}_y^2 = \frac{1}{LN_1}\left[\sum_{i=1}^{N_1}\|\mathbf{y}_i\|^2 + \mathtt{trace}\left(\hat{\mathbf{W}}_y^\top \hat{\mathbf{W}}_y\mathbf{C}_1\right) - 2\,\mathtt{trace}\left(\mathbf{Y}\hat{\mathbf{W}}_y\mathbf{Z}_1^\top\right)\right]. \tag{7.16}$$

The whole algorithm is summarized in Algorithm 7.2. When $N_2 = 0$, i.e., we have no unlabeled data, the learning algorithm reduces to SPPCA learning in Algorithm 7.1.

7.2.3 EM Learning in Dual Form

As discussed in Chapter 4, when the number of data points is less than the number of features, i.e., $N < M$, it is more efficient to consider the *dual* solution for PCA in which we perform SVD to the Gram matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$. The canonical PCA is sometimes called the *primal* solution. For SPPCA we have similar dual solution, and it can be directly derived from the EM learning in previous subsections, similarly to the way we extend PPCA to PKPCA in Chapter 5. Since SPPCA can be viewed as a special case of S^{2}PPCA, we only focus on S^{2}PPCA model here.

Let $\mathbf{C} = \mathbf{C}_1 + \mathbf{C}_2$, and

$$\mathbf{X} = \begin{pmatrix}\mathbf{X}_1\\ \mathbf{X}_2\end{pmatrix}, \quad \mathbf{Z} = \begin{pmatrix}\mathbf{Z}_1\\ \mathbf{Z}_2\end{pmatrix}, \quad \mathbf{K} = \mathbf{X}\mathbf{X}^\top = \begin{pmatrix}\mathbf{X}_1\mathbf{X}^\top\\ \mathbf{X}_2\mathbf{X}^\top\end{pmatrix} = \begin{pmatrix}\mathbf{K}_1\\ \mathbf{K}_2\end{pmatrix},$$

then (7.13) can be written as $\hat{\mathbf{W}}_x = \mathbf{X}^\top\mathbf{Z}\mathbf{C}^{-1}$. This leads to

$$\hat{\mathbf{W}}_x^\top\hat{\mathbf{W}}_x = \mathbf{C}^{-1}\mathbf{Z}^\top\mathbf{K}\mathbf{Z}\mathbf{C}^{-1}, \tag{7.17}$$

$$\hat{\mathbf{W}}_x^\top\mathbf{x} = \mathbf{C}^{-1}\mathbf{Z}^\top\mathbf{k}(\mathbf{X},\mathbf{x}), \tag{7.18}$$

which are the building blocks for the non-linear extension. In the E-step, we first rewrite $\mathbf{A}$ as

$$\hat{\mathbf{A}} = \frac{1}{\sigma_x^2}\mathbf{C}^{-1}\mathbf{Z}^\top\mathbf{K}\mathbf{Z}\mathbf{C}^{-1} + \frac{1}{\sigma_y^2}\mathbf{C}_1^{-1}\mathbf{Z}_1^\top\mathbf{Y}\mathbf{Y}^\top\mathbf{Z}_1\mathbf{C}_1^{-1} + \mathbf{I}.$$

Applying (7.17) and (7.18) in sufficient statistics (7.5), we get

$$\hat{\mathbf{Z}}_1 = \left[\frac{1}{\sigma_x^2}\mathbf{K}_1\mathbf{Z}\mathbf{C}^{-1} + \frac{1}{\sigma_y^2}\mathbf{Y}\mathbf{Y}^\top\mathbf{Z}_1\mathbf{C}_1^{-1}\right]\hat{\mathbf{A}}^{-1}, \tag{7.19}$$

by collecting $\langle\mathbf{z}_i\rangle$ in columns and transposing it. Sufficient statistics (7.6) can be written in terms of $\mathbf{C}_1$:

$$\hat{\mathbf{C}}_1 = \sum_{i=1}^{N_1}\langle\mathbf{z}_i\mathbf{z}_i^\top\rangle = N_1\hat{\mathbf{A}}^{-1} + \hat{\mathbf{Z}}_1^\top\hat{\mathbf{Z}}_1. \tag{7.20}$$

Algorithm 7.3 Learning in S^{2}PPCA Model - Dual Form

Require: N_1 labeled data points $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N_1}$ and N_2 unlabeled points $\{\mathbf{x}_{i'}\}_{i'=N_1+1}^{N}$, with inputs $\mathbf{x} \in \mathbb{R}^M$ and observed outputs $\mathbf{y} \in \mathbb{R}^L$.

Require: A kernel function $\kappa(\cdot,\cdot)$. A desired dimension $K < M$.

1: Calculate kernel matrix $\mathbf{K}$ via $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ and center it using (4.3).
2: Calculate output sample mean $\boldsymbol{\mu}_y$ using (7.7) and let $\mathbf{y}_i \Leftarrow \mathbf{y}_i - \boldsymbol{\mu}_y$.
3: Initialize $\mathbf{Z}$, $\mathbf{C}$ and model parameters $\boldsymbol{\Omega}$ randomly.
4: **repeat** {The EM-step}
5: Calculate $\mathbf{Z}_1$ and $\mathbf{C}_1$ using (7.19) and (7.20);
6: Calculate $\mathbf{Z}_2$ and $\mathbf{C}_2$ using (7.21) and (7.22);
7: Update σ_x^2 and σ_y^2 via (7.23) and (7.24);
8: **until** the change of $\boldsymbol{\Omega}$ is smaller than a threshold.

Output: Parameters $\boldsymbol{\Omega}$ and projection vectors $\{\mathbf{z}_i\}_{i=1}^{N}$. For test data $\mathbf{x}_*$ the mean projection $\mathbf{z}_* = \mathbf{C}\left(\mathbf{Z}^\top\mathbf{K}\mathbf{Z} + \sigma_x^2\mathbf{C}^2\right)^{-1}\mathbf{Z}^\top\mathbf{k}(\mathbf{X}, \mathbf{x}_*)$ with $\mathbf{k}(\mathbf{X}, \mathbf{x}_*) = [\kappa(\mathbf{x}_1, \mathbf{x}_*), \ldots, \kappa(\mathbf{x}_N, \mathbf{x}_*)]^\top$ and centered via (4.4).

Similarly, we obtain the following two updates for unlabeled data:

$$\hat{\mathbf{Z}}_2 = \frac{1}{\sigma_x^2}\mathbf{K}_2\mathbf{Z}\mathbf{C}^{-1}\left[\frac{1}{\sigma_x^2}\mathbf{C}^{-1}\mathbf{Z}^\top\mathbf{K}\mathbf{Z}\mathbf{C}^{-1} + \mathbf{I}\right]^{-1}, \tag{7.21}$$

$$\hat{\mathbf{C}}_2 = N_2\left[\frac{1}{\sigma_x^2}\mathbf{C}^{-1}\mathbf{Z}^\top\mathbf{K}\mathbf{Z}\mathbf{C}^{-1} + \mathbf{I}\right]^{-1} + \hat{\mathbf{Z}}_2^\top\hat{\mathbf{Z}}_2. \tag{7.22}$$

In M-step, we only need to update variances σ_x^2 and σ_y^2 as

$$\hat{\sigma}_x^2 = \frac{1}{MN}\left[\mathtt{trace}\left(\mathbf{K}\right) - \mathtt{trace}\left(\hat{\mathbf{Z}}^\top\mathbf{K}\hat{\mathbf{Z}}\hat{\mathbf{C}}^{-1}\right)\right], \tag{7.23}$$

$$\hat{\sigma}_y^2 = \frac{1}{LN}\left[\mathtt{trace}\left(\mathbf{Y}\mathbf{Y}^\top\right) - \mathtt{trace}\left(\hat{\mathbf{Z}}^\top\mathbf{Y}\mathbf{Y}^\top\hat{\mathbf{Z}}\hat{\mathbf{C}}^{-1}\right)\right], \tag{7.24}$$

which can be easily verified from (7.9) and (7.10).

As in the case of PKPCA, all interesting terms in the EM algorithm take input data into account only via inner product. Then we can define a kernel function $\kappa(\cdot,\cdot)$ for non-linear mappings with S^{2}PPCA model [62]. To make the data centered in the kernel induced feature space, we need to modify the kernel matrix the same way as in (4.3) and (4.4). The whole algorithm is summarized in Algorithm 7.3.

In Figure 7.3 and Figure 7.4 we show the dual-form SPPCA and S^{2}PPCA behaviors on a 2D toy data. The data consist of 4 clusters, and we assume the classification problem is not linearly separable. RBF kernel is used with $\alpha = 0.5$. In Figure 7.3 all the data points are labeled, but kernel PCA cannot use this information and only projects the data based on the clustering structure. SPPCA, on the other hand, uses the class information and can project the data into a first dimension which can exactly separate the two classes.

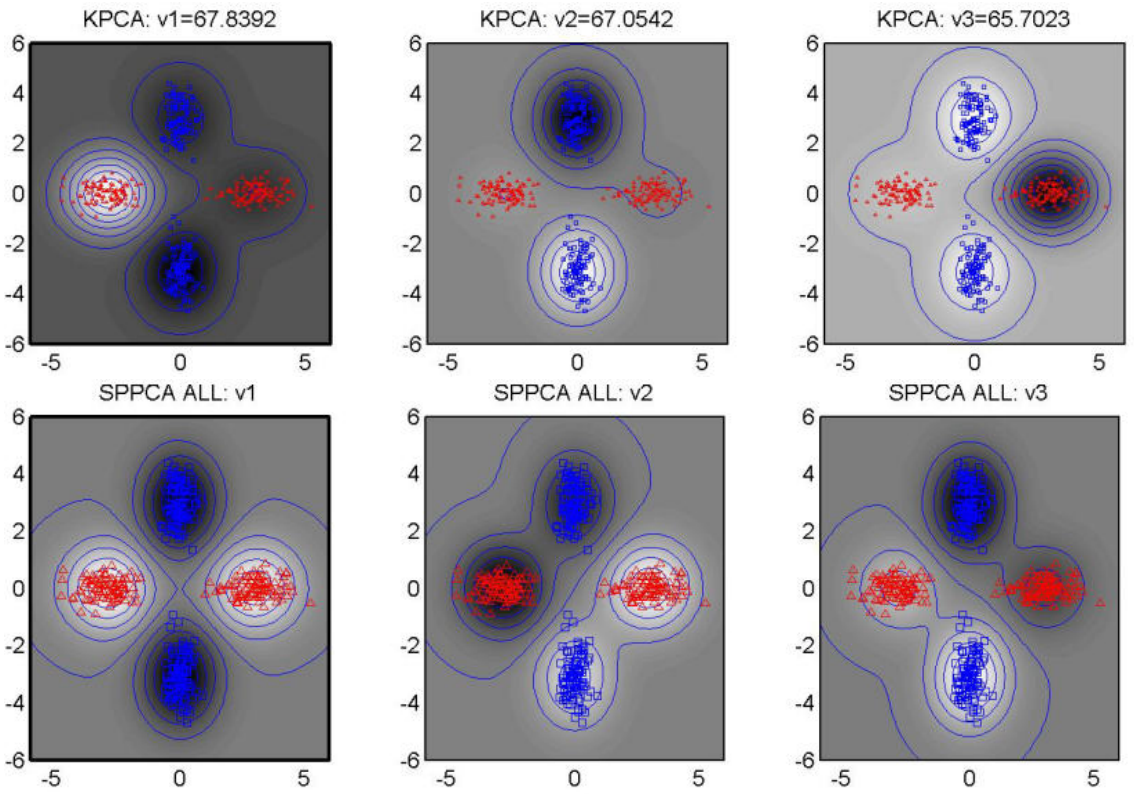


Figure 7.3: Illustration of dual-form SPPCA on a 2D toy data of four clusters. RBF kernel is used with $\alpha = 0.5$. Upper row shows the results of kernel PCA, and bottom row shows the results of SPPCA with classes ($+1$ for red and -1 for blue) as outputs.

The second and third dimensions are related to the clustering structure of the data. In Figure 7.4, only 5 data points of each cluster are labeled, and they are at the higher absolute value side of each cluster. Based on these 20 data points, SPPCA can detect the two classes (upper left), but all the contours are circles and the projection completely ignores the structure of unlabeled data. By considering both labeled data and unlabeled data, S^{2}PPCA can not only detect the two classes (bottom left), but also model the clustering structure of the data (bottom middle and right). All the contours are not circles and are influenced by unlabeled data.

7.2.4 Computational Issues

In the primal form (i.e., Algorithm 1 and 2), the time complexity for both algorithms is $\mathcal{O}\big(m(M+L)NK\big)$, with m the number of iterations. It is linear in the number of data points N and the input dimension M. The space complexity is $\mathcal{O}\big((M+L)N\big)$, which is also linear both in N and M. The projection for a test data point is just a linear operation

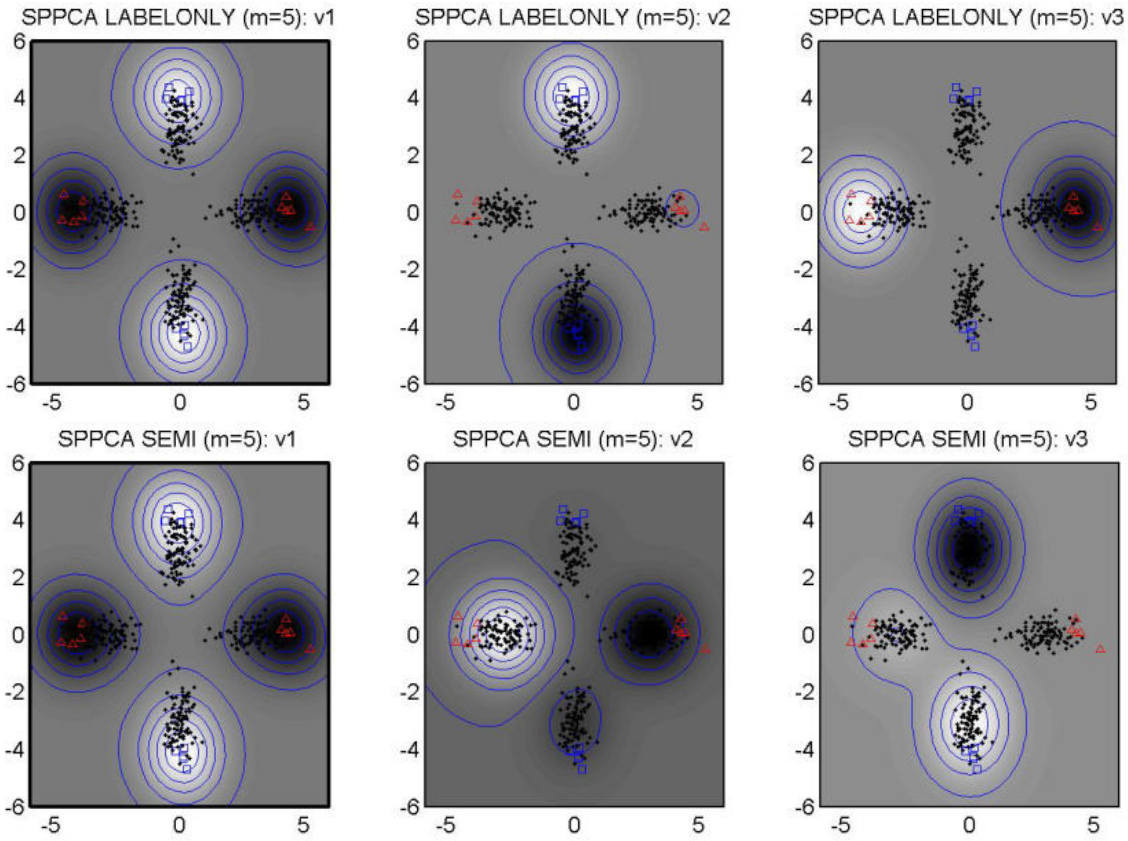


Figure 7.4: Illustration of dual-form SPPCA and S^{2}PPCA on a 2D toy data of four clusters. Only 5 data are labeled in each cluster. RBF kernel is used with $\alpha = 0.5$. Upper row shows the results of SPPCA with only labeled data, and bottom row shows the results of S^{2}PPCA using both labeled and unlabeled data.

and costs $\mathcal{O}(MK)$ time.

In the dual form, the time complexity is $\mathcal{O}(mN^2K)$ plus $\mathcal{O}(N^2M)$ which is the one-time calculation of Gram matrix, and the space complexity is $\mathcal{O}\big(N^2\big)$. Both of them are now quadratic in the number of data points N. The time for projecting a test data point is now $\mathcal{O}(NM)$. As the case for PCA, in situations where $M > N$, i.e., we have more features than the number of data points, the dual form is more efficient than the primal form.

7.3 Theoretical Justification

In this section we provide some theoretical analysis for SPPCA model and show how the supervised information influences the projection.

7.3.1 Primal Form Solution

Recall that matrix $\mathbf{\Phi}$ is a $(M+L)\times(M+L)$ diagonal matrix with all the noise levels in diagonal, i.e., $\mathbf{\Phi} = \mathtt{diag}(\sigma_x^2,\ldots,\sigma_x^2,\sigma_y^2,\ldots,\sigma_y^2)$. We obtain the following theorem for mapping matrix $\mathbf{W}_x$ and $\mathbf{W}_y$ in SPPCA model. This makes it easier to compare SPPCA with related models such as PCA.

Theorem 7.3.1. *Let* $\mathbf{S}$ *denote the normalized sample covariance matrix for centered observations* $\{(\mathbf{x}_i,\mathbf{y}_i)\}_{i=1}^N$,

$$\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{\Phi}^{-\frac{1}{2}}\begin{pmatrix}\mathbf{x}_i\\ \mathbf{y}_i\end{pmatrix}\begin{pmatrix}\mathbf{x}_i\\ \mathbf{y}_i\end{pmatrix}^{\top}\mathbf{\Phi}^{-\frac{1}{2}} = \begin{pmatrix}\frac{1}{\sigma_x^2}\mathbf{S}_x & \frac{1}{\sigma_x\sigma_y}\mathbf{S}_{xy}\\ \frac{1}{\sigma_x\sigma_y}\mathbf{S}_{yx} & \frac{1}{\sigma_y^2}\mathbf{S}_y\end{pmatrix},$$

and $\lambda_1 \geq \ldots \geq \lambda_{(M+L)}$ *be its eigenvalues with eigenvectors* $\mathbf{u}_1,\ldots,\mathbf{u}_{(M+L)}$, *then if the latent space is* K*-dimensional, the following results hold:*

(i) *In SPPCA model* $\mathbf{W}_x$ *and* $\mathbf{W}_y$ *are calculated as*

$$\begin{aligned}\mathbf{W}_x &= \sigma_x\mathbf{U}_x(\mathbf{\Lambda}_K-\mathbf{I})^{\frac{1}{2}}\mathbf{R},\\ \mathbf{W}_y &= \sigma_y\mathbf{U}_y(\mathbf{\Lambda}_K-\mathbf{I})^{\frac{1}{2}}\mathbf{R},\end{aligned} \tag{7.25}$$

where $\mathbf{\Lambda}_K = \mathtt{diag}(\lambda_1,\ldots,\lambda_K)$, $\mathbf{U}_x$ $(\mathbf{U}_y)$ *contains the first* M *(last* L*) rows of* $[\mathbf{u}_1,\ldots,\mathbf{u}_K]$, *and* $\mathbf{R}$ *is an arbitrary* $K\times K$ *orthogonal rotation matrix.*

(ii) *Projection* $\mathbf{z}_*$ *for centered new input* $\mathbf{x}_*$ *is given as*

$$\mathbf{z}_* = \frac{1}{\sigma_x}\mathbf{R}^{\top}(\mathbf{\Lambda}_K-\mathbf{I})^{-\frac{1}{2}}\left[\mathbf{U}_x^{\top}\mathbf{U}_x+(\mathbf{\Lambda}_K-\mathbf{I})^{-1}\right]^{-1}\mathbf{U}_x^{\top}\mathbf{x}_*.$$

Proof. The proof is similar to the PPCA solution given in [70] and we only give a sketch here. The mapping matrices can be obtained by finding a stationary point in the EM algorithm in Section 7.2. For this proof we use notation

$$\mathbf{W} := \begin{pmatrix}\frac{1}{\sigma_x}\mathbf{W}_x\\ \frac{1}{\sigma_y}\mathbf{W}_y\end{pmatrix},$$

and (7.3) can be rewritten as $\mathbf{A} = \mathbf{W}^{\top}\mathbf{W}+\mathbf{I}$. Plugging this and (7.5), (7.6) into (7.8) yields an update equation only related to $\mathbf{W}$:

$$\hat{\mathbf{W}} = \mathbf{S}\mathbf{W}\left[\mathbf{W}^{\top}\mathbf{S}\mathbf{W}+\mathbf{W}^{\top}\mathbf{W}+\mathbf{I}\right]^{-1}\left(\mathbf{W}^{\top}\mathbf{W}+\mathbf{I}\right).$$

At the stationary point, this simplifies to $\mathbf{SW} = \mathbf{WW}^\top\mathbf{W} + \mathbf{W}$. Let $\mathbf{W} = \mathbf{UDV}^\top$ be the SVD of $\mathbf{W}$. Then each column $\mathbf{u}$ of $\mathbf{U}$ satisfies $d\mathbf{Su} = (d + d^3)\mathbf{u}$, with d the corresponding singular value. Following the same discussions in Appendix A.1∼A.4 of [70], we conclude that solving an eigenvalue problem for $\mathbf{S}$ gives the mapping matrix $\mathbf{W}$ as in (i), where $\mathbf{u}$'s and d's are from the leading eigenvectors and eigenvalues of $\mathbf{S}$. Plugging them into (7.4) gives the mapping $\mathbf{z}_*$ for $\mathbf{x}_*$ as in (ii). □

In the special case that $L = 0$, the model is unsupervised and $\mathbf{S} = \frac{1}{\sigma_x^2}\mathbf{S}_x$ holds. Then (7.25) degrades to $\sigma_x\mathbf{U}_x(\mathbf{\Lambda}_K - \mathbf{I})^{\frac{1}{2}}\mathbf{R}$, which recovers the PPCA solution. $\mathbf{U}_x$ is seen to be column orthogonal in this case, and the mapping $\mathbf{z}_*$ of $\mathbf{x}_*$ is standard PCA mapping when $\sigma_x^2 \to 0$ and $\mathbf{R} = \mathbf{I}$. This proves Proposition 5.1.1 which is a corollary of this theorem.

When $L > 0$, SPPCA solutions explain not only the sample covariance of inputs $\mathbf{S}_x$, but also the *intra-covariance* of outputs $\mathbf{S}_y$ (if $L > 1$) and the *inter-correlations* between inputs and outputs, $\mathbf{S}_{xy}$ and $\mathbf{S}_{yx}$. Therefore, one column of $\mathbf{W}_x$ is the direction that best explains the whole system from the perspective of inputs, and thus are *biased* by the outputs. Unlike the case of PCA, the learned $\mathbf{W}_x$ in SPPCA needs not to be column orthogonal. This means we are only learning an *affine* mapping for $\mathbf{x}$. If necessary, it is straightforward to find the orthogonal basis by performing SVD to matrix $\mathbf{W}_x$.

In both cases the learned $\mathbf{W}_x$ has an arbitrary rotation factor $\mathbf{R}$. This is due to the spherical noise model for $\mathbf{x}$, and the mapping is invariant under a rotation of latent space, as can be seen from the equation for $\mathbf{z}_*$. Therefore the SPPCA model can only find the *latent principal subspace*, which has also been discussed in Chapter 5 and [70] for PPCA. Thus the EM algorithm in Section 7.2 can find different mappings with different initializations, but they define the same subspace and do not change the structure of projected data. If necessary, this ambiguity can be removed by eigen-decomposing

$$\frac{1}{\sigma_x^2}\mathbf{W}_x^\top\mathbf{W}_x + \frac{1}{\sigma_y^2}\mathbf{W}_y^\top\mathbf{W}_y = \mathbf{R}^\top(\mathbf{\Lambda}_K - \mathbf{I})\mathbf{R}$$

to uncover the rotation factor $\mathbf{R}$. A final comment is that it is of theoretical importance but may be not applicable to applications, since we have to solve an eigenvalue problem for a square matrix of size $M + L$.

7.3.2 Dual Form Solution

In the dual form, we do not obtain the mapping matrix $\mathbf{W}_x$, but the projected vectors directly. The following theorem gives the solution in the dual form.

Theorem 7.3.2. *Let* $\widehat{\mathbf{K}} = \frac{1}{\sigma_x^2}\mathbf{K} + \frac{1}{\sigma_y^2}\mathbf{YY}^\top$, *and* $\lambda_1 \geq \ldots \geq \lambda_N$ *be its eigenvalues with eigenvectors* $\mathbf{v}_1, \ldots, \mathbf{v}_N$, *then if the latent space in SPPCA is* K*-dimensional,*

(i) *The projections of training data, which are encoded in rows of matrix* $\mathbf{Z}$, *are calculated as*

$$\mathbf{Z} = \sqrt{N}\mathbf{V}_K(\mathbf{I} - N\mathbf{\Lambda}_K^{-1})^{\frac{1}{2}}\mathbf{R} \tag{7.26}$$

where $\mathbf{\Lambda}_K = \mathtt{diag}(\lambda_1, \ldots, \lambda_K)$, $\mathbf{V}_K = [\mathbf{v}_1, \ldots, \mathbf{v}_K]$, *and* $\mathbf{R}$ *is an arbitrary* $K \times K$ *orthogonal matrix.*

(ii) *Projections* $\mathbf{z}_*$ *for new input* $\mathbf{x}_*$ *is given as*

$$\mathbf{z}_* = \sqrt{N}\mathbf{R}^\top \mathbf{D}^{-\frac{1}{2}} \left(\mathbf{V}_K^\top \mathbf{K} \mathbf{V}_K + \mathbf{D}\right)^{-1} \mathbf{V}_K^\top \mathbf{k}(\mathbf{X}, \mathbf{x}_*),$$

where $\mathbf{D} = \mathbf{I} - N\mathbf{\Lambda}_K^{-1}$.

Proof. We define $\mathbf{B} = \mathbf{Z}\mathbf{C}^{-1}$ and rewrite (7.19) and (7.20) using only $\mathbf{B}$. This leads to

$$N\mathbf{B}\left(\mathbf{I} + \mathbf{B}^\top \widehat{\mathbf{K}} \mathbf{B}\right) = \widehat{\mathbf{K}}\mathbf{B},$$

which is the same problem as we obtained in the proof of Theorem 5.2.1 except that matrix $\mathbf{K}$ is changed to $\widehat{\mathbf{K}}$. Then following that proof leads to the projections for training data as in (i), and (ii) holds if we apply the obtained $\mathbf{Z}$ and $\mathbf{C}$ to the dual-form projection equation. □

It is seen that when there is no output in SPPCA, i.e., $\widehat{\mathbf{K}} = \frac{1}{\sigma_x^2}\mathbf{K}$, SPPCA reduces to kernel PCA as desired. It recovers Theorem 5.2.1 in Chapter 5. This theorem also presents a nice explanation for SPPCA model: We just use the output information to modify the kernel matrix of input data, and control the trade-off via the ratio of the noise levels. The complexity of the model remains the same (i.e., quadratic in N) no matter how many output dimensions we have.

7.3.3 Discussion

Previous two subsections give some theoretical results for SPPCA model. There exists however no such a closed-form solution for S^{2}PPCA. One can only empirically analyze the behavior of this model.

In the EM learning algorithm we are learning the maximum likelihood (ML) estimate for the two mapping matrices $\mathbf{W}_x$ and $\mathbf{W}_y$. In probabilistic modeling we can assign a prior to them to reduce overfitting. For instance, we can assign an isotropic Gaussian prior for each column of $\mathbf{W}_x$, and if we consider the *maximum a posteriori* (MAP) estimate this prior corresponds to a smooth term in the update equations. For simplicity we do not consider this prior here.

In this paper we mainly discuss the simplest form for function $\mathbf{f}$, which is linear in $\mathbf{z}$. One can define other forms for specific tasks, but then we lose the nice closed-form solutions described in Theorem 7.3.1 and 7.3.2, and in the E-step of the EM learning the posterior distribution of $\mathbf{z}$ is no longer a Gaussian (see (7.2)). To solve this problem we can apply the EM-EP learning algorithm [49] to approximate each likelihood $P(y_\ell|\mathbf{z}) = P(y_\ell|f_\ell(\mathbf{z}, \mathbf{\Theta}), \sigma_y^2)$ as a Gaussian for $\mathbf{z}$ sequentially. Then the learning has higher time complexity due to the expectation-propagation (EP) [54] step. Empirically comparing this algorithm with the basic ones would be part of the future work.

Table 7.1: Statistics of the multi-class data sets

	Category	# Data	# Dim	# Class
Yale	Face	165	1024	15
ORL	Face	400	1024	40
PIE	Face	11554	1024	68
YaleB	Face	2414	1024	38
11_Tumors	Gene	174	12533	11
14_Tumors	Gene	308	15009	26
Lung_Cancer	Gene	203	12600	5
20Newsgroup	Text	19928	25284	20
TDT2	Text	8692	35452	20

Table 7.2: Statistics of the multi-label data sets

	Category	# Data	# Dim	# Class
Yeast	Gene	2417	103	14
RCV1	Text	23149	15500	103

7.4 Empirical Studies

In this section we empirically investigate the performance of SPPCA models. The supervised tasks here are multi-class classification and multi-label classification. Our basic setting is that we train a supervised projection model using the input features and label information, and then test the classification performance for test data using the projected features. Since the test data are assumed known in the training phase, for S^{2}PPCA we will be able to use these unlabeled data to train the mapping.

7.4.1 Data Sets

We test the proposed model on 9 multi-class and 2 multi-label classification problems. These problems include face recognition, gene classification and text categorization. Some statistics of these data sets are shown in Table 7.1 and 7.2.

For face recognition we use four data sets Yale, ORL, PIE and YaleB (the extended Yale Face Database B).[2] The Yale data set contains 165 grayscale images in GIF format of 15 individuals. There are 11 images per subject, one per different facial expression or configuration such as center-light, left-light, happy or surprised. The ORL database contains 10 different images of each of 40 distinct subjects. For some subjects, the images were taken at different times with varying lighting and facial details. The PIE databases we use contains 170 images for each of 68 people. These images are the five near frontal poses under different illuminations and expressions. For YaleB we have 38 individuals and around 64 near frontal images under different illuminations per individual. All the faces

[2]These data sets can be downloaded from `http://www.ews.uiuc.edu/ dengcai2/Data/data.html`.

Table 7.3: Results for Multi-class Classification Tasks. Bold face indicates lowest error rate. Symbols ⋆ indicate that the best method is significantly better than the competitors (p-value 0.01 in Wilcoxon rank sum test).

Task	Full	PCA	LDA	PLS	SPPCA	S^{2}PPCA
Yale	0.5656 ± 0.0394	0.6690 ± 0.0333	$\mathbf{0.6133 \pm 0.0471}$	0.6440 ± 0.0383	0.7007 ± 0.0402	0.7121 ± 0.0393
ORL	0.3308 ± 0.0347	0.5593 ± 0.0263	0.5302 ± 0.0444	0.5505 ± 0.0294	0.5459 ± 0.0305	$\mathbf{0.5287 \pm 0.0286}$
PIE	0.6988 ± 0.0085	0.9325 ± 0.0032	$\mathbf{0.7066 \pm 0.0177}$	0.8781 ± 0.0058	0.8780 ± 0.0116	0.8452 ± 0.0037
YaleB	0.6360 ± 0.0160	0.9895 ± 0.0023	$\star\mathbf{0.5328 \pm 0.0251}$	0.9546 ± 0.0066	0.9701 ± 0.0088	0.9800 ± 0.0034
11_Tumors	0.3161 ± 0.0566	0.5409 ± 0.0490	$\mathbf{0.4505 \pm 0.0755}$	N/A	0.5226 ± 0.0636	0.5130 ± 0.0491
14_Tumors	0.6084 ± 0.0360	0.7363 ± 0.0286	0.7161 ± 0.0481	N/A	0.7312 ± 0.0371	$\mathbf{0.7138 \pm 0.0296}$
Lung_Cancer	0.3680 ± 0.1148	0.3768 ± 0.0939	$\mathbf{0.3225 \pm 0.1658}$	N/A	0.4287 ± 0.1338	0.3896 ± 0.0923
20Newsgroup	0.6135 ± 0.0155	0.9070 ± 0.0177	0.9140 ± 0.0208	N/A	$\mathbf{0.9030 \pm 0.0162}$	0.9126 ± 0.0116
TDT2	0.1875 ± 0.0233	0.6664 ± 0.0657	0.7834 ± 0.0782	N/A	0.6236 ± 0.0739	$\star\mathbf{0.3686 \pm 0.0349}$

(a) Projection dimension $K = 5$.

Task	Full	PCA	LDA	PLS	SPPCA	S^{2}PPCA
Yale	0.5656 ± 0.0394	0.5993 ± 0.0312	$\mathbf{0.5279 \pm 0.0460}$	0.5698 ± 0.0386	0.6101 ± 0.0447	0.5916 ± 0.0433
ORL	0.3308 ± 0.0347	0.4049 ± 0.0293	0.3625 ± 0.0468	0.4048 ± 0.0349	0.3832 ± 0.0409	$\mathbf{0.3509 \pm 0.0287}$
PIE	0.6988 ± 0.0085	0.8573 ± 0.0051	$\mathbf{0.5496 \pm 0.0185}$	0.8062 ± 0.0068	0.7105 ± 0.0161	0.6942 ± 0.0047
YaleB	0.6360 ± 0.0160	0.9308 ± 0.0046	$\star\mathbf{0.3846 \pm 0.0282}$	0.8762 ± 0.0108	0.7976 ± 0.0242	0.7986 ± 0.0117
11_Tumors	0.3161 ± 0.0566	0.3682 ± 0.0655	0.3926 ± 0.0667	N/A	0.3801 ± 0.0624	$\star\mathbf{0.3297 \pm 0.0664}$
14_Tumors	0.6084 ± 0.0360	0.6868 ± 0.0288	0.6212 ± 0.0430	N/A	0.6322 ± 0.0363	$\mathbf{0.6120 \pm 0.0331}$
Lung_Cancer	0.3680 ± 0.1148	0.3493 ± 0.0996	$\mathbf{0.3225 \pm 0.1658}$	N/A	0.6235 ± 0.1520	0.3517 ± 0.1063
20Newsgroup	0.6135 ± 0.0155	0.9039 ± 0.0172	0.8943 ± 0.0292	N/A	0.8931 ± 0.0242	$\mathbf{0.8548 \pm 0.0138}$
TDT2	0.1875 ± 0.0233	0.5531 ± 0.0742	0.6878 ± 0.1068	N/A	0.5346 ± 0.0885	$\star\mathbf{0.2794 \pm 0.0327}$

(b) Projection dimension $K = 10$.

Task	Full	PCA	LDA	PLS	SPPCA	S^{2}PPCA
Yale	0.5656 ± 0.0394	0.5437 ± 0.0414	0.5793 ± 0.0438	0.5216 ± 0.0435	0.5093 ± 0.0391	$\mathbf{0.5001 \pm 0.0589}$
ORL	0.3308 ± 0.0347	0.3323 ± 0.0310	0.2944 ± 0.0398	0.3366 ± 0.0331	0.3271 ± 0.0372	$\mathbf{0.2755 \pm 0.0286}$
PIE	0.6988 ± 0.0085	0.7999 ± 0.0060	$\mathbf{0.4352 \pm 0.0186}$	0.7454 ± 0.0092	0.5912 ± 0.0146	0.5361 ± 0.0090
YaleB	0.6360 ± 0.0160	0.8304 ± 0.0096	$\star\mathbf{0.3004 \pm 0.0227}$	0.7695 ± 0.0148	0.5619 ± 0.0276	0.5652 ± 0.0172
11_Tumors	0.3161 ± 0.0566	0.3267 ± 0.0635	0.3926 ± 0.0667	N/A	0.4470 ± 0.0691	$\mathbf{0.3012 \pm 0.0582}$
14_Tumors	0.6084 ± 0.0360	0.6379 ± 0.0360	0.5822 ± 0.0388	N/A	$\mathbf{0.5669 \pm 0.0347}$	0.5674 ± 0.0372
Lung_Cancer	0.3680 ± 0.1148	0.3584 ± 0.0953	$\mathbf{0.3225 \pm 0.1658}$	N/A	0.6487 ± 0.1540	0.4092 ± 0.1107
20Newsgroup	0.6135 ± 0.0155	0.9160 ± 0.0220	0.8001 ± 0.0425	N/A	$\mathbf{0.6254 \pm 0.0420}$	0.6568 ± 0.0146
TDT2	0.1875 ± 0.0233	0.4582 ± 0.1441	0.1524 ± 0.0622	N/A	0.1566 ± 0.0509	$\mathbf{0.1520 \pm 0.0210}$

(c) Projection dimension $K = 20$.

are manually aligned, cropped and resized to 32×32 pixels. We then normalize each image to have Euclidean distance 1.

We consider three gene expression datasets 11_Tumors, 14_Tumors and Lung_Cancer for gene classification.[3] 11_Tumors describes 11 various human tumor types, and 14_Tumors describes 14 tumor types with 12 normal tissue types. For Lung_Cancer we need to classify 4 lung cancer types and normal tissues. The characteristic of these data is that the number of data points is small, but the input dimensionality is very high.

The two textual datasets we use are taken from 20Newsgroup and TDT2. 20Newsgroup contains 20,000 news articles posted in 20 news groups. We remove the words that occur less than 5 times, and obtain 19,928 documents with 25,284 words. The TDT2 corpus we use consists of the documents collected during the first half of 1998 and taken from 6 sources, including 2 newswires (APW, NYT), 2 radio programs (VOA, PRI) and 2 television programs (CNN, ABC). It consists of 11,021 documents which are classified into 96 semantic categories. In our experiments, we keep the largest 20 categories and remove those documents that are assigned to more than one categories. This leaves us 8,692 documents with totally 35,452 words. For both of these datasets we use TF-IDF features and normalize each document to have Euclidean distance 1.

For multi-label classification we use Yeast and RCV1. The Yeast dataset is formed by micro-array expression data and phylogenetic profiles with 2,417 genes in total and 103 input dimensions. There are 14 groups and each gene can belong to multiple groups. The other data is a subset of the RCV1-v2 text data set, provided by Reuters and corrected by Lewis et al. [52]. We use the training set provided by Lewis, which contains 103 labels, 23,149 documents with 15,500 words after we remove words that occur less than 5 times. We also extract TF-IDF features and normalize each document to have length 1.

7.4.2 Experimental Setting

For the multi-class classification tasks, we randomly pick up a small number of labeled data points for training (2 for those datasets with less than 500 data points, and 5 for the others), and test the classification error rate on the unlabeled data. We will in general compare the following six algorithms if applicable:

- PCA: Unsupervised projection. Note that in our experiments we use both the labeled and unlabeled data to perform the mapping.
- LDA: Linear discriminant analysis.
- PLS: Partial least squares.
- SPPCA: Supervised probabilistic PCA.
- S^{2}PPCA: Semi-supervised probabilistic PCA. We allow S^{2}PPCA to use the test data to train the mapping.

[3] They are available at `http://www.gems-system.org`.

- FULL: All the features are used without projection.

For all the projection methods, we project the data onto 5, 10 and 20 dimensional space, and train a nearest neighbor classifier for the test points using new features with Euclidean distance. For FULL we directly train the nearest neighbor classifier using original features. For PLS, SPPCA and S^{2}PPCA, we translate the one column output to the "One of C" setting, i.e., each class has one column with binary labels.

For multi-label classification, we pick up 5 positive examples from each label to obtain the training data. For all projection methods we project to 5, 10 and 20 dimensions, and then train a linear SVM classifier for each label. The comparison metrics are F1-Macro, F1-Micro and AUC (Area Under ROC Curve) score. The candidate algorithms are almost the same as multi-class setting, except LDA which is not applicable to this task. The C in SVM is fixed as 100, and from our experience it is not sensible for all these algorithms.

In all these comparisons, the iteration number for SPPCA and S^{2}PPCA is set to 1000. Both the noise levels σ_x^2 and σ_y^2 are set to 10^{-5} initially. It turns out that PLS gets memory problems when applied to large dimensions. We repeat each experiments 50 times independently,[4] and the results are illustrated in .

7.4.3 Analysis of Results

The first observation is that in most cases the supervised PCA model is better than unsupervised PCA model. This means by using the output information, we are able to derive a more meaningful projection for the supervised tasks. When the dimensionality is larger (e.g., 20), SPPCA and S^{2}PPCA obtain the best results for most of the tasks.

When we compare SPPCA model with other supervised projection methods, SPPCA is consistently better than PLS, but in some tasks worse than LDA (e.g., for YaleB). The reasons may be that SPPCA models are still based on the PCA assumptions for input features, so the mapping is strongly biased by PCA. When PCA projection directions are almost useless for classification, like for YaleB dataset, the SPPCA projections are also not informative enough. In this case discriminative methods like LDA can often do a good job. In other situations where PCA does help, SPPCA can in general be better than pure discriminative methods.

When we compare SPPCA and S^{2}PPCA, in most cases S^{2}PPCA gets better results. For some tasks the difference is very big (e.g., for TDT2). This indicates that by incorporating the unlabeled data we can learn a better mapping. But S^{2}PPCA is in general slower than SPPCA because it has to consider all the test data in the training phase. In this case 1000 iterations may be not enough to get the algorithm converge. This may also be part of the reason why S^{2}PPCA is inferior to other methods for some tasks like PIE.

Most of the supervised projection methods can get a better performance than FULL even if they only project the data into a very low dimensional space. This is important

[4]For the four face recognition tasks we use the available split versions.

Table 7.4: Results for Multi-label Classification Tasks. Bold face indicates best performance.

K	Model	F1-Macro	F1-Micro	AUC
5	Full	0.3813 ± 0.0102	0.5161 ± 0.0154	0.5571 ± 0.0094
	PCA	0.2318 ± 0.0354	0.5600 ± 0.0220	0.5279 ± 0.0108
	PLS	0.3432 ± 0.0231	0.5795 ± 0.0233	0.5556 ± 0.0094
	SPPCA	0.3823 ± 0.0120	0.5332 ± 0.0188	0.5641 ± 0.0087
	S^2PPCA	$\mathbf{0.3927 \pm 0.0134}$	$\mathbf{0.5890 \pm 0.0126}$	$\mathbf{0.5842 \pm 0.0104}$
10	Full	0.3813 ± 0.0102	0.5161 ± 0.0154	0.5571 ± 0.0094
	PCA	0.3113 ± 0.0304	$\mathbf{0.5916 \pm 0.0146}$	0.5493 ± 0.0101
	PLS	0.3756 ± 0.0154	0.5517 ± 0.0177	0.5610 ± 0.0095
	SPPCA	0.3924 ± 0.0117	0.5459 ± 0.0180	0.5685 ± 0.0084
	S^2PPCA	$\mathbf{0.3985 \pm 0.0103}$	0.5914 ± 0.0106	$\mathbf{0.5896 \pm 0.0107}$
20	Full	0.3813 ± 0.0102	0.5161 ± 0.0154	0.5571 ± 0.0094
	PCA	0.3723 ± 0.0171	0.5537 ± 0.0204	0.5614 ± 0.0097
	PLS	0.3799 ± 0.0123	0.5208 ± 0.0158	0.5585 ± 0.0102
	SPPCA	0.3859 ± 0.0133	0.5517 ± 0.0151	0.5640 ± 0.0097
	S^2PPCA	$\mathbf{0.3976 \pm 0.0142}$	$\mathbf{0.6012 \pm 0.0190}$	$\mathbf{0.5921 \pm 0.0119}$

(a) Results for Yeast data set.

K	Model	F1-Macro	F1-Micro	AUC
5	Full	0.2796 ± 0.0055	0.5053 ± 0.0095	0.6030 ± 0.0063
	PCA	0.0467 ± 0.0063	0.3540 ± 0.0106	0.5208 ± 0.0072
	PLS	N/A	N/A	N/A
	SPPCA	0.1155 ± 0.0071	0.4433 ± 0.0089	0.5568 ± 0.0079
	S^2PPCA	$\mathbf{0.1312 \pm 0.0118}$	$\mathbf{0.4620 \pm 0.0236}$	$\mathbf{0.5762 \pm 0.0162}$
10	Full	0.2796 ± 0.0055	0.5053 ± 0.0095	0.6030 ± 0.0063
	PCA	0.0843 ± 0.0124	0.4003 ± 0.0162	0.5347 ± 0.0072
	PLS	N/A	N/A	N/A
	SPPCA	0.1797 ± 0.0112	0.4474 ± 0.0124	0.5872 ± 0.0087
	S^2PPCA	$\mathbf{0.1956 \pm 0.0110}$	$\mathbf{0.4735 \pm 0.0198}$	$\mathbf{0.6012 \pm 0.0098}$
20	Full	0.2796 ± 0.0055	0.5053 ± 0.0095	0.6030 ± 0.0063
	PCA	0.1320 ± 0.0086	0.4419 ± 0.0095	0.5504 ± 0.0061
	PLS	N/A	N/A	N/A
	SPPCA	0.2297 ± 0.0119	0.4690 ± 0.0126	0.6044 ± 0.0054
	S^2PPCA	$\mathbf{0.2536 \pm 0.0117}$	$\mathbf{0.4921 \pm 0.0102}$	$\mathbf{0.6090 \pm 0.0076}$

(b) Results for RCV1 data set.

because we can not only speed up the system, but also improve the performance. PCA in our experiments uses the input features of both labeled and unlabeled data, thus it sometimes can get better results than Full method (e.g., for Yale and Lung_Cancer).

The results for multi-label classification show that S^2PPCA is consistently better than other methods. S^2PPCA also shows very good scalability in our experiments, since for 20Newsgroup and RCV1 it need to handle 20,000 documents with more than 15,000 fea-

tures. Most of the other algorithms fail on these large datasets.

7.5 Summary

In this chapter we proposed a supervised PCA algorithm and a semi-supervised PCA algorithm, and derived an efficient EM algorithm for model learning. Empirical results show that the proposed model obtain good performance and scale well for large data sets.

Part III

Probabilistic Clustering-Projection Models

Chapter 8

Overview of Joint Clustering-Projection Models

Part I and Part II discuss data clustering and feature projection, respectively. For clustering we aim to group similar data points together, and for projection we want to find an informative low dimensional feature representation for the data. They are seen to model the data from different perspectives, and most research work only focuses on one of them. But can we model both of them simultaneously and achieve a joint clustering-projection model?

The motivation of this idea is two-fold. If we have already a good projection model, i.e., the data are now distributed in an informative low dimensional space, a clustering model is more likely to obtain better results. This is because we can remove the noisy dimensions and decrease the complexity of learning in mixture modeling, since mixture modeling for lower dimensional data is more stable. On the other hand, if a clustering of all the data points is known, we can also incorporate the grouping information into the projection model and treat each cluster as a whole instead of considering each data point individually. This extra information is likely to improve the learned feature mapping.

In real-world applications, many people take a projection model such as PCA as a pre-processing step, and train a clustering model on the reduced space. Since the clusters of data contain additional information, it is desired to use this information to train a better projection model. If we continue this process, we actually have an iterative way of adapting a projection model and a clustering model, and this algorithm can be used to train a so-called *joint clustering-projection model*. At convergence, both clustering and projection operations should not change the current status.

In the literature there exists some work on this direction, and we briefly review them in this chapter. As projection can mean different things for continuous data and discrete data, for this review we also explicitly separate these two types of data.

8.1 Joint Clustering-Projection Models for Continuous Data

In [20] the authors consider to use PCA as the projection model and train a k-means clustering for the data in the reduced space. After the K centroids are obtained from the data-cluster relationship, a subspace which spans these centroids is chosen to learn a new projection model. The whole process is repeated until a stable situation is obtained. As discussed in Chapter 2 and Chapter 5, k-means is training a simplified version of Gaussian mixture models, and PCA is the deterministic solution for a PPCA model. Taking this into account, we can generalize the joint model in [20] to a pure probabilistic model which assumes a GMM in the latent variable space of a PPCA model. More flexible covariance matrices can be learned in this new model, and we can also allow the number of mixture components to go to infinity to approximate the non-parametric DP prior. The iterative updates can be obtained directly from an EM learning algorithm for this model.

Note that this joint clustering-projection model is not the same as mixture of PPCA [69], in which several PPCA components exist and model different parts of the data. The mixture of PPCA model assumes that each PPCA has its own projection model, and one data point belongs to one component if this PPCA component models its local projection very well. The joint clustering-projection model, however, assumes a *global* projection model for all the data.

8.2 Joint Clustering-Projection Models for Discrete Data

As reviewed in Chapter 4, projection on discrete data differs from the case on continuous space, where, for example, the most popular technology PCA tries to find the orthogonal dimensions (or factors) that explains the *covariance* of data dimensions. However, one cannot make the same orthogonal assumption on the low-dimensional factors of discrete data and put the interests on the covariance anymore. Instead, it is desired to find the *independent* latent factors that explain the *co-occurrence* of dimensions (e.g., words). In text modeling, if we refer the factors as topics, the projections actually represent each document as a data point in a low-dimensional topic space, where a co-occurrence factor actually suggests more or less a cluster of words (i.e., a group of words often occurring together). Intuitively, if the projected topic space is informative enough, it should also be highly indicative to reveal the clustering structure of documents. On the other hand, a truly discovered clustering structure reflects the shared topics within document clusters and the distinguished topics across document clusters, and thus can offer evidence for the projection side. Therefore, it is highly desired to consider the two problems in a unified model.

From a probabilistic point of view, a Multinomial distribution is suitable to describe the data-cluster memberships. As discussed in Chapter 4, the projection of words should also be discrete and takes a Multinomial distribution. Buntine et al. [12] viewed the projection for discrete data as a discrete PCA problem. They also noticed its relationship

with clustering and pointed out that the Multinomial PCA model takes clustering and projection as two extreme cases [13]. Another closely related work is the so-called two-sided clustering, like [40] and [19], which aims to cluster words and documents simultaneously. In [19] it is implicitly assumed a one-to-one correspondence between the two sides of clusters. [40] is a probabilistic model for discrete data, but it has similar problems as in pLSI and not generalizable to new documents.

Chapter 9

The PCP Model for Discrete Data

As discussed in Part I and Part II, both data clustering and feature projection are important tasks and have been extensively studied in data mining and machine learning communities. However, as discussed in Chapter 8, it is ideal to consider both of them in a unified framework and improve the performance of both tasks.

In this chapter we propose a novel probabilistic clustering-projection (PCP) model to jointly handle the projection and clustering for discrete data.[1] The projection of words is explicitly formulated with a matrix of model parameters. Document clustering is then incorporated using a mixture model on the projected space, and we model each mixture component as a Multinomial over the latent topics. In this sense this is a *clustering model using projected features* for documents if the projection matrix is given, and a *projection model with structured data* for words if the clustering structure is known. A nice property of the model is that we can perform clustering and projection *iteratively*, incorporating new information on one side to the updating of the other. We will show that they are corresponding to a variational Bayesian algorithm that improves the data likelihood iteratively. It is a difficult problem to do model selection in clustering analysis, i.e., to choose the correct number of clusters. We will point out the connection to the Dirichlet process prior discussed in Chapter 3, and thus the model inherits the nice property of automatically determining the number of clusters.

This chapter is organized as follows. Section 9.1 introduces the PCP model and explicitly points out the clustering and projection effects. In Section 9.2 we present inference and learning algorithm. Then Section 9.3 presents experimental results and Section 9.4 summarizes the model.

[1] This chapter is based on [84] and [80].

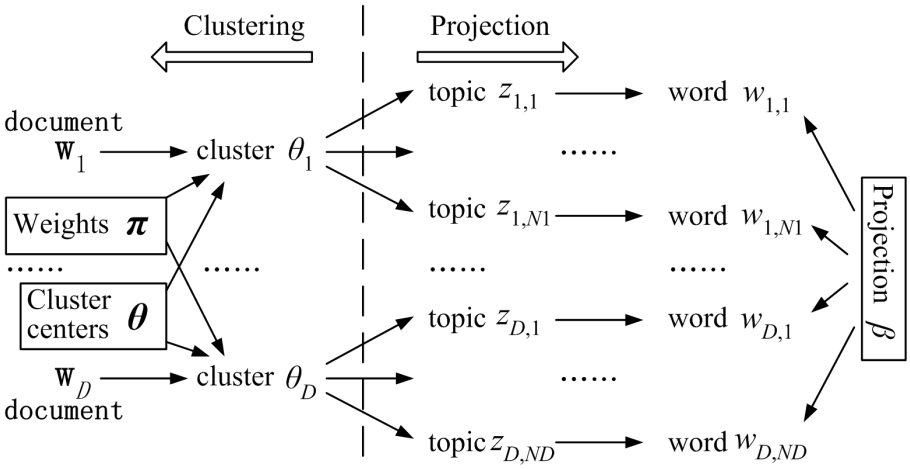


Figure 9.1: Informal sampling process for the PCP model. Dark arrows show dependencies between entities and the dashed line separates the clustering and projection effects. Model parameters α and $\boldsymbol{\lambda}$ are not shown for simplicity.

9.1 The PCP Model

We use some new notations in this chapter. For document modeling we consider a corpus $\mathcal{D}$ containing D documents, with vocabulary $\mathcal{V}$ having V words. Following the notation in [10], each document d is a sequence of N_d words that is denoted by $\mathbf{w}_d = \{w_{d,1}, \ldots, w_{d,N_d}\}$, where $w_{d,n}$ is a variable for the n-th word in $\mathbf{w}_d$ and denotes the index of the corresponding word in $\mathcal{V}$. Note that a same word may occur several times in the sequence $\mathbf{w}_d$.

To simplify explanations, we use "clusters" for components in document clustering structure and "topics" for projected space for words. Let M denote the number of clusters and K the dimensionality of topics. Roman letters d, m, k, n, j are indices for documents, clusters, topics, words in $\mathbf{w}_d$, and words in $\mathcal{V}$. They are up to D, M, K, N_d, V, respectively. Letter i is reserved for temporary index.

9.1.1 The Probabilistic Model

The PCP model is a generative model for a document corpus. Figure 9.1 illustrates the sampling process in an informal way. To generate one document d, we first choose a cluster from the M clusters. For the m-th cluster, the cluster center is denoted as $\boldsymbol{\theta}_m$ and defines a topic mixture over the topic space. Therefore $\boldsymbol{\theta}_m$ is a K-dimensional vector and satisfies $\theta_{m,k} \geq 0$, $\sum_{k=1}^{K} \theta_{m,k} = 1$ for all $m = 1, \ldots, M$. The probability of choosing a specific cluster m for document d is denoted as π_m, and $\boldsymbol{\pi} := \{\pi_1, \ldots, \pi_M\}$ satisfies $\pi_m \geq 0$, $\sum_{m=1}^{M} \pi_m = 1$.

When document d chooses cluster m, it defines a document-specific topic mixture $\boldsymbol{\theta}_d$, which is obtained exactly from the cluster center $\boldsymbol{\theta}_m$. Note that everything is discrete and

two documents belonging to the same cluster will have the same topic mixtures. Words are then sampled independently given topic mixture $\boldsymbol{\theta}_d$, in the same way as in LDA. Each word $w_{d,n}$ is generated by first choosing a topic $z_{d,n}$ given the topic mixture, and then sampling the word given the projection β. β is the $K \times V$ matrix where $\beta_{k,j}$ specifies the probability of generating word j given topic k, $\beta_{k,j} = P(w^j = 1|z^k = 1)$. Therefore each row $\beta_{k,:}$ defines a Multinomial distribution for all words over topic k and satisfies $\beta_{k,j} \geq 0$, $\sum_{j=1}^{V} \beta_{k,j} = 1$.

To complete the model, we put a Dirichlet prior $\texttt{Dir}(\boldsymbol{\lambda})$ for all the cluster centers $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M$, and a symmetric Dirichlet prior $\texttt{Dir}(\alpha/M, \ldots, \alpha/M)$ for the mixing weights $\boldsymbol{\pi}$. This prior is used similarly in Chapter 2 and will approach the Dirichlet process prior in the asymptotic case. Note that they are sampled only once for the whole corpus.

Finally we obtain the probabilistic model formally illustrated in Figure 9.2 (left), using standard plate model. c_d takes value $\{1, \ldots, M\}$ and acts as the indicator variable saying which cluster document d takes on out of the M clusters. All the model parameters are $\alpha, \boldsymbol{\lambda}$, β and amount to $1 + M + K \times (V - 1)$. The following procedure describes the sampling process for the whole corpus:

1. Choose model parameter $\alpha, \boldsymbol{\lambda}, \beta$;
2. For the m-th cluster, choose $\boldsymbol{\theta}_m \sim \texttt{Dir}(\boldsymbol{\lambda}), m = 1, \ldots, M$;
3. Choose the mixing weight $\boldsymbol{\pi} \sim \texttt{Dir}(\alpha/M, \ldots, \alpha/M)$;
4. For each document $\mathbf{w}_d$:
   (a) Choose a cluster m with mixing weights $\boldsymbol{\pi}$, and obtain $\boldsymbol{\theta}_d = \boldsymbol{\theta}_m$;
   (b) For each of the N_d words $w_{d,n}$:
      i. Choose a topic $z_{d,n} \sim \texttt{Mult}(\boldsymbol{\theta}_d)$;
      ii. Choose a word $w_{d,n} \sim \texttt{Mult}(\beta_{z_{d,n},:})$.

Denote $\boldsymbol{\Theta}$ as the set of M cluster centers $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M\}$, the likelihood of the corpus $\mathcal{D}$ can be written as

$$\mathcal{L}(\mathcal{D}|\alpha, \boldsymbol{\lambda}, \beta) = \int_{\boldsymbol{\pi}} \int_{\boldsymbol{\Theta}} \prod_{d=1}^{D} P(\mathbf{w}_d|\boldsymbol{\Theta}, \boldsymbol{\pi}, \beta) dP(\boldsymbol{\Theta}|\boldsymbol{\lambda})\, dP(\boldsymbol{\pi}|\alpha), \tag{9.1}$$

where $P(\boldsymbol{\Theta}|\boldsymbol{\lambda}) = \prod_{m=1}^{M} P(\boldsymbol{\theta}_m|\boldsymbol{\lambda})$, and the likelihood of document d is a mixture:

$$P(\mathbf{w}_d|\boldsymbol{\Theta}, \boldsymbol{\pi}, \beta) = \sum_{c_d=1}^{M} P(\mathbf{w}_d|\boldsymbol{\Theta}, c_d, \beta) P(c_d|\boldsymbol{\pi}). \tag{9.2}$$

Given mixture component c_d, likelihood term $P(\mathbf{w}_d|\boldsymbol{\Theta}, c_d, \beta)$ is then given by

$$P(\mathbf{w}_d|\boldsymbol{\theta}_{c_d}, \beta) = \prod_{n=1}^{N_d} \sum_{z_{d,n}=1}^{K} P(w_{d,n}|z_{d,n}, \beta) P(z_{d,n}|\boldsymbol{\theta}_{c_d}). \tag{9.3}$$

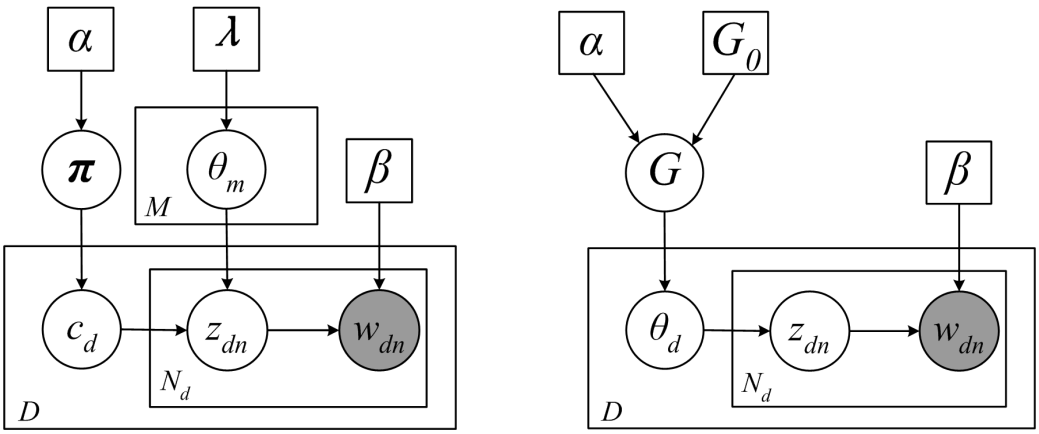


Figure 9.2: The plate models of PCP model with finite mixture prior (left) and Dirichlet process prior (right).

9.1.2 PCP as a Clustering Model

As can be seen from (9.2) and (9.3), PCP is a clustering model when the projection β is assumed known. The essential terms now are the probabilities of clusters $P(m|\boldsymbol{\pi}) = \pi_m$, probabilistic clustering assignment for documents $P(\mathbf{w}_d|\boldsymbol{\theta}_m, \beta)$, and cluster centers $\boldsymbol{\theta}_m$, for $m = 1, \ldots, M$. Note from (9.3) that cluster centers $\boldsymbol{\theta}_m$ are not modeled directly with words like $P(w|\boldsymbol{\theta}_m)$, but with topics, $P(z|\boldsymbol{\theta}_m)$. This means we are not clustering documents in word space, but in *topic space*. This is analogous to clustering continuous data on the latent space found by PCA [20], and K is exactly the dimensionality of this space. To obtain the probability that document d belongs to cluster m, we project each word into topic space, and then calculate the distance to cluster center $\boldsymbol{\theta}_m$ by considering all the words in $\mathbf{w}_d$. This explains (9.3) from perspective of clustering.

To improve generalization and avoid overfitting, we put priors to $\boldsymbol{\theta}_m$ and $\boldsymbol{\pi}$ and treat them as hidden variables, as usually done in mixture modeling. The prior distributions are chosen to be Dirichlet that is *conjugate* to Multinomial. This will make model inference and learning much easier (see Section 9.2).

9.1.3 PCP as a Projection Model

A projection model aims to learn projection β, mapping words to topics. As can be seen from (9.3), the topics are not modeled directly with documents $\mathbf{w}_d$, but with cluster centers $\boldsymbol{\theta}_m$. Therefore if clustering structure is already known, PCP will learn β by using the richer information contained in cluster centers, not just individual documents. In this sense, PCP can be explained as a *projection model with structured data* and is very attractive because clustered documents are supposed to contain less noise and coarser granularity. This will make the projection more accurate and faster.

As a projection model, PCP is more general than pLSI because document likelihood (9.3) is well defined and generalizable to new documents. Although LDA uses similar equation as (9.3), the topic mixture $\boldsymbol{\theta}_d$ is only sampled for current document and no inter-similarity of documents is directly modeled. Documents can only exchange information via the hyperparameter for $\boldsymbol{\theta}_d$'s, and thus its effect to β is only implicit. On the contrary, PCP directly models similarity of documents and incorporate all information to learn β.

As discussed in [10], projection β can be smoothed by putting a common prior to all the rows. If only the *maximum a posteriori* (MAP) estimate of β is considered, the effect of smoothing turns out to add a common factor to each entry of β before normalization each row. This is also straightforward in PCP model and we will not discuss it in detail for simplicity. In the experiments we will use this smoothing technique.

9.1.4 Connections to Dirichlet Processes

In this subsection we analyze the PCP model in the limiting case that M tends to infinity, similarly as the infinite mixture models in Chapter 3. When we fix M in our model, the topic mixture $\boldsymbol{\theta}_d$ for a specific document $\mathbf{w}_d$ is equivalently seen to sample from a mixture model:

$$G(\boldsymbol{\theta}_d) := P(\boldsymbol{\theta}_d|\boldsymbol{\pi}, \{\boldsymbol{\theta}_m\}_{m=1}^M) = \sum_{m=1}^M \pi_m \delta_{\boldsymbol{\theta}_m}(\boldsymbol{\theta}_d),$$

where $\delta_{\boldsymbol{\theta}_m}(\boldsymbol{\theta})$ is the point mass distribution and takes value 1 for $\boldsymbol{\theta} = \boldsymbol{\theta}_m$ and 0 otherwise. In this sense, distribution $G(\cdot)$ defines a *discrete prior* for all document specific topic mixtures $\boldsymbol{\theta}_d$'s, and the plate model in Figure 9.2(a) can be equivalently illustrated as in Figure 9.2(b), where model parameters α and $\boldsymbol{\lambda}$ now take the role of tuning the discrete but *unknown* distribution $G(\cdot)$.

It is shown in Chapter 3 that when M tends to infinity, the unknown distribution G tends to be a sample from a Dirichlet process (DP), constrained by the concentration parameter α and base distribution $G_0 := \mathtt{Dir}(\boldsymbol{\lambda})$. Following the conventional notation for Dirichlet process, in our model we sample $\boldsymbol{\theta}_d$ as follows:

$$\begin{aligned} \boldsymbol{\theta}_d &\sim G, \quad \text{for } d = 1, \ldots, D; \\ G &\sim \mathcal{DP}(\alpha, G_0). \end{aligned}$$

The DP defines a *non-parametric prior* for $\boldsymbol{\theta}_d$'s, and as the sampled distribution G is discrete, a natural clustering structure will occur when several documents happen to choose the same $\boldsymbol{\theta}$ in the sampling process. Moreover, concentration parameter α is able to control the discreteness of sampled distribution G, and thus control the number of clusters. Larger α usually results in more clusters.

Remark 9.1.1. *We compare the plate model of PCP with DP prior (Figure 9.2 right) and that of LDA (Figure 4.4 in Chapter 4). Instead of assigning a Dirichlet prior for $\boldsymbol{\theta}$'s as in LDA, PCP model assigns a non-parametric prior G and have two hyperparameters to control G. As mentioned in Remark 3.2.1, PCP can be viewed as a Dirichlet enhanced*

LDA model. This also explains why PCP model is more flexible than LDA (see discussions in [80]).

When M is finite, however, our model is not equivalent to defining Dirichlet process prior for $\boldsymbol{\theta}_d$'s, but it is a good approximation if we choose a sufficiently large M, due to the *Dirichlet-Multinomial allocation* (DMA) approximation for Dirichlet processes. An important property of DMA is that it demonstrates similar clustering effect as with Dirichlet process prior, and α can also be used to control the granularity of the clustering structure (see Section 3.3.4). This motivates us to start with a large number for M and let the model automatically fit this number during learning procedure. It turns out that the data likelihood is improved each time the effective cluster number decreases in the iterative process, and that only a small number of clusters will have non-zero probabilities at convergence. α can be chosen *a priori* for different purposes of applications, or learned to fit the data. This so-called VBDMA algorithm is used for inference and learning in the next section.

9.2 Inference and Learning

In this section we consider model inference and learning. As seen from Figure 9.2, for inference we need to calculate the *a posteriori* distribution of latent variables

$$\hat{P}(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c}, \mathbf{z}) := P(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c}, \mathbf{z} | \mathcal{D}, \alpha, \boldsymbol{\lambda}, \beta),$$

including both effects of clustering and projection. Here for simplicity we denote $\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c}, \mathbf{z}$ as groups of $\pi_m, \boldsymbol{\theta}_m, c_d, z_{d,n}$, respectively. This requires to compute (9.1), where the integral is however analytically infeasible. A straightforward Gibbs sampling method can be derived, but it turns out to be very slow and inapplicable to high dimensional discrete data like text, since for each word we have to sample a latent variable z. Therefore in this section we suggest an efficient *variational method* by introducing variational parameters for latent variables [47]. Then we can maximize the data likelihood by iteratively updating these parameters and obtain a *variational Bayesian* algorithm until convergence. The interesting thing is that this algorithm is equivalent to performing clustering and projection iteratively, which we will discuss in detail. This, on one hand, provides an intuitive explanation to the complicated updating equations and let us gain some insight of variational method; on the other hand, it characterizes the object function of the iterative clustering-projection procedure and theoretically guarantees convergence.

9.2.1 Variational Bayesian Learning

The idea of variational Bayesian algorithm is to propose a joint distribution $Q(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c}, \mathbf{z})$ for latent variables conditioned on some free parameters, and then enforce Q to approximate the *a posteriori* distributions of interests by minimizing the KL-divergence

$D_{\mathrm{KL}}(Q\|\hat{P})$ with respect to those free parameters. We propose a variational distribution Q over latent variables as the following

$$Q(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c}, \mathbf{z}|\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\psi}, \boldsymbol{\phi}) = Q(\boldsymbol{\pi}|\boldsymbol{\eta}) \prod_{m=1}^{M} Q(\boldsymbol{\theta}_m|\gamma_m) \prod_{d=1}^{D} Q(c_d|\psi_d) \prod_{n=1}^{N_d} Q(z_{d,n}|\phi_{d,n}), \tag{9.4}$$

where $\boldsymbol{\eta}, \boldsymbol{\gamma}, \boldsymbol{\psi}, \boldsymbol{\phi}$ are groups of *variational parameters*, each tailoring the variational *a posteriori* distribution to each latent variable. In particular, $\boldsymbol{\eta}$ specifies an M-dimensional Dirichlet for $\boldsymbol{\pi}$, γ_m specifies a K-dimensional Dirichlet for distinct $\boldsymbol{\theta}_m$, ψ_d specifies an M-dimensional Multinomial for indicator c_d of document d, and $\phi_{d,n}$ specifies a K-dimensional Multinomial over latent topics for word $w_{d,n}$. It turns out that minimization of the KL-divergence is equivalent to maximization of a lower-bound of the log likelihood $\ln P(\mathcal{D}|\alpha, \boldsymbol{\lambda}, \beta)$, derived by applying Jensen's inequality [47]:

$$\begin{aligned}\mathcal{L}_Q(\mathcal{D}) = \mathbb{E}_Q[\ln P(\boldsymbol{\pi}|\alpha)] + \sum_{m=1}^{M} \mathbb{E}_Q[\ln P(\boldsymbol{\theta}_m|\boldsymbol{\lambda})] + \sum_{d=1}^{D} \mathbb{E}_Q[\ln P(c_d|\boldsymbol{\pi})] \\ + \sum_{d=1}^{D}\sum_{n=1}^{N_d} \mathbb{E}_Q[\ln P(w_{d,n}|z_{d,n}, \beta)P(z_{d,n}|\boldsymbol{\Theta}, c_d)] - \mathbb{E}_Q[\ln Q(\boldsymbol{\pi}, \boldsymbol{\Theta}, \mathbf{c}, \mathbf{z})].\end{aligned} \tag{9.5}$$

The optimum is found by setting the partial derivatives with respect to each variational and model parameter to be zero, which corresponds to the variational E-step and M-step, respectively. All the terms in the lower-bound can be derived similarly as those in Chapter 2, and for simplicity we omit the details in this chapter. In the following we separate these equations into two parts and interpret them from the perspective of clustering and projection, respectively.

9.2.2 Updates for Clustering

As we mentioned in Section 9.1.2, the specific variables for clustering are document-cluster assignments c_d, cluster centers $\boldsymbol{\theta}_m$, and cluster probabilities $\boldsymbol{\pi}$. It turns out that their corresponding variational parameters are updated as follows:

$$\psi_{d,m} \propto \exp\left\{\sum_{k=1}^{K}\left[\left(\Psi(\gamma_{m,k}) - \Psi(\sum_{i=1}^{K}\gamma_{m,i})\right)\sum_{n=1}^{N_d}\phi_{d,n,k}\right] + \Psi(\eta_m) - \Psi(\sum_{i=1}^{M}\eta_i)\right\}, \tag{9.6}$$

$$\gamma_{m,k} = \sum_{d=1}^{D}\psi_{d,m}\sum_{n=1}^{N_d}\phi_{d,n,k} + \lambda_k, \qquad \eta_m = \sum_{d=1}^{D}\psi_{d,m} + \frac{\alpha}{M}, \tag{9.7}$$

where $\Psi(\cdot)$ is again the digamma function, the first derivative of the log Gamma function. $\psi_{d,m}$ are the *a posteriori* probabilities $P(c_d = m)$ that document d belongs to cluster m, and define a *soft cluster assignment* for each document. $\gamma_{m,k}$ characterize the cluster centers $\boldsymbol{\theta}_m$ and are basically the k-th coordinate of $\boldsymbol{\theta}_m$ on the topic space. Finally η_m control the mixing weights for clusters and define the probability of cluster m. $\phi_{d,n,k}$ are

the variational parameters that measure the *a posteriori* probability that word $w_{d,n}$ in document d is sampled from topic k. They are related to projection of words and assumed fixed at the moment.

These equations seem to be complicated and awful, but they turn out to be quite intuitive and just follow the standard clustering procedure. In particular,

- $\psi_{d,m}$ is seen from (9.6) to be a multiplication of two factors P_1 and P_2, where P_1 includes the γ terms in the exponential and P_2 the η terms:

$$P_1 \propto \exp\left\{\sum_{k=1}^{K}\left[\left(\Psi(\gamma_{m,k}) - \Psi(\sum_{i=1}^{K}\gamma_{m,i})\right)\sum_{n=1}^{N_d}\phi_{d,n,k}\right]\right\},$$

$$P_2 \propto \exp\left\{\Psi(\eta_m) - \Psi(\sum_{i=1}^{M}\eta_i)\right\}.$$

  Since η_m controls the probability of cluster m, P_2 acts as a *prior term* for $\psi_{d,m}$; P_1 can be seen as the *likelihood term*, because it explicitly measures the probability of generating $\mathbf{w}_d$ from cluster m by calculating the inner product of projected features and cluster centers. Therefore, (9.6) directly follows from Bayes' rule, and a normalization term is needed to ensure $\sum_{m=1}^{M}\psi_{d,m} = 1$.

- $\gamma_{m,k}$ is updated by summing over the *prior position* λ_k and the *empirical location*, the weighted sum of projected documents that belong to cluster k.

- Similar to $\gamma_{m,k}$, η_k is empirically updated by summing over the *belongingnesses* of all documents to cluster k. α/M acts as a prior or a *smoothing term*, shared by all the clusters.

Since these parameters are coupled, clustering is done by iteratively updating (9.6) and (9.7). Note that the words are incorporated into the clustering process *only* via the projected features $\sum_{n=1}^{N_d}\phi_{d,n,k}$. This means that the clustering is performed not in word space, but in the more informative topic space. Documents are first projected into a low-dimensional space and then clustered, and in principal we can incorporate well-defined projections into the clustering procedure to accelerate convergence.

9.2.3 Updates for Projection

If $\boldsymbol{\psi}, \boldsymbol{\gamma}, \boldsymbol{\eta}$ are fixed, projection parameters $\boldsymbol{\phi}$ and β are updated as:

$$\phi_{d,n,k} \propto \beta_{k,w_{d,n}} \exp\left\{\sum_{m=1}^{M}\psi_{d,m}\left[\Psi(\gamma_{m,k}) - \Psi(\sum_{i=1}^{K}\gamma_{m,i})\right]\right\}, \tag{9.8}$$

$$\beta_{k,j} \propto \sum_{d=1}^{D}\sum_{n=1}^{N_d}\phi_{d,n,k}\delta_j(w_{d,n}), \tag{9.9}$$

where $\delta_j(w_{d,n}) = 1$ if $w_{d,n}$ takes word index j, and 0 otherwise. Please recall that $\phi_{d,n,k}$ is the *a posteriori* probability that word $w_{d,n}$ is sampled from topic k, and $\beta_{k,j}$ measures the probability of generating word j from topic k. Normalization terms are needed to ensure $\sum_{k=1}^{K} \phi_{d,n,k} = 1$ and $\sum_{j=1}^{V} \beta_{k,j} = 1$, respectively. Update (9.9) for $\beta_{k,j}$ is quite intuitive, since we just sum up all the documents that word j occurs, weighted by their generating probabilities from topic k. For update of $\phi_{d,n,k}$ in (9.8), $\beta_{k,w_{d,n}}$ is the probability that topic k generates word $w_{d,n}$ and is thus the *likelihood* term; the rest exponential term defines the *prior*, i.e., the probability that document d selects topic k. This is calculated by taking into account the clustering structure and summing over all cluster centers with corresponding soft weights. Therefore, the projection model is learned via clusters of documents, not simply individual ones. This makes it possible to integrate the rich intrinsic structure of documents into projection learning and alleviate the impact of outlier documents. Finally we iterate (9.8) and (9.9) until convergence to obtain the optimal projection.

9.2.4 Discussion

As guaranteed by variational Bayesian algorithm, iteratively performing the given clustering and projection operations will improve the data likelihood monotonically until convergence, where a local maxima is obtained. The convergence is usually very fast, and like other variational methods, the quality of the maxima and the convergence rate depend on the initialization of model parameters. For a good starting point, it would be beneficial to get an estimate of the projection matrix before starting, using other simple projection models like pLSI.

In the last two subsections we interpret the variational Bayesian algorithm in terms of iteratively performing clustering and projection operations. Although the intuitive explanations are not formal, they help to understand the model as well as the variational algorithm.

The remaining parameters α and $\boldsymbol{\lambda}$ control the mixing weights $\boldsymbol{\pi}$ and cluster centers $\boldsymbol{\theta}_m$ *a priori*, and they can also be learned by setting their partial derivatives to be zero. However, there are no analytical updates for them and we have to use computational methods like Newton-Raphson method as in [10]. Finally we summarize the whole algorithm in Algorithm 9.1.

9.3 Empirical Study

In this section we illustrate experimental results for the PCP model. In particular we compare it with other models in the following three perspectives:

- **Document Modeling**: How good is the generalization in PCP model?
- **Word Projection**: Is the projection really improved in PCP model?

Algorithm 9.1 Variational Bayesian Learning for PCP Model

Require: A corpus of D documents with V words. Document d has N_d words in total.
Require: A topic number $K > 0$. Model parameters α and $\boldsymbol{\lambda}$.

Choose a large enough mixture number M (e.g., set $M = D$). Randomly initialize $\phi_{d,n,k}$, $\gamma_{m,k}$ and η_k such that $\sum_{k=1}^{K} \phi_{d,n,k} = 1$, for all $d = 1, \ldots, D$, $n = 1, \ldots, N_d$.
repeat
  {**Clustering Step**}
  Calculate *sufficient projection term* $\sum_{n=1}^{N_d} \phi_{d,n,k}$ for each document d;
  repeat
    Update cluster assignments $\psi_{d,m}$ via (9.6);
    Update cluster centers $\gamma_{m,k}$ and mixing weights η_k via (9.7);
  until the improvement of log-likelihood is smaller than a threshold.
  {**Projection Step**}
  Calculate *sufficient clustering term* $\sum_{m=1}^{M} \psi_{d,m} \left[\Psi(\gamma_{m,k}) - \Psi(\sum_{i=1}^{K} \gamma_{m,i})\right]$ for each document d;
  repeat
    Update word projections $\phi_{d,n,k}$ via (9.8);
    Update projection matrix β via (9.9);
  until the improvement of log-likelihood is smaller than a threshold.
until the improvement of log-likelihood is smaller than a threshold.

Output: Document-cluster membership $\psi_{d,m}$'s, cluster weights calculated as $\pi_m = \eta_m / \sum_{m'=1}^{M} \eta_{m'}$, cluster centers γ_m's, and word-topic projection matrix β.

- **Document Clustering**: Will the clustering be better in PCP model?

We will make comparisons based on two text data sets. The first one is Reuters-21578, and we select all the documents that belong to the five categories *money-fx*, *interest*, *ship*, *acq* and *grain*. After removing stop words, stemming and picking up all the words that occur at least in 5 documents, we finally obtain 3948 documents with 7665 words. The second data set consists of four groups taken from 20Newsgroup, i.e., *autos*, *motorcycles*, *baseball* and *hockey*. Each group has 1000 documents, and after the same preprocessing we get 3888 documents with 8396 words. In the following we use "Reuters" and "Newsgroup" to denote these two data sets, respectively. Before giving the main results, we illustrate one case study for better understanding of the algorithm.

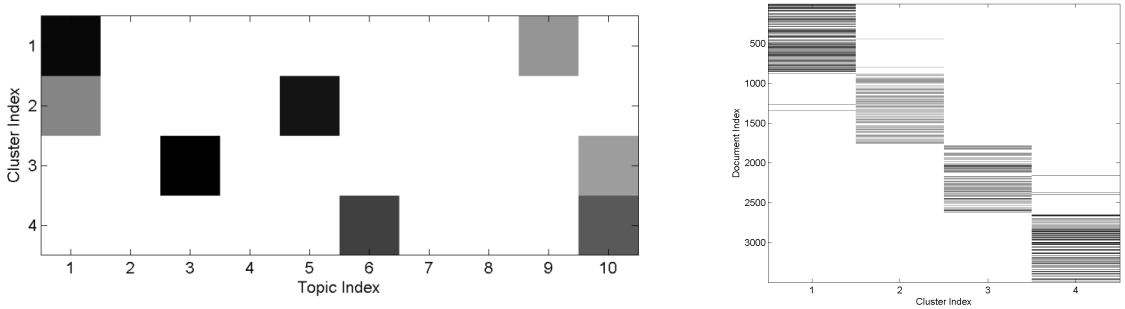
9.3.1 Case Study

We run the PCP model on the Newsgroup data set, and set topic number $K = 50$ and cluster number $M = 20$. α is set to 1 and $\boldsymbol{\lambda}$ is set with each entry being $1/K$. Other initializations are chosen randomly. The algorithm runs until the improvement on $\mathcal{L}_Q(\mathcal{D})$ is less than 0.01% and converges after 50 steps.

Figure 9.3 illustrates part of the results. In (a) 10 topics are shown with 10 words that

1	2	3	4	5	6	7	8	9	10
car	ball	game	gm	bike	team	car	pit	car	team
engin	runner	basebal	rochest	clutch	hockei	tire	det	price	year
ford	hit	gant	ahl	back	nhl	brake	bo	dealer	win
problem	base	pitch	st	gear	leagu	drive	tor	year	morri
mustang	write	umpir	john	front	game	radar	chi	model	cub
good	fly	time	adirondack	shift	season	oil	nyi	insur	game
probe	rule	call	baltimor	car	citi	detector	van	articl	write
write	articl	strike	moncton	time	year	system	la	write	jai
ve	left	write	hockei	work	star	engin	stl	cost	won
sound	time	hirschbeck	utica	problem	minnesota	spe	buf	sell	clemen

(a)



(b) (c)

Figure 9.3: A case study of PCP model on Newsgroup data. (a) shows 10 topics and 10 associated words for each topic with highest generating probabilities. (b) shows 4 clusters and the topic mixture on the 10 topics. Darker color means higher value. (c) gives the assignments to the 4 clusters for all the documents which are sorted by their true category labels.

have highest assigned probabilities in β. The topics are seen to be very meaningful and each defines one projection for all the words. For instance, topic 5 is about "bike", and 1, 7, 9 are all talking about "car" but with different subtopics: 1 is about general stuffs of car; 7 and 9 are specifying car systems and purchases, respectively. Besides finding topic 6 that covers general terms for "hockey", we even find two topics that specify the hockey teams in US (4) and Canada (8). These topics provide the building blocks for document clustering.

Figure 9.3(b) gives the 4 cluster centers that have highest probabilities after learning. They define topic mixtures over the whole 50 topics, and for illustration we only show the given 10 topics as in (a). Darker color means higher weight. It is easily seen that they are corresponding to the 4 categories *autos*, *motorcycles*, *baseball* and *hockey*, respectively. If we sort all the documents with their true labels, we obtain the document-cluster assignment matrix as shown in Figure 9.3(c). Documents that belong to different categories are clearly separated, and the clustering structure is uncovered.

Table 9.1: Perplexity comparison for pLSI, LDA and PCP on Reuters and Newsgroup

	Reuters						Newsgroup					
K	5	10	20	30	40	50	5	10	20	30	40	50
pLSI	1995	1422	1226	1131	1128	1103	2171	2018	1943	1868	1867	1924
LDA	1143	892	678	599	562	533	2083	1933	1782	1674	1550	1513
PCP	**1076**	**882**	**670**	**592**	**555**	**527**	**2039**	**1871**	**1752**	**1643**	**1524**	**1493**

9.3.2 Document Modeling

In this subsection we investigate the generalization of PCP model. We compare PCP with pLSI and LDA on the two data sets, where 90% of the data are used for training and the rest 10% are held out for testing. The comparison metric is *perplexity*, which is conventionally used in language modeling and defined as

$$\text{Perp}(\mathcal{D}_{\text{test}}) = \exp\left(-\ln P(\mathcal{D}_{\text{test}}) / \sum_d |\mathbf{w}_d|\right),$$

where $|\mathbf{w}_d|$ is the length of document d, and the sum is over all documents in the testing set $\mathcal{D}_{\text{test}}$. A lower perplexity score indicates better generalization performance.

We follow the formula in [10] to calculate perplexity for pLSI. For PCP model, we take the similar approach as in LDA, i.e., we run the variational inference and calculate the lower bound (9.5) as the likelihood term. M is set to be the number of training documents for initialization. As suggested in [10], a smoothing term for β is used and optimized for LDA and PCP. All the three models are trained until the improvement is less than 0.01%. We compare all three algorithms using different K's, and the results are shown in Table 9.1. PCP outperforms both pLSI and LDA in all the runs, which indicates that the model fits the data better and has better generalization performance.

9.3.3 Word Projection

All the three models pLSI, LDA and PCP can be seen as projection models and learn the mapping β. To compare the quality, we train a support vector machine (SVM) on the low-dimensional representations of these models and measure the classification rate. For pLSI, the projection for document d is calculated as the *a posteriori* probability of latent topics conditioned on d, $P(z|d)$. This can be computed using Bayes' rule as $P(z|d) \propto P(d|z)P(z)$. In LDA it is calculated as the *a posteriori* Dirichlet parameters for d in the variational E-step [10]. In PCP model this is simply the projection term $\sum_{n=1}^{N_d} \phi_{d,n,k}$ which is used in clustering.

We train a 10-topic model on the two data sets and then train a SVM for each category. Note that we are reducing the feature space by 99.8%. In the experiments we gradually improve the number of training data from 10 to 200 (half positive and half negative) and randomize 50 times. The performance averaged over all categories is shown in Figure 9.4

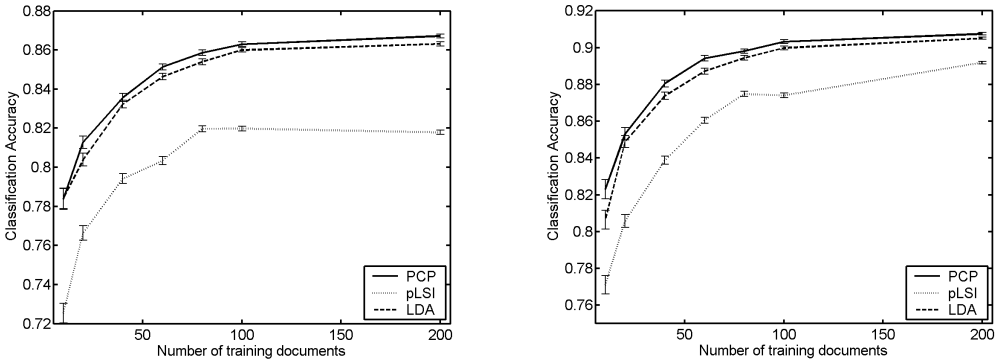


Figure 9.4: Classification results on Reuters (left) and Newsgroup (right).

Table 9.2: Comparison of clustering using different methods

	NMF	LDA+k-means	PCP
Reuters	0.246	0.331	**0.418**
Newsgroup	0.522	0.504	**0.622**

with mean and standard deviation. It is seen that PCP obtains better results and learns a better word projection.

9.3.4 Document Clustering

In our last experiment we demonstrate the performance of PCP model on document clustering. For comparison we implement the original version of NMF algorithm [51] which can be shown as a variant of pLSI, and a k-means algorithm that uses the learned features by LDA. For NMF we tune its parameter to get best performance. The k-means and PCP algorithms are run with the true cluster number, and we tune the dimensionality K to get best performance.

The experiments are run on both two data sets. The true cluster number is 5 for Reuters and 4 for Newsgroup. For comparison we use the normalized mutual information [77], which is just the mutual information divided by the maximal entropy of the two cluster sets. The results are given in Table 9.2, and it can be seen that PCP performs the best on both data sets. This means iterating clustering and projection can obtain better clustering structure for documents.

9.4 Summary

In this chapter we proposed a probabilistic clustering-projection model for discrete co-occurrence data, which unifies clustering and projection in one probabilistic model. Iteratively updating the two operations turns out to be the variational inference and learning under Bayesian treatments. Experiments on two text data sets show promising performance for the proposed model.

Chapter 10

Conclusion

Probabilistic modeling for unsupervised learning is a very important research area. In this thesis we introduced several advanced probabilistic models for clustering and projection, and made the following contributions:

- In the first part we studied the general finite mixture models of exponential family distributions from a Bayesian perspective. The proposed Gibbs sampling and variational Bayesian inference algorithms are applicable to all exponential family distributions.

- We clarified the connection between finite mixture model with symmetric Dirichlet prior and the Dirichlet process prior. This allows us to achieve the same sparsity in finite mixture modeling, and to control the sparsity via a single parameter.

- We proposed a variational Bayesian learning algorithm called VBDMA for approximated inference in a DP model. We then empirically compared the proposed method with another one called VBTDP, and showed that they are both applicable to large-scale problems but with different obtained sparsity.

- In the second part we extended the probabilistic PCA to non-linear kernel PCA, and derived an EM algorithm for kernel PCA which is more efficient and can deal with larger data sets. Furthermore an incremental kernel PCA can be proposed from this algorithm.

- We derived the MORP algorithm for supervised dimensionality reduction which is motivated from a latent variable model. The model takes PCA and kernel PCA as special cases and obtain better results than other supervised projection methods.

- We introduced a probabilistic version of MORP which is called SPPCA. A natural extension of SPPCA is called S^2PPCA which can handle semi-supervised projection. To our knowledge this is the first elegant semi-supervised projection framework.

- In the last part we considered clustering and projection jointly and introduced a probabilistic clustering-projection model for discrete data. The model is shown to obtain better results for both clustering of data and projection of features.

Unsupervised learning like clustering and projection is a big area, and this thesis only addresses this problem from a probabilistic perspective. Sometimes the probabilistic assumptions are not perfectly satisfied, and the deterministic approaches are usually more robust. So in the future it is interesting to figure out when to apply those methods, and to uncover the relationship between probabilistic models and the various deterministic methods. Extending the current solutions to more complex setting and larger data sets is also an important research problem.

Bibliography

[1] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of ACM SIGMOD Internation Conference on Management of Data (SIGMOD'99)*, 1999.

[2] C. E. Antoniak. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *Annals of Statistics*, 2(6), Nov. 1974.

[3] H. Attias. A variational Bayesian framework for graphical models. In *Advances in Neural Information Processing Systems 12*. MIT Press, 2000.

[4] J. D. Banfield and A. E. Raftery. Model-based gaussian and non-gaussian clustering. *Biometrics*, 49:803–821, 1993.

[5] D. Bartholomew and M. Knott. *Latent Variable Models and Factor Analysis*. Oxford University Press, 1999.

[6] C. Basu, H. Hirsh, and W. W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the Fifteenth National Conference on Artificial Intelligencen AAAI/IAAI*, pages 714–720, 1998.

[7] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems 14*.

[8] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[9] D. M. Blei and M. I. Jordan. Variational methods for the Dirichlet process. Proceedings of the 21st International Conference on Machine Learning, 2004.

[10] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[11] L. M. Bregman. The relaxation method of finding the common points of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7:200–217, 1967.

[12] W. Buntine and A. Jakulin. Applying discrete PCA in data analysis. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, 2004.

[13] W. Buntine and S. Perttu. Is multinomial PCA multi-faceted clustering or dimensionality reduction? In *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*, pages 300–307, 2003.

[14] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: A Bayesian classification system. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 54–64, 1988.

[15] R. J. Connor and J. E. Mosimann. Concepts of independence for proportions with a generalization of the Dirichlet distribution. *J. Amer. Stat. Ass.*, 64:194–206, 1969.

[16] A. Corduneanu and C. M. Bishop. Variational Bayesian model selection for mixture distributions. In T. Richardson and T. Jaakkola, editors, *Eighth International Workshop on Artificial Intelligence and Statistics*, pages 27–34. Morgan Kaufmann, 2001.

[17] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[18] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm (with discussion). *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

[19] I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 269–274, 2001.

[20] C. Ding, X. He, H. Zha, and H. D. Simon. Adaptive dimension reduction for clustering high dimensional data. In *ICDM*, pages 147–154, 2002.

[21] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, 2000.

[22] M. D. Escobar and M. West. Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90(430), 1995.

[23] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[24] T. Evgeniou and M. Pontil. Regularized multi-task learning. In *Proceedings SIGKDD*, 2004.

[25] T. S. Ferguson. A Bayesian analysis of some nonparametric problems. *Annals of Statistics*, 1:209–230, 1973.

[26] C. Fraley and A. E. Raftery. How many clusters? which clustering method? answers via model-based cluster analysis. *The Computer Journal*, 41(8):578–588, 1998.

[27] K. Fukumizu, F. R. Bach, and M. I. Jordan. Dimensionality reduction for supervised learning with reproducing kernel hilbert spaces. *Journal of Machine Learning Research*, 5(Jan):73–99, 2004.

[28] A. Gelman, J. B. Carlin, H.S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Chapman and Hall-CRC, 1996.

[29] Z. Ghahramani and M. J. Beal. Graphical models and variational methods. In D. Saad and M. Opper, editors, *Advanced mean Field Methods — Theory and Practice*. MIT Press, 2000.

[30] Z. Ghahramani and M. J. Beal. Variational inference for Bayesian mixtures of factor analysers. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*. MIT Press, 2000.

[31] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval Journal*, 4(2):133–151, 2001.

[32] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins Univeristy Press, 1996.

[33] P. J. Green and S. Richardson. Modelling heterogeneity with and without the Dirichlet process. unpublished paper, 2000.

[34] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[35] D. R. Hardoon, J. Shawe-Taylor, and O. Friman. KCCA feature selection for fmri analysis. Technical Report CSD-TR-03-02, Royal Holloway University of London, 2003.

[36] D. R. Hardoon, S. Szedmak, and J. Shawe-Taylor. Canonical correlation analysis; an overview with application to learning methods. Technical Report CSD-TR-03-02, Royal Holloway University of London, 2003.

[37] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Satatistical Learning*. Springer Verlag, 2001.

[38] K. A. Heller and Z. Ghahramani. Bayesian hierarchical clustering. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 297–304, 2005.

[39] T. Hofmann. Probabilistic Latent Semantic Indexing. In *Proceedings of the 22nd Annual ACM SIGIR Conference*, pages 50–57, Berkeley, California, August 1999.

[40] T. Hofmann and J. Puzicha. Statistical models for co-occurrence data. Technical Report AIM-1625, 1998.

[41] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, 1933.

[42] H. Hotelling. Relations between two sets of variables. *Biometrika*, 28:321–377, 1936.

[43] Y. Huang, K. Yu, M. Schubert, S. Yu, and H.-P. Kriegel. Hierarchy-regularized latent semantic indexing. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM'2005)*, 2005.

[44] H. Ishwaran and L. F. James. Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, 96(453):161–173, 2001.

[45] H. Ishwaran and L. F. James. Gibbs sampling methods for stick-breaking priors. *J. Amer. Stat. Assoc*, 96:161–173, 2001.

[46] I. T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 2002.

[47] M. I. Jordan, Z. Ghahramani, T. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[48] S. D. Kamvar, D. Klein, and C. D. Manning. Interpreting and extending classical agglomerative clustering algorithms using a model-based approach. In *Proceedings of the 19th International Conference on Machine Learning*, pages 283–290, 2002.

[49] H. Kim and Z. Ghahramani. The EM-EP algorithm for Gaussian process classification. In *Proceedings of the Workshop on Probabilistic Graphical Models for Classification at ECML*, 2003.

[50] J. B. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Publications, Beverly Hills, CA, 1978.

[51] D. D. Lee and H. S. Seung. Learning the parts of objects with nonnegative matrix factorization. *Nature*, 401:788–791, 1999.

[52] D. D. Lewis, Y. Yang, T. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[53] D. J. C. MacKay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1991.

[54] T. P. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

[55] R. M. Neal. Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9:249–265, 2000.

[56] K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science, Sixth Series 2*, pages 559–572, 1901.

[57] R. Rosipal and L. J. Trejo. Kernel Partial Least Squares Regression in Reproducing Kernel Hilbert Space. *Journal of Machine Learning Research*, 2(12):97–123, 2001.

[58] S. Roweis. EM algorithm for PCA and SPCA. In *Advances in Neural Information Processing Systems 10*. MIT Press, 1998.

[59] S. Roweis and Z. Ghahramani. A unifying review of linear gaussian models. *Neural Computaion*, 11:305–345, 1999.

[60] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, pages 2323–2326, 2000.

[61] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[62] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.

[63] A. Schwaighofer, V. Tresp, and K. Yu. Hierarchical bayesian modelling with Gaussian processes. In *Advances in Neural Information Processing Systems 17*. MIT Press, 2005.

[64] J. Sethuraman. A constructive definition of Dirichlet priors. *Statistica Sinica*, 4:639–650, 1994.

[65] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge Univeristy Press, 2004.

[66] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[67] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

[68] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-Posed Problems*. Wiley, New York, 1977.

[69] M. E. Tipping and C. M. Bishop. Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443–482, 1999.

[70] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statisitical Scoiety*, B(61):611–622, 1999.

[71] M. Turk and A. Pentland. Face recognition using eigenfaces. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–591, 1991.

[72] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[73] J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. Vapnik. Kernel dependency estimation. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press.

[74] H. Wold. Soft modeling by latent variables; the nonlinear iterative partial least squares approach. *Perspectives in Probability and Statistics, Papers in Honour of M.S. Bartlett*, 1975.

[75] H. Wold. Partial least squares. *Encyclopedia of the Statistical Sciences*, pages 581–591, 1985.

[76] T.-T. Wong. Generalized Dirichlet distribution in Bayesian analysis. *Appl. Math. Comput.*, 97(2-3):165–181, 1998.

[77] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th Annual International ACM SIGIR Conference*, pages 267–273, 2003.

[78] Z. Xu, V. Tresp, K. Yu, S. Yu, and H.-P. Kriegel. Dirichlet enhanced relational learning. In *Proceedings of the 22nd International Conference on Machine Learning (ICML'2005)*, 2005.

[79] K. Yu, V. Tresp, and S. Yu. A nonparametric hierarchical bayesian framework for information filtering. In *Proceedings of the 27th Annual International ACM SIGIR Conference (SIGIR'2004)*, 2004.

[80] K. Yu, S. Yu, and V. Tresp. Dirichlet enhanced latent semantic analysis. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTATS'2005)*, 2005.

[81] K. Yu, S. Yu, and V. Tresp. Multi-label informed latent semantic indexing. In *Proceedings of the 28th Annual International ACM SIGIR Conference (SIGIR'2005)*, 2005.

[82] K. Yu, S. Yu, and V. Tresp. Multi-output regularized projection. In *Proceedings of the International IEEE Conference on Computer Vision and Pattern Recognition (CVPR2005)*, 2005.

[83] K. Yu, S. Yu, and V. Tresp. Soft clustering on graphs. In *Neural Information Processing Systems 18 (NIPS'05)*, 2006.

[84] S. Yu, K. Yu, V. Tresp, and H.-P. Kriegel. A probabilistic clustering-projection model for discrete data. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'2005)*, 2005. Best Paper Runner-Up.

[85] S. Yu, K. Yu, V. Tresp, and H.-P. Kriegel. Multi-output regularized feature projection. *IEEE Transactions on Knowledge and Data Engineering*, 18(22), 2006.

[86] S. Yu, K. Yu, V. Tresp, and H.-P. Kriegel. Variational Bayesian Dirichlet-Multinomial Allocation for Exponential Family Mixtures. In *Proceedings of the 17th European Conference on Machine Learning (ECML'2006)*, 2006.

[87] S. Yu, K. Yu, V. Tresp, H.-P. Kriegel, and M. Wu. Supervised probabilistic principal component analysis. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (KDD'2006)*, 2006.

[88] H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. Technical report, Statistics Department, Stanford University, 2005.

Index