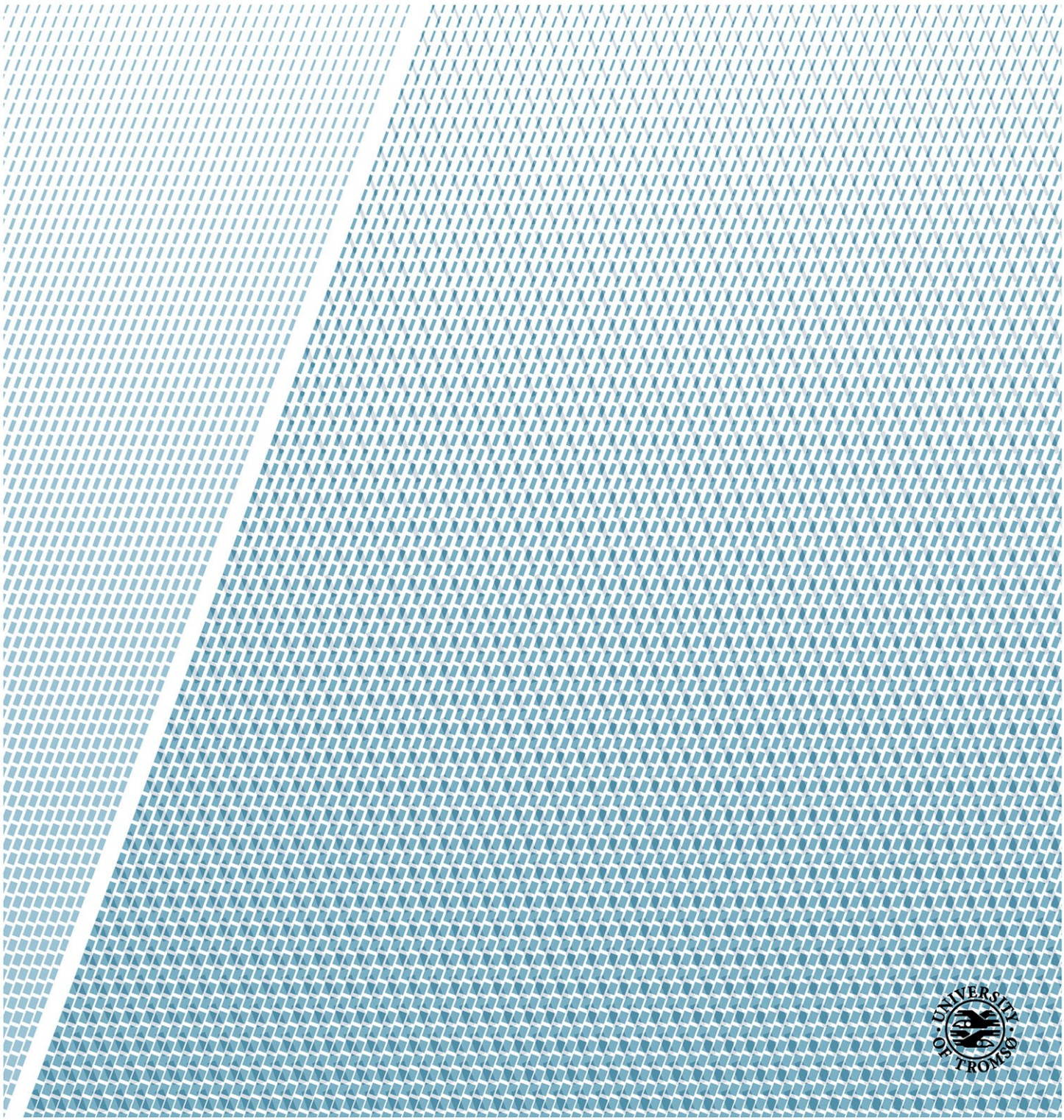
UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty of Science and Technology
Department of Mathematics and Statistics

Dirichlet Process Cluster Kernel

—

Tobias Olsen Foslid
STA-3900 Master's Thesis in Statistics



Abstract

This thesis aims to apply the Dirichlet process mixture model to the cluster kernel framework. The probabilistic cluster kernel is extended with a Bayesian nonparametric model to avoid critical parameters within the model. The Dirichlet process cluster kernel demonstrate advantages compared to the probabilistic cluster kernel in both classification and clustering. Additionally, the two dimensional projection using kernel PCA and the Dirichlet process cluster kernel show compact clusters with a higher degree of cluster discrimination.

The second main contribution of the thesis is an application of the cluster kernel methodology in semi-supervised learning. The Dirichlet process cluster kernel demonstrates a high degree of descriptive representation.

Acknowledgements

First I would like to thank my supervisors, Associate Professor Robert Jenssen, Dr. Filippo Maria Bianchi and Sigurd Løkse for your great advices, ideas and patience throughout the work on the thesis project. Robert, without your inspiring lectures I would never have chosen to study machine learning. To Filippo, thank you for the continuous discussions and helpful comments. Finally thank you to Sigurd for your mathematical insight into kernels, and trying to pass it onto me. And to all of you, thank you for not giving up on me, and if you did, for not showing it.

To all my friends (you know who you are); thank you for the continuous coffee breaks. Without you the thesis would be finished yesterday. I do not think I would have finished my thesis without the endless discussions, long nights of drinking bears and hanging out. To Lars: Without you, the office hours would feel empty.

Most of all, thank you to my girlfriend Aina, for supporting me through this crazy year. I am looking forward to spending more time with you.

Special mentions goes out to my good friend Sigurd, here is your special mention.

Contents

List of Algorithms

List of Figures

List of Tables

List of Acronyms

DPMM Dirichlet Process Mixture Model 1
GMM Gaussian Mixture Model 2
PCK Probabilistic Cluster Kernel 2
DPCK Dirichlet Process Cluster Kernel 2
PCA principal components analysis 10
SVM Support vector machines 14
RBF Radial basis function 14
KPCA kernel principal components analysis 15
MCMC Markov chain Monte Carlo 25
MH-algorithm Metropolis-Hastings algorithm 25
KDE Kernel Density Estimation 29
CRP Chinese restaurant process 34
GIW Gaussian Inverse-Wishart 48
NMI Normalized Mutual Information 76

Summary of Data Sets

In the thesis there will be discussions mainly about a few different data sets. In table 1 the different data sets that are used throughout the thesis is summarized. All data sets can be found either at the UCI data set repository [1] or at Kaggle.com.

Name:	N	D	Classes	Source:	Ref
Default of Credit Card Clients	30 000	25	2	UCI	[2]
Credit Card Fraud Detection	284 807	31	2	Kaggle	[3]
Breast Cancer Wisconsin (Diagnostic)	699	31	2	UCI	[4]
SPECT Heart	267	23	2	UCI	
Titanic Survival	712	8	2	Kaggle	
Iris Species	150	4	3	UCI	
IMDB 5000 Movie	3756	28	3	Kaggle	
Contraceptive Method Choice	1473	9	3	UCI	
Image Segmentation	2310	19	7	UCI	
Ecoli	336	8	8	UCI	
Nutrition Facts for McDonald's Menu	260	23	9	Kaggle	
Abalone	4177	9	28	UCI	

Table 1.: Description of the different data sets used within the thesis.

For computational reasons each dataset is represented as a subsample of 300 observations for both the train and test set, unless otherwise specified. The specifics used to create each subset can be seen in appendix A.

1. Introduction

1.1. Background

In our daily lives, we are surrounded by increasingly large amounts of data. In line with the amount of data accumulated every day, we are in need of methods to automatically analyze and explore data. There is an increasing demand for accurate, efficient and usable methods for automated learning from data. We use automated learning algorithms everyday. We can find such algorithms everywhere from driving patterns to bus timetables.

Many tasks within the automated learning methodology is unsupervised, which means there is no prior information about the structure of the data, or class labels. Unsupervised methods include clustering, the task of grouping data into groups where the observations within one group is similar to each other. Without knowing anything about the underlying class labels methods to optimize the clustering model performance cannot be applied. Such methods include parameter grid search [5] or cross validation [6, 7]. Therefore there is a need for methodologies without a dependency to critical parameters. One such methodology is ensemble clustering, where the same data is clustered multiple times before combing the results. The main idea is that if the same observations is clustered together multiple times, there will be more evidence that they should be in the same cluster.

Within unsupervised frameworks and clustering we are often faced with the problem of choosing the number of groups in the data. Does an image contain one, two or maybe five objects? One methodology for dealing with these kinds of problems with a statistical framework is to use a nonparametric model. A nonparametric model is a model where the number of parameters increase (unbounded) with the data, e.g. as we see wider images we see more objects within the image. The nonparametric method in focus in this thesis, is the Dirichlet Process Mixture Model (DPMM). The Dirichlet process was first introduced by Ferguson in 1973 [8]. Then, a year later the mixtures of Dirichlet processes for application within Bayesian nonparametric problems was introduced [9]. However it was not until much later with the rise of approximate statistics and Markov chain Monte Carlo methods that Dirichlet processes gained traction with DPMM [10, 11].

Another issue with many machine learning frameworks is that they are derived such that they can only learn linear boundaries between classes. However, data are often not linearly separable. One methodology to deal with these problems are kernel methods [12], where a kernel function is used to implicitly map the data to a high dimensional space, called feature space. The main motivation for kernel methods is that in feature

space, classes have a higher probability to be linearly separable. Kernel methods rely on a kernel where each element represent the similarity between two observations in feature space. However, most kernel functions are dependent on a shape parameter, the value of the shape parameter changes the feature space in a critical way. Within an unsupervised framework there are few possibilities to further tune the shape parameters.

1.2. Aims and Motivation of the Thesis

References for the methods discussed in this section can be found in the related work, section 1.3

Combining the ideas from ensemble clustering and kernel methods give rise to the Probabilistic Cluster Kernel (PCK) framework. The PCK is a data driven kernel function, that learn similarities between observations by clustering the same data multiple times. Within each ensemble a Gaussian Mixture Model (GMM) is fitted to the data, and the posterior probabilities are predicted. Whereby the different clusterings are combined by taking the inner products between pairs of observations to create a final kernel matrix. Within the PCK framework there are two parameters, where one decide the number of clusters for the GMM. A benefit of the cluster kernel framework is that the learned kernel represent similarities between observations. One drawback with the current cluster kernel framework is that the final kernel requires two parameters to be learned. To learn the finite cluster kernel the number of clusters have to be decided a priori, thus making prior assumptions on the underlying shape of the data.

The aim of the thesis is to use a nonparametric Bayesian model to create a cluster kernel without any prior assumptions on the data. The thesis proposes to apply the DPMM to the existing PCK to create a nonparametric cluster kernel, the Dirichlet Process Cluster Kernel (DPCK). The justification for using a more advanced clustering model within the cluster kernel framework is to be less biased towards prior parameter choices. Additionally, the DPCK is applied to some semi-supervised learning problems, where the descriptive power of the cluster kernel is examined.

In figure 1.1 the outline of the cluster kernel process is illustrated. The highlighted parts represent some of the contributions of the thesis.

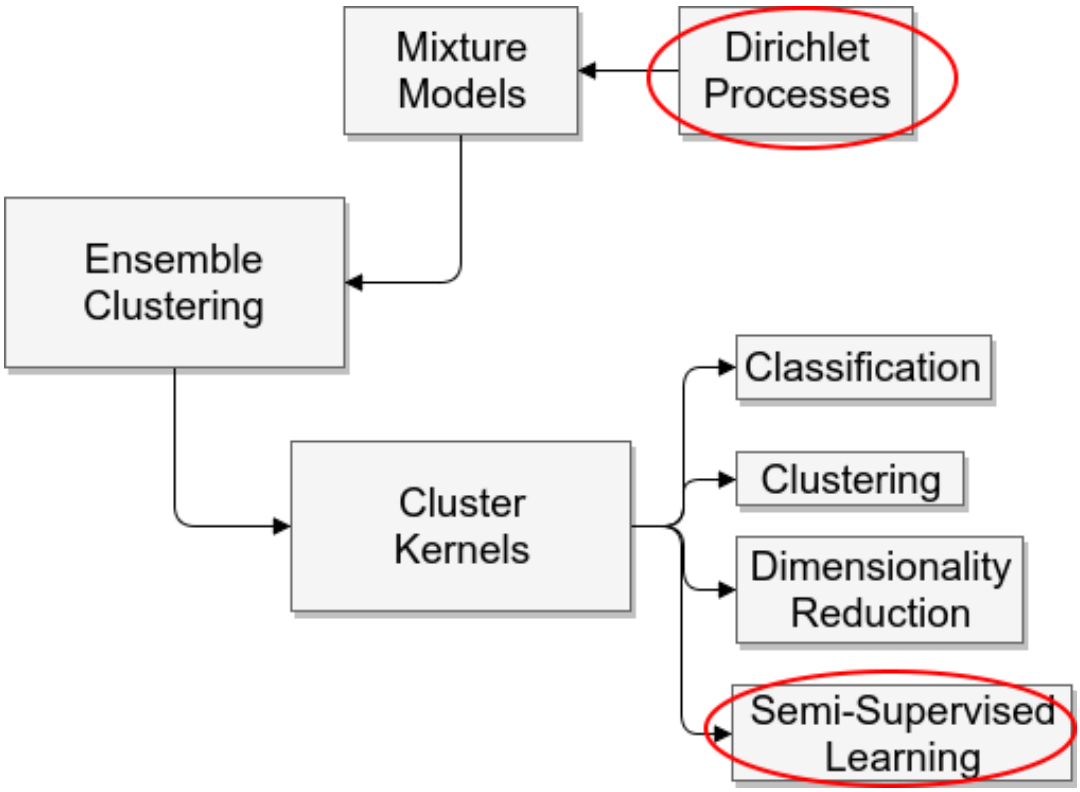


Figure 1.1.: Outline of cluster kernel pipeline and application. Red circles represent where the main contributions of the thesis lies.

The thesis project aim at introducing the DPMM within the cluster kernel framework. This creates the DPCK, a nonparametric cluster kernel that infers the number of cluster within each ensemble. Experiments are conducted to compare the DPCK against the existing PCK. Finally, the DPCK is applied in a semi-supervised classification problem. The main contributions of the thesis include:

- **Chapter 5:** The methodology of the Dirichlet Process Cluster Kernel (DPCK) is developed.
- **Chapter 7:** Experiments are conducted to compare the DPCK to the PCK.
- **Chapter 8:** New application of cluster kernels, where the DPCK is applied to semi-supervised classification problems.

1.3. Related Work

Cluster kernels is an ensemble clustering framework first introduced in [13], and further developed within remote-sensing [14, 15]. The aim of the cluster kernel methodology is to learn a data driven kernel function, to represent similarities between observations without prior assumptions of the data. However, the PCK relies on a parameter that

controls the number of clusters and therefore making prior assumptions about the shape of the data. This thesis aim to extend the current cluster kernel methodology with the DPMM, such that each model infers the number of clusters automatically.

The DPMM is vastly represented in the literature [16–18], with successful application in a wide range of topics. These topics range from microarray data [19] to natural language processing with clustering verbs [20, 21] to tissue classification of magnetic resonance imaging (MRI) [22]. Within the natural langugate processing they want to cluster verbs with similar meaning together. They adapt the DPMM sampler to work with constraints, in the form of must-links and cannot-links. They modify the collapsed Gibbs sampler to not violate any of the must-links or cannot-links, thus using the prior information within the clustering scheme. This constrained adaptation could be an interesting approach in a semi-supervised framework for the DPCK. This will be briefly discussed in chapter 11.

Dirichlet processes have not only been used in clustering, they are often used as a prior on the number of learners to use in an ensemble problem. In [23] they introduce a framework where they split data into subgroups and use the Dirichlet process to infer the number of subgroups of data to train the models on. In [24] they want to learn a logistic-regression model [25, 26] for multi-task classification. Multi-task learning is a way to exploit commonalities between different learning tasks. In [24] they develop a model with a Dirichlet process prior over the number of logistic-regression models to use. The model finds similarities between the different tasks and infer on the number of logistic-regression models to use. This approach of inferring the number of ensembles to use with a Dirichlet process looks interesting. However, the specific methodologies proposed in these papers only work in supervised problems.

There exists little directly related work on cluster kernels and Dirichlet processes. Something close is found in [27], where they cluster phrase categories using an ensemble of DPMM. However, they do not use the posterior probabilities to create the kernel. Instead they combine the different clusterings by counting the number of times each pair of observations are clustered together. To obtain the final clustering they apply a spectral clustering algorithm, described here [28].

1.4. Outline of the Thesis

The thesis is divided into four parts.

Part 1: Theory, Concepts and Methods.
The first part of the thesis describes the background theory and concepts.

- **Chapter 2:** Discussion of background theory in kernels, clustering and the ensemble clustering framework.
- **Chapter 3:** First the theory behind statistical modeling is introduced, then random processes and the Dirichlet process are discussed. Finally, the mixture models are defined.

- **Chapter 4:** The chapter combines the previous theory and concepts to discuss the PCK framework.

Part 2: Proposed Method; The Dirichlet Process Cluster Kernel
The second part contains one of the contributions in the thesis. The theory and motivation behind the DPCK is explained.

- **Chapter 5:** The framework of the DPCK is proposed and the algorithm is derived.

Part 3: Experiments.
The third part contains the experiments conducted in the thesis.

- **Chapter 6:** Investigation of the parameters in the DPMM.
- **Chapter 7:** Experimental comparison of the PCK and the DPCK.
- **Chapter 8:** The DPCK is applied to some semi-supervised problems.

Part 4: Conclusion.
The fourth and final part contains a summary of the thesis and a discussion about the results obtained. Finally, some future work is discussed.

- **Chapter 9:** Summary of the thesis.
- **Chapter 10:** Discussion of the performance of the DPCK. With discussion about application in semi-supervised classification problems.
- **Chapter 11:** Discussion of future work based on and around the framework of DPCK.

Part I.

Theory, Concepts and Methods

2. Kernels, Clustering and Ensemble Clustering

In data analysis we are concerned with predictive modeling, given some training data we want to predict the behavior of unseen test data. We call this task **learning**, and we make a clear distinction between learning problems that are **supervised** (classification) and **unsupervised** (clustering). In supervised learning we have labeled data, the ground truth for the data categories. Unsupervised learning is to analyze data with unlabeled data [29]. Generally we speak of two different data analysis techniques: **exploratory** and **confirmatory**. In exploratory analysis we wish to understand the underlying structure or characteristics of the data set [30]. Confirmatory is when you wish to confirm or validate a set of hypotheses [31]. In some applications it is natural to have some mix between labeled data and unlabeled data, as such a field of growing interest is called **semi-supervised** learning. In semi-supervised learning we often have a small portion of the data as labeled data, while the rest remain unlabeled. The unlabeled data are also used in the learning process. In clustering semi-supervised learning is used by encoding prior knowledge by specify pair-wise constraints, a weaker way of encoding the class labels [32].

The focus of the thesis is unsupervised learning and clustering. Organizing data into sensible groupings is one of the most fundamental methods of understanding and learning within exploratory data analysis. Cluster analysis is the formal study of methods and algorithms for grouping objects according to measured or perceived characteristics or similarities [33]. We know that clustering is an unsupervised method, meaning that we do not use category labels to find structures in data. As such clustering is exploratory in nature and a challenging problem with a vast literature available. A lot of problems can be formulated as clustering challenges. Some of these problems is image segmentation [34], document clustering [35] and structure analysis of genome data (DNA) [36].

In this chapter we are going to introduce some methods for analyzing data. The first method is dimensionality reduction a way to visualize high dimensional data or reducing data complexity. Secondly we want to do a brief overview of kernels and kernel methods in machine learning. Kernel methods is going to be the basic framework of the thesis, as we later want to create a kernel by clustering the same data multiple time. Thirdly we are going to introduce some clustering algorithms, to conceptualize the ideas previously discussed. Finally we introduce cluster ensembles a flexible and robust framework for combining multiple clusterings of the same data.

2.1. Dimensionality Reduction

Within data analysis we are often interested in visualizing data, to explore and understand the data visually or avoid the curse of dimensionality [37][1]. In some cases we need to reduce the dimensions of our data to remove redundant information which increase computing speed and predictive accuracy. Applications for dimensionality reduction techniques are many but for the thesis we will focus on using it for visualizing data in two dimensions.

2.1.1. PCA

One method used to reduce the dimensions is called principal components analysis (PCA) [39]. In PCA we want to project the data onto the principal components such that we keep as much variance as possible. We want to find an orthogonal set of L basis vectors ($\mathbf{w}_j$) and the corresponding scores ($\mathbf{z}_i$). Such that the linear transformation $\mathbf{W} \cdot \mathbf{x}$ minimize the average reconstruction error (mean square error) [40]. Thus we have obtained a new basis for the data where we also have a score for the information contained in each basis (in terms of reconstruction error or loss of information). Once the optimal solution is obtained we can remove n basis vectors with the lowest corresponding scores and have a new space with fewer dimensions containing the most information.

The solution obtained when $\mathbf{W}$ contains the L largest eigenvalues with corresponding eigenvectors of the covariance matrix is called PCA [40]. In figure 2.1 we see the principal components of a bivariate Gaussian sample, the length correspond to the scores of information (eigenvalues). If we projected all the data down on the largest principal component we would loose less variance then if we used the smaller one.

[1] The curse of dimensionality is a term to describe to problem caused by the exponential increase in volume associated with extra dimensions to Euclidean space [38]

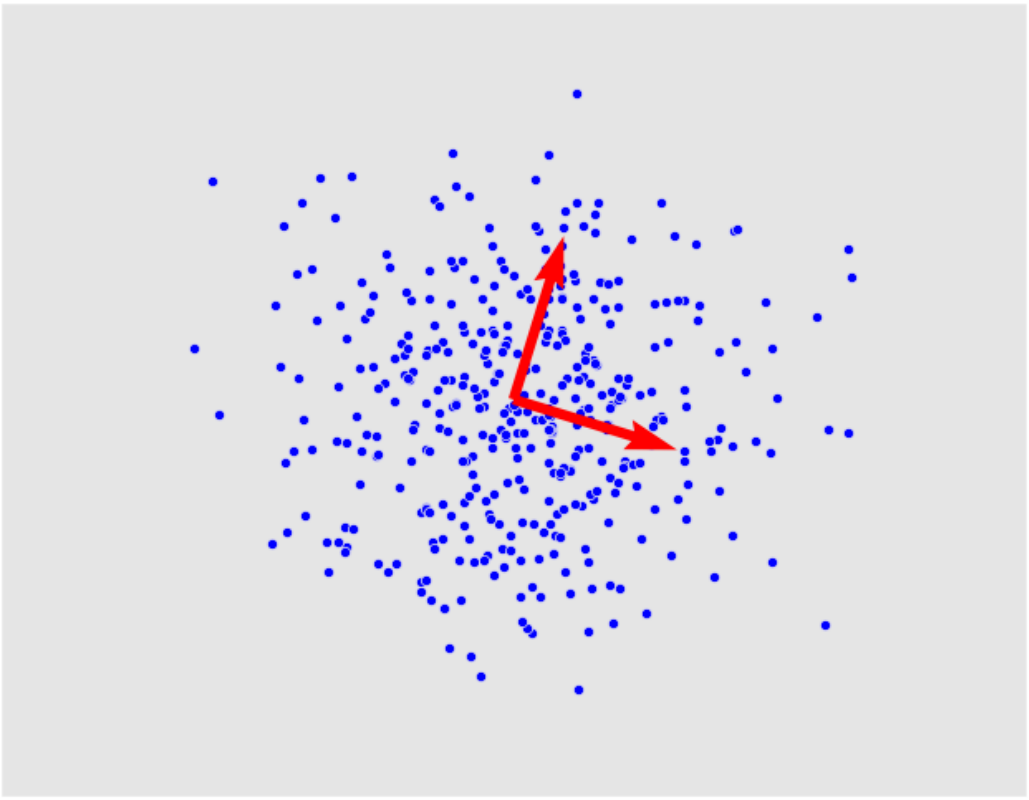


Figure 2.1.: The principal components (in red) of a multivariate Gaussian sample.

Example

In figure 2.2 we see all four different dimensions plotted against each other for the data set Iris species. We see that one class is separated from the other two, but the last two is not separable from each other. The problem that arise is how to visualize data when the number of dimensions grow, as plotting all them all is quickly not feasible. Thus we need methods to reduce the dimensionality when working with data. In figure 2.2 we see that projection the Iris data set on two principal components give good results. We see that one class is linearly separable from the other two, but the two last classes can not be linearly separated.

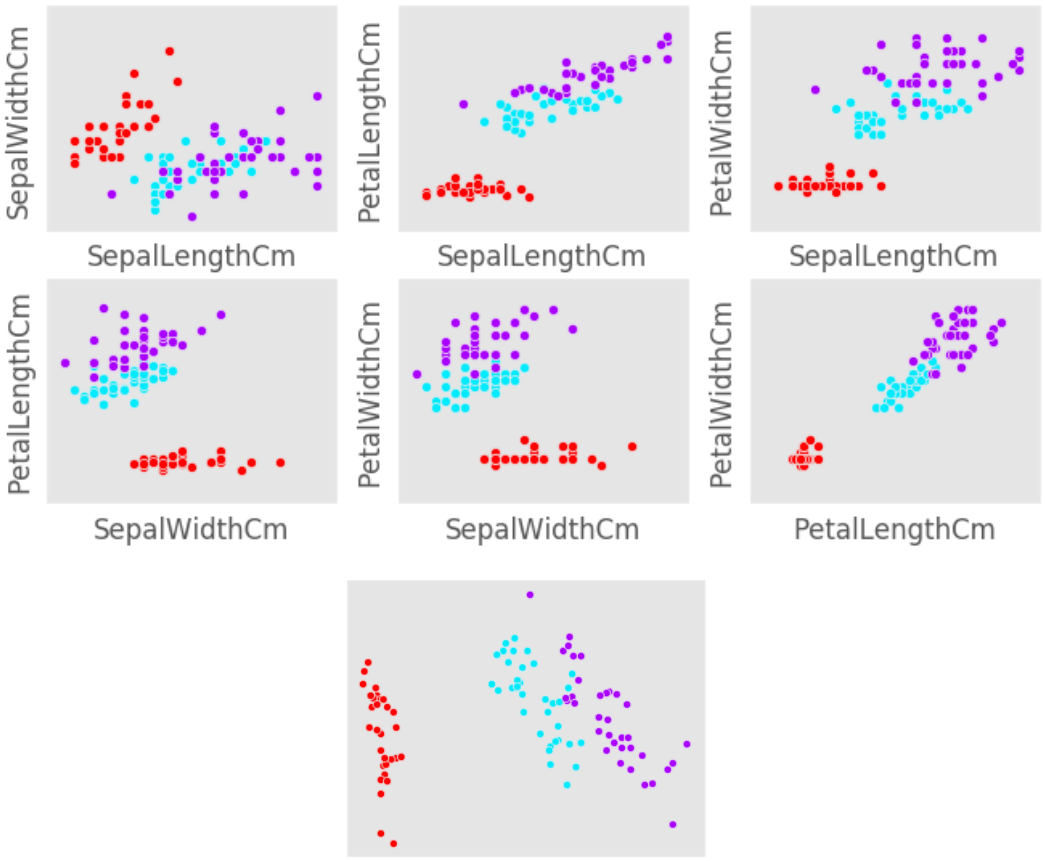


Figure 2.2.: The top two rows display different dimensions of the Iris data set. The figure of the last row is the data projected onto the two largest eigenvectors of the empirical covariance matrix (PCA).

In figure 2.3 we see a data set before and after using some feature preprocessing steps [2]. The figure illustrate that we can apply dimensionality reduction and visualize the effect of us working on it. We see that after working with the data set we did improve the discrimination for the two classes, and that we can visualize these effects.

[2]When preprocessing we are interested in adding and manipulating features such that different classes become more discriminant. Specific steps are outside the scope of the thesis.

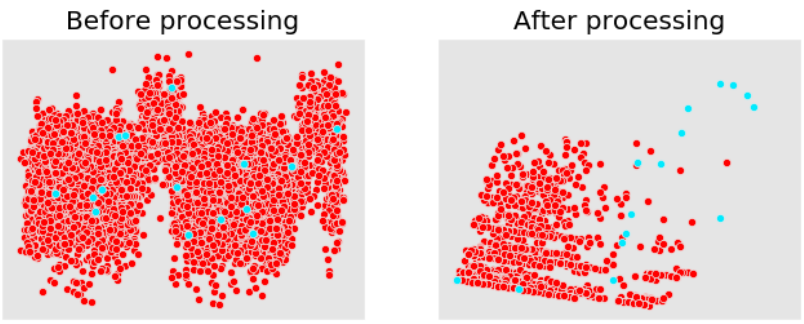


Figure 2.3.: Figure displays a data set projected down on the two largest eigenvectors of the covariance matrix, before and after doing some feature pre-processing. Different colors represent different classes.

2.2. Kernels

The theory behind kernels and reproducing kernels is beyond the scope of the thesis, for a discussion about these topics see [41–43].

In this section we try to summarize the main properties of a kernel that we need later in the thesis. We summarize the main properties through examples and demonstrations with the use of Kernel methods.

2.2.1. Kernel Methods

Kernel methods is a wide array of methods designed to learn non-linear boundaries with linear learning models [40]. The methods are designed in two parts: one that performs an implicit mapping from input space to feature space The second is a linear learning algorithm, an algorithm that discover linear boundaries [44]. We represent the implicit mapping as inner products in feature space in a matrix, called a kernel K. We describe one element of K as

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \tag{2.1}$$

Where $\Phi(\cdot)$ is the map from input space to feature space. In figure 2.4 we see a visualization of one such map. The map in this figure is created using the function which we call a polynomial kernel $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + c)^d$ (in the case of figure 2.4 with degree two and $c = 0$). The explicit map for the polynomial kernel is

$$\mathbf{x} = (x_1, x_2) \to (z_1, z_2, z_2) := (x_1^2, \sqrt{2}x_1x_2, x_2^2) \tag{2.2}$$

To explicitly map all the observations to the three dimensional feature space would be a tedious task, and impossible if we want to choose a infinite dimensional feature space. Luckily for us we can utilize the fact that we choose an algorithm that is optimized through inner products. Thus when changing the inner products to happen in feature space, we can replace the calculations with a call to the kernel matrix. Doing this we

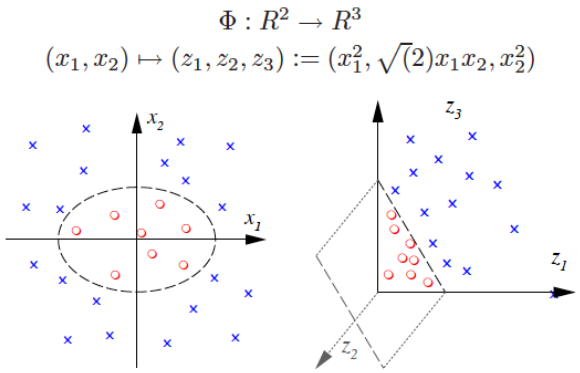


Figure 2.4.: Figure showing the main idea of kernel based methods, to map observations to a high dimensional space where they are linearly separable. The kernel trick is to only calculate the inner products in the new space, R^3. Source: Sven Laur, "SVM and Kernel methods for Graphs", Graph Mining 2011.

efficiently find a linear boundary in a high (or infinite) dimensional space. We call this the **kernel trick**.

In (2.3) we see how we represent one element of the kernel matrix K for the polynomial kernel function.

$$K(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle = \left\langle (x_1^2, \sqrt{2}x_1x_2, x_2^2), (y_1^2, \sqrt{2}y_1y_2, y_2^2) \right\rangle = (x_1y_1 + x_2y_2)^2 = (\mathbf{x} \cdot \mathbf{y})^2 \tag{2.3}$$

Some well known kernel functions are the polynomial kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + c)^d$ and the Radial basis function (RBF) kernel $k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right)$ [45]. One issue with these kernel functions is that they depend on some parameters that drastically changes the resulting feature space.

We know from [46] that **any** positive semidefinite matrix K can be thought of as a kernel. Thus we do not need to explicitly define any kernel function, we just need to create a positive semidefinite matrix in some way (that represent distances or similarities between observations). These two properties of kernels and existing kernel functions are fundamental in section 4.2. Where we construct such a kernel matrix by doing clustering on the same data set multiple times.

2.2.2. Support Vector Machines

Support vector machines are a well known classification algorithm widely known for its ability to use kernels to improve accuracy. The finer details about this algorithm is outside the score of the thesis, for further details see [47–50].

The Support vector machines (SVM) learns a linear decision boundary (the boundary separating two classes) between classes using inner products to optimize the distance to

the boundary. Thus we can use kernel functions to replace the inner products in input space to feature space.

2.2.3. Example using Kernel Principal Component Analysis and Support Vector Machines

Recall that in section 2.1.1 we introduced a way to create a low-dimensional (linear) embedding of some data [51]. In PCA we project on the principal components defined on the empirical covariance matrix. Recall that we find the empirical covariance matrix, and project on the eigenvectors to calculate the principal components.

However we can perform PCA in feature space following the reasoning in [52]. We find the empirical covariance matrix in feature space in (2.4) and note that the solution to this eigenvalue problem depend only on the inner products in this space. Thus we use the kernel trick to find a non-linear embedding of the data, we call this method the kernel principal components analysis (KPCA).

$$\bar{\boldsymbol{S}} = \frac{1}{N}\sum_{i=1}^{N}\Phi(\mathbf{x}_i)\Phi(\mathbf{x}_i)^T \tag{2.4}$$

In figure 2.5 we see the decision boundary when training a SVM using no kernel, the polynomial kernel (degree 5) and the RBF kernel. The top row show a plot of the data projected down on the top two eigenvectors of the kernel matrix using KPCA. We observe that we can greatly increase the classification accuracy, without a great loss in speed, using kernel function and the kernel trick.

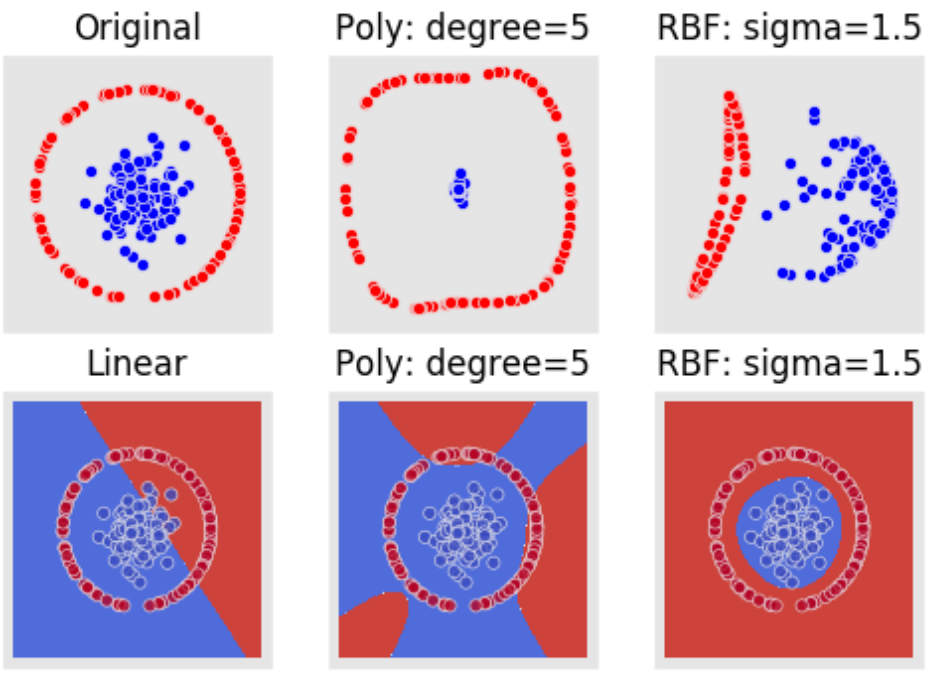


Figure 2.5.: Figure showing artifical data and the effect of mapping the data using the polynomial kernel and the RBF kernel, then projecting using KPCA. The bottom row shows the boundary when training the support vector machine with different kernels. Colored points correspond to ground truth.

2.3. Clustering

In [53] they define the goal of clustering; as to separate a finite unlabeled data set into a finite and discrete set of natural data structures. Having unlabeled observations increase the difficulty of the problem, because what the best result is depend on who is interpreting the results (and how!). As such clustering is a highly subjective process which rule out an absolute judgment as to the efficiency of different clustering algorithms. This case is further supported in [54], where they argue that in cluster analysis a group of objects are split into a number of homogeneous subgroups on the basis of an *subjectively* chosen measure of similarity.

The operation of clustering and grouping things occur naturally for people, and we are very good at finding them, and it is something we do on a daily basis. An example is through different groupings of species, as one group can be dogs and another can be cats. Looking through observations of different animals, we could argue that based on some set of specific characteristics (weight, hight, eye color ...) there should be a way to separate dogs from cats. In many cases we would know how many groupings we are looking for like in the dog or cat example. In reality this might not be true, and you

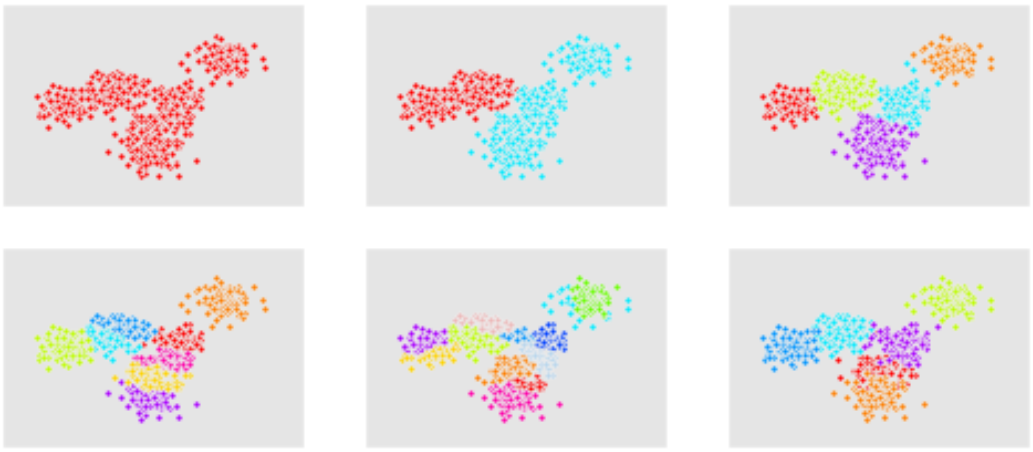


Figure 2.6.: Figure illustrating the difficulty of choosing the appropriate amount of clusters.

have to guess, estimate or use other tools to find the number of groupings.

If one want to separate all the animals into land and non-land animals, based on one characteristic; if they are observed on land or in water. Immediately we would have a problem with animals observed both on land and in water. We might improve our results by adding an additional possible grouping, hybrid animals. This example illustrate one of the main application of clustering, as it is unsupervised it can be used as a data mining tool to help build knowledge and understanding of your data. In figure 2.6 we try to visualize the difficulty of choosing the appropriate number of clusters.

The normal application to clustering is when you assume that each observation only belong to one group, this is called hard clustering. But some algorithms allow for each observation to belong to more than one group, you have a measure of membership (observation i belong to group a and b with weights $0.3, 0.7$ respectively). This method of clustering is called **fuzzy clustering** or soft clustering. Intuitively we can think of fuzzy clustering as a way of assigning pseudo probabilities for each observation to belong in the different clusters [40], it is worth noting that from a fuzzy clustering it is trivial to get a hard clustering for each observation, just assign to the most likely cluster. Additionally for fuzzy clustering we can say something about the uncertainty of assignments by looking at the unused weights.

2.4. Clustering Methods

Different clustering methods partition observations into a number of clusters or groups, based on different measures of (dis)similarity. From [53] we know that a assignments obtained from running a clustering algorithm, should have the property that observations in the same cluster should be similar to each other, while observations in different clusters should be dissimilar.

In this section we will first go through some basic clustering algorithms, then some more advanced ones using kernel methods.

2.4.1. K-Means

One of the most used clustering methods is K-Means , as the name indicate the methods is based on having K different means. We assign each observation to the closest mean based on some dissimilarity measure between means. Commonly we use the euclidean distance as a dissimilarity measure. Clearly we can modify the algorithm to work with similarity measures and different measures might be used for different tasks. K-Means is a versatile clustering algorithm applied with success in many different areas, with small modifications in (dis)similarity measures and/or definition of cluster means. Examples for application is document clustering [35, 55], image segmentation [56, 57] and bioinformatics to analyze Microarrray data [58]. We note that using euclidean distance create a linear boundary between classes.

The objective function that is minimized running the K-Means algorithms is the sum of squared errors between the mean of each cluster and all observations in different clusters [33] . The cost function for K-Means is the squared error between each mean, $\boldsymbol{\mu}_k$, and each cluster assignment set, $\mathbf{c}_k = \{\mathbf{x}_i \mid z_i = k\}$, written as:

$$J(\mathbf{c}_k) = \sum_{\mathbf{c}_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 \tag{2.5}$$

It can be shown that the K-Means algorithm minimize the sum over all, K different, $J(\mathbf{c}_k)$.

$$J(\mathbf{C}) = \sum_{k=1}^{K} J(\mathbf{c}_k) = \sum_{k=1}^{K} \sum_{\mathbf{c}_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 \tag{2.6}$$

Additionally we know that good initial means are essential for the success of partitioned clustering algorithms as K-Means [59], and K-Means is only guarantied to converge to local minima due to the random selection of initial cluster means [60]. However there is a high probability for global maxima if the cluster separation is high [33].

Algorithm 1 K-Means batch algorithm

1: Initialize $\boldsymbol{\mu}_k$ ▷ Draw K observations as initial means.
2: **while** Not converged **do**
3: $z_i = \arg\min_k \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$ ▷ Assign each observation to the closest cluster.
4: $\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{\mathbf{c}_k} \mathbf{x}_i$ ▷ Update each cluster mean with all observations belonging to the cluster.

In algorithm 1 we see that the standard K-Means algorithm can be modified with different distance measures and definition of cluster means without much trouble.

2.4.2. Spectral Clustering

Spectral clustering is a family of clustering methods which exploits the eigenvalues and eigenvectors of a similarity matrix to create a partitioning of the data. Spectral clustering have been applied to a wide range of problems, within image segmentation [34], document clustering [61] and self-driving cars [62]. The reason being the simplicity of implementing the algorithm. Additionally the clustering obtained is able to learn linear boundaries in high dimensional space. However even tho spectral clustering is an easy algorithm to implement, the theory behind is involved. Much of the theory behind spectral clustering lie outside the scope of the thesis. For a more in-depth discussion about spectral clustering see [63, 64].

In general we say that spectral clustering consist of four steps as described in algorithm 2. In general there is multiples way to create similarity graphs and graph Laplacians both normalized and not. As such there exists many different algorithms. We are going to briefly discuss these methods before showing an example.

Different Similarity Graphs

There are several popular constructions to transform a data set represented as a similarity matrix into a graph. Constructing Similarity Graphs we want want to model the **local** structures, e.g. local relationships between data points. Two such methods is the *The ϵ-neighborhood* graph and *k-mutual nearest neighbor* graph.

To create the ϵ-neighborhood graph we connect all observations with a distance $< \epsilon$. The k-mutual nearest neighbor graph is constructed by connecting observation i with j if j is one of the k closest neighbors of i. Or if j is one of the closest neighbors of i.

Graph Laplacians

The topic of Graph Laplcaians is outside the scope of the thesis, the interested reader can see [65, 66].

Spectral Clustering Algorithm

Algorithm 2 General Algorithmic Scheme for Spectral Clustering

1: **function** SpectralClustering((Similarity matrix, k))
2: Construct similarity graph from similarity matrix.
3: Compute Graph Laplacian.
4: Calculate k largest Eigenvectors $\mathbf{u}_i$ of Graph Laplacian and create the matrix $\mathbf{U} = (\mathbf{u}_1, ..., \mathbf{u}_k)$.
5: Cluster $\mathbf{U}$ using K-Means , where each row represent one observation.

2.4.3. Kernel K-Means

One drawback of K-Means is that it can only find linear boundaries between classes. A approach for tackling such problems is kernel K-Means . As the name indicate kernel K-Means lies within kernel methods from section 2.2. Thus we use K-Means in high dimensional space to learn nonlinear boundaries.

For generalization purposes we derive an weighted kernel K-Means . Let $\alpha_i = w(\mathbf{x}_i)$ be the weight for observation $\mathbf{x}_i$. Setting all weights equal to one we get an unweighted algorithm. For some kernel function Φ we introduce the weighted kernel K-Means objective function following [67]. Let $\boldsymbol{\pi}_j$ contain the indexes of observations assigned to cluster j.

$$\begin{aligned} J\left(\{\boldsymbol{\pi}_j\}_{j=1}^k\right) &= \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \boldsymbol{\pi}_j} \alpha_i \|\Phi(\mathbf{x}) - \mathbf{m}_j\|^2 \\ \text{Where } \mathbf{m}_j &= \frac{\sum_{\mathbf{x}_n \in \boldsymbol{\pi}_j} \alpha_n \Phi(\mathbf{x}_n)}{\sum_{\mathbf{x}_n \in \boldsymbol{\pi}_j} \alpha_n} \end{aligned} \tag{2.7}$$

Where we expand the distance calculation in feature space as

$$\begin{aligned} \|\Phi(\mathbf{x}_i) - \mathbf{m}_j\|^2 &= \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_i) - \\ \frac{2\sum_{\mathbf{x}_n \in \boldsymbol{\pi}_j} \alpha_n \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_n)}{\sum_{\mathbf{x}_n \in \boldsymbol{\pi}_j} \alpha_n} &+ \frac{\sum_{\mathbf{x}_n, \mathbf{x}_c \in \boldsymbol{\pi}_j} \alpha_n \alpha_c \Phi(\mathbf{x}_n) \cdot \Phi(\mathbf{x}_c)}{\left(\sum_{\mathbf{x}_n \in \boldsymbol{\pi}_j} \alpha_n\right)^2} \end{aligned} \tag{2.8}$$

Thus the Euclidean distance from $\Phi(\mathbf{x}_i)$ to a cluster mean $\mathbf{m}_j$ can be calculated efficiently using the kernel matrix. As discussed in [68] the kernel K-Means cost function is identically to standard K-Means except for the fact that distances are computed using the kernel matrix. If we choose all α's to be one and Φ to be the identity function algorithm 3 simplify to the standard K-Means .

Algorithm 3 Weighted Kernel K-Means batch algorithm

1: Initialize k clusters from input $\{\boldsymbol{\pi}_j\}_{j=1}^k$ else randomly.
2: **while** Not converged **or** max iterations reached **do**
3: Compute $d(\mathbf{x}_i, \mathbf{m}_c)$ from (2.8) for all observations and clusters.
4: Assign all points to clusters $z_i = \arg\min_c d(\mathbf{x}_i, \mathbf{m}_c)$.
5: Update cluster means with respect to new assignments.

Additionally they prove in [69, 70] that a general weighted Kernel K-Means objective is mathematically equivalent to a weighted graph partitioning objective. Thus we can use weighted Kernel K-Means to directly optimize the graph partitioning objectives, without computing the eigenvectors.

2.5. Ensemble Clustering

One characteristic of clustering, is the lack of truth about the data. Therefore optimizing the clustering process with respect to clustering algorithms and parameters are a

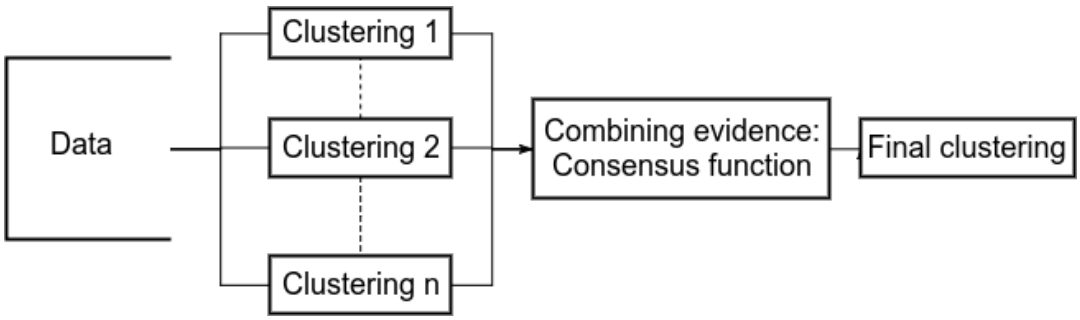


Figure 2.7.: Diagram displaying the general process of ensemble clustering.

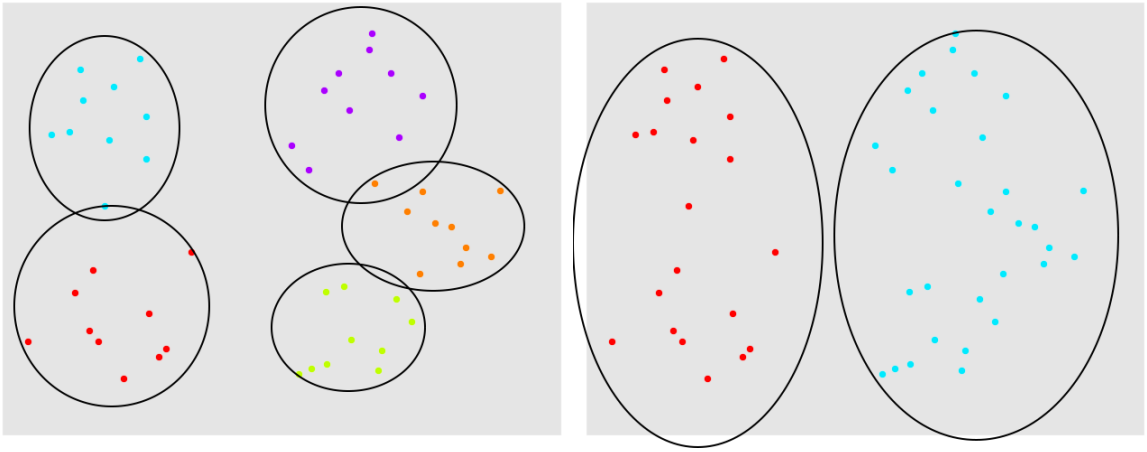
highly subjective task. Ensemble methods is one framework that try to deal with these problems. The basic idea, is to cluster the same data multiple times with different algorithms and parameters and combine the results. The justification is that as we cluster the same observations in the same clusters we accumulate evidence that this is true [71]. Ensemble methods have been adopted and used in classification with good results for a long period of time. Mainly through boosting [72, 73] and bagging [74].

Clustering is an unsupervised method, we can not use methods from supervised methods to optimize parameters and initializations. As such the main goal of ensemble clustering is to have an algorithm which is robust to both parameters and initializations [75]. In figure 2.7 we see a diagram of the general process of ensemble clustering. Looking at the figure we divide the process up into three parts. First we have to create n different clusterings, secondly we have to combine the results from multiple clusterings. Finally we have to recover a final clustering based on all n results. However, it is worth nothing that ensemble clustering comes at a cost, as we have to create n different clusterings we increase the total computation time needed. Further the n different clusterings might use to much computer resources to be feasible for large data sets.

In this section we discuss different ways to create clustering ensembles (different clusterings), then we discuss some challenges with combining the results. Finally we briefly show some methods to recover the final clustering.

2.5.1. Producing Clustering Ensembles

When producing different clustering ensembles we need the different ensembles to be diverse enough to capture different structures of the data. Initiating the same algorithm with different parameters and initial conditions we capture different structures within the data [76, 77]. In figure 2.8 we see an illustration of this. In figure 2.8 a) we see that we learn local scales with five clusters. In figure 2.8 b) we learn global structures by using two clusters. One way of doing this is to cluster the data into different number of clusters for each ensemble, where two clusters would map to global structures and 15 would give local structures. In figure 2.9 we see some methods which we can use to create different clustering ensembles. To project to different subsets we can randomly choose a subsample from the original sample. To increase diversity we can sample with and without replacement, and change the subsample size.



(a) (b)

Figure 2.8.: a) Clustering partition with local scales. b) Clustering partition with global scales.

Different subspaces can be obtained by many approaches. We can choose to only use a subset of the variables available, or we can project on $p \leq D$ principal components using PCA. Another methods for embedding is Laplacian eigenmaps [78], a method to preserve local structures in the projection.

To get diversity in the cluster ensembles we can change the hyper parameters in each ensemble, such as varying the number of clusters etc. Different clustering algorithms can be combined with all the other methods mentioned.

We can combine the different methods to further increase diversity. For example we can choose a subset of data and project it down on the two largest eigenvectors using PCA and initiate with a random number of clusters. Then we can do something totally different for the next ensemble, using the full sample but only choosing two of the variables on a different clustering algorithm with a different number of clusters.

It should be mentioned that even if the problem scales with the amount of clustering ensembles, each ensemble is produced independently of the others. Producing the clustering ensembles is an embarrassingly parallel problem [71], [79].

2.5.2. Combining the Evidence

As we saw in the previous section there is many ways to ensure diversity in our clustering ensembles. The challenge that arise is to combine the ensembles and accumulate the evidence in such a way that it is invariant to relabeling and the number of clusters. We call the function that combines the ensembles the consensus function. From the literature we know that there exists many different consensus functions.

In [71] they use a counting function, where they count each time two observations is in the same cluster and normalizing by the number of ensembles. This create a $N \times N$ matrix representing similarities between observations.

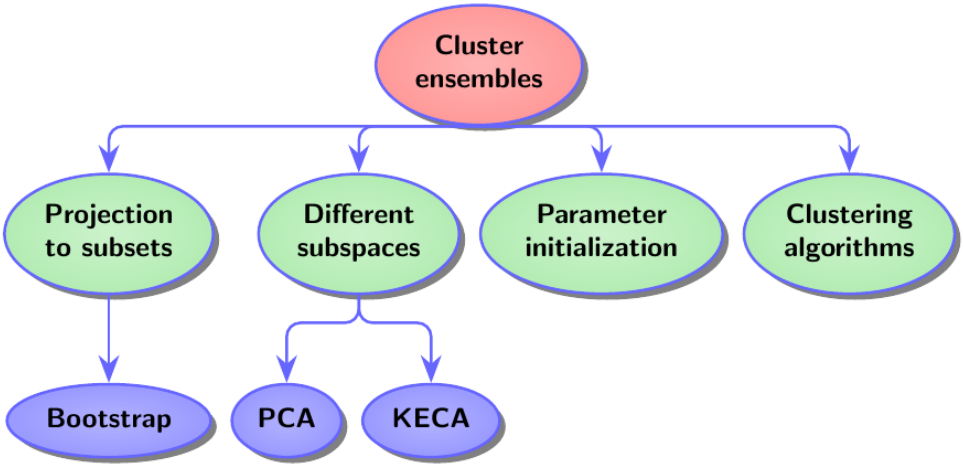


Figure 2.9.: Some methods to create different cluster ensembles.

Other consensus functions used in the literature is, but not limited to: relabeling and voting [80, 81], Co-association matrix based methods [82], graph and hypergraph based methods [83], finite mixture models based methods [84] and many more.

2.5.3. Final Clustering

Once we have the outcome of the consensus function we need to have a method to recover one final clustering and make use the accumulated evidence. This depend fully on the consensus function used in the previous step, due to different consensus function having different outputs. The counting function previously explained output a similarity matrix, and from section 2.2 we know that this represent a kernel matrix. Thus we can use a number of methods to recover the final clustering. We can use is spectral clustering from 2.4.2 or kernel K-Means from section 2.4.3 as these methods work by inputting a similarity matrix.

Furthermore we note that the use of these kernels is not limited to clustering, as they can be used in both supervised learning, through SVMs, and dimensionality reduction with KPCA.

3. Dirichlet Process Mixture Models

We briefly discussed some of the obstacles within unsupervised learning and clustering. As we have no ground labeling available we have no way to verify the final clustering. Choosing the number of clusters is an hard task as the dimension of the data increases, when we can not visually plot the data. The aim of this chapter, is to first discuss statistical background theory and methods such that we can introduce a statistical clustering model. Finally we aim to introduce the nonparametric mixture model, the DPMM, this is the base clustering model we will apply to the current cluster kernel framework. One advantage of the DPMM is that it infers the number of clusters when learning the model.

Before introducing the DPMM, we review and discuss some statistical theory. We start by introducing the notion of statistical distributions and statistical models. Then we expand the framework to Bayesian statistics, and give some examples. Further, we explore random processes, especially we spend some time on laying the framework of the Dirichlet process. Then we move into Markov chains and the world of numerical statistics using Markov chain Monte Carlo (MCMC), where we derive the Metropolis-Hastings algorithm (MH-algorithm) and the Gibbs sampler. Additionally we discuss bootstrapping for creating a confidence interval and doing hypothesis testing. Finally we derive the collapsed Gibbs sampler for the GMM, before taking the infinite limit to get the DPMM.

3.1. Statistical Distributions

It can be more intuitive to visualize and grasp properties of an univariate distribution, compared to multivariate distributions. The aim of this section is to review the univariate distributions of the multivariate ones we are going to use later. Intuitively we want to use the simple, intuitive, visualization to generalize some properties to the multivariate distributions.

3.1.1. Student's t-Distribution

Later we are going to look at the multivariate Student's-distribution. Therefore it is convenient to recall the main properties of the univariate Student's t-distribution before diving into the more advanced model.

The Student's t-distribution arises when estimating the mean of a normal distributed sample, where the σ is unknown. As such we say that the t-distribution describes samples draw from a full normal distributed sample. In equation (3.1) we see the probability

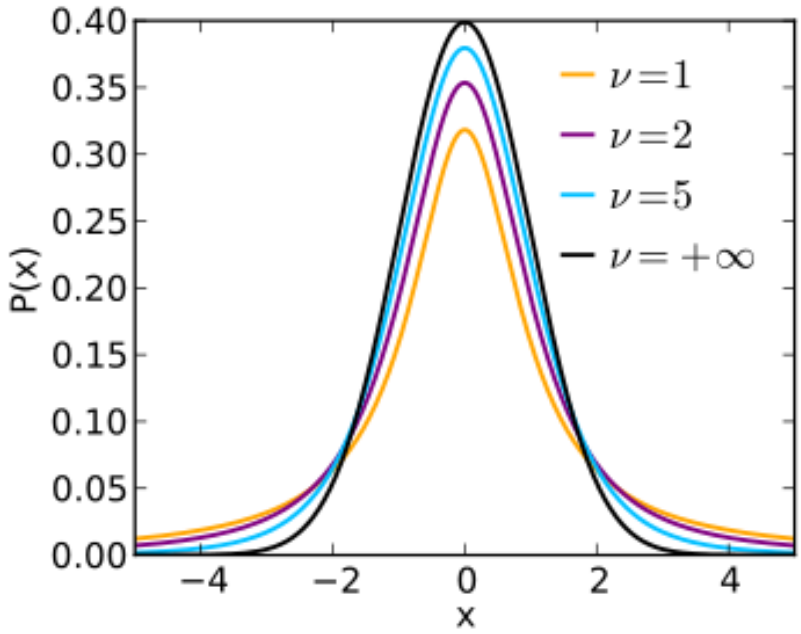


Figure 3.1.: Probability density function of the Student's t-distribution for different degrees of freedom ν. Source: Student's distribution - Wikipedia.

density function for the Student's t-distribution. The degree of freedom comes from the number of samples such that: $\nu = N - 1$, where N is the number of samples. We see in figure 3.1 that as we increase the samples N we approximate the normal distribution, in applied context we say that around 20-30 they are approximately equal. Additionally we observe that the Student's t-distribution is just a wider normal distribution (more uncertain).

$$T(x;\ \nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\ \Gamma(\frac{\nu}{2})} \left(1 + \frac{x^2+1}{\nu}\right) \tag{3.1}$$

In figure 3.1 we see the probability density function of the Student's t-distribution for different values of freedom ν.

3.1.2. Dirichlet Distribution

In chapter 3, we want to introduce a model based on the Dirichlet process. The Dirichlet process is going to share a lot of the properties with the Dirichlet distribution. Just as the Dirichlet distribution does with the beta distribution. Before going through the details of the Dirichlet process we briefly discuss the main properties of the Dirichlet distribution.

The Dirichlet Distribution is a multivariate Beta distribution, in fact the marginal distribution of a Dirichlet distribution is the Beta distribution. As such the Dirichlet distribution is a way to sample probability vectors that sum to one. If we let $\mathbf{p} \sim \text{Dir}(\alpha)$

then $\mathbf{p} = \{p_1, p_2, ..., p_k\}$, where $\sum_{i=1}^{k} p_i = 1$. If $k = 2$ then $\text{Dir}(\alpha)$ simplifies to the Beta distribution.

From [85] we recite the main properties of the Dirichlet distribution.

$$\mathbf{p} \sim \text{Dir}(\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{k} p_i^{\alpha_i - 1} \qquad \boldsymbol{\alpha} = (\alpha_1, ..., \alpha_k) \tag{3.2}$$

$$\mathbb{E}(\mathbf{p}) = \frac{\boldsymbol{\alpha}}{\alpha_0}, \quad \text{Var}(\mathbf{p}) = \frac{\boldsymbol{\alpha}(\alpha_0 - \boldsymbol{\alpha})}{\alpha_0^2(\alpha_0 + 1)} \qquad \alpha_0 = \sum_i \alpha_i \tag{3.3}$$

$$Q_i \sim \text{Beta}(\alpha_i, \alpha_0 - \alpha_i) \qquad \text{Marginal Distribution} \tag{3.4}$$

$$(Q_{-i} \mid Q_i) \sim (1 - Q_i)\text{Dir}(\boldsymbol{\alpha}_{-i}) \qquad \text{Conditional Distribution} \tag{3.5}$$

$$(Q_1, Q_2, ..., Q_i + Q_j, .., Q_k) \sim \text{Dir}(\alpha_1, \alpha_2, ..., \alpha_i + \alpha_j, ..., \alpha_k) \qquad \text{Aggregation Property} \tag{3.6}$$

In figure 3.2 we see plots for different values of $\boldsymbol{\alpha}$. We can think of α_i as a weight parameter, if one of the α_i are greater then the other then we will have more samples from that area. Equal values will result in an uniform sample, inside the probability simplex. The proportions between α_i decides the variance, we see in figure 3.2 that as α_i increases the variances decreases. The Dirichlet distributed sample is generated using the Pólya's Urn algorithmic scheme, for additional details see [86, 87].

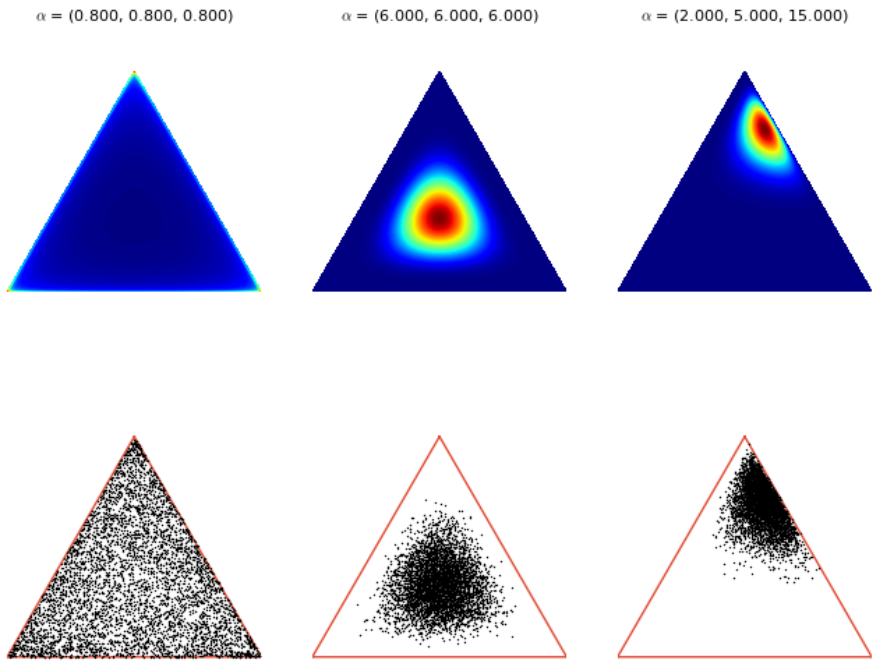


Figure 3.2.: Figure displays the Dirichlet distribution simplex for different values of α.

Stick-breaking

The stick-break approach to generating a random vector with a $\text{Dir}(\boldsymbol{\alpha})$ distribution is going to prove useful when we introduce the Dirichlet process in section 3.3.2. Suppose we start with a stick with length one and we want to break it into n pieces such that the lengths of the n pieces follow a $\text{Dir}(\boldsymbol{\alpha})$ distribution [85]. From (3.5) we observe that the conditional distribution for the second piece given the first is:

$$(Q_i, Q_{i+1}, ..., Q_k) \mid (Q_1, ..., Q_{i-1}) \sim \prod_{j=1}^{i-1}(1 - Q_j)\text{Dir}(\alpha_i, \alpha_{i+1}, ..., \alpha_k) \tag{3.7}$$

and the marginal distribution from (3.4) give us the distribution for piece i.

$$Q_i \mid (Q_1, Q_2, ..., Q_{i-1}) \sim \prod_{j=1}^{i-1}(1 - Q_j)\text{Beta}(\alpha_i, \sum_{j=i+1}^{k} \alpha_j) \tag{3.8}$$

Figure 3.3 illustrate the distribution of each piece as we break it and figure 3.4 show some realizations for $\boldsymbol{\alpha} = (10, 5, 3)$.

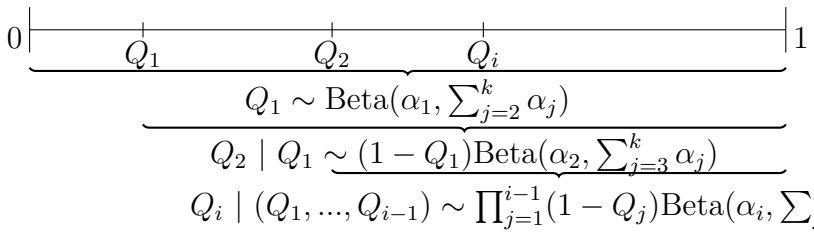


Figure 3.3.: Figure displaying how the stick is distributed according to the Stick-breaking representation.

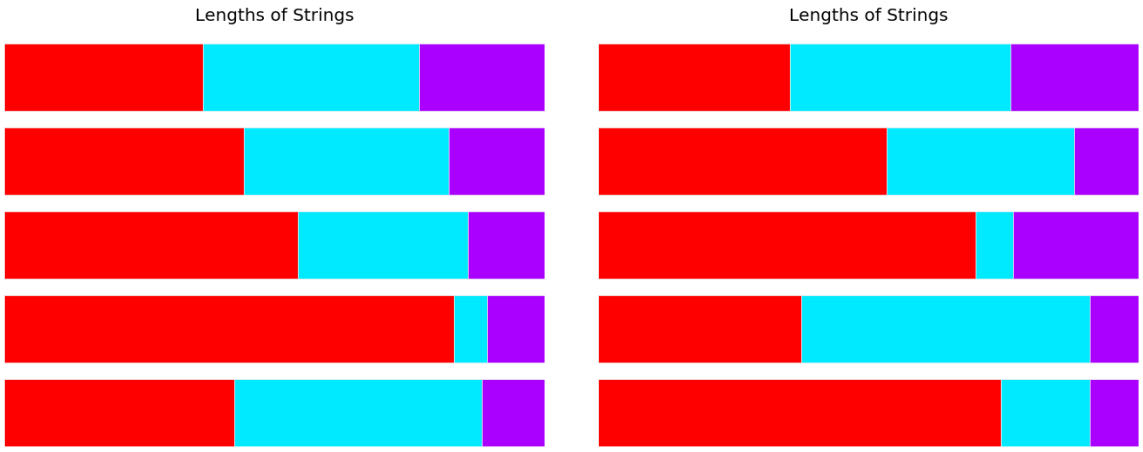


Figure 3.4.: Some realizations of the Dirichlet distributed stick for $\boldsymbol{\alpha} = (10, 5, 3)$

3.2. Statistical Models

The term model will be used from time to time throughout the thesis. Talking about a model we talk about a statistical model on some sample space Ω. This is in fact a set of **probability measures** on Ω parameterized by some parameter θ [88].

Terms that will be used and discussed are **parametric** and **nonparametric** models. According to [88] the difference is on the dimension of the parameter space; if the dimension of θ is finite we call the model parametric. Equally the model is called a nonparametric model if θ has infinite dimension. Additionally we call a distribution on an infinite dimensional space a **stochastic process**.

3.2.1. Statistical Inference

In the classical statistical approach we say that inference is done in three steps:

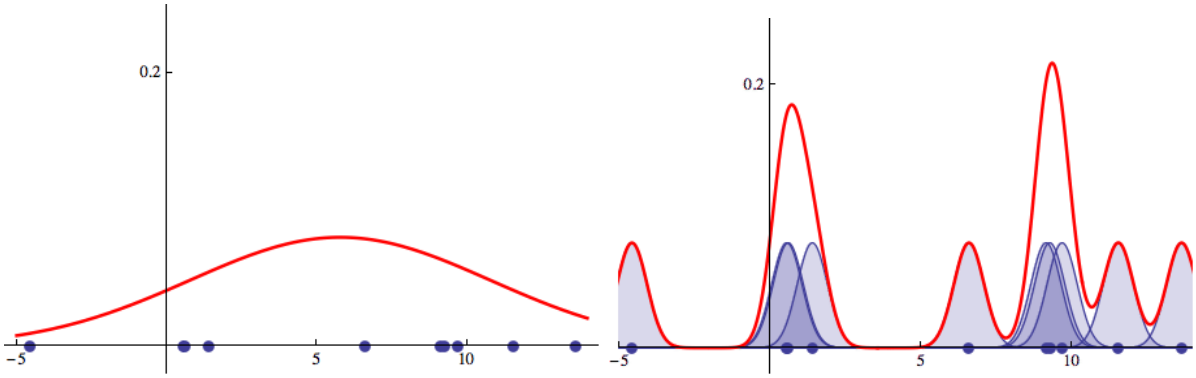
1. Assume that n observations are from a sample space Ω, such that $\mathbf{X} = \{\mathbf{x}_i \mid \mathbf{x}_i \in \Omega\}$
2. Model the observations as random variables from a probability measure in the model, i.e.
$$\mathrm{X}_1, ..., \mathrm{X}_n \sim_{iid} P_\theta \tag{3.9}$$
3. Draw conclusion about the value of θ, and hence the distribution of P_θ, from the observations.

3.2.2. Example of Statistical Modeling

To illustrate the difference between parametric and nonparametric methods we study a classical parametric and nonparametric density estimation example. Suppose we observe some real valued data $\mathbf{x}_1, ..., \mathbf{x}_n$, and would like to get an estimate of the underlying density. Using classical parametric density estimation we assume that our observations come from some normal distribution. We then use maximum likelihood estimation to estimate the mean and standard deviation.

For the nonparametric density estimation we use a method called Kernel Density Estimation (KDE), where we assume that the observations are from a mixture of normal distributions. We add a new normal distribution with mean $\mathbf{x}_i$ for each observation with some fixed σ and average all of these to estimate the density. Figure 3.5 illustrates the two approaches.

Comparing the number of parameters ($\mathrm{Dim}(\theta)$) for each of the estimators, we see that the Gaussian maximum likelihood estimate have two degrees of freedom. The mean and standard deviation are fixed for all sample sizes n thus it is a parametric approach. Using KDE we need an additional parameter for each additional data point, the position of each observation. The dimension of the parameter space increase linearly with n, and we say that KDE is a nonparametric approach.



(a) (b)

Figure 3.5.: Density estimation using Gaussians. a) Maximum likelihood estimation. b) KDE. Source: [88]

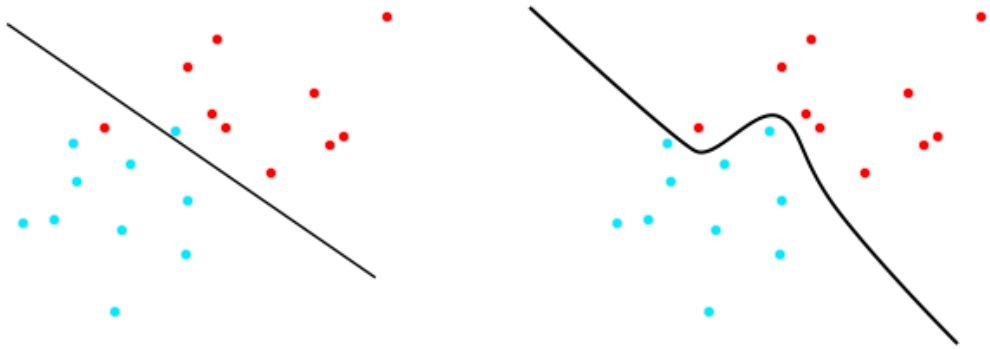
3.2.3. Bayesian Framework

In Bayesian modeling, probabilities are used to describe inferences and to quantify the degree of belief about the parameters [89]. We model parameters as random variables where the randomness reflects our lack of knowledge about them, so called prior probability distribution (representing the degree of belief about model parameters prior to observing any data). Once the prior and the likelihood distribution have been decided the model is completely specified, we use the following notation for a Bayesian model.

$$f(\theta \mid \mathbf{X}) = \frac{f(\mathbf{X} \mid \theta) f(\theta)}{\int f(\mathbf{X} \mid \theta) f(\theta) \, \mathrm{d}\theta} \tag{3.10}$$

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}} \tag{3.11}$$

Therefore using a Bayesian framework we allow for prior knowledge to be incorporated within the model. As discussed in [90] we need to be careful when choosing a prior, as an over-specified prior will overfit and drag the posterior toward the more complex model (leaving the base model with no support in the posterior even where the base model is the true underlying model). In figure 3.6 we see an example of this, where we want to find the boundary between two Gaussians. We see here that finding the optimal solution for new observations being sampled from the two Gaussians, a) would generalize better (the boundary between two Gaussians is a line).



(a) (b)

Figure 3.6.: Example of overfitting to the data.

Normally we divide Bayesian inference into three general steps [89]:

1. Formulating the probabilistic model, i.e. the joint distribution over data and parameters, including priors.
2. Infer the parameters of the posterior distribution.
3. Checking model fit and computing quantities of interest.

Finally we summarize the model of which we assume our observations to come from in a Bayesian model:

$$\begin{aligned} \Theta &\sim f(\theta) \\ X_1, ..., X_n \mid \Theta &\sim_{iid} f(\mathbf{X} \mid \theta) \end{aligned} \tag{3.12}$$

It is important to note that the observations is only conditionally independent.

3.2.4. Conjugate Priors

We recall that in a Bayesian framework we need to make assumptions about the distribution of parameters. Choosing a prior distribution in the same family of distributions as the likelihood implies that the posterior will have the same distribution as the prior. This kind of prior is called **conjugate prior** and it is widely used in statistics for its nice mathematical convenience [91]. A conjugate prior gives a closed form expression for the posterior.

The conjugate prior for a binomial distribution is the beta distribution. To show an example of a conjugate prior let $f(\mathbf{X} \mid p) \sim \text{Bin}(k, p)$ with known k, and $f(p) \sim \text{Beta}(a, b)$

for a data set $\mathbf{X} = \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix}$.

$$f(p \mid \mathbf{X}) \propto f(\mathbf{X} \mid p)f(p) = \prod_{i=1}^{k} \left[p^{x_i}(1-p)^{n-x_i}\right] \cdot p^{a-1}(1-p)^{b-1} = p^{a+\sum_{i=1}^{k} x_i - 1}(1-p)^{b+n-\sum_{i=1}^{k} x_i - 1} \tag{3.13}$$

We observe that the posterior distribution is a beta distribution with updated parameters (thus the beta distribution is the conjugate prior for the binomial distribution). In figure 3.7 we see some example plots of the beta distribution.

$$f(\mathbf{X} \mid p)f(p) \sim \text{Beta}\left(\sum_{i=1}^{k} x_i + a,\ b + n - \sum_{i=1}^{k} x_i\right) \tag{3.14}$$

For a comprehensive review of conjugate priors for univariate and multivariate distributions see [92].

3.2.5. Example Using Bayesian Inference and Conjugate Priors

As an example using conjugate priors and Bayesian inference we will construct a model using the Bayesian framework to modeling batting-players in baseball. Assume that we have observed three players batting ten times each during a game, each batting can be a miss or hit. We want to model the batting average, expected value of the posterior distribution, for each player. We can model each player with a hit or miss as binomial distributed with known number of batting attempts n. The probability, p of a hit is unknown and we can use a beta distribution as the prior.

$$f(p \mid \mathbf{X}) \propto f(\mathbf{X} \mid p)f(p) \qquad \text{Posterior} \tag{3.15}$$
$$f(\mathbf{X} \mid p) = \text{Bin}(n, p), \qquad \text{Likelihood} \tag{3.16}$$
$$f(p) = \text{Beta}(a, b), \qquad \text{Prior} \tag{3.17}$$

Where $\mathbf{X}$ is the data matrix containing the player scores and $f(p)$ is the prior distribution for making a hit. From (3.14) we get the following posterior distribution for each player.

$$f(p \mid \mathbf{X}) = \text{Beta}\left(\sum_{i=1}^{k} x_i + a,\ b + n - \sum_{i=1}^{k} x_i\right) \tag{3.18}$$

As discussed in section 3.2.3 we can incorporate prior knowledge in our model. For our specific model we choose our parameters a and b in the following sense; the mean

batting average is around 0.27, and should reasonable range between 0.21 and 0.35 (**prior knowledge**). If we set $a = 81$ and $b = 219$ we get the right expected value and standard deviation for the beta distribution. In figure 3.7 we see the prior distribution and the posterior distribution for each player, calculated from (3.14) and table 3.1.

From visual inspection of figure 3.7 we observe that even one player achieve 10 hits, the batting average is still reasonable within what we would expect! We see that with a good chosen prior we get good results even for small sample sizes, we can think of this fact as giving our model a head start using prior knowledge. One thing to note is that when the sample size increases the prior knowledge is less useful for the model, looking at the posterior parameter $\sum_{i=1}^{k} x_i + a$, we see that in a larger sample $\sum_{i=1}^{n} x_i >> a$.

	Miss	Hit
Player 1	5	5
Player 2	8	2
Player 3	0	10

Table 3.1.: Table of hitting scores for three different players.

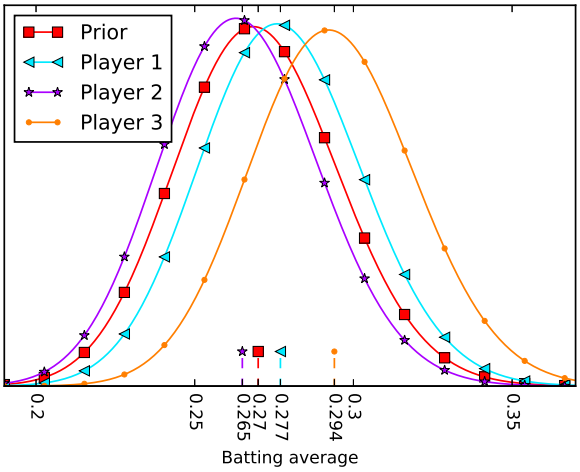


Figure 3.7.: Plot of the beta prior and posterior distribution for each of the players from table 3.1. The batting average prior is $\mu_p = 0.27$ and for each player the posterior is $\mu_1 = 0.277$, $\mu_2 = 0.266$ and $\mu_3 = 0.294$.

3.3. Random Processes

3.3.1. Stochastic Process

From 3.2 we know that a parameter space with infinite dimension is called a stochastic process. Additionally we know that a stochastic process is a collection of random variables indexed by some set, where all the random variables are defined on the same underlying set [85]. The random variables should be defined in such a way that there exists a joint distribution over the collection of random variables. Intuitively we can think of a stochastic process as a *random* process.

To give a better understanding of what a stochastic process is, we look at a simple example for a stochastic process. From the literature we define a a random walk model. A stochastic model where we start at time zero and at every time step $i+1$ we can go either up or down with probability p and $1-p$ respectively. The random variables is the value/position, where the index is at time i: $i = \{0, 1, ..., n\}$. In figure 3.8 we some realizations of this random walk model for different values of p.

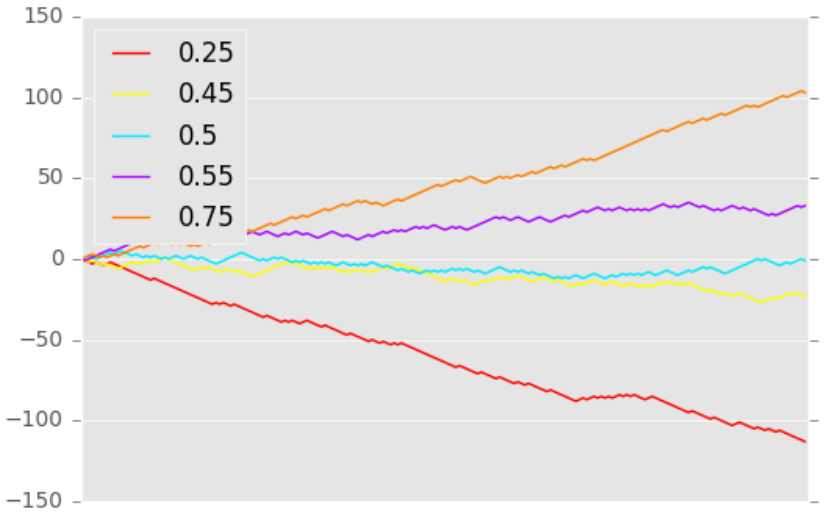


Figure 3.8.: Display of some realizations for a random walk model. Initiated at zero, with a probability p to go up and $1-p$ to go down.

3.3.2. Dirichlet Process

The aim of this section is to give an overview of Dirichet processes, for a more in depth discussion see [8, 93]. First we discuss the properties of the Dirichlet process analog to the Dirichlet distribution. Then we give an intuitive example of a related process. Finally we show an example of a sample generated from the Dirichlet process.

In section 3.1.2 we saw that the Dirichlet distribution is a multidimensional generalization of the beta distribution. Analog to this the Dirichlet process is an infinite dimensional generalization of the Dirichlet process. The marginal distribution of a Dirichlet process is Dirichlet distributed. With words we say that the Dirichlet process is a way to provide a random distribution over distributions over infinite sample spaces. We say that each draw from a Dirichlet process is itself a distribution [85]. If we have a bag of infinite number of dices, where each dice have some probabilities (and thus a distribution) assign to each side. We say that a draw from a Dirichlet process is one dice, where each dice (draw) can have different distributions. However, the mean of the Dirichlet process can be a dice with $p = 1/6$ for all sides, and we note that the mean of the Dirichlet process is a distribution.

To give an intuitive example we look at a related process, called Chinese restaurant process (CRP). The CRP is constructed such that the process represent a distribution

over the infinite partition of the integers. We start with a Chinese restaurant with an infinite number of tables, and let customers enter one at a time. The first customer enters and sit at the first empty table. Then we let the following customers have a chance to sit at a new table, or at a existing table with n customers. We let the probability of choosing a new table be proportional to some value α and the probability of sitting at an existing table to be proportional to the number of persons sitting at the specific table. At any point in this process the assignment of customers defines a **random finite partition** over $\mathbb{N}$ [94].

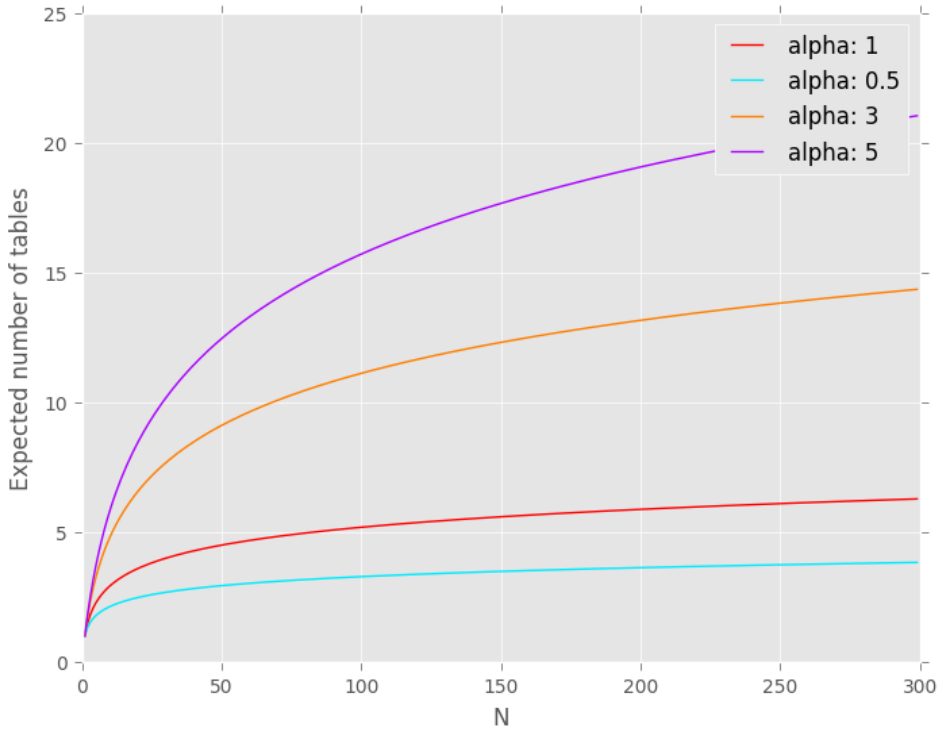


Figure 3.9.: Expected number of tables in a Chinese restaurant process for different values of alpha.

The posterior distribution given N customers have entered the restaurant is

$$t_{k+1} \mid t_1, ..., t_K \sim \frac{1}{\alpha + N}\left(\alpha H + \sum_{j=1}^{N} \delta_{t_i}(\cdot)\right) = \frac{1}{\alpha + N}\left(\alpha H + \sum_{i=1}^{K} N_i \delta_{t_i}(\cdot)\right) \tag{3.19}$$

where $t_1, ..., t_K$ is the K different tables. Additionally δ_{t_i} represent the position for table t_i drawn from the base distribution H (intuitively it is the location of the table inside the restaurant, but for this example we ignore the specific position). Furthermore we note that for $j \in [1, N]$ we do not have unique positions (sum up each customer). For $i \in [1, K]$ we sum up all the repeated positions (sum up each customer at the table) thus N_i is the number of customers at table i.

The table t_1 will be repeated by t_{k+1} with a probability N_1, customers sitting at the table. Thus the Dirichlet process inhabit a rich-gets-richer phenomenon, where large tables grow faster. In figure 3.9 we see the expected number of tables as a function of N customers for the CRP.

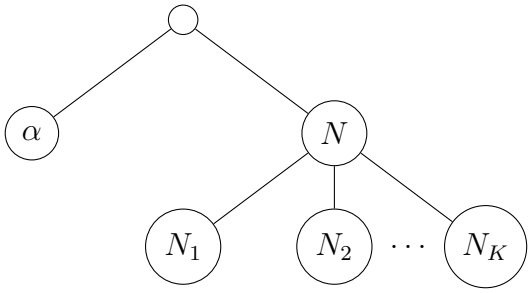


Figure 3.10.: Figure illustrating how customers enter the restaurant in a CRP, they can sit at a new table with probability (proportional to) α or sit at an existing table with probability (proportional to) $N = \sum_{i=1}^{K} N_i$, where N_i is the number of people sitting at table i.

Using the CRP from figure 3.10 and (3.19) we can create a realization from a Dirichlet process by assuming that each table is a bivariate normal distribution with some mean (position of each table lie in the xy-plane). When a customer enters he can sit at an existing table meaning that we draw a value from that bivariate normal distribution. If he chooses to sit at a new table we draw a mean for that table from the base distribution: the bivariate normal distribution. In figure 3.11 we see a realization using this base distribution CRP, Each plot represent the realizations at different number of observed customers. Note that in figure 3.11 we see the rich-gets-richer phenomenon clearly.

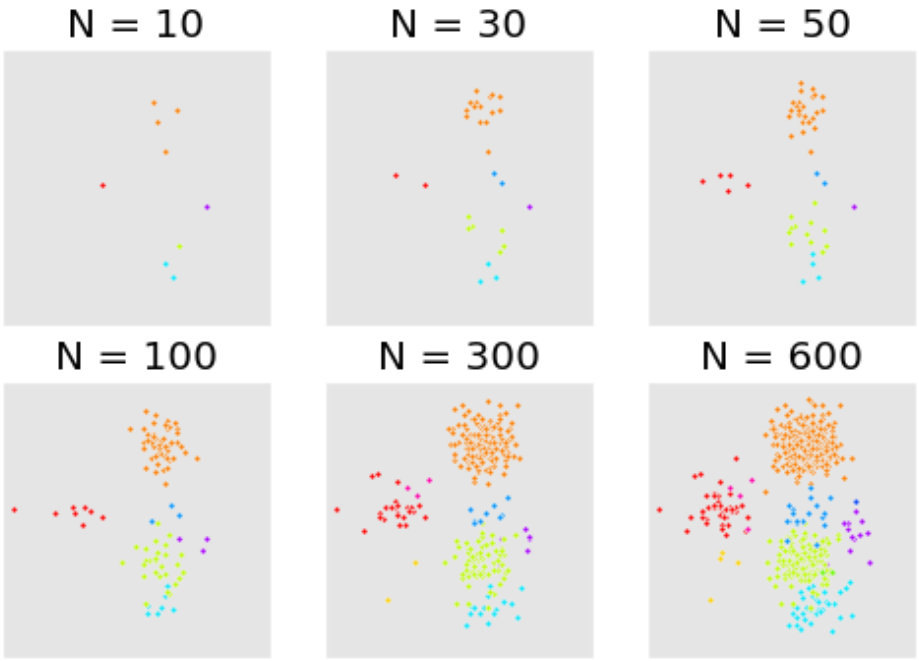


Figure 3.11.: Figure showing realizations using a CRP at different amount of observations.

3.4. Markov Chains

In section 3.5 we want to cleverly construct Markov chains to do numerical statistical modeling. However, before introducing those methods we refresh a few concepts from Markov chain theory. First we define the Markov property, then we discuss some of the possible states a Markov chain can have. Finally we introduce the notion of stationary distributions, which we will need in section 3.5.

We start by defining the stochastic process:

$$\{X_n \mid n \in \mathbb{N}\} \tag{3.20}$$

where X_n is some random variable, where we say that X_0 is the **initial state** of the process, and let $X_n = i$ denote that the process is in **state i** at time n. Next we say that the **Markov property** is when the next step only depends on the present state. We call this stochastic process a **Markov chain** if the Markov property hold for all time steps n. For a discrete Markov chain we show the Markov property:

$$P_{ij} = P(X_{n+1} = j \mid X_n = i, ..., X_0 = i_0) = P(X_{n+1} = j \mid X_n = i) \quad \forall\ n \geq 0 \tag{3.21}$$

We call P_{ij} the **transition probability** between state i and j, and is independent of n. The probability of moving from state i to state j. Additionally we call the matrix

containing all one step transition probabilities a **transition matrix**, $\mathbf{P}$.

$$\mathbf{P} = \begin{pmatrix} P_{00} & \dots & P_{0k} \\ \vdots & \ddots & \vdots \\ P_{k0} & \dots & P_{kk} \end{pmatrix}, \qquad P_{ij} \geq 0 \; \forall \; i, j \geq 0 \quad \text{and} \quad \sum_{j=0}^{\infty} P_{ij} = 1, \; \forall \; i \tag{3.22}$$

A Markov chain is fully specified by its initial state and the transition matrix. Additionally we denote the n step transition matrix as $(\mathbf{P})^n = \mathbf{P}^n$ [95]. These are the probabilities that the Markov chain moves from state i to j in n steps.

3.4.1. Classification of States

In Markov chain theory it is important to classify state of the Markov chain. In figure 3.12 we see a simple diagram of a Markov chain. For the thesis, one important state of a Markov chain is irreducibility. Later we will need this property to justify the existence of a stationary distribution.

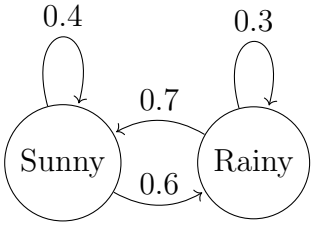


Figure 3.12.: Display of a Markov chain.

$$\mathbf{P} = \begin{pmatrix} 0.4 & 0.6 \\ 0.7 & 0.3 \end{pmatrix}$$

(a)

$$\mathbf{P}^5 = \begin{pmatrix} 0.5373 & 0.4627 \\ 0.5398 & 0.4602 \end{pmatrix} \qquad \mathbf{P}^{20} = \begin{pmatrix} 0.5385 & 0.4615 \\ 0.5385 & 0.4615 \end{pmatrix}$$

(b) (c)

Figure 3.13.: a) One step transition matrix. b) Five step transition matrix. c) 20 step transition matrix.

Two states in a Markov chain is said to communicate if they are accessible from each other, e.g. you can go from sunny to raining with some probability > 0. Additionally we say that all states that communicate is in the same class. Finally we say that a Markov chain is irreducible if there is only one class (all states communicate).

For a stationary distribution to exist we need an irreducible time reversible Markov chain. A time reversible Markov chain is a Markov chain where the proportion of moves

from state i to state j is equal the proportion of moves from j to i. Mathematically we define a time reversible Markov chain as a chain where:

$$\pi_i P_{ij} = \pi_j P_{ji}, \quad \forall\ i,\ j \tag{3.23}$$

π_i is the limiting probabilities for state i, and represent the proportion of time the chain is in state i.

If a Markov chain is both *irreducible* and *time reversible* then we know that the stationary distribution exists [95]. The reason for why we need the stationary distribution to exist, is obvious in section 3.5. In the example Markov chain in figure 3.13 we can see the approximated stationary distribution in figure 3.13 c).

3.5. Markov Chain Monte Carlo Methods

Bayesian statistics began with the with the introduction of Markov chain Monte Carlo (MCMC) methods. One of the main steps in Bayesian methodology is to infer parameters of the posterior distribution. Analytically tractable posterior distributions is often not the case, as such Bayesian statistics had its rise together with numerical methods [90].

In the following sections we first introduce Monte Carlo methods. The framework for numerically approximating expectations, given that you can sample from the distribution. Then we will discuss how we can construct Markov chains with a specific stationary distribution. Such that when we run the Markov chain we obtain a sample from a specific (target) distribution, we can use the previously discussed numerical methods of MCMC. Thus we introduce a robust framework for approximating posterior distributions.

We will use this framework later to discuss statistical clustering methods based on mixture models. Within these clustering schemes we need to numerically approximate some posterior distributions, where we will utilize MCMC methods.

3.5.1. Monte Carlo Method

We start by introducing Monte Carlo methods, based on approximating $\mathbb{E}(h(X))$ by drawing samples $\{X_i : i = 1, ..., n\}$, recall the connection between integration and expected value.

$$\mu_h = \mathbb{E}(h(X)) = \int_{-\infty}^{\infty} h(x) f_X(x)\, \mathrm{d}x \approx \frac{1}{n} \sum_{i=1}^{n} h(X_i) = \hat{\mu}_h \tag{3.24}$$

when X_i is an iid sample from f, for a function h. If we can generate a sample from f then we can evaluate the integral for *any* function h (If we can evaluate the function $h[f(\cdot)]$). In general, drawing samples independently from the distribution is not feasible, since the distribution can be non standard (it is shown that as long as we draw samples X_i throughout the support of the distribution they do not have to be independent) [96].

Further we observe that we can rewrite an integral in the following way

$$\int_{\mathbb{D}} f(x)\,\mathrm{d}x = \int_{\mathbb{D}} h(x)\frac{f(x)}{h(x)}\,\mathrm{d}x = \int_{\mathbb{D}} h(x)g(x)\,\mathrm{d}x\,, \qquad g(x) = \frac{f(x)}{h(x)}. \tag{3.25}$$

therefore from (3.25) we can approximate almost any integral, with the criteria being that we can sample iid from the function h and evaluate $g(h(\cdot))$. This way of using the integral to approximate the expected value of a sample from a distribution is called the **Monte Carlo method**. Monte Carlo methods is closely related to other numerical methods for solving integrals, except that here we sample our grid randomly instead of having it fixed (achieving many grid points in high probability space). It is worth noting that by varying the function f we can approximate many quantities of interest, such as expectation, variance and the median.

3.5.2. Example

Monte Carlo methods is useful in many applications and science areas, we can even approximate π using this toolbox in the following way: We know that the area of a circle with radius one is π, it is also equal to the integral:

$$A_I = \int_{-1}^{1}\int_{-1}^{1} \mathbb{I}(x^2 + y^2 \leq 1)\,\mathrm{d}x\,\mathrm{d}y \tag{3.26}$$

where $\mathbb{I}()$ is a indicator function. If we sample $x, y \sim \mathrm{Unif}(0, 1)$ we look at one forth of the full circle, such that we can use Monte Carlo integration to approximate π as:

$$\pi = A_I \approx \frac{4}{n}\sum_{i=1}^{n} \mathbb{I}(x_i^2 + y_i^2 \leq 1) \tag{3.27}$$

This method only require us to being able to do simple algebraic operations and sample $u \sim \mathrm{Unif}(0, 1)$. In figure 3.14 we see samples for the circle/square and the error as a function of n, together with a 95% confidence interval. We observe that to the number of MCMC samples needed to get accurate results is very high.

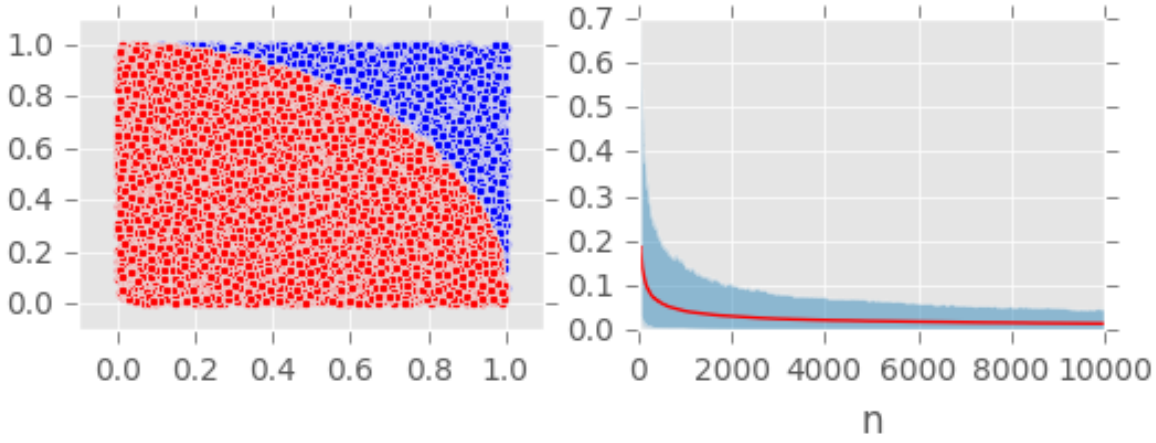


Figure 3.14.: Figure on the left show samples within the square, where red indicate inside circle and red outside. On the right we see the mean error as a function of n, with a 95% confidence interval.

3.5.3. Constructing Markov Chains

The success of Monte Carlo methods is the feasibility to sample from the specific distribution f. It turns out that we can construct Markov chains in a clever way, that their stationary distribution is the target distribution f. This imply that when we run the Markov chain for a sufficient amount of time, we get a sample from f [96]. There are many ways of constructing these chains, and they are all special cases of the framework introduced by Metropolis and Hastings [95]. Approximating statistics in this way is called MCMC, more specific we say that MCMC refers to the framework of producing an ergodic Markov chain with a specific stationary distribution. In relation to the random walk model explaining what a stochastic process is, equally in MCMC we perform a random walk through the probability distribution in question, favoring values with higher probabilities.

How does the initial state X_0 affect the Markov chain in general? We see in [96] that as the chain moves on it will gradually forget, and depend less, on the initial state. This implies that as the chain moves on the sampled points $\{X_i\}$ will look more and more like a sample from the stationary distribution. We say that after a sufficiently long burn-in period of N_{burn} iterations the points $\{X_i \mid i = N_{burn} + 1, ..., N\}$ will be a dependent sample approximately from the stationary distribution [96] .

The two most known algorithms for MCMC is called MH-algorithm and Gibbs sampling, where the Gibbs sampler is a special case of MH-algorithm.

Metropolis Hastings Algorithm

For each step i, the next state x_{i+1} is chosen by first sampling a candidate x^*_{i+1} from a **proposal** distribution (proposal distribution may depend on x_i). Then we accept the

candidate with probability:

$$\alpha(x_i, x_{i+1}^*) = \min\left(1, \frac{p(x_{i+1}^*)q(x_i \mid x_{i+1}^*)}{p(x_i)q(x_{i+1}^* \mid x_i)}\right) \tag{3.28}$$

Where $q()$ is the proposal distribution and $p()$ is the target distribution. From [96] we see that remarkably, the proposal distribution can have any form and the stationary distribution of the chain will be the target distribution.

Algorithm 4 Metropolis-Hastings algorithm

Initialize x_0
for $i = 1$ to N **do**
    Sample $u \sim \text{Unif}(0, 1)$
    Sample $x_{i+1}^* \sim q(x_{i+1}^* \mid x_i)$ ▷ where $q()$ is the proposal distribution
    if $u < \alpha(x_i, x_{i+1}^*) = \frac{p(x_{i+1}^*)q(x_i \mid x_{i+1}^*)}{p(x_i)q(x_{i+1}^* \mid x_i)}$ **then**
        $x_{i+1} = x_{i+1}^*$
    else
        $x_{i+1} = x_i$

As an example of the MH-algorithm we want to sample from a mixture of two Gaussian, as such we chose the proposal distribution as another Gaussian with mean x_i and high variance in each iteration. As such we define the target distribution (stationary distribution) and proposal:

$$p(x) \propto 3\exp\left(-\frac{1}{5}x^2\right) + 7\exp\left(-\frac{1}{5}(x-10)^2\right) \qquad \text{Target distribution} \tag{3.29}$$

$$q(x \mid y) \sim \text{Norm}\,(\mu = y, \sigma = 50) \qquad \text{Proposal distribution} \tag{3.30}$$

Using (3.29) and (3.30) we can see the histogram of the MH-algorithm samples and the target distribution after different iteration points in figure 3.15. It is important to note the large amount of MCMC samples required to converge to the target distribution. As we can read in [97] even if the MH-algorithm is very simple, it requires careful design of the proposal distribution to converge in a feasible amount of samples.

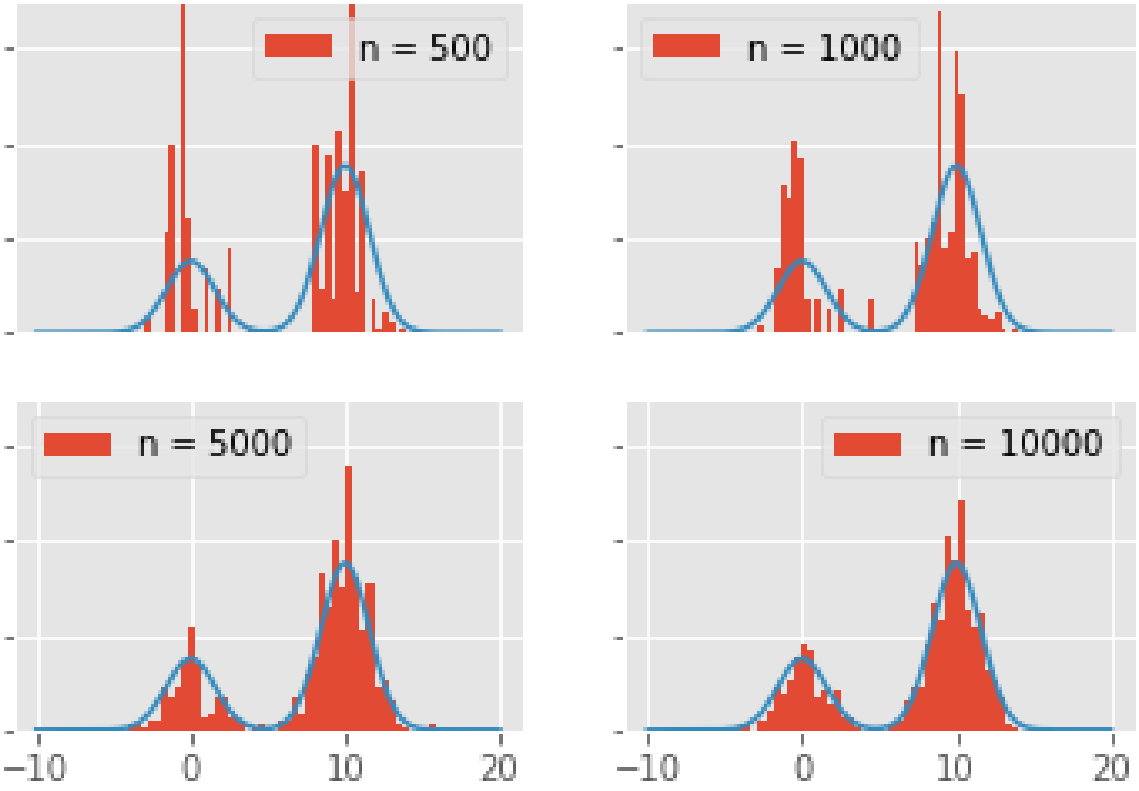


Figure 3.15.: Figure showing target distribution and histogram of the MH-algorithm generated samples at different iteration points.

The Gibbs Sampler

Instead of updating the whole of x at every iteration step, it is often more convenient and efficient to divide x into components $x = (x_1, ..., x_n)$. We denote $x_{-j} = (x_1, ..., x_{j-1}, x_{j+1}, ..., x_n)$, all components except x_j. As such we construct the Gibbs sampler, which is a special case of the MH-algorithm where you choose a specific proposal distribution. If we chose the proposal distribution for updating the jth *component* of x to be

$$q(x^*_{i+1} \mid x_i) = \begin{cases} p(x^*_{i+1,j} \mid x_{i,-j}) & \text{If } x^*_{i+1,-j} = x_{i,-j} \\ 0 & \text{Otherwise} \end{cases} \tag{3.31}$$

Then the corresponding acceptance probability can be shown to be equal to one [97].

Algorithm 5 Gibbs sampler

1: Initialise $\mathbf{x}_0 \in \mathbb{R}^n$
2: **for** $i = 1$ to N **do**
3: **for** $j = 1$ to n **do**
4: Sample $x_{i+1,j} \sim p\left(x_j \mid x_{(i+1),0}, ..., x_{(i+1),(j-1)}, ..., x_{(i+1),(j+1)}, ..., x_{(i+1),n}\right)$

Where $x_{i,j}$ is indexed such that i represent iterations step and j is the dimension. As an example to better understand the Gibbs sampler we want to generate a sample from a two-dimensional Gaussian:

$$\mathbf{x} \sim N_2(\mu, \Sigma) = N_2\left((\mu_1, \mu_2), \begin{bmatrix}\Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22}\end{bmatrix}\right) \tag{3.32}$$

If we are only able to sample from the univariate Gaussian distribution and if we recall that the conditional distribution of a bivariate Gaussian is univariate, we can create a Gibbs sampler to iteratively generate a sample from $\mathbf{x}$. The full conditional is written as:

$$x_1 \mid x_2 \sim N\left(\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}\right) \tag{3.33}$$

Let the bivariate Gaussian be:

$$\mathbf{x} \sim N_2\left((2, 5), \begin{bmatrix}1 & -0.5 \\ -0.5 & 1\end{bmatrix}\right) \tag{3.34}$$

Initiating with $x_{0,2} = 2$ we calculate

$$x_{1,1} \mid x_{0,2} = 2 \sim N(\mu_1 + \Sigma_{12}\Sigma 22^{-1}(x_{0,2} - \mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}) = N(3.5, 0.75) \tag{3.35}$$

We sample and draw $x_{1,1} = 3$. We calculate the next step in the Gibbs sampler:

$$x_{1,2} \mid x_{1,1} = 3 \sim N(5 + (-0.5)(3 - 2), 1 - 0.25) = N(4.5, 0.75) \tag{3.36}$$

If we keep following this logic we generate a sample from the bivariate Gaussian in (3.34). In figure 3.16 we see the generated samples after different numbers of iterations. Note that we used no burn-in period, burn-in period is the first b samples from the Markov chain. In more complex problems we want to discard the b first samples because we have not entered the stationary distribution of the chain yet.

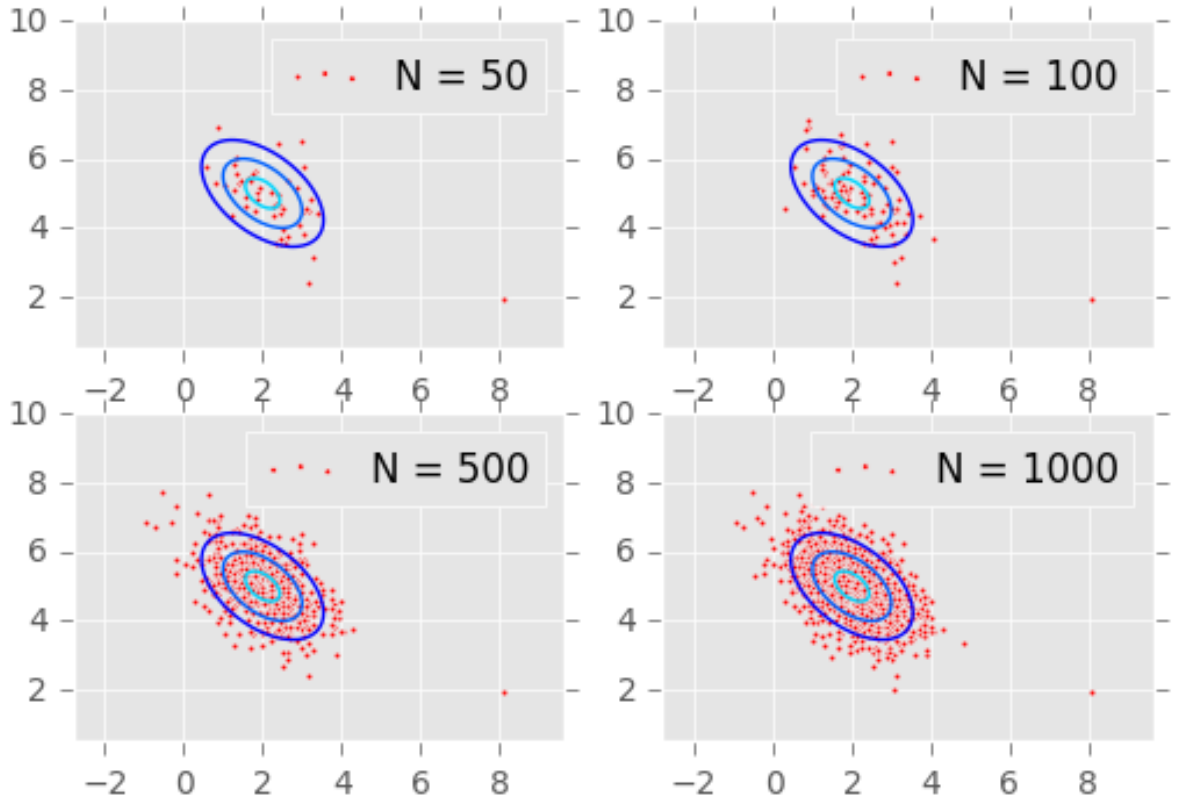


Figure 3.16.: Figure showing samples generated using a Gibbs sampler for a bivariate Gaussian after different number of iterations. No burn-in period used.

3.6. Bootstrap

In this section we are briefly going to discuss the empirical bootstrap. The empirical boostrap is a statistical (Monte Carlo) technique made popular by Bradley Efron [98, 99]. We will develop the needed theory to bootstrap an empirical confidence interval. We will use these bootstrapped confidence intervals to further analyze the experiments in chapter 7.

The bootstrap is a data-based simulation method to do statistical inference, typically avoiding parametric assumptions [100]. The general idea of bootstrap is to sample with replacement from the observed data set and estimate the **empirical** distribution of the predictor. In figure 3.17 we see a digram of the bootstrap principle.

The bootstrap principle is to resample B times with replacement form the original sample. Each bootstrap sample should be the same size as the original sample[1].

To give some form of justification to why the empirical bootstrap work we can see how an empirical distribution approximates the true distribution better as we get more

[1]The resample should be the same size because the variation of the statistic will depend on the size of the sample. We want to approximate the original sample and should use the same size.

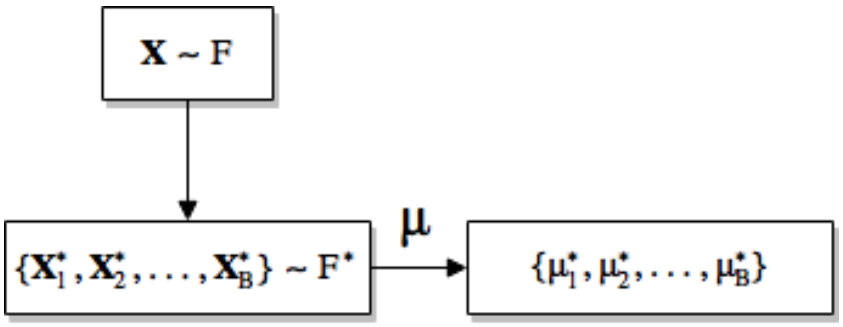
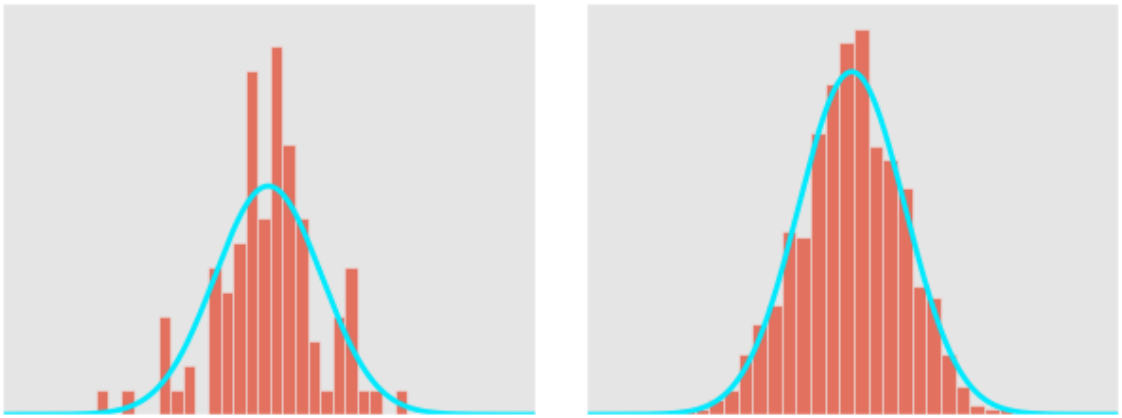


Figure 3.17.: Digram of the bootstrap principle.



(a) (b)

Figure 3.18.: Empirical and true distribution for two a) $n = 100$ observations and b) $n = 1000$ samples.

and more samples. In figure 3.18 we see the true distribution (F) and the empirical distribution (F^*) for a) $n = 100$ and b) $n = 1000$ samples. As we can see the empirical distribution approximates the true distribution better as we get more samples. We expect the empirical distribution of a predictor to behave in the same way, thus as we follow the bootstrap principle we estimate the distribution of the predictor. Thus we can find mean, variation and confidence intervals of the predictor using bootstrapping. Note that we do **not** assume any prior distribution for the predictor.

3.6.1. Empirical Bootstrap Confidence Intervals

We can find the empirical confidence interval without assuming the underlying distribution using bootstrapping. To find the confidence interval we want to know how much the distribution of some predictor μ^* varies around the mean of the predictor μ. That is, we want to know the distribution of $\zeta = \mu^* - \mu$. If we knew the distribution (or assumed it) we could calculate the percentiles of ζ. Recall that the (1-α)100% confidence interval

is:

$$p(\zeta_{1-\alpha/2} \leq \mu^* - \mu \leq \zeta_{\alpha/2}) = 1 - \alpha \tag{3.37}$$

However we can estimate the distribution of ζ using bootstrapping and approximate the percentiles. Note that since we estimate ζ using bootstrap, we resample from the original samples. We can then resample how many times we want, and estimate the distribution with high precision.

Now we can estimate $\zeta_i^* = \mu_i^* - \mu$, where μ is the prediction of the original sample, and μ_i^* is the prediction of bootstrap sample i. This we can create B such samples and find the empirical percentiles of all the ζ_i^*. Then we have the confidence interval as $[\mu - \zeta_{1-\alpha/2}^*, \mu - \zeta_{\alpha/2}^*]$.

3.6.2. Empirical Bootstrap Hypothesis Testing

Similar to confidence intervals we can use bootstrapping to do hypothesis testing. For the thesis we will focus only on a two sided test to check if two population means is zero. This is the only test we will use in the thesis, to test if the predicted values of one model is different then the other.

The basic idea is to first transform the data such that H_0 is true, e.g. make the difference in mean zero. Then we resample B samples from that data. We then calculate the difference between the two means in each of the bootstrap sample given. Finally we count the number of times the means under H_0 is larger then the original difference. Let $\mu_{\mathbf{X}_i}^*$ denote the mean of bootstrap sample i of sample $\mathbf{X}$ given that H_0 is true.

$$p\text{-value} = \frac{\sum_{i=1}^{B} \left(\left| \mu_{\mathbf{X},i}^* - \mu_{\mathbf{Y},i}^* \right| > |\mu_{\mathbf{X}} - \mu_{\mathbf{Y}}| \right) + 1}{B + 1} \tag{3.38}$$

3.7. Gaussian Mixture Models

The aim of this section is to derive the collapsed Gibbs sampler for a finite mixture model. Following the notation from [16] we know that a finite GMM with K components can be written as:

$$\begin{aligned} f(X \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{w}) &\sim \sum_{k=1}^{K} w_k \mathcal{N}(X; \mu_k, \Sigma_k) && \text{Likelihood} \\ f(\mathbf{w} \mid \alpha) &\sim Dir(\alpha) = Dir(\alpha/K, \cdots, \alpha/K) && \text{Weight prior} \\ f(\mu_k, \Sigma_k \mid \beta_0) &= f(\mu_k \mid \Sigma_k)\pi(\Sigma_k) && \text{Parameter prior} \end{aligned} \tag{3.39}$$

where β_0 is the hyper parameters for the prior.

From the model in (3.39) we introduce the stochastic latent variable z_i, indicating which of the k components $\mathbf{x}_i$ belong to. Then, we indicate with $w_k = P(z_i = k)$ the probability that $\mathbf{x}_i$ belongs to component k, which can be expressed as:

$$p(\mathbf{z} \mid \mathbf{w}) \sim \text{Categorical}(z_1, ..., z_K \mid N, w_1, ..., w_K) \tag{3.40}$$

3.7.1. Conjugate Priors for Mean and Covariance

We assume the that the parameter for each component (mean and covariance) are drawn from the same prior distribution. Specifically, we choose the prior for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ to be a Gaussian and an inverse-Wishart distribution respectively. Since the covariance of a Gaussian mixture and its mean are dependent their joint distribution must be specified accordingly [17]. Given the join dependent distribution that we have chosen to define prior for the mean and covariance, the full joint distribution is a completely conjugate prior density, named a Gaussian Inverse-Wishart (GIW) distribution:

$$\begin{aligned} p(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \beta_0) &= p(\boldsymbol{\mu} \mid \boldsymbol{\Sigma}, m_0, \kappa_0) p(\boldsymbol{\Sigma} \mid S_0, \nu_0) \sim \mathcal{N}\left(\boldsymbol{\mu}; m_0, \frac{1}{k_0}\boldsymbol{\Sigma}\right) \cdot IW(\boldsymbol{\Sigma}; S_0, \nu_0) \\ &\propto |\boldsymbol{\Sigma}|^{\frac{1}{2}} \exp\left\{-\frac{\kappa_0}{2}(\boldsymbol{\mu} - m_0)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu} - m_0)\right\} \cdot |\boldsymbol{\Sigma}|^{\frac{\nu_0 + D + 1}{2}} \exp\left\{-\frac{1}{2}\operatorname{Tr}\left(S_0 \boldsymbol{\Sigma}^{-1}\right)\right\} \Rightarrow \\ p(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \beta_0) &\sim GIW(\boldsymbol{\mu}, \boldsymbol{\Sigma}; \beta_0), \quad \beta_0 = (m_0, \kappa_0, \nu_0, S_0) \end{aligned} \tag{3.41}$$

As a way to understand how this prior behave we have the following interpretation for the parameters in β_0 [17]; m_0 is our prior mean for $\boldsymbol{\mu}$, κ_0 is how strongly we believe this prior to be correct. S_0 is proportional to our prior mean for $\boldsymbol{\Sigma}$ and ν_0 is how strongly we believe this prior to be correct. Additionally, we require that $\nu_0 > D - 1$.

3.7.2. Posterior of Parameters

The posterior distribution $p(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \mathbf{X})$ in each mixture component is again GIW distributed.

$$\begin{aligned} p(\boldsymbol{\mu}, \boldsymbol{\Sigma} \mid \mathbf{X}) \propto p(\mathbf{X} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(\mathbf{X}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \cdot GIW(\boldsymbol{\mu}, \boldsymbol{\Sigma}; \beta_0) = \\ GIW(\boldsymbol{\mu}, \boldsymbol{\Sigma}; \beta_N) \end{aligned} \tag{3.42}$$

with the updated parameters

$$\begin{aligned} \beta_N &= \{\mathbf{m}_N, \kappa_N, \nu_N, \mathbf{S}_N\} \\ \mathbf{m}_N &= \frac{\kappa_0 \mathbf{m}_0 + N\bar{\mathbf{x}}}{\kappa_N} \\ k_N &= \kappa_0 + N \\ \nu_N &= \nu_0 + N \\ \mathbf{S}_N &= \mathbf{S}_0 + \mathbf{S} + \kappa_0 \mathbf{m}_0 \mathbf{m}_0^T - \kappa_N \mathbf{m}_N \mathbf{m}_N^T \\ \mathbf{S} &= \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T \end{aligned} \tag{3.43}$$

where $\bar{\mathbf{x}}$ is the mean of the observations $\mathbf{X}$ with cardinality N.

The full joint distribution is in fact $p(\mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = p(\mathbf{X} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. We get the **posterior predictive distribution** for a new observation, $\mathbf{x}^*$;

$$p\left(\mathbf{x}^* \mid \mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \frac{p\left(\mathbf{x}^*, \mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right)}{p\left(\mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right)} \tag{3.44}$$

3.7.3. Constructing Sampling Scheme

So far we have not been dependent on using a conjugate prior, looking at the posterior predictive in (3.44) and the fact that we choose a conjugate prior; we can analytically integrate out $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Therefore using the conjugate prior GIW we will construct a **collapsed Gibbs sampler** for the GMM. The collapsed Gibbs sampler get its name by integrating out (marginalizes over) one or more variables when sampling the cluster assignments.

The labels z_i can be assigned by using a Gibbs sampling scheme. Since we can calculate the conditional distribution of the categorical distribution of $\mathbf{z}$, we can sample labels one at a time from:

$$z_i \mid \mathbf{z}_{-i} \sim f(z_i \mid \mathbf{z}_{-i}, \mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{w}, \alpha, \beta) \tag{3.45}$$

For the GMM we want to create a Gibbs sampler such that we sample one at a time from the conditional distribution for each observation given all the other observations, $z_i \mid z_{-i}$. Intuitively this is equivalent to iteratively observer each observation *for the first time*. From (3.45) we can marginalize out the mean and covariance, such that in each step for each observation we sample from the categorical distribution;

$$z_i \mid \mathbf{z}_{-i} \sim \{f\left(z_i = k \mid z_{-i}, \mathbf{X}, \alpha, \beta\right)\}_{n=1}^{N} \tag{3.46}$$

In order to perform the sampling scheme we need to expand the term in (3.46) and find the analytical expression, we extend as:

$$\begin{aligned} &f(z_i = k \mid \mathbf{z}_{-i}, \mathbf{X}, \alpha, \beta_0) \propto f(z_i = k \mid \mathbf{z}_{-i}, \alpha, \beta_0) \cdot f(\mathbf{X} \mid z_i = k, \mathbf{z}_{-i}, \alpha, \beta_0) \\ &= f(z_i = k \mid \mathbf{z}_{-i}, \alpha) \cdot f(\mathbf{x}_i \mid \mathbf{X}_{-i}, z_i = k, \mathbf{z}_{-i}, \beta_0) \cdot f(\mathbf{X}_{-i} \mid z_i = k, \mathbf{z}_{-i}, \beta_0) \\ &\propto f(z_i = k \mid \mathbf{z}_{-i}, \alpha) \cdot f(\mathbf{x}_i \mid \mathbf{X}_{-i}, z_i = k, \mathbf{z}_{-i}, \beta_0) \\ &\qquad\quad \text{Term 1} \qquad\qquad \text{Term 2} \end{aligned} \tag{3.47}$$

The two terms in (3.47) are in fact calculated independently.

Term 1 The first term is the likelihood that the new observation is assigned to component k, given all other assignments and α.

$$f(z_i = k \mid \mathbf{z}_{-i}, \alpha) = \frac{f(z_i = k, \mathbf{z}_{-i} \mid \alpha)}{f(\mathbf{z}_{-i} \mid \alpha)} = \frac{f(\mathbf{z} \mid \alpha)}{f(\mathbf{z}_{-i} \mid \alpha)} = \frac{N_{k-i} + \alpha/K}{N + \alpha - 1} \tag{3.48}$$

The step to finding the analytical solution in term 1 is done using the standard Dirichlet integral [16].

Term 2 The second term is similar to the marginalized posterior predictive in (3.44):

$$f(\mathbf{x}_i \mid \mathbf{X}_{-i}, z_i = k, \mathbf{z}_{-i}, \beta_0) = f(\mathbf{x}_i \mid \mathbf{X}_{k,-i}, \beta_0) = \frac{f(\mathbf{X}_k \mid \beta_0)}{f(\mathbf{X}_{k,-i} \mid \beta_0)} \tag{3.49}$$

where $\mathbf{X}_{k,-i}$ represents the set of elements assigned to the k-th mixture, without considering the i-th element. Notice that if $z_i \neq k$, $\mathbf{X}_{k,-i} = \mathbf{X}_k$. This marginalized posterior predictive distribution is known to have a multivariate Student's t distribution [40];

$$f(\mathbf{x}_i \mid \mathbf{X}_{k,-i}, \beta_N) \sim \mathcal{T}\left(\mathbf{x}_i | m_N, \frac{\kappa_N + 1}{\kappa_N(\nu_N - D + 1)} S_N, \nu_N - D + 1\right) = \mathcal{T}\left(\mathbf{x}_i | \beta_k\right) \tag{3.50}$$

3.7.4. Collapsed Gibbs Sampler for a Finite Gaussian Mixture Model

Algorithm 6 Collapsed Gibbs sampler for a finite Gaussian mixture model.

```
Initialize z.
for T iterations do
    for i = 1 to N do
        Remove x_i from current mixture component z_i = ∅.
        Remove empty mixture components.
        for k = 1 to K do
            calculate f(z_i = k | z_{-i}, X, α, β_0) ▷ From term 1 (3.48) and term 2 (3.49).
        Draw z_i = k_new from {p(z_i = k | z_{-i}, X, w, μ, Σ, α, β_0)}_{k=1}^K.
        Add x_i to the new mixture component z_i = k_new
```

3.8. Dirichlet Process Mixture Model

The content of this section is one of the main building blocks of the contribution in the thesis. We want to develop the nonparametric mixture model, the DPMM. In the thesis we introducing the transition from a finite GMM to the infinite DPMM. We do this transition by taking the finite GMM and observing the model as we take the limit on the number of clusters to infinite, $K \to \infty$.

If we consider the categorical distribution we sample from in the finite model described in (3.47). We have shown how it is possible to sample from this distribution using a collapsed Gibbs iterative scheme, even tho we now want to study how the two terms change as we let the potential number of mixture components go to ∞ it is important to note that we only observe a **finite** amount of non-empty mixture components. We denote this variable number as k_{obs}, the number of non-empty mixture components currently being observed (at current iteration).

Term 1: The probability $P(z_i = k \mid z_{-i}, \alpha)$ is governed by the CRP and it can be derived by taking the limit $K \to \infty$. Accordingly, the probability of assigning the element $\mathbf{x}_i$ to an existing mixture component versus assigning to an empty unused mixture component (of which we have an infinite amount of):

$$P(z_i = k \mid z_{-i}, \alpha) \propto \begin{cases} N_{k-i} & \text{if } k \text{ is a non-empty mixture} \\ \alpha & \text{if } k \text{ is a new mixture} \end{cases} \tag{3.51}$$

with N_{k-i} the number of pattern which are assigned to the component k, excluding the element x_i. Note that the probability of using a non-empty mixture component can be obtained trivially, by taking the limit $K \to \infty$. For what concerns the probability of observing a new mixture component(using an empty mixture component), we sum $\alpha/K(N + \alpha - 1)$ for each empty component, then get the total probability of which we take the limit.

$$\lim_{K\to\infty} \sum_{k=1}^{K_\emptyset} \frac{\alpha/K}{(N + \alpha - 1)} = \frac{1}{N + \alpha - 1} \left(\frac{\alpha}{K} + \frac{\alpha}{K} + ... + \frac{\alpha}{K} \right) = \frac{\alpha}{(N + \alpha - 1)} \tag{3.52}$$

Observe that $K_\emptyset(K)$ (the number of empty components) is a strictly increasing function of K, and as we let K go to infinity we observe that the number empty mixtures goes to infinite, such that we get $\lim_{K\to\infty} K_\emptyset = K$.

Term 2: The second term $p(x_i \mid \mathbf{X}_{-i}, z_i = k, z_{-i}, \beta)$ does only depend on the number of observed mixture components K_{obs} and it can be computed as in the previous case. The only difference is that if x_i is assigned to a new component $z_i = k*$, the predictive distribution in Eq. 3.50 must be computed considering $\mathbf{X} = \emptyset$.

We can now construct a Gibbs sampler for a DPMM by samplinger from the $K_{obs} + 1$ dimensional multinomial distribution

$$z_i \sim \{P(z_i = k \mid z_{-i}, X, w, \mu, \Sigma, \alpha, \beta_0)\}_{k=0}^{K_{obs}} = (\hat{w}_0, \hat{w}_1, ..., \hat{w}_{K_{obs}}) \tag{3.53}$$

Where $\hat{w}_0$ is the probability of choosing a new mixture component.

3.8.1. Collapsed Gibbs Sampler for the Infinite Gaussian Mixture Model

Algorithm 7 Collapsed Gibbs sampler for an infinite Gaussian mixture model.

1: Choose an initial z.
2: **for** T iterations **do**
3: **for** $i = 1$ to N **do**
4: Remove x_i's statistics from component z_i.
5: Remove empty mixture components.
6: **for** $k = 0$ to K **do**
7: $P(z_i = k \mid z_{-i}, X, w, \mu, \Sigma, \alpha, \beta)$. ▷ as given in (3.53).
8: Sample k_{new} from $P(z_i = k \mid z_{-i}, X, w, \mu, \Sigma, \alpha, \beta)$ after normalizing.
9: Add x_i statistics to the new component $z_i = k_{new}$.

3.9. Non Conjugate Priors

The collapsed Gibbs sampler we derived in section 3.8 requires that we use a conjugate prior and analytically integrate out component parameters. Often researches choose priors for mathematically simplicity and retractable posteriors. Or they choose priors all ready used in existing literature, a spiral of confirmation bias. For many applications we need to define non conjugate priors to best explain our model and support the data at hand. For a much more in-depth discussion about priors see [90, 101, 102].

In section 3.8 we constructed the algorithm for training a mixture model using a Dirichlet process prior, given that we used a conjugate prior. In this section we are going to review some of the basic ideas behind handling non conjugate priors.

3.9.1. Adding Metropolis-Hastings Steps

One of the naive approaches for non conjugate priors is to implement Metropolis-Hastings updates for the unknown posterior distribution. Using a non conjugate prior makes the posterior predictive function in (3.44) analytically impossible to marginalize. Such that we can use Metropolis-Hastings updates as discussed in section 3.5.3 to approximate the posterior. For more details on this approach see [103].

3.9.2. Using Auxiliary Parameters

The second method for non conjugate priors that we are going to discuss in the thesis is to use auxiliary parameters within the Gibbs sampling scheme. Auxiliary parameters in this context is to introduce latent variable(s) and create the full join distribution. Such that the full conditional for the latent variable(s) are standard distributions and can be sampled directly. For more details about applications and results see [103, 104].

3.10. The Road to Faster Dirichlet Process Mixture Models Samplers

In 3.8 we derived Algorithm 7 to fit a DPMM for clustering. One of the drawbacks with the algorithmic scheme we derived is the iterative nature of MCMC methods. Another drawback is that the sampler propose local changes since we sample one cluster assignment at a time. Therefore the sampler exhibit poor convergence [105]. For larger data sets we need to create schemes that allow for parallelization and faster convergence.

One approach to faster algorithms is to move away from MCMC sampling methods and into approximate variational inference. These kind of algorithms does not have convergence guarantees such as MCMC methods. However, in the literature they tend to be the go-to algorithms for large data sets since these algorithms tend to be parallelizable. For a much more in-depth discussion about variational algorithms for fitting a DPMM see [106–108].

If we want to stay within in the MCMC world, there are proposed methods to increase convergence speed and create (partly) parallelizable MCMC samplers for the DPMM. In [109] they propose a fast MCMC sampler for Dirichlet process mixture models that can be parallelized. They combine restricted Gibbs iterations with super-clusters [110–112] with split/merge moves via sub-clusters [113, 114]. They show that the sampler enforce the correct stationary distribution of the Markov chain (without the need for approximations).

4. Probabilistic Cluster Kernel

In this chapter we discuss the data generated kernel function PCK. We create the kernel by clustering with a probabilistic clustering algorithm, using multiple initialization with different number of clusters [14]. The Probabilistic Cluster Kernel (PCK) is a robust framework for creating a matrix representing similarities, robust to prior parametric assumptions. Recall from section 2.2, that the RBF kernel is highly dependent on the parameter γ. The motivation of the PCK is to create a kernel function that is independent of prior parameter choices. In figure 4.1 we see a diagram for the process used to create the PCK. One of the strengths with the PCK is that the process creates a similarity matrix, as discussed previously these kernels have a wide range of application.

In section 2.5 we discussed the framework of ensemble clustering, where we introduced a framework to use multiple clusterings to make a final robust and consistent clustering. In this chapter we combine ensemble clustering methods with the GMM introduced in section 3.7. The main idea is to learn a kernel function through clustering the data multiple times, e.g. data driven kernel function. The posterior probabilities predicted by each model for each observation are used to learn similarities.

The main idea of the PCK can be traced back to [13]. In the early days they calculated the (hard) cluster memberships, and counted the number of times each pair of observations was clustered together. Later it was adopted and further developed within remote sensing [15, 115]. It was in the latter that they started to use the fuzzy clustering assignments, or posterior probabilities obtained from the GMM to compute the kernel matrix.

4.1. Mathematical Definition

We learn the Probabilistic Cluster Kernel (PCK) by using a GMM, where we fit each model using the EM-algorithm [116, 117]. We let $q = \{1, ..., Q\}$ be Q different initial-

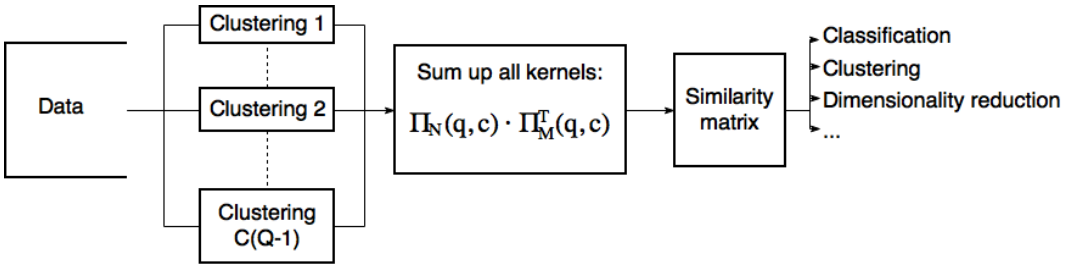


Figure 4.1.: Display of the process used to create the PCK.

izations, and $c = \{2, ..., C\}$ be the number of clusters for a specific initialization. We train each model by iterating c in the interval $[2, C]$, and for each of these cs we do Q different initializations. We ensure diversity by using different subsamples for each $q \in Q$. Furthermore we sample a different number of features uniformly for each subsample. However, we ensure that we fit each model on atleast two features. For each fitted model (q, c) we calculate the posterior probabilities $\boldsymbol{\pi}_i(k, c)$ for each observation $\mathbf{x}_i$ and calculate the inner product between all the different observations. Thus one ensemble create one kernel matrix, and we combine these kernels by averaging over all the kernels [14]. We use the following notation for the PCK kernel function;

$$K_{pck}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{Z} \sum_{q=1}^{Q} \sum_{c=2}^{C} \boldsymbol{\pi}_i(q, c) \cdot \boldsymbol{\pi}_j^T(q, c) \tag{4.1}$$

where Z is a normalization constant, and $\boldsymbol{\pi}_i(q, c)$ is the posterior probability for observation i in initialization q, having c clusters. For simplicity we will denote $\boldsymbol{\Pi}_N(q, c)$ as the matrix containing $\boldsymbol{\pi}_i(q, c)$ in row i, thus the shape of $\boldsymbol{\Pi}_N(q, c)$ is $(N \times c)$.

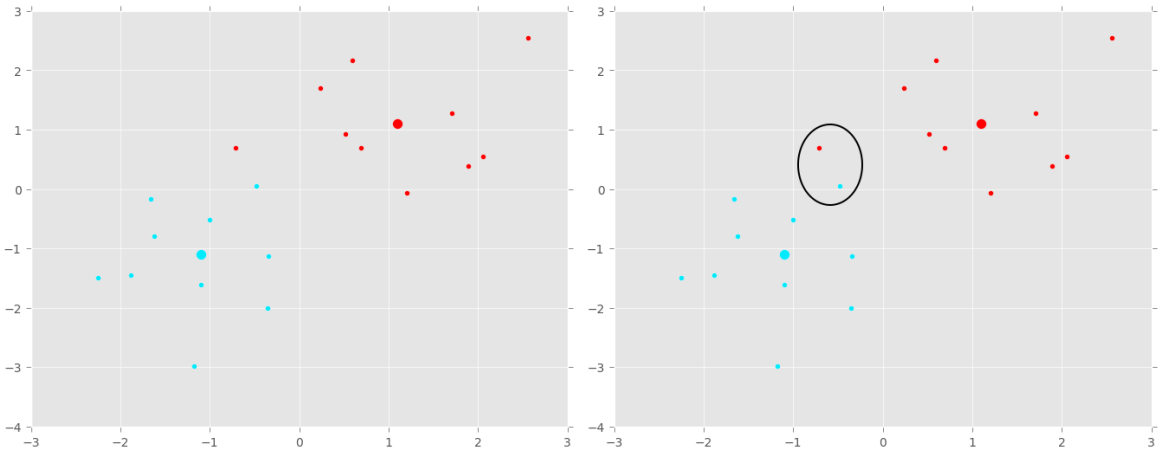
The PCK is motivated by creating a kernel function without prior parametric assumptions. However, we see that as we decide on the upper bound C we biased the model to prior assumptions about the parametric shape of the data.

4.1.1. Counting versus Inner Products as Consensus Function

One question that arise from the construction of the PCK is why to use the inner products of the posterior probabilities. Looking at figure 4.2 a) we see some data where we have learned two clusters. The big dots represent them means of each cluster. If this was one clustering inside an ensemble and we use counting to combine the results, we would loose the similarities of the two observations circled in figure 4.2 b). While using the inner products of the posterior probabilities we account for the relationship between the two observations. Let us call them $\mathbf{x}_1$ and $\mathbf{x}_1$, to calculate the kernel function for this ensemble in (4.2).

$$\begin{aligned} \mathbf{x}_1 &= (0.55, 0.45) \\ \mathbf{x}_2 &= (0.45, 0.55) \\ \mathbf{x}_1 \cdot \mathbf{x}_2 &= 0.495 \end{aligned} \tag{4.2}$$

Thus using a inner product consensus function, we keep more of the information about similarities within each cluster ensemble. The counting consensus function would add zero similarity between the two observations.



(a) (b)

Figure 4.2.: Toy example to illustrate two predicted mixture components.

4.2. Algorithm

In the training phase we fit each model and save the model parameters, and the subsets of features used within each model. The model parameters can then be used in the test phase to create the final PCK. For initiating the parameters for each GMM we can use both fixed parameters for each model or random sampling of the values. For the purpose of the thesis we will use fixed parameters for each ensemble when creating the PCK.

Algorithm 8 PCK: training phase.

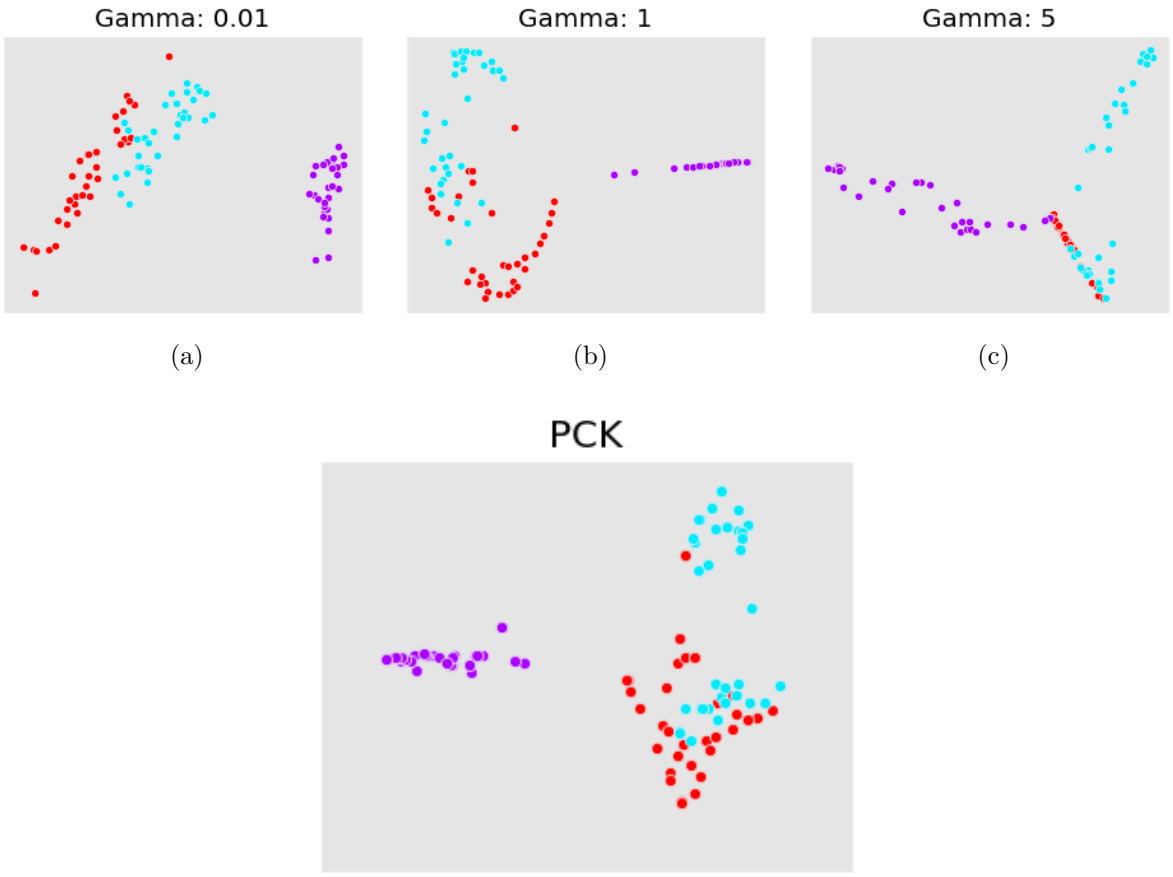
1: **function** PCK(Train set, C, Q)
2: Initiate GMM parameters for each model $\mathbf{\Theta}_{cq}$.
3: **for** q in $1 : Q$ **do**
4: **for** c in $2 : C$ **do**
5: Initiate GMM with c clusters and parameters $\mathbf{\Theta}_{cq}$.
6: Fit GMM using a random subsample. Additionally randomly select the features to use.

Algorithm 9 PCK: test phase.

function PCK(Train set, test set, subsets of features for each model, GMM parameters.)
    Initiate kernel matrix $K = \mathbf{0}_{N \times M}$.
    for q in $1 : Q$ **do**
        for c in $2 : C$ **do**
            Compute posterior probabilities for train set, $\mathbf{\Pi}_N$.
            Compute posterior probabilities for test set, $\mathbf{\Pi}_M$.
            Update kernel matrix $K = K + \mathbf{\Pi}_N(q, c) \cdot \mathbf{\Pi}_M^T(q, c)$

4.3. Dimensionality Reduction using the PCK

In figure 4.3 we see the difficulty of selecting the appropriate value for γ in a RBF kernel. If we have prior information about the data, like the true class labels, we could tune γ by visual inspection or according to some measure. However, in an unsupervised context we need methods that do not rely on prior knowledge. In 4.3 d) we see that the PCK discriminated two of the classes, and some discrimination between the two last classes. Additionally we do an important observation, in figure 4.3 d) we see that there is no free lunch in machine learning the PCK does not magically obtain the best results. As we know from the free lunch theorems, there are no general-purpose universal optimization strategy [118, 119].



(a) (b) (c)

(d)

Figure 4.3.: Data projected on the top two eigenvectors using KPCA, using the Iris species data set. a-c) RBF kernels with different γ. d) PCK.

Part II.

Proposed Method; The Dirichlet Process Mixture Model

5. The Dirichlet Process Cluster Kernel

The content of this chapter is one of the main contributions of the thesis. The contribution is to apply the DPMM to the PCK framework. In chapter 4 we saw that the motivation for the PCK was to learn a data driven kernel function without prior assumptions about the data. However, for the PCK we have to decide on the upper bound C, the number of clusters to use. In doing so we assume some parametric shape on the data.

In this chapter we propose to replace the finite mixture model, GMM with the DPMM discussed in 3.8. The main idea is that each ensemble within the cluster kernel should learn the number of clusters based on the data available. The hypothesis is that we create a stable cluster kernel without a priori deciding on any critical shape parameters. The new cluster kernel is a data driven kernel function that learns similarities in the data by clustering.

5.1. Proposed Algorithm

To learn the DPCK kernel matrix we fit different DPMM using the collapsed Gibbs sampler described in algorithm 7. To ensure diversity and robustness in the ensembles we sample Q different sets of random hyper-parameters and initial conditions. By re-sampling with replacement from the original data (called bootstrapping or bagging) we obtain a collection of training sets to further increase robustness to parameter choice and diversity in the ensembles [120]. We set the size of each bootstrap sample to the same as our original training sample. Additionally we sample a random number of features in the interval $[2, D]$, where D is the dimension of the data set. We sample these uniformly

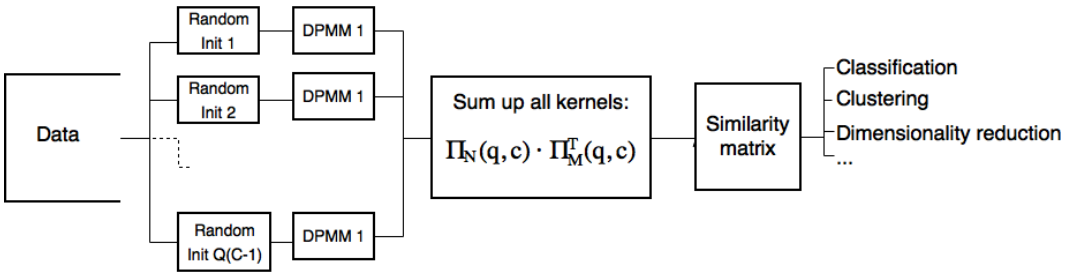


Figure 5.1.: Digram illustrating how the Probabilistic Cluster Kernel is constructed using Dirichlet Process Mixture Models.

without replacement.

We initiate each model with a fixed, predefined number of clusters k. With the notation of initiating the DPMM with k clusters, we mean to initiate the label assignments using K-Means for one iteration with k randomly selected cluster means.

To create the DP-CK kernel we take the inner products of the posterior probabilities between all observations to create the DPCK kernel matrix. This is equal to the approach used for creating the PCK in section 4.2.

Algorithm 10 DPCK: Training.

function DP-CK(Train set, Q)
    for q in $1 : Q$ **do**
        Draw parameters for the DPMM, $\boldsymbol{\beta}_0$.
        Initiate model with parameters $\boldsymbol{\beta}_0$.
        Fit model on different subsamples and with a random number of features.

Algorithm 11 DPCK: Test.

function DP-CK(Train set, test set, subsets of features for each model, DPGMM parameters.)
    Initiate kernel matrix $K = \mathbf{0}_{N \times M}$.
    for q in $1 : Q$ **do**
        Compute posterior probabilities for train set, $\mathbf{\Pi}_N$.
        Compute posterior probabilities for test set, $\mathbf{\Pi}_M$.
        Update kernel matrix $K = K + \mathbf{\Pi}_N(q) \cdot \mathbf{\Pi}_M^T(q)$

Part III.

Experiments

In this part we run some experiments to investigate if the DPCK is an improvement to the PCK. We run the different cluster kernels on a series of different data sets to investigate the differences between the two cluster kernels. One of the goals of this part is to examine different kinds of problems such that we later can discuss what kinds of data sets and problems we can expect the DPCK to be an improvement.

The main goal of this part is to present quantitative and statistical evidence to the performance of the DPCK versus PCK. We will structure each chapter in this part such that we first present some motivation for the specific experiments. Then we display the experimental setup that are going to be used. Finally we show the results of the experiments and briefly discussing important results. The different chapters contains different experiments with somewhat different subgoals. We will structure the chapters in this part in the following way:

- **Chapter 6:** Investigation of parameter sensitivity of the DPMM.
- **Chapter 7:** Investigation of the DPCK versus the PCK.
  - **Section: 7.2:** DPCK stability of parameters.
  - **Section: 7.3:** Performance comparison on different data sets.
    - **Section 7.3.1:** Classification problems.
    - **Section 7.3.2:** Clustering problems.
    - **Section 7.3.3:** Dimensionality reduction using KPCA.
- **Chapter 8:** Application of the DPCK in semi-supervised classification problems.

6. Parameter Investigation for the Dirichlet Process Mixture Model

In this chapter we run some experiments to investigate how the collapsed Gibbs sampler depend on its parameters. As we know from previous sections, the Gibbs sampler we constructed for the DPMM depends on one parameter for the mixture weights and four parameters for each mixture components density. Investigating the stability of the model parameters can help us understand how the final DPCK ensemble of multiple DPMMs behave. Additionally we want to say something about the range for each parameter and how they are related to the final clustering.

The mixture weights are determined by the scale parameter α where we get less mixtures when α is small and opposite when α is large. When using the CRP analogy α determines the number of people at each table. Higher value for α leads to more tables with less people.

We want to look at the parameters for each mixture, recall the marginalized posterior predictive function from section 3.7.3.

$$f(\mathbf{x}_i \mid \mathbf{X}_{k,-i}, \beta_N) \sim \mathcal{T}\left(\mathbf{x}_i | m_N, \frac{\kappa_N + 1}{\kappa_N(\nu_N - D + 1)}\mathbf{S}_N, \nu_N - D + 1\right) \tag{6.1}$$

Here β_N is dependent on the initial parameters $\beta_0 = \{\mathbf{m}_0, \kappa_0, \nu_0, S_0\}$, they have to be set by the user prior to running the clustering algorithm.

Starting with the mean of each mixture we get define $\mathbf{m}_0$ to be the mean of all observations, e.g. zero for centered data sets. Additionally we see that the parameter κ_0 is related to the mean as:

$$m_N = \frac{\kappa_0 m_0 + N\bar{\mathbf{x}}}{\kappa_0 + N} \approx \begin{cases} \bar{\mathbf{x}} & , 0 < \kappa_0 << 1 \\ \frac{m_0}{N} + \bar{\mathbf{x}} & , \kappa_0 = 1 \\ \frac{m_0 + \bar{\mathbf{x}}}{2} & , 1 << \kappa_0 \approx N \end{cases} \tag{6.2}$$

In (6.2) we see how the mean of each mixture approximates for different values of κ_0. We see in (6.2) that κ_0 represent how much we believe our initial mean. Choosing a low value we get mixture components only depending on the current observations assigned to the cluster. While increasing the value we get an average of the two. We see at κ_0 the mixture mean be close to $\bar{\mathbf{x}}$ as we assign more observations to the specific mixture. We expect small values close to 1 for κ_0 to perform good.

We choose the prior covariance $\mathbf{S}_0$ to be the identity matrix. The degree of freedom ν_0 is the scale parameter for the covariance, thus some value on our belief in the prior

	Low	Medium	High
α	0.01	1	100
κ_0	0.001	1	10
ν_0	10	25	80
k_{init}	5	10	25

Table 6.1.: Table containing values used to explore model parameters α, κ_0, ν_0 and k_{init}..

covariance. Choosing this value to be small, the variance of each mixture component is large. As ν_0 goes to ∞ each mixture components variance goes to Σ (multivariate Gaussian). We note that $\nu_0 = D + 15$ should give reasonable results.

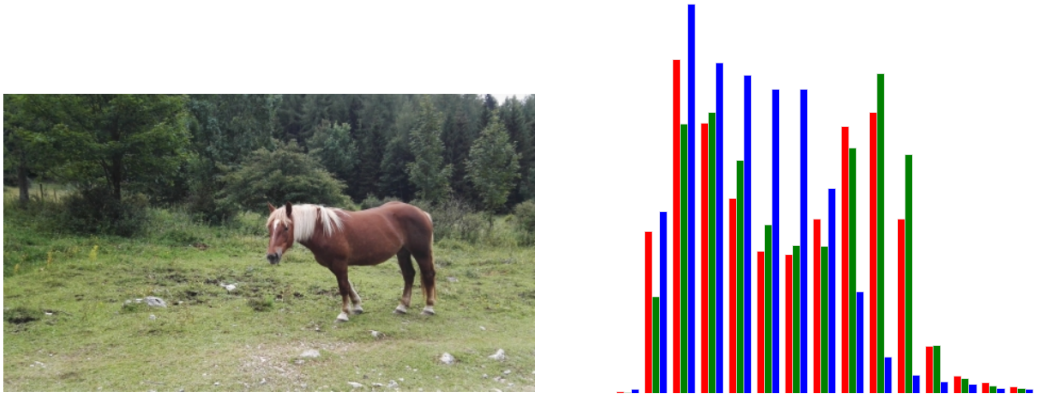
Our collapsed Gibbs sampler has another dependency with the number of clusters we initiate with k_{init}. This can be explained by the fact that any posterior of mixtures can be approximated arbitrarily well if we increase the numbers of mixtures [121]. Thus when we start with a high number of clusters, removing clusters can decrease the total likelihood. Additionally we know that the collapsed Gibbs sampler derived only sample one cluster assignment at a time, therefore we can only do small local changes in each iteration. When we write $k_{init} = a$ we mean that we initiate the Gibbs sampler using one iteration of K-Means with $K = a$, where we randomly sample the a different means.

6.1. Experimental setup

We want to investigate how the algorithm depends on the parameters. To do this we choose three different values for each parameter, one low, one medium and one high. In table 6.1 we see the different values we choose. We hope that by varying over these three values we can see the general behavior of the algorithm. To visualize the effect we choose to segment an image using our algorithm. In total we ran 81 different segmentations. In figure 6.2 we see nine randomly sampled segmentations. We want to discuss how the algorithm handles values that we believe to be to high or to low.

6.1.1. Data Set

For these experiments we will use the collapsed Gibbs sampler to segment an image of a horse. The size for the image is 500 $\times$ 280 pixels (140 000 observations in $\mathbb{R}^3$.). In figure 6.1 we see the original image together with the histogram of the different colors. What makes this an interesting problem is that the Dirichlet process allow us to observe different amount of clusters, for different segmentations.



(a) (b)

Figure 6.1.: a) Original image. b) Histogram of RGB colors.

For the image we do not use any preprocessing, as such we only segment based on RGB color values. To improve accuracy we could apply one of the many preprocessing techniques available in the literature [122, 123]. These often include adding edges and color of nearby pixels as features to add geometric context within the pixels [124].

6.2. Results

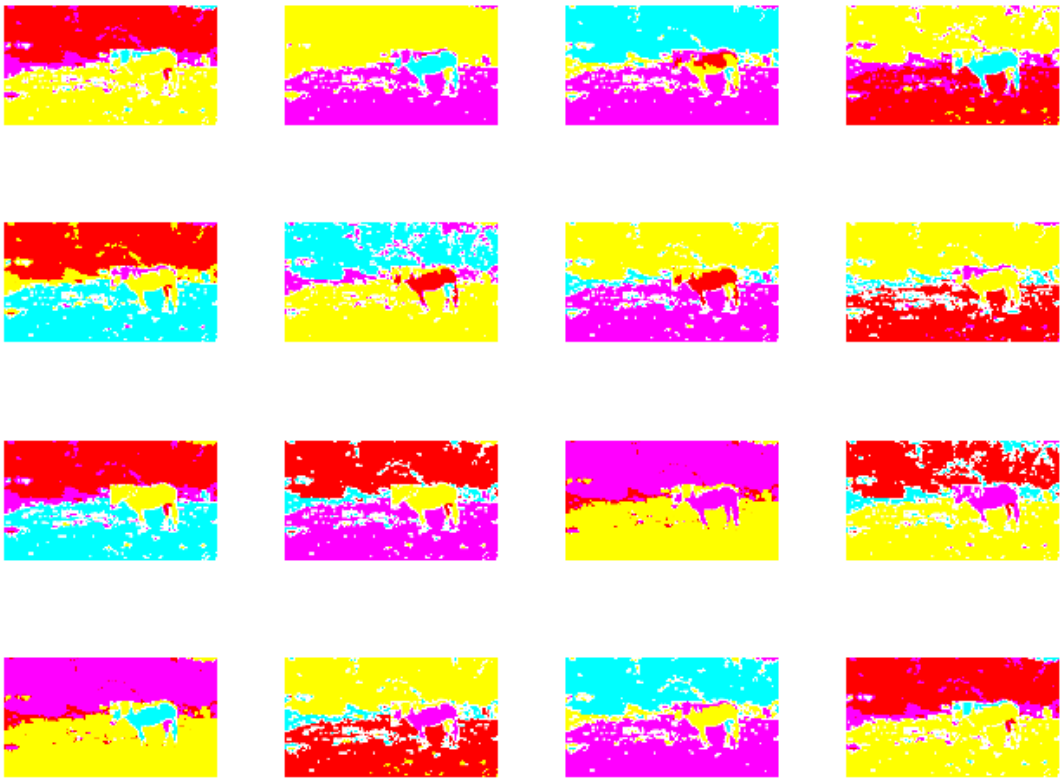
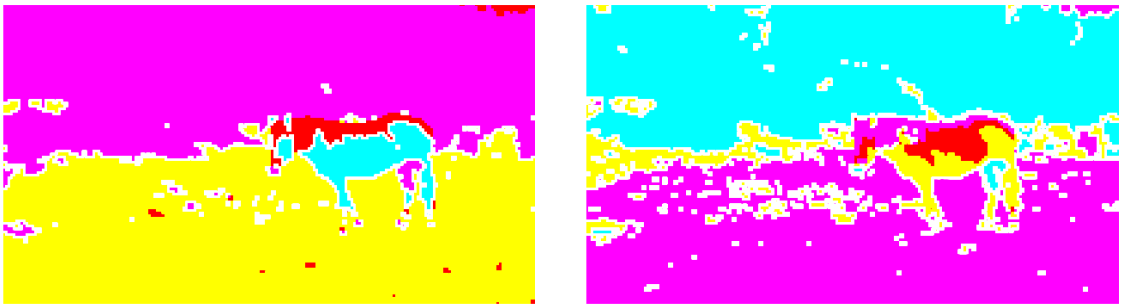


Figure 6.2.: Display of 16 randomly drawn segmentations of all 81.

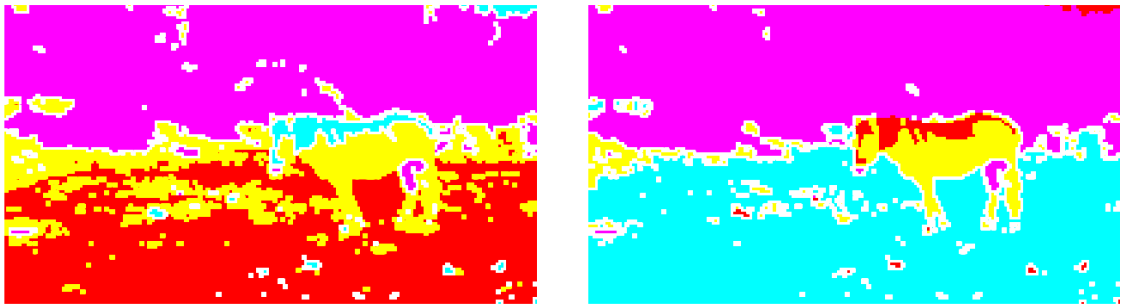
In figure 6.3 we see how the segmentation changes as we increase α from 0.001 to 100. We see that as α increases we observe more of the small objects in the form of rocks and such. This is what we would expect, since the probability of observing a new mixture component is proportional to the value of α.



(a) (b)

Figure 6.3.: Fixed: $\kappa_0 = 0.001$, $\nu_0 = 25$ and $k_{init} = 5$. a) Segmented image obtained when $\alpha = 0.01$. b) Segmented images obtained when $\alpha = 100$.

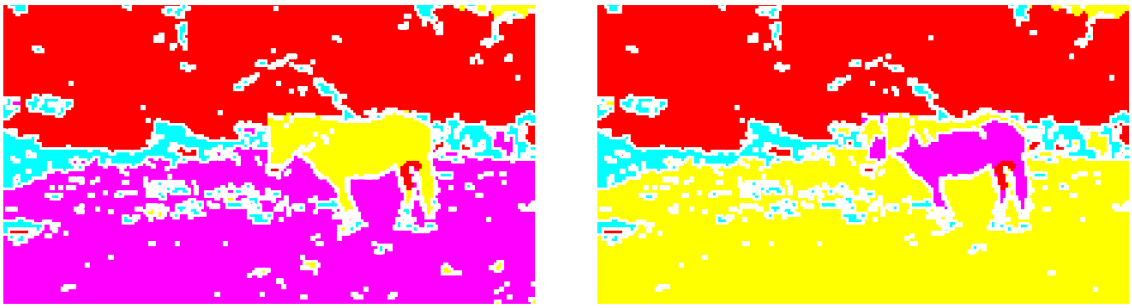
In figure 6.4 we see how the segmentation changes as we increase κ_0. One important thing to see here is that a lower value for κ_0 will decrease the sensitivity of each cluster, e.g. observations within a cluster can be less similar. Thus we observe that all the little rocks and grass pieces is now contained in one cluster. While in 6.4 b) we see that they are segmented together with the grass, for the most part.



(a) (b)

Figure 6.4.: Fixed $\alpha = 1$, $\nu_0 = 25$ and $k_{init} = 5$. a) Segmented image obtained when $\kappa_0 = 0.001$. b) Segmented image obtained when $\kappa_0 = 10$.

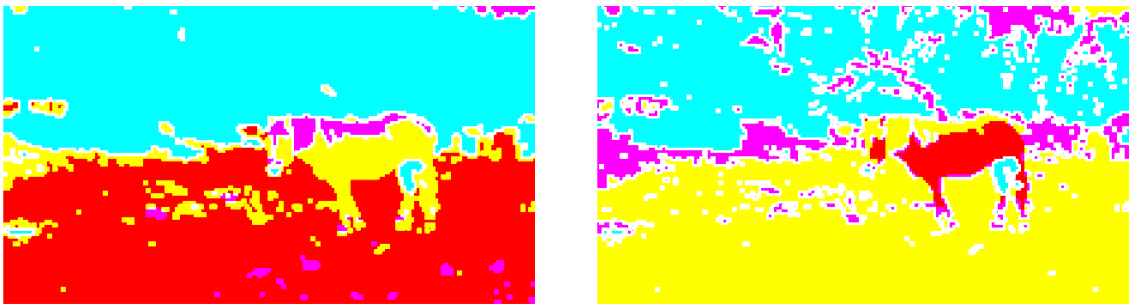
Observing from figure 6.5 that as ν_0 increases, the spread of each mixture decreases. This is as expected looking at term 1 and term 2 in (3.47), as the variance of each mixture decrease as ν_0 increases. The likelihood for observations further away from mixture components will decrease and we will observe more compact mixture components.



(a) (b)

Figure 6.5.: Fixed: $\alpha = 1$, $\kappa_0 = 1$, and $k_{init} = 5$. a) Segmented image obtained when $\nu_0 = 10$. b) Segmented image obtained when $\nu_0 = 80$.

From figure 6.6 we see the difference when initiating with a low number of mixture components versus a high number of clusters. In general we see that initiating with a high number of mixture components seems to introduce a higher number of final mixture components. Even tho choosing a high number of clusters to initiate with we get reasonable results. Thus we can argue that the model is not very sensitive to the parameters.



(a) (b)

Figure 6.6.: Fixed: $\alpha = 1$, $\kappa_0 = 1$, and $\nu_0 = 25$. a) Segmented image obtained when $k_{init} = 5$. b) Segmented image obtained when $k_{init} = 25$.

7. Investigation of the DPCK

In this chapter we investigate the PCK and DPCK in terms of predictive accuracy and clustering accuracy. First we will present some evidence to the stability of the DPCK with respect to the parameters k_{init} and total number of ensembles. Then we illustrate some benefits of using a nonparametric Bayesian model for learning the cluster kernel compared to the parametric GMM.

The structure of this chapter will be such that we start by going through the experimental setup, and explain the details used to learn both cluster kernels. Additionally we will inspect the predictive accuracy and cluster accuracy for a range of different data sets. Investigating how the DPCK and PCK learns the similarities. We then investigate how the DPCK depends on the initial number of clusters used for each ensemble and how the results correlate with the total number of ensembles. These results will be presented in forms of some prediction results as a function of the number of ensembles and inside a table with the score for each data set. Finally we visualize the difference in the learned kernels using KPCA and looking at the projected data.

7.1. Experimental Setup

For learning the cluster kernels we follow the described and proposed algorithms in chapter 4 and 5. As such algorithms 8 and 9 for the PCK and algorithms 10 and 11 for the DPCK.

Learning the DPCK we fix the number of cluster ensembles to 70 and the number of initial clusters to $k_{init} = 11$ (if not otherwise specified). Those numbers where experimentally chosen, as we will briefly show in section 7.2. Additionally we fix $\mathbf{m}_0$ as the mean of all the observations and use a identity matrix as prior covariance. We fix the number of iterations for each ensemble to be 20, we have seen through experiments that 20 iterations is a good trade between accuracy and speed. We sample the DPMM parameters for each ensemble from the intervals in (7.1). Where D is the dimension of data e.g. $\mathbf{x}_i \in \mathcal{R}^D$.

$$\begin{aligned} \alpha &\sim \text{Unif}(10^{-2}, 1) \\ \kappa_0 &\sim \text{Unif}(10^{-3}, 1) \\ \nu_0 &\sim \text{Unif}(D + 25, D + 100) \end{aligned} \tag{7.1}$$

For the PCK we fix $Q = 5$ and $C = 15$, thus creating a total number of ensembles of 75. We do this for all experiments unless otherwise specified.

All the experiments are repeated 10 times and we use the mean value. Additionally we create a 95% confidence interval by using the empirical bootstrap method discussed

in section 3.6.1. In the related literature on cluster kernels we see that they often use the standard deviation. Some problems with this is that the standard deviation is symmetric. In some cases this might not give the full picture, as the prediction accuracy can not be greater then 1. Additionally using the empirical bootstrap we do not have to assume any prior distribution on the predictions. For each data set we calculate the p-value; for the null-hypothesis (H_0) that the means are equal. We do this by using the bootstrap method discussed in section 3.6.2.

7.1.1. Classification

To investigating the performance of the DPCK we test the prediction accuracy of the learned PCK and DPCK cluster kernels. First we learn the different kernels as defined in the previous section. Then we train a SVM using the learned kernels on a training set. Then we predict the labels for the test set and measure the accuracy score. The accuracy is measured using the mean accuracy. For ground truth $Y = [0, 0, 1, 1]$ and predicted $\hat{Y} = [0, 1, 1, 1]$ we get the mean accuracy as $\frac{1}{4}\sum_{i=1}^{4} f(Y_i, \hat{Y}_i) = 0.5$ where f is the function:

$$f(Y, X) = \begin{cases} 1, & Y = X \\ 0, & Y \neq X \end{cases} \tag{7.2}$$

7.1.2. Clustering

For the clustering comparison between the PCK and DPCK we use a spectral clustering algorithm with the number of clusters equal the true underlying number. To get a clustering accuracy we calculate the Normalized Mutual Information (NMI) score on the labeling obtained by the spectral clustering algorithm.

NMI is a score between 0 and 1 for cluster accuracy where 0 is no mutual information, and 1 is perfect correlation. One of its main properties is that it is independent of the value of the labels. For additional information see [125].

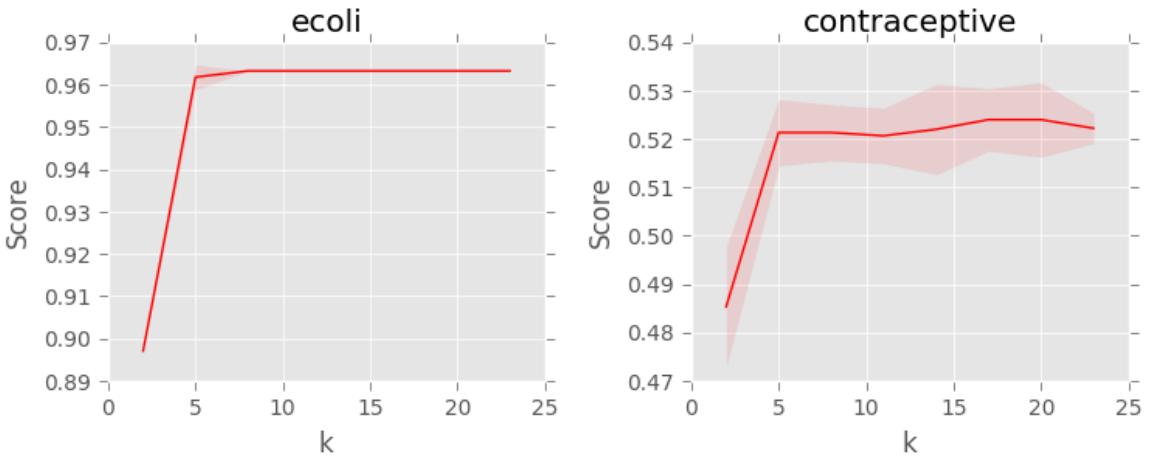
7.1.3. Dimensionality Reduction

For visually inspecting the kernels we project the data down on the two largest eigenvectors using KPCA. We do this for a selection of the data sets.

7.2. Dirichlet Process Cluster Kernel stability

In this section we want to say something about how the DPCK depend on its parameters. Specifically how the results correlate with choosing k_{init} and the total number of ensembles. In figure 7.1 we see how the final prediction accuracy depend on initial number of clusters we use for each ensemble. We see that choosing k_{init} to be < 5 can give bad results. For any value higher then this it looks stable. Furthermore we know that the higher value for k_{init} we choose, the more computations we need to do. Thus we

choose the value of 11 for all our experiments, as a nice trade-off between accuracy and speed. Additionally we want to see if there is any difference between the rate at which the ensemble converges (how fast or lower accuracy). Additionally we not that these stability tests where done for 70 different ensembles. Thus for all other experiments we choose the total number of ensembles to be 70.



(a) (b)

Figure 7.1.: Display of prediction accuracy for different values of k_{init} when creating the DPCK. Shaded areas represent standard deviation.

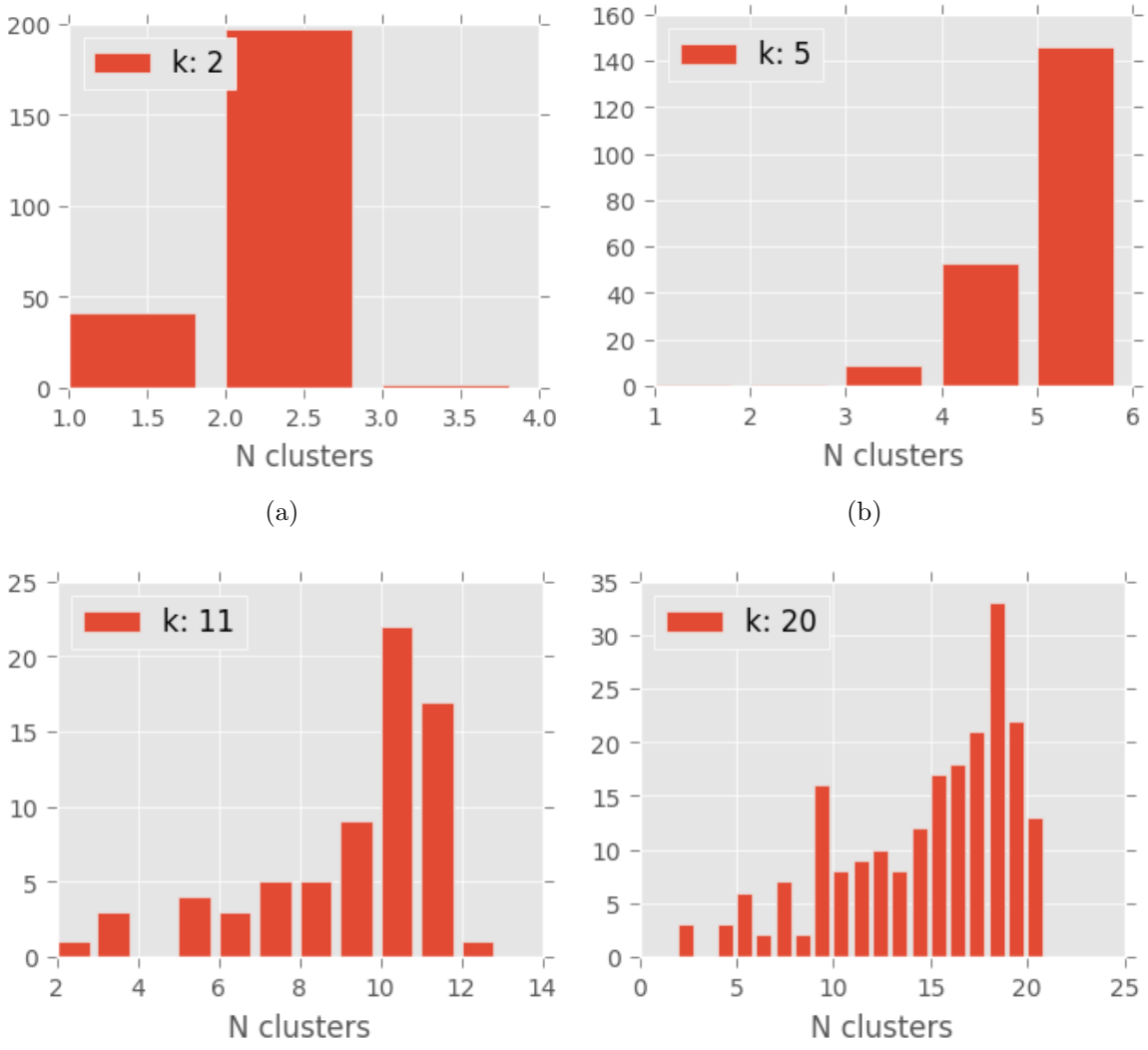


Figure 7.2.: Counts for how many clusters used by the different ensembles with the DPCK on segmentation data set. Segmentation data set have seven different clusters. a) Initiating with 2 clusters. b) Initiating with 5 clusters. c) Initiating with 11 clusters. d) Initiating with 20 clusters.

In figure 7.2 we see the number of clusters used with each ensemble within the DPCK. Observe that the algorithm rarely add additional clusters. Even tho the segmentation data set have 7 true clusters, the different ensembles rarely add clusters to fit to this number. The collapsed Gibbs sampler does not add clusters very often. Thus initiating with a high number of clusters we get some of the same effect as in the PCK. We use both few number of clusters, and many to learn local and global structures.

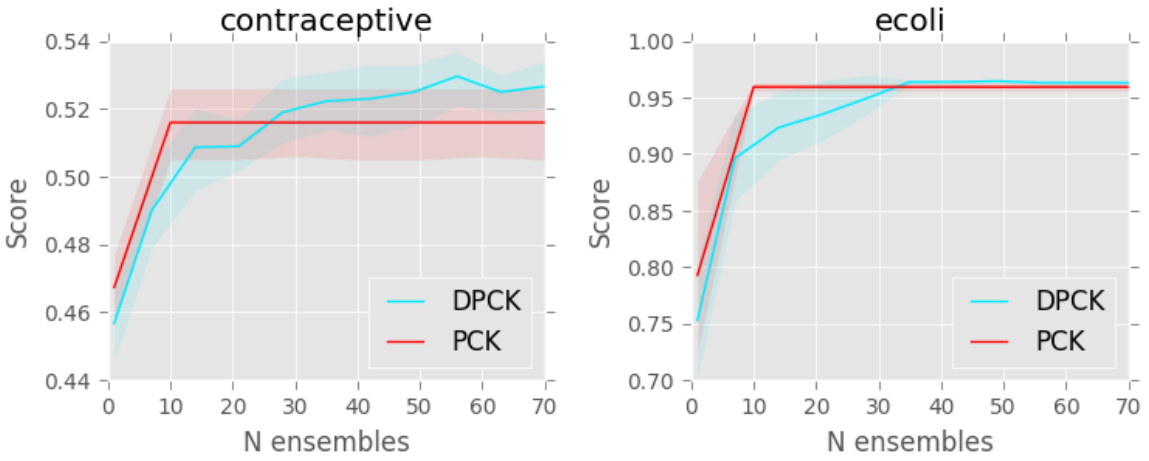
7.3. Comparing the Probabilistic Cluster Kernel to the Dirichlet Process Cluster Kernel

In table 7.1 we see the classification accuracy for 9 different data sets. The scores are calculated using a SVM on the DPCK and PCK. In table 7.2 we see the NMI values for 9 different data sets. The p-value is calculated for the null-hypothesis of equal score/NMI. If the p-value is small (typically 0.05) we say that the score/NMI is significantly different[1].

Additionally we want to see how the different cluster kernels converges to the final value. We plot the prediction accuracy and NMI as a function of total ensembles for different data sets.

7.3.1. Classification

In figure 7.3 we see that the DPCK and PCK get pretty equal results. It appears that the two cluster kernels converges at the same rate as well. From table 7.1 we see that the two kernels does not give significantly different results. Figure 7.3 might indicate that we could increase the accuracy by adding more ensembles to the DPCK. While the PCK have reached its full potential.



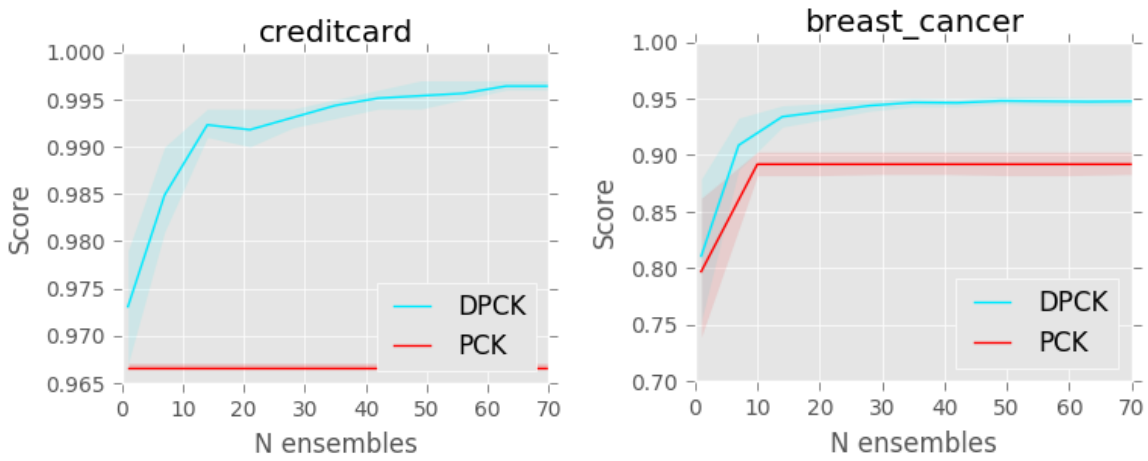
(a) (b)

Figure 7.3.: a) Predictive accuracy on the contraceptive data set. Comparison of the accuracy as a function of the total number of ensembles between the DPCK and PCK. b) Predictive accuracy on the ecoli data set. Comparison of the accuracy as a function of the total number of ensembles between the DPCK and PCK.

In figure 7.4 we see that for some data sets the DPCK performs very good. In 7.4 a)

[1]Do note that even if the p-value is large, the null-hypothesis is not necessary true.

we see that the DPCK converges very fast, to a accuracy larger then the PCK. In 7.4 b) we see that the DPCK converges faster and to a better solution with less variation. The p-value from table 7.1 and the confidence interval suggest that the DPCK is significantly better for these two data sets.



(a) (b)

Figure 7.4.: a) Predictive accuracy on the credit card data set. Comparison of the accuracy as a function of the total number of ensembles between the DPCK and PCK. b) Predictive accuracy on the breast cancer data set. Comparison of the accuracy as a function of the total number of ensembles between the DPCK and PCK.

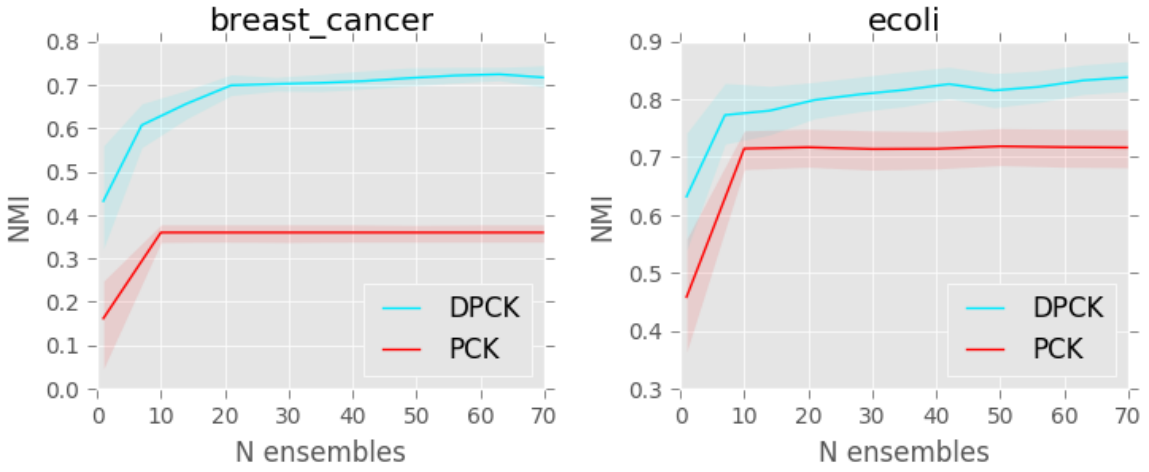
Name	DPCK	95%	PCK	95%	p-value
ecoli	**0.963**	(0.963, 0.963)	0.960	(0.956, 0.963)	0.067
segmentation	0.826	(0.811, 0.84)	**0.838**	(0.814, 0.867)	0.436
imdb	0.666	(0.664, 0.669)	**0.685**	(0.674, 0.696)	0.001
UCI_creditcard	**0.780**	(0.775, 0.784)	**0.780**	(0.769, 0.79)	0.931
creditcard	**0.996**	(0.996, 0.997)	0.967	(0.967, 0.967)	0.000
contraceptive	**0.527**	(0.52, 0.534)	0.516	(0.505, 0.526)	0.087
iris	0.981	(0.979, 0.983)	**0.986**	(0.978, 0.994)	0.297
breast_cancer	**0.948**	(0.944, 0.951)	0.892	(0.882, 0.903)	0.000
spect	0.428	(0.387, 0.468)	**0.535**	(0.423, 0.653)	0.080
titanic	0.829	(0.824, 0.834)	**0.844**	(0.834, 0.854)	0.010
mcdonald	0.780	(0.765, 0.792)	**0.821**	(0.808, 0.833)	0.000
abalon	0.198	(0.189, 0.207)	**0.232**	(0.222, 0.243)	0.000

Table 7.1.: Prediction accuracy for different data sets using the DPCK and PCK. The highest score for each data set is highlighted in bold.

In table 7.1 we see the prediction accuracy using the PCK and DPCK on different data sets. We see that there is only a few data sets where we have statistical significance for different means. The PCK seems to perform better then the DPCK when the data has categorical features. While the DPCK performs better when we only have continuous features. Additionally the DPCK might perform better when the data set has an uneven amount of observation in each cluster or one of the clusters is more compact (in terms of variance).

7.3.2. Clustering

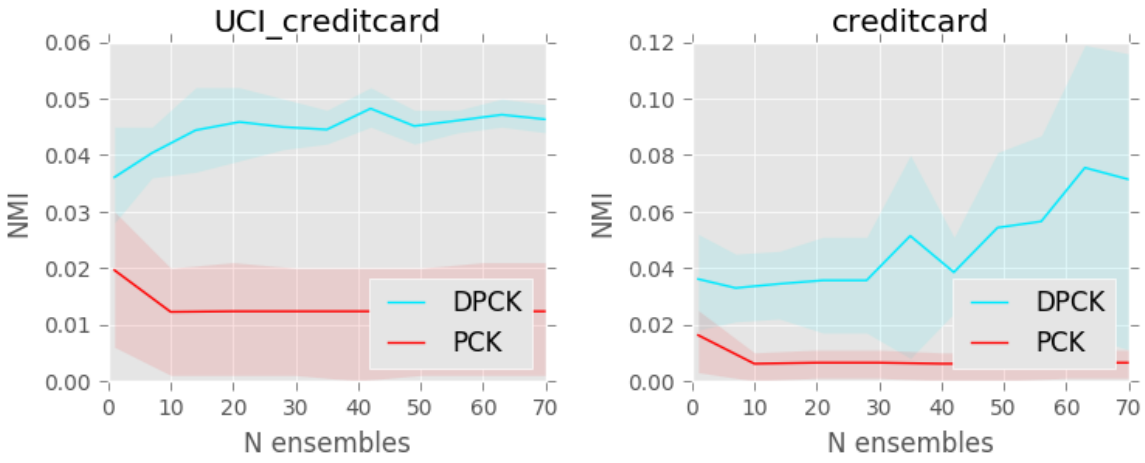
In figure 7.5 we see that the PCK have a wider confidence interval, and having a lower NMI value. In terms of convergence the two cluster kernels seems to obtain the best value after 10-15 ensembles.



(a) (b)

Figure 7.5.: a) Clustering NMI score for the Breast cancer data set. Comparison of the NMI as a function of the total number of ensembles between the DPCK and PCK. b) Clustering NMI score for the Ecoli data set. Comparison of the NMI as a function of the total number of ensembles between the DPCK and PCK.

In figure 7.6 we see that both the cluster kernels have problems scoring high using NMI for both the credit card data sets. Additionally the PCK seems to have problems learning the similarities in any good way, as the score decreases with more ensembles. The DPCK give more stable results, but still scoring in the 4% range. As the DPCK reaches 70 ensembles for the credit card fraud detection data set the results seems to be very unstable. This tells us that we might need more ensembles for this specific data set.



(a) (b)

Figure 7.6.: a) Clustering NMI score for the UCI creditcard data set. Comparison of the NMI as a function of the total number of ensembles between the DPCK and PCK. b) Clustering NMI score for the creditcard data set. Comparison of the NMI as a function of the total number of ensembles between the DPCK and PCK.

We can see in table 7.2 that in many cases we get a very low NMI score. Additionally we see that the DPCK give higher scores for most data sets, and there are indicators for significantly better mean. It is interesting to see that the DPCK do better on unsupervised problems.

Name	DPCK	95%	PCK	95%	p-value
ecoli	**0.838**	(0.814, 0.866)	0.719	(0.685, 0.751)	0.000
segmentation	**0.658**	(0.634, 0.681)	0.606	(0.575, 0.646)	0.019
imdb	0.028	(0.028, 0.028)	**0.054**	(0.046, 0.061)	0.000
UCI_creditcard	**0.046**	(0.044, 0.049)	0.012	(0.0, 0.021)	0.000
creditcard	**0.071**	(0.011, 0.116)	0.006	(0.001, 0.011)	0.014
contraceptive	0.045	(0.043, 0.047)	**0.052**	(0.045, 0.058)	0.040
iris	0.702	(0.695, 0.71)	**0.716**	(0.683, 0.745)	0.400
breast_cancer	**0.717**	(0.695, 0.745)	0.360	(0.338, 0.378)	0.000
spect	**0.219**	(0.204, 0.233)	0.190	(0.147, 0.231)	0.197
titanic	**0.164**	(0.161, 0.168)	0.096	(0.078, 0.11)	0.000
mcdonald	**0.558**	(0.538, 0.581)	0.430	(0.411, 0.451)	0.000
abalon	0.258	(0.254, 0.262)	**0.259**	(0.255, 0.264)	0.659

Table 7.2.: NMI for different data sets using the DPCK and PCK. The highest score for each data set is highlighted in bold.

7.3.3. Dimensionality Reduction

In this section we visualize the kernels by looking at the two top principal components using KPCA for different kernels. For these experiments we use the DPCK, PCK and a linear kernel. The goal here is to visually inspect the cluster discrimination differences between the kernels, and general structurally differences.

The PCK and DPCK is created in a similar manner, but there is a major difference between the single models. The PCK forces local and global scales by iterating through different number of clusters to use. As such we assume to less compact clusters from the PCK. Creating the DPCK we recall the number of clusters used in each ensemble that we saw in figure 7.2. There we saw that in general the number of clusters for a single model is close to the initial number of clusters. Therefore we assume more compact clusters, as we stay within the same scale for each ensemble.

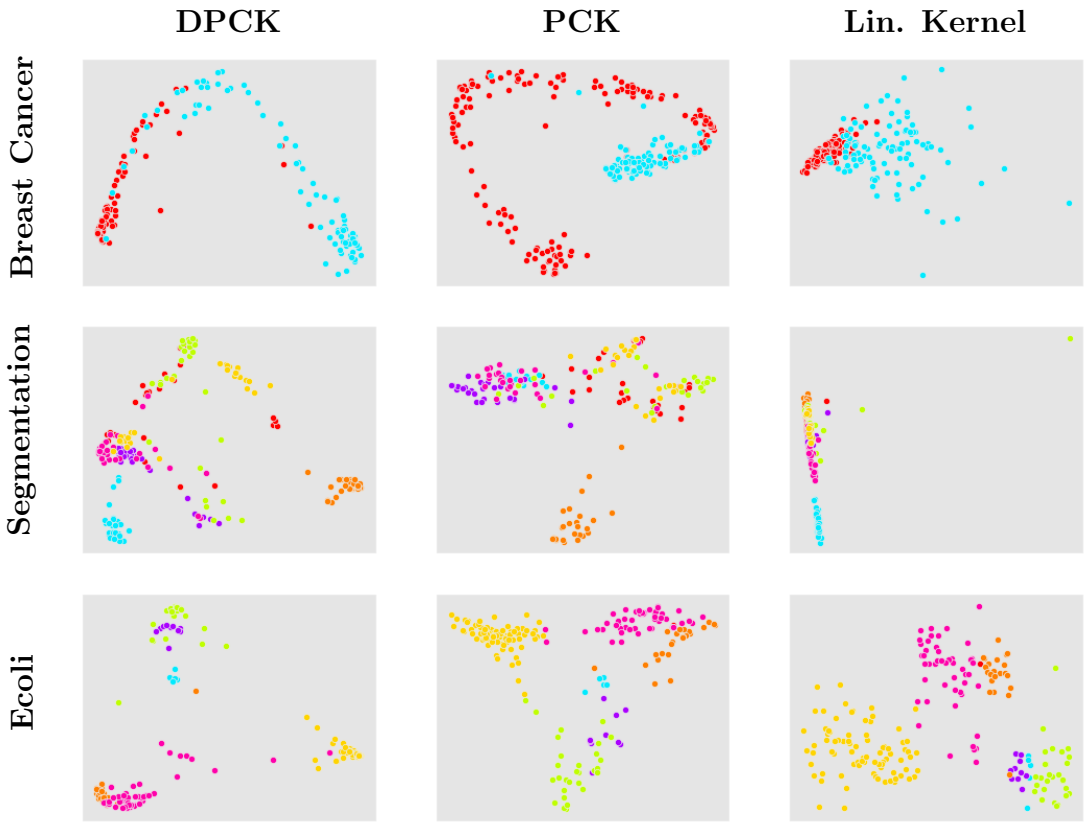


Figure 7.7.: Projection of three data sets onto the top two principal components using KPCA on the DPCK, PCK and a linear kernel. Colors indicate true class labels.

In figure 7.7 we see the difference between the principal components of the two cluster kernels. The PCK tend to give wider clusters, while the PCK give more compact clusters. It is interesting to see that the PCK and DPCK manage to isolate different clusters for the breast cancer data. The PCK focus on the small compact cluster, while the

DPCK evenly spread the two clusters. We see in the segmentation kernels that the DPCK manage to mostly isolate four clusters, while the linear kernel isolate only one. Additionally we see here that the DPCK seems to produce more compact clusters. The difference in the spread of each clusters is seen for the other data sets as well. The linear kernel have problems discriminating the green, teal and purple clusters, both the DPCK and PCK manage that. Since the spread of each cluster is larger for the PCK, the discrimination is higher using the DPCK.

8. Semi-Supervised Learning

In many applications we often have a small amount of labeled samples available, and a large amount of unlabeled samples. In such cases we are often interested in learning using all the data, e.g. using all the information available. A method for doing this is to use the full data set to learn an unsupervised kernel.The learned kernel should represent the structure of the few labeled observations better [126, 127]. The basic idea is to first map all of the data to some feature space using one of the many kernel functions available. Then to train a learning algorithm on the few labeled observations, using the created kernel. Where the feature space is a better representation of the structure between the observation.

This chapter contains a contribution in that we apply the DPCK in some semi-supervised classification problems. We hypothesize that the DPCK learns similarities and structures in a more general way, then other kernel functions that have critical shape parameters (like the RBF). Additionally the DPCK learns the cluster kernel based on the data, without critical parameters. The DPCK should have some advantages over using existing kernel functions for creating the kernel.

- We create a kernel without any prior assumptions on the structure of the data. Using other kernel functions we often have to tune a critical shape parameter, like γ for a RBF.
- Since the DPCK is learned from clustering the data, we hypothesize that the final clustering should contain the full structure in a better way.

In this chapter we run experiments to show the effectiveness of the DPCK to capture similarities of unlabeled observations. We investigate the prediction accuracy in a semi-supervised framework when we have a small number of labeled observations.

8.1. Experimental Setup

We create the DPCK with the same parameters as chapter 7. For these experiments we do the following:

1. Create DPCK following algorithm 10 and 11.
2. Randomly choice N_k observations to be the labeled observations, with some size $< N$.
3. Train SVM using the N_k observations from the DPCK.

4. Test using $N - N_k$ observations from the DPCK, where the test kernel now has shape $(N - N_k) \times N_k$.

For each data set we repeat the random sampling of labeled observations 20 times, and calculate the mean for all of these. Additionally we plot the 95% confidence interval, using the empirical bootstrapped confidence interval discussed in section 3.6.1.

To further examine the performance of the learned kernel, we compare it with a RBF with $\gamma = 0.5$. For these experiments we do not tune γ in any way, we choose the value rather arbitrary.

8.2. Results

In figure 8.1 we see the prediction accuracy for four different data sets. We note that as we add more constraints (here as a function of % of the total sample size) the prediction accuracy increases. One interesting fact to note is that for the Iris species data set the accuracy improves more then 70%, from around 0.7 to 0.95.

In table 8.1 we see the prediction accuracy for four different data sets together with the standard deviation. We fix the number of labeled observations to 5% and 10% of the total sample size. Interestingly we see that as we increase the number of labeled observations the DPCK have a higher increase in prediction accuracy.

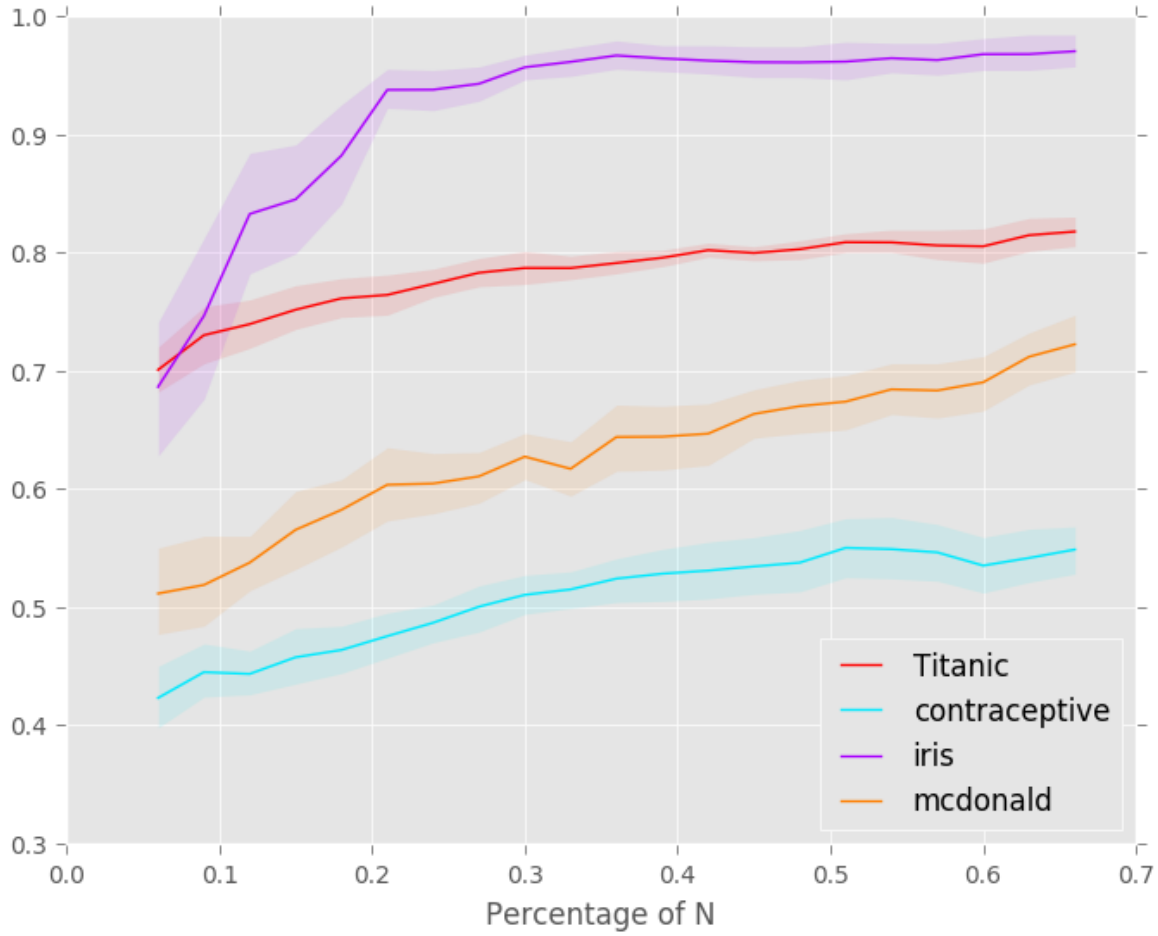


Figure 8.1.: Prediction accuracy as a function of the percentage of labeled data points. Shaded regions represent the 95% confidence interval

	DPCK 5%	DPCK 10%	RBF 5%	RBF 10%
Titanic	0.667 ± 0.058	0.722 ± 0.060	0.608 ± 0.081	0.625 ± 0.068
breast_cancer	0.909 ± 0.070	0.934 ± 0.022	0.648 ± 0.061	0.656 ± 0.084
ecoli	0.773 ± 0.069	0.806 ± 0.074	0.390 ± 0.110	0.488 ± 0.061
segmentation	0.369 ± 0.112	0.468 ± 0.104	0.199 ± 0.034	0.237 ± 0.040

Table 8.1.: Prediction accuracy when training using 5%/10% of the observations. Comparison between the DPCK and RBF kernel, with $\gamma = 0.5$. The values represent mean ± standard deviation over 20 runs.

Part IV.

Conclusion

9. Summary

The aim for the thesis was to consider the application of a nonparametric mixture model in the cluster kernel framework. To learn the existing PCK, the number of clusters had to be determined a priori to learning the kernel, thus making parametric assumptions. The DPMM was applied to the PCK framework to create the DPCK. A cluster kernel without the critical C parameter. Further, experiments was conducted to investigate the behavior of the DPCK by using the learned kernel in both classification and clustering problems. The DPCK was compared against the PCK in all learning tasks. Results illustrated that the DPCK learned more compact clusters and performed good as a similarity kernel when applied to clustering problems. Additionally, some new contributions where made when the cluster kernel framework was applied to semi-supervised classification. In the semi-supervised experiments the DPCK demonstrated that the learned similarities was a very general representation of the data.

In the first part of the thesis theory behind kernels, clustering and ensemble clustering where explored. Then the statistical methodology was introduced together with the DPMM. Finally, all these results where combined to demonstrate the PCK framework.

The second part contained one of the main contributions of the thesis. Here the proposed DPCK was proposed and an algorithm was proposed for learning the cluster kernel. The new cluster kernel is learned without prior assumptions about any shape or number of clusters.

The third part contained all the experiments of the thesis. First experiments where conducted to examine the stability of the DPMM with respect to its parameters. In these experiments we saw that the DPMM was very stable, and looked to not have any critical parameters. Additionally we ran experiments on multiple data sets to compare the DPCK with the PCK, using the learned kernels on both classification and clustering problems. The DPCK demonstrated good results when applied for clustering, but the PCK performed slightly better for classification. However, for some types of data set the DPCK looked to perform better. Finally the proposed DPCK was applied to some semi-supervised classification problems. Here we saw that the DPCK learned the full structure of the data, and had better performance then a single RBF kernel.

10. Discussion

10.1. Dirichlet Process Mixture Models

First we ran experiments to investigate the DPMM and how the parameters changes the results. We saw in section 6.2 that the DPMM have dependencies with five parameters. Due to the DPMM predicting different number of mixture components from different runs, we choice to visually inspect the segmentations to inspect the results. There exists methods to measure the accuracy of different clustering partitions with different numbers of components, however for the objective of this chapter we found that visually inspecting the segmentations was sufficient. Exploring the parameters one at a time, we saw that the final segmentation obtained by the DPMM was feasible for a wide range of parameter values. We saw that choosing multiple parameters with outliers values (low/high) gave feasible results in the end. However we do note that some parameters where more influential then the others. We saw that the parameter κ_0 was the most influential parameter. Where the difference between low and high values was big.

To summarize the discussion about the DPMM we conclude that the model was very stable for the experiments we ran. The parameter range for where the model would behave nicely and converge to a good solution looks to be wide.

10.2. Investigating the Performance of the DPCK

In chapter 7 we saw that the DPCK learned similarities in a unsupervised way, without prior parametric assumptions. Additionally we saw that the DPCK was very good compared to the PCK on some kinds of problems. In this section we want to discuss what we learned from the experiments.

- The DPCK have good classification accuracy for some types of data sets.
- The DPCK have good clustering accuracy for most types of data sets.
- The DPCK create compact clusters and is good at discriminating between different clusters.
- The proposed DPCK learned using the collapsed Gibbs sampler is limited by the implemented algorithm.

10.2.1. The Dirichlet Process Cluster Kernel

One of the contributions of the thesis was to investigate the use of a nonparametric clustering model within the cluster kernel framework. To examine the performance of the proposed DPCK, we ran experiments using both classification and clustering to assess the cluster kernel. Additionally we visually inspected the projection on the two top eigenvectors using KPCA, on a selection of data sets.

From the results obtained in the classification problems in section 7.3.1, we saw that the PCK was better for data sets containing categorical features. Data sets having continuous features gave the edge to the DPCK. Additionally we saw that the DPCK produced predictions with a smaller confidence interval, suggesting that the results are more stable.

From the results for the clustering experiments in section 7.3.1 we saw that the DPCK was in general equal or slightly better for most data sets. It was interesting to see that the DPCK learned a better kernel for unsupervised methods. In general we saw that the DPCK produced kernels with more compact clusters, as such we suspect that the more discriminated clusters helps in an unsupervised method.

For visually inspecting the two dimensional projections we saw that the DPCK created a more compact representation of different clusters, while the PCK tended to create wider clusters. Additionally, the DPCK seemed to give more discrimination between clusters in some cases. This might also be confirmation bias, as having more compact clusters will make it look more separated. For a learning algorithm the two kernels might be similar in these cases. However, if we look at the classes the two kernels failed to discriminate. The PCK tends to give wider representations of the clusters. When the kernels group multiple clusters together, the wider representation might make it easier to discriminate some of the observations within the grouping. This might be a reason for why the PCK seemed to have an edge for classification.

It is important to note that using a more complex model within the cluster kernel framework increases the total computation time in a significant way. One of the major difficulties with the thesis project was the time it took to learn the DPCK. Clearly the reason for the major time increase for the DPMM was due to the algorithm itself. As discussed in section 3.10 there exists much faster algorithms. Using one of these would make the DPCK be feasible for anything but small test data sets. Such that following the proposed framework; if resources and time is an issue the PCK might give reasonably good enough results.

Heuristics

Another thing to consider is the heuristics we use to create the DPCK. We sample three parameters uniformly from the ranges:

$$\begin{aligned} \alpha &\sim \text{Unif}(10^{-2}, 1) \\ \kappa_0 &\sim \text{Unif}(10^{-3}, 1) \\ \nu_0 &\sim \text{Unif}(D + 25, D + 100) \end{aligned} \tag{10.1}$$

We do note that for α a log-uniformly distribution between $[10^{-2}, 10^2]$ could be good, looking at the results discussed in section 10.1. Additionally as we saw that κ_0 was one of the most influential parameters, we could sample $\kappa_0 \sim \text{Exp}(\gamma)$ such that we allowed values greater then 1 (but very rarely). Or we could sample κ_0 log-uniformly in the same manner as proposed for α. This would further increase the diversity within the cluster kernel, and explore more of the parameter space.

10.2.2. Different Data Sets

We see from the results in chapter 7 that the DPCK is expected to perform better for some kinds of data sets. Looking at the results from clustering and classification we suspect that the DPCK performs better for some specific data sets. i) Data sets where the clusters have different compactness, in terms of variance. We speculate that the reason for this is caused by the flexibility of the single ensemble covariance structure. The PCK might do better here if we used a prior for the covariance structure within the single GMM. ii) Because of the Dirichlet process prior on the mixture weights, we know that the marginal distributions for the weights is Dirichlet distributed. As such we expect the mixture weights to fit good on data with uneven number of observations within each cluster.

Further we saw that the PCK seemed to learn a better kernel when the data had categorical features.

10.2.3. Algorithmic Limitations

In section 7.3 we saw the distribution for the number of clusters used in the DPCK, for different initial number of clusters used. Here we saw that the collapsed Gibbs sampler did not add additional clusters, even when the true number of clusters was higher then the initial. This can be due to the fact that the collapsed Gibbs sampler only propose local changes. As we only sample one cluster assignment at a time, we only see small local changes in the algorithm. Adding clusters in the algorithm happen frequently, however, to not remove the cluster in the next iteration we have to assign a new observation to the cluster before doing one iteration. If the algorithm fails this, the cluster vanishes. Based on this we can suspect that the implemented algorithm have slow convergence to the global maximals and might be stuck in local maximals for a significant amount of time.

In section 3.10 we discussed methods to improve the speed of the collapsed Gibbs sampler for the DPMM. An algorithm where we allow the model to split and merge clusters will mean that we are not so dependent on the initiation for each ensemble. We suspect that with an fast split/merge algorithm we will gain an even larger improvement over the parametric PCK. Improving the base algorithm will decrease the total time needed to construct the DPCK. Therefore an faster algorithm will allow us to test larger and more interesting data sets.

In all our experiments we fixed the number of iterations of each ensemble to 20 due to time constraints. This might prove to be a much to low number for an MCMC method,

such that with an improved algorithm we could check these hypothesis within a feasible time-frame.

10.3. Semi-Supervised Classification

In section 8.2 we wanted to further examine the descriptive power of the DPCK. To do this we applied the cluster kernel to some semi-supervised problems. We learned the kernel using the full data, then we trained a classifier on 5%/10% of the data. We calculated the predictive accuracy on the rest of the data, and compared against an unsupervised RBF kernel. The results showed that the DPCK seemed to have very good generalization power. Because when we changed number of observations to train the data on, the relative increase in predictive accuracy was larger for the DPCK compared to the RBF kernel. This might indicate that the learned similarities in the DPCK are a more descriptive representation of the data.

11. Future Work

From the thesis we can propose many interesting approaches for future work/research.

11.1. Split/Merge Algorithm

The biggest improvement to the proposed DPCK would be to implement a faster algorithm that exploit split/merge moves. One possible choice would be the fast algorithm proposed in [109], as discussed in section 3.10. The author suspect that as we allow split/merge moves, the algorithm converges faster and to a better solution. Thus, the final DPCK should depend less on the initial number of clusters.

11.1.1. Different Data Sets

One of the limitation working with the thesis was the computational time for the collapsed Gibbs sampler that was implemented. With a faster and better algorithm one could investigate larger data sets. We suspect that as the data sets grow larger, the difference between the PCK and DPCK increases. Both because tuning the parameters C and Q for the PCK can get increasingly hard as we get more data, also because of the flexibility of the DPMM.

11.2. Semi-Supervised Cluster Kernels

In [128] they propose an interesting boosting-like approach for unsupervised clustering ensembles. They create the weight distribution for each observation according to the consistency of the clustering assignment it got in the previous ensembles. Thus as they create ensembles they sample observations to increasingly focus on the problematic regions. In [129] they have a similar approach, just for feature selection (recursive feature elimination scheme). Another interesting approaches to boosting-like approaches for unsupervised learning is to calculate the out of model feature importance score. Some examples for such scores is the explained variance from PCA, or Laplacian score [130]. Laplacian score is a way to use the graph Laplacian to infer on the importance of each feature using the local structure. If we want to focus om semi-supervised cluster kernel approaches like relative feature importance from learned models, like in [131]. Or one could calculate the Fisher score [132].

As we saw the DPCK was very good at learning general data structures, compared to other kernel functions within the semi-supervised framework. It would be interesting

to look at the constrained DPMM discussed in [20, 21] and combing the constrained sampler with the boosting like subsamples of observations and features with the DPCK. Either fully unsupervised learning process for the cluster kernel or through supervised methods on the few labeled observations . However the unsupervised cluster kernel approach is probably more feasible, as the number of labeled examples is usually small.

11.3. Missing Data

Since we are working with a Bayesian model, putting a prior on the missing data should be possible for these classes of models. In [77] they create a cluster kernel for multivariate time series with missing data. The missing data approach to cluster kernels looks to give promising results, within an ensemble framework the lack of data should not impact the results as much as it would when using a single model. As such looking into how to model missing data inside the DPMM and DPCK framework would be an interesting approach for future work.

A. Databook

Databook

```
In [37]: import matplotlib.pyplot as plt
         import numpy as np
         import random
         import pandas as pd
         from os import listdir
         from sklearn.decomposition import PCA, KernelPCA
         #from sklearn.manifold import SpectralEmbedding as PCA
         plt.style.use('classic')
         plt.style.use('ggplot')
         col_list = ['#FF0000', '#00EAFF', '#AA00FF',
                     '#FF7F00', '#BFFF00', '#FF00AA',
                     '#FFD400', '#6AFF00', '#EDB9B9',
                     '#0040FF', '#8F2323', '#8F6A23',
                     '#6B238F', '#4F8F23', '#000000',
                     '#737373']
         data_path = '/home/toby/master/data/data_set/'
```

```
In [38]: def plot_classes(df, labels, ax):
             df = np.array(df)
             labels = np.array(labels)
             k = 0
             for i in np.unique(labels)[::-1]:
                 ax.scatter(df[labels == i, 0], df[labels == i, 1],
                            c=col_list[k%len(col_list)])
                 k += 1

         def plot_dim_rel(df, labels, fig):
             size = df.shape[1]
             labels = np.array(labels)
             index_pairs = [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]
             for i in range(6):
                 ax = fig.add_subplot(2, 3, i+1)
                 plot_classes(df[:, [index_pairs[i][0], index_pairs[i][1]]], labels, ax)
                 plt.axis('off')
```

1 UCI Credit Card Defaults

This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.

1.1 Variables

There are 25 variables:

- ID: ID of each client
- LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
- SEX: Gender (1=male, 2=female)
- EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
- MARRIAGE: Marital status (1=married, 2=single, 3=others)
- AGE: Age in years
- PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months, 9=payment delay for nine months and above)
- PAY_2: Repayment status in August, 2005 (scale same as above)
- PAY_3: Repayment status in July, 2005 (scale same as above)
- PAY_4: Repayment status in June, 2005 (scale same as above)
- PAY_5: Repayment status in May, 2005 (scale same as above)
- PAY_6: Repayment status in April, 2005 (scale same as above)
- BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
- BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
- BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
- BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
- BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
- BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
- PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
- PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
- PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
- PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
- PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
- PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
- default.payment.next.month: Default payment (1=yes, 0=no)

1.2 Processing Data

```
In [39]: df = pd.read_csv(data_path + 'UCI_creditcard/UCI_Credit_Card.csv')
         labels = df.iloc[:, -1]
         df = df.drop(['ID', df.columns[-1]], axis=1)
         df['labels'] = labels
         df[['LIMIT_BAL', 'BILL_AMT1', 'BILL_AMT2','BILL_AMT3',
            'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6',
            'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
            'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']] /= df[['BILL_AMT1', 'BILL_AMT2','BILL_AMT3',
            'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6',
            'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
            'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']].max().max()
         print('Shape of data set: ', df.shape)
```

```
Shape of data set:  (30000, 24)
```

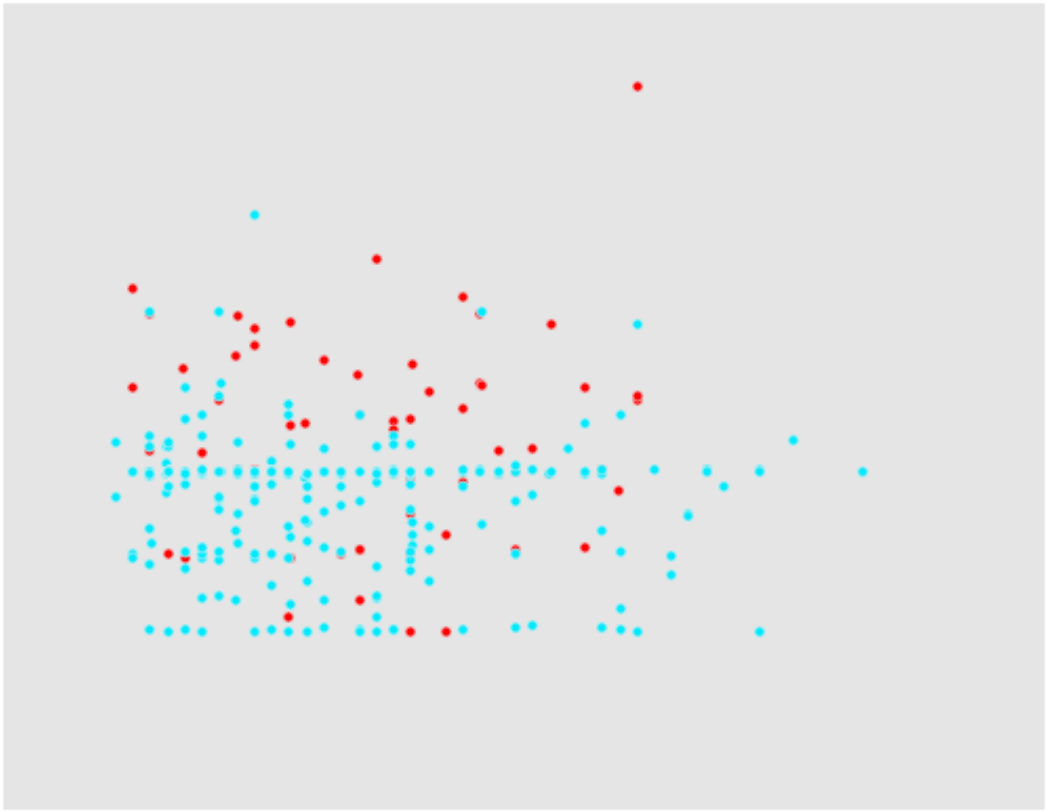
We create a subsample for $N = 300$ for both training and test phases.

```
In [40]: random.seed(123)
         np.random.seed(123)
         indexes = [i for i in range(df.shape[0])]
         random.shuffle(indexes)
         df_train = df.iloc[indexes[:300], :]
         df_test = df.iloc[indexes[300:600], :]
         print(np.unique(df_train['labels'], return_counts=True))
         print(np.unique(df_test['labels'], return_counts=True))
         df_train.to_csv(data_path + 'UCI_creditcard/UCI_creditcard_train.csv', index=False)
         df_test.to_csv(data_path + 'UCI_creditcard/UCI_creditcard_test.csv', index=False)
```

```
(array([0, 1]), array([236,  64]))
(array([0, 1]), array([229,  71]))
```

1.3 Visualizing Data

```
In [41]: fig = plt.figure()
         ax = fig.add_subplot(111)
         pca = PCA(2)
         data = pca.fit_transform(df_train.iloc[:, :-1])
         plot_classes(data, df_train.iloc[:, -1], ax)
         ax.set_xticks([])
         ax.set_yticks([])
         plt.show()
```



2 Credit Card Fraud Detection

The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

2.1 Variables

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

2.2 Processing Data

```
In [42]: df = pd.read_csv(data_path + 'creditcard/creditcard.csv')
         labels = df['Class'].values

         def time_converter(time):
             return np.floor(time/(3600))%24, np.floor(time/(60))%60

         hour, min_ = time_converter(df['Time'])
         df['Hour'] = hour
         #df['Min'] = min_
         #df['Amount'] = df['Amount']/df['Amount'].max()
         df = df.drop(['Class', 'Time'], axis=1)
         df['labels'] = labels
         print(df.shape)
         df.head()
```

```
(284807, 31)
```

Out[42]:

	V1	V2	V3	V4	V5	V6	V7	\
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	

	V8	V9	V10	...	V22	V23	V24	\
0	0.098698	0.363787	0.090794	...	0.277838	-0.110474	0.066928	
1	0.085102	-0.255425	-0.166974	...	-0.638672	0.101288	-0.339846	
2	0.247676	-1.514654	0.207643	...	0.771679	0.909412	-0.689281	
3	0.377436	-1.387024	-0.054952	...	0.005274	-0.190321	-1.175575	
4	-0.270533	0.817739	0.753074	...	0.798278	-0.137458	0.141267	

	V25	V26	V27	V28	Amount	Hour	labels
0	0.128539	-0.189115	0.133558	-0.021053	149.62	0.0	0
1	0.167170	0.125895	-0.008983	0.014724	2.69	0.0	0
2	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0.0	0
3	0.647376	-0.221929	0.062723	0.061458	123.50	0.0	0
4	-0.206010	0.502292	0.219422	0.215153	69.99	0.0	0

[5 rows x 31 columns]

We create a subsample of $N = 300$ where we fix the number of frauds to 10 per data set.

```
In [43]: np.random.seed(123)
         n_scams = 10
         n_non_scams = 300-n_scams
         df_processed = df.iloc[labels == 0, :].iloc[np.random.randint(0, np.sum(labels == 0), n
```

```
        df_processed = df_processed.append(df.iloc[labels == 1, :].iloc[np.random.randint(0, np
        print(df_processed.shape)
        df_processed.to_csv(data_path + 'creditcard/creditcard_train_300.csv', index=False)

        np.random.seed(2)
        df_processed = df.iloc[labels == 0, :].iloc[np.random.randint(0, np.sum(labels == 0), n
        df_processed = df_processed.append(df.iloc[labels == 1, :].iloc[np.random.randint(0, np
        print(df_processed.shape)
        df_processed.to_csv(data_path + 'creditcard/creditcard_test_300.csv', index=False)
```

```
(300, 31)
(300, 31)
```

2.3 Visualizing Data

```
In [44]: fig = plt.figure()
         ax = fig.add_subplot(111)
         pca = PCA(2)
         data = pca.fit_transform(df_processed.iloc[:, :-1])
         plot_classes(data, df_processed.iloc[:, -1], ax)
         ax.set_xticks([])
         ax.set_yticks([])
         plt.show()
```

3 UCI Segmentation

The instances were drawn randomly from a database of 7 outdoor images. The images were handsegmented to create a classification for every pixel.

Each instance is a 3×4 region.

3.1 Variables

1. region-centroid-col: the column of the center pixel of the region.
2. region-centroid-row: the row of the center pixel of the region.
3. region-pixel-count: the number of pixels in a region = 9.
4. short-line-density-5: the results of a line extractoin algorithm that counts how many lines of length 5 (any orientation) with low contrast, less than or equal to 5, go through the region.
5. short-line-density-2: same as short-line-density-5 but counts lines of high contrast, greater than 5.

6. vedge-mean: measure the contrast of horizontally adjacent pixels in the region. There are 6, the mean and standard deviation are given. This attribute is used as a vertical edge detector.
7. vegde-sd: (see 6)
8. hedge-mean: measures the contrast of vertically adjacent pixels. Used for horizontal line detection.
9. hedge-sd: (see 8).
10. intensity-mean: the average over the region of (R + G + B)/3
11. rawred-mean: the average over the region of the R value.
12. rawblue-mean: the average over the region of the B value.
13. rawgreen-mean: the average over the region of the G value.
14. exred-mean: measure the excess red: (2R - (G + B))
15. exblue-mean: measure the excess blue: (2B - (G + R))
16. exgreen-mean: measure the excess green: (2G - (R + B))
17. value-mean: 3-d nonlinear transformation of RGB. (Algorithm can be found in Foley and VanDam, Fundamentals of Interactive Computer Graphics)
18. saturatoin-mean: (see 17)
19. hue-mean: (see 17)

3.2 Processing Data

```
In [45]: df = pd.read_csv(data_path + 'segmentation/training_data.csv')
         df_test = pd.read_csv(data_path + 'segmentation/test_data.csv')

         def string_to_category(string):
             if string == 'BRICKFACE':
                 return 0
             elif string == 'CEMENT':
                 return 1
             elif string == 'FOLIAGE':
                 return 2
             elif string == 'GRASS':
                 return 3
             elif string == 'PATH':
                 return 4
             elif string == 'SKY':
                 return 5
             elif string =='WINDOW':
                 return 6
             else:
```

```
            return np.NaN

    labels_train = df['labels'].apply(string_to_category)
    labels_test = df_test['labels'].apply(string_to_category)
    random.seed(123)
    indexes = [i for i in range(df_test.shape[0])]
    random.shuffle(indexes)
    df = df.drop(['labels',
                  'REGION-PIXEL-COUNT',
                  'SHORT-LINE-DENSITY-5',
                  'SHORT-LINE-DENSITY-2'], axis=1)
    df_test = df_test.drop(['labels',
                            'REGION-PIXEL-COUNT',
                            'SHORT-LINE-DENSITY-5',
                            'SHORT-LINE-DENSITY-2'], axis=1)
    df['labels'] = labels_train
    df_test['labels'] = labels_test
```

```
In [46]: df_test = df_test.iloc[indexes[:df.shape[0]], :]
         print('Train shape: ', df.shape)
         print('Test shape: ', df_test.shape)
         df.to_csv(data_path + 'segmentation/training_data_processed.csv', index=False)
         df_test.to_csv(data_path + 'segmentation/test_data_processed.csv', index=False)
         df.head()
```

```
Train shape:  (210, 17)
Test shape:  (210, 17)
```

Out[46]:

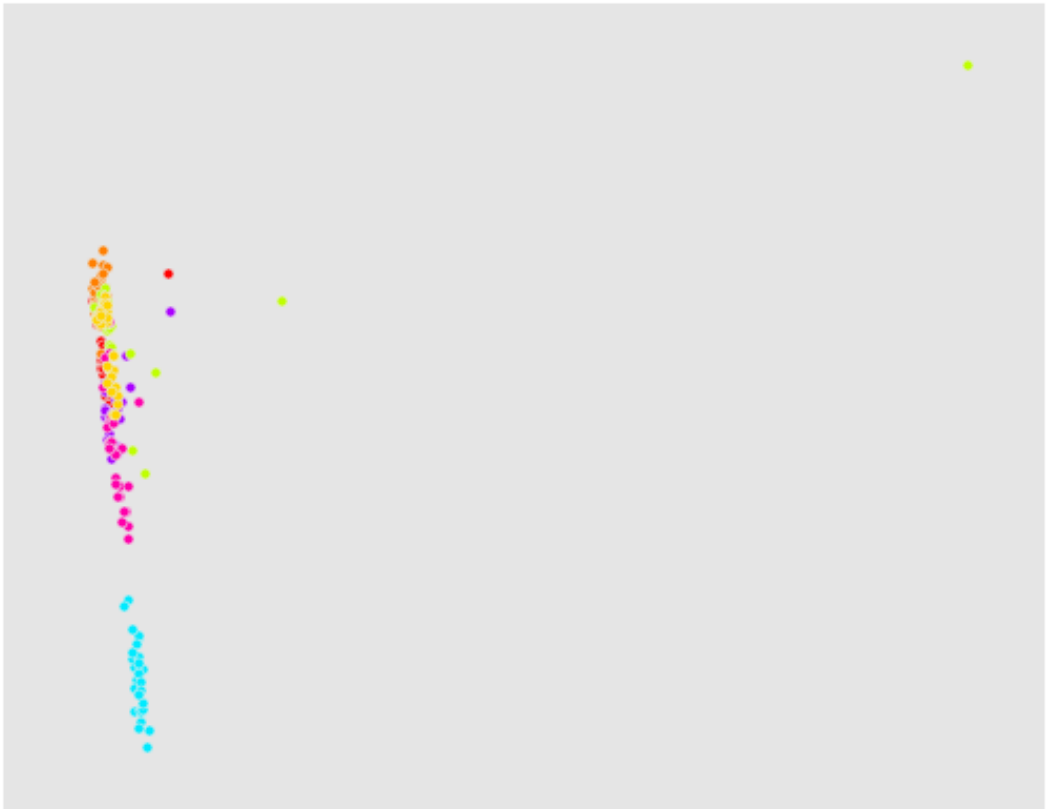
	REGION-CENTROID-COL	REGION-CENTROID-ROW	VEDGE-MEAN	VEDGE-SD	HEDGE-MEAN	HEDGE-SD	INTENSITY-MEAN	RAWRED-MEAN	RAWBLUE-MEAN	RAWGREEN-MEAN	EXRED-MEAN	EXBLUE-MEAN	EXGREEN-MEAN	VALUE-MEAN	SATURATION-MEAN
0	140.0	125.0	0.277778	0.062963	0.666667	0.311111	6.185185	7.333334	7.666666	3.555556	3.444444	4.444445	-7.888889	7.777778	0.545635
1	188.0	133.0	0.333333	0.266667	0.500000	0.077778	6.666666	8.333334	7.777778	3.888889	5.000000	3.333333	-8.333333	8.444445	0.538580
2	105.0	139.0	0.277778	0.107407	0.833333	0.522222	6.111111	7.555555	7.222222	3.555556	4.333334	3.333333	-7.666666	7.555555	0.532628
3	34.0	137.0	0.500000	0.166667	1.111111	0.474074	5.851852	7.777778	6.444445	3.333333	5.777778	1.777778	-7.555555	7.777778	0.573633
4	39.0	111.0	0.722222	0.374074	0.888889	0.429629	6.037037	7.000000	7.666666	3.444444					

```
4    2.888889     4.888889     -7.777778    7.888889          0.562919
```

```
   HUE-MEAN  labels
0 -1.121818       0
1 -0.924817       0
2 -0.965946       0
3 -0.744272       0
4 -1.175773       0
```

3.3 Visualizing Data

In [47]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
pca = PCA(2)
data = pca.fit_transform(df.iloc[:, :-1])
plot_classes(data, df.iloc[:, -1], ax)
ax.set_xticks([])
ax.set_yticks([])
plt.show()
```



4 UCI Iris Species

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

4.1 Variables

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
    - Iris Setosa
    - Iris Versicolour
    - Iris Virginica

4.2 Processing Data

```
In [48]: df = pd.read_csv(data_path + 'iris_species/Iris.csv')
         labels = df['Species']
         df = df.drop(['Species', 'Id'], axis=1)

         def converter(species_string):
             if species_string == 'Iris-setosa':
                 return 0
             elif species_string == 'Iris-versicolor':
                 return 1
             elif species_string == 'Iris-virginica':
                 return 2
             else:
                 return np.NaN

         df['labels'] = labels.apply(converter)
         df.shape

Out[48]: (150, 5)
```

Create train set with $N = 100$ and test set containing $N = 50$ observations

```
In [49]: random.seed(123)
         indexes = [i for i in range(150)]
         random.shuffle(indexes)
```

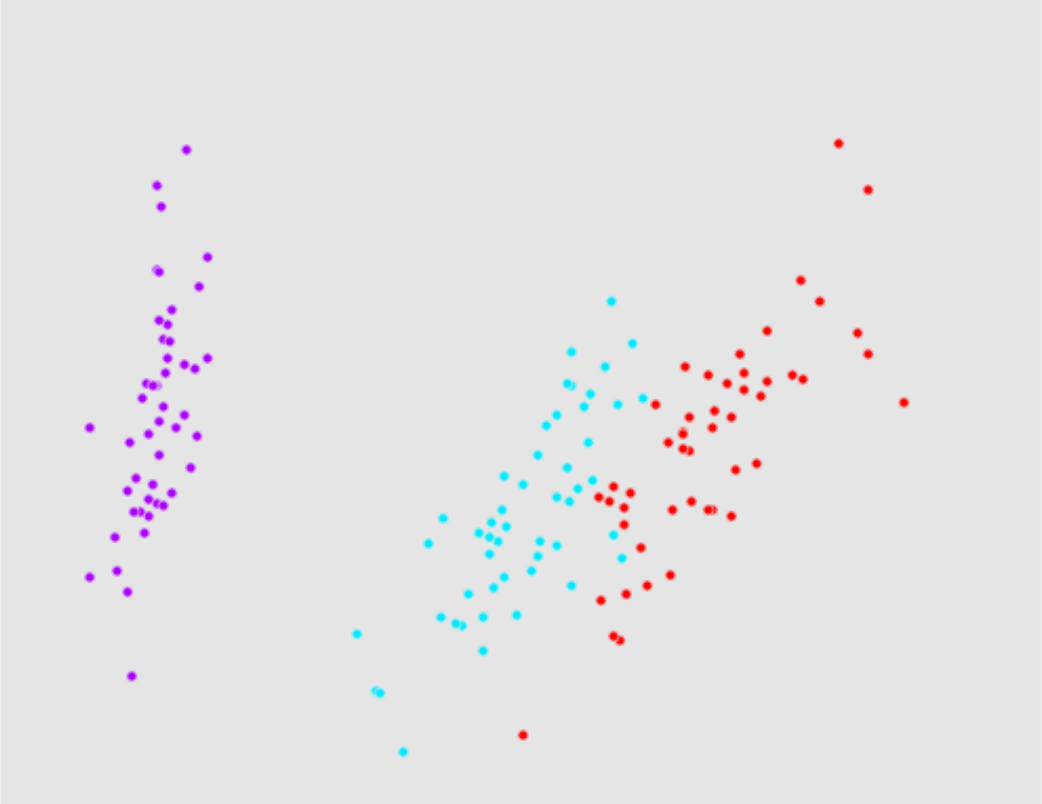
```
df_train = df.iloc[indexes[:], :]
df_test = df.iloc[indexes[30:], :]
df_train.to_csv(data_path + 'iris_species/Iris_train.csv', index=False)
df_test.to_csv(data_path + 'iris_species/Iris_test.csv', index=False)
df_train.head()
```

Out[49]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	labels
79	5.7	2.6	3.5	1.0	1
132	6.4	2.8	5.6	2.2	2
7	5.0	3.4	1.5	0.2	0
144	6.7	3.3	5.7	2.5	2
120	6.9	3.2	5.7	2.3	2

4.3 Visualizing Data

In [50]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
pca = PCA(2)
data = pca.fit_transform(df_train.iloc[:, :-1])
plot_classes(data, df_train.iloc[:, -1], ax)
ax.set_xticks([])
ax.set_yticks([])
plt.show()
```



5 Breast Cancer

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

5.1 Variables

- ID number
- Diagnosis (M = malignant, B = benign)
- 3-32) Ten real-valued features are computed for each cell nucleus:
  - radius (mean of distances from center to points on the perimeter)
  - texture (standard deviation of gray-scale values)
  - perimeter
  - area
  - smoothness (local variation in radius lengths)
  - compactness (perimeter^2 / area - 1.0)
  - concavity (severity of concave portions of the contour)
  - concave points (number of concave portions of the contour)
  - symmetry
  - fractal dimension ("coastline approximation" - 1)

5.2 Processing Data

```
In [51]: np.random.seed(123)
         random.seed(123)
         df = pd.read_csv(data_path + 'breast_cancer/raw_data.csv', names=[0, 'labels']+[i for i
         labels = df['labels']
         df = df.drop(['labels', 0], axis=1)
         def converter_bs(species_string):
             if species_string == 'M':
                 return 0
             elif species_string == 'B':
```

```
            return 1
        else:
            return np.NaN
    df['labels'] = labels.apply(converter_bs)
    n = df.shape[0]
    indexes = [i for i in range(n)]
    random.shuffle(indexes)
    df_train = df.iloc[indexes[:int(n/2)], :]
    df_test = df.iloc[indexes[int(n/2):], :]
    df_train.to_csv(data_path + 'breast_cancer/train_data.csv', index=False)
    df_test.to_csv(data_path + 'breast_cancer/test_data.csv', index=False)
    df_train.head()
```

Out[51]:

	1	2	3	4	5	6	7	8	\
52	11.940	18.24	75.71	437.6	0.08261	0.04751	0.019720	0.01349	
369	22.010	21.90	147.20	1482.0	0.10630	0.19540	0.244800	0.15010	
67	11.310	19.04	71.80	394.1	0.08139	0.04701	0.037090	0.02230	
424	9.742	19.12	61.93	289.7	0.10750	0.08333	0.008934	0.01967	
210	20.580	22.14	134.70	1290.0	0.09090	0.13480	0.164000	0.09561	

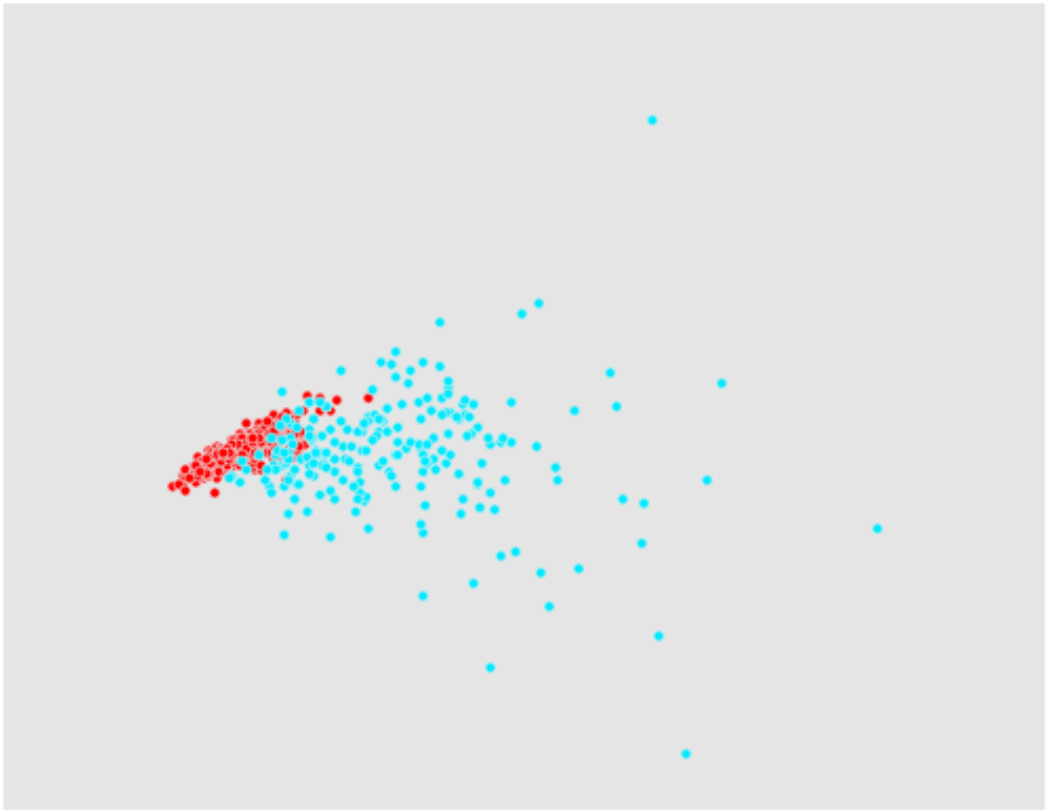
	9	10	...	22	23	24	25	26	27	\
52	0.1868	0.06110	...	21.33	83.67	527.2	0.1144	0.08906	0.09203	
369	0.1824	0.06140	...	25.80	195.00	2227.0	0.1294	0.38850	0.47560	
67	0.1516	0.05667	...	23.84	78.00	466.7	0.1290	0.09148	0.14440	
424	0.2538	0.07029	...	23.17	71.79	380.9	0.1398	0.13520	0.02085	
210	0.1765	0.05024	...	27.84	158.30	1656.0	0.1178	0.29200	0.38610	

	28	29	30	labels
52	0.06296	0.2785	0.07408	1
369	0.24320	0.2741	0.08574	0
67	0.06961	0.2400	0.06641	1
424	0.04589	0.3196	0.08009	1
210	0.19200	0.2909	0.05865	0

[5 rows x 31 columns]

5.3 Visualizing Data

In [52]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
pca = PCA(2)
data = pca.fit_transform(df.iloc[:, :-1])
plot_classes(data, df.iloc[:, -1], ax)
ax.set_xticks([])
ax.set_yticks([])
plt.show()
```



6 SPECT Heart Data

The dataset describes diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images. Each of the patients is classified into two categories: normal and abnormal. The database of 267 SPECT image sets (patients) was processed to extract features that summarize the original SPECT images. As a result, 44 continuous feature pattern was created for each patient. The pattern was further processed to obtain 22 binary feature patterns. The CLIP3 algorithm was used to generate classification rules from these patterns. The CLIP3 algorithm generated rules that were 84.0% accurate (as compared with cardilogists' diagnoses).

SPECT is a good data set for testing ML algorithms; it has 267 instances that are descibed by 23 binary attributes

6.1 Variables

1. OVERALL_DIAGNOSIS: 0,1 (class attribute, binary)
2. F1: 0,1 (the partial diagnosis 1, binary)
3. F2: 0,1 (the partial diagnosis 2, binary)

4. F3: 0,1 (the partial diagnosis 3, binary)
5. F4: 0,1 (the partial diagnosis 4, binary)
6. F5: 0,1 (the partial diagnosis 5, binary)
7. F6: 0,1 (the partial diagnosis 6, binary)
8. F7: 0,1 (the partial diagnosis 7, binary)
9. F8: 0,1 (the partial diagnosis 8, binary)
10. F9: 0,1 (the partial diagnosis 9, binary)
11. F10: 0,1 (the partial diagnosis 10, binary)
12. F11: 0,1 (the partial diagnosis 11, binary)
13. F12: 0,1 (the partial diagnosis 12, binary)
14. F13: 0,1 (the partial diagnosis 13, binary)
15. F14: 0,1 (the partial diagnosis 14, binary)
16. F15: 0,1 (the partial diagnosis 15, binary)
17. F16: 0,1 (the partial diagnosis 16, binary)
18. F17: 0,1 (the partial diagnosis 17, binary)
19. F18: 0,1 (the partial diagnosis 18, binary)
20. F19: 0,1 (the partial diagnosis 19, binary)
21. F20: 0,1 (the partial diagnosis 20, binary)
22. F21: 0,1 (the partial diagnosis 21, binary)
23. F22: 0,1 (the partial diagnosis 22, binary)

6.2 Processing Data

```
In [53]: np.random.seed(123)
         random.seed(123)
         df = pd.read_csv(data_path + 'spect/raw_train.csv', names=['labels'] + [i for i in rang
         df_test = pd.read_csv(data_path + 'spect/raw_test.csv', names=['labels'] + [i for i in
         labels = df['labels']
         labels_t = df_test['labels']
         df = df.drop(['labels'], axis=1)
         df_test = df_test.drop(['labels'], axis=1)
         df['labels'] = labels
         df_test['labels'] = labels_t
         df.to_csv(data_path + 'spect/train_data.csv')
         df_test.to_csv(data_path + 'spect/test_data.csv')
         print('Train shape: ', df.shape, 'Test shape: ', df_test.shape)
         df.head()
```

```
Train shape:  (80, 23) Test shape:  (187, 23)
```

Out[53]:

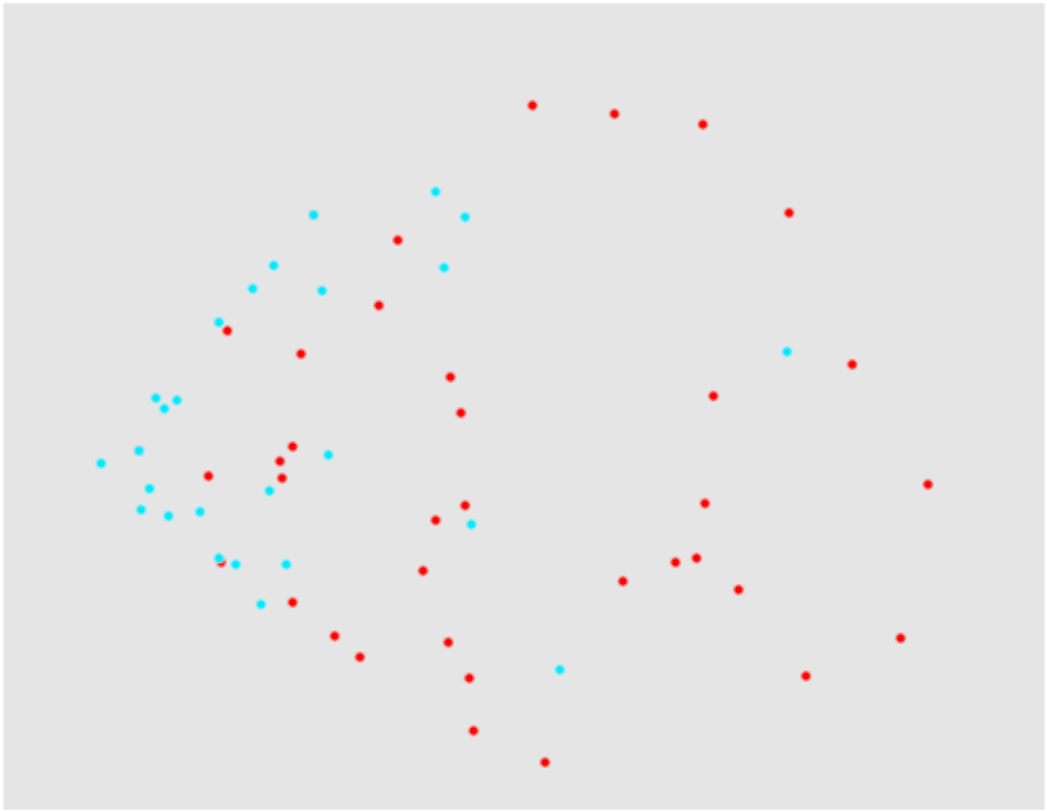
	0	1	2	3	4	5	6	7	8	9	...	13	14	15	16	17	18	19	20	21	\
0	0	0	0	1	0	0	0	1	1	0	...	1	0	0	0	0	0	0	0	0	
1	0	0	1	1	0	0	0	1	1	0	...	1	0	0	0	0	0	0	0	1	
2	1	0	1	0	1	0	0	1	0	1	...	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	1	1	1	
4	0	0	0	0	0	0	0	1	0	0	...	0	1	1	0	0	0	0	0	0	

	labels
0	1
1	1
2	1
3	1
4	1

[5 rows x 23 columns]

6.3 Visualizing Data

In [54]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
pca = PCA(2)
data = pca.fit_transform(df.iloc[:, :-1])
plot_classes(data, df.iloc[:, -1], ax)
ax.set_xticks([])
ax.set_yticks([])
plt.show()
```



7 IMDB Movies

There are 2399 unique director names, and thousands of actors/actresses.

7.1 Variables

Below are the 28 variables:

- movie_title
- color
- num_critic_for_reviews
- movie_facebook_likes
- duration
- director_name
- director_facebook_likes

- actor_3_name
- actor_3_facebook_likes
- actor_2_name
- actor_2_facebook_likes
- actor_1_name
- actor_1_facebook_likes
- gross
- genres
- num_voted_users
- cast_total_facebook_likes
- facenumber_in_poster
- plot_keywords
- movie_imdb_link
- num_user_for_reviews
- language
- country
- content_rating
- budget
- title_year
- imdb_score
- aspect_ratio

7.2 Processing Data

```
In [55]: def ratio_converter(imdb_score):
             if imdb_score < 4: #Shitty movie
                 return 0
             elif 4 <= imdb_score < 7: #Meh movie
                 return 1
             elif imdb_score >= 7: #Good movie
                 return 2

         df = pd.read_csv(data_path + 'imdb_movies/movie_metadata.csv')
         df = df[df.isnull().sum(axis=1) == 0]
         print(df.shape)
```

```
true_values = df['imdb_score']
labels = df['imdb_score'].apply(ratio_converter)
dummie_df = pd.get_dummies(df[['color']])
df = df.iloc[:, [2, 3, 4, 5, 7, 8, 12, 13, 15, 18, 22, 23, 24, 25, 26, 27]]
df[dummie_df.columns] = dummie_df
df = df.drop(['imdb_score'], axis=1)
df -= df.min()
df /= df.max()
df['labels'] = labels
print(df.shape)
np.random.seed(123)
random.seed(123)
n = 1500
indexes = [i for i in range(df.shape[0])]
random.shuffle(indexes)
df_train = df.iloc[indexes[:n], :]
true_values = true_values[indexes[:n]]
true_values.to_csv(data_path + 'imdb_movies/true_values.csv', index=False)
df_test = df.iloc[indexes[n:], :]
df_train.to_csv(data_path + 'imdb_movies/train_imbd_1500.csv', index=False)
df_test.to_csv(data_path + 'imdb_movies/test_imbd_1500.csv', index=False)
df.head()
```

```
(3756, 28)
(3756, 18)
```

Out[55]:

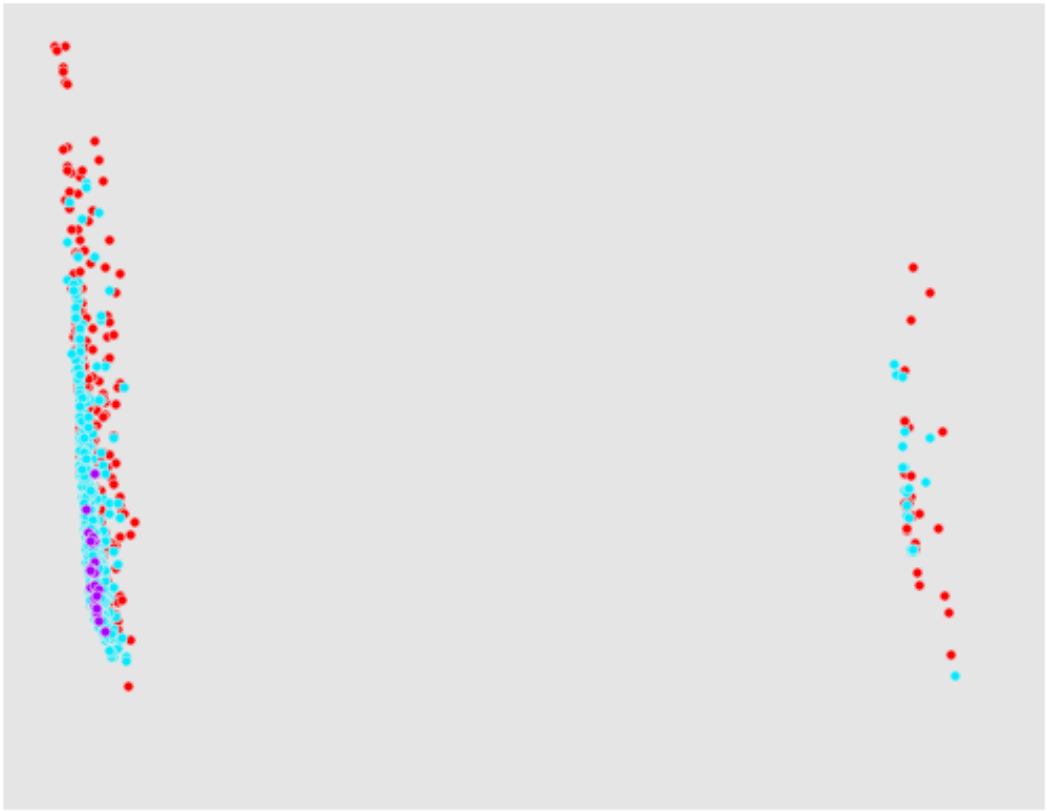
	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_likes	gross	num_voted_users
0	0.889026	0.481229	0.000000	0.037174	0.001563	1.000000	0.524429
1	0.369914	0.450512	0.024478	0.043478	0.062500	0.406840	0.278829
2	0.739827	0.378840	0.000000	0.007000	0.017188	0.263080	0.163213
3	1.000000	0.433447	0.956522	1.000000	0.042188	0.589253	0.677200
5	0.567201	0.324232	0.020652	0.023043	0.001000	0.096066	0.125535

	cast_total_facebook_likes	facenumber_in_poster	num_user_for_reviews
0	0.007361	0.000000	0.603244
1	0.073622	0.000000	0.244066
2	0.017816	0.023256	0.195807
3	0.162561	0.000000	0.533426
5	0.002852	0.023256	0.145174

```
     budget  title_year  actor_2_facebook_likes  aspect_ratio  \
0  0.019402    0.921348                0.006832      0.040486
1  0.024559    0.898876                0.036496      0.078947
2  0.020056    0.988764                0.002869      0.078947
3  0.020466    0.955056                0.167883      0.078947
5  0.021587    0.955056                0.004613      0.078947

   movie_facebook_likes  color_ Black and White  color_Color  labels
0              0.094556                     0.0          1.0       2
1              0.000000                     0.0          1.0       2
2              0.243553                     0.0          1.0       1
3              0.469914                     0.0          1.0       2
5              0.068768                     0.0          1.0       1
```

7.3 Visualizing Data

```
In [56]: fig = plt.figure()
         ax = fig.add_subplot(111)
         pca = PCA(2)
         data = pca.fit_transform(df_train.iloc[:, :-1])
         plot_classes(data, df_train.iloc[:, -1], ax)
         ax.set_xticks([])
         ax.set_yticks([])
         plt.show()
```



8 Abalon

Predicting the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope -- a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict the age. Further information, such as weather patterns and location (hence food availability) may be required to solve the problem.

From the original data examples with missing values were removed (the majority having the predicted value missing), and the ranges of the continuous values have been scaled for use with an ANN (by dividing by 200).

8.1 Variables

- **Name / Data Type / Measurement Unit / Description**
- Sex / nominal / – / M, F, and I (infant)
- Length / continuous / mm / Longest shell measurement
- Diameter / continuous / mm / perpendicular to length

- Height / continuous / mm / with meat in shell
- Whole weight / continuous / grams / whole abalone
- Shucked weight / continuous / grams / weight of meat
- Viscera weight / continuous / grams / gut weight (after bleeding)
- Shell weight / continuous / grams / after being dried
- Rings / integer / – / +1.5 gives the age in years

8.2 Processing Data

```
In [70]: df = pd.read_csv('/home/toby/master/data/data_set/abalon/raw.csv')
         def sex_converter(sex_string):
             if sex_string == 'M':
                 return 0
             elif sex_string == 'F':
                 return 1
             elif sex_string == 'I':
                 return 2
         df['Sex'] = df['Sex'].apply(sex_converter)
         random.seed(123)
         indexes = [i for i in range(df.shape[0])]
         random.shuffle(indexes)
         df_train = df.iloc[indexes[:300], :]
         df_train.to_csv(data_path + 'abalon/train.csv', index=False)
         df_test = df.iloc[indexes[300:600], :] #labels == 21 is not precent in train set, but i
         df_test.to_csv(data_path + 'abalon/test.csv', index=False)
         df.head()
```

Out[70]:

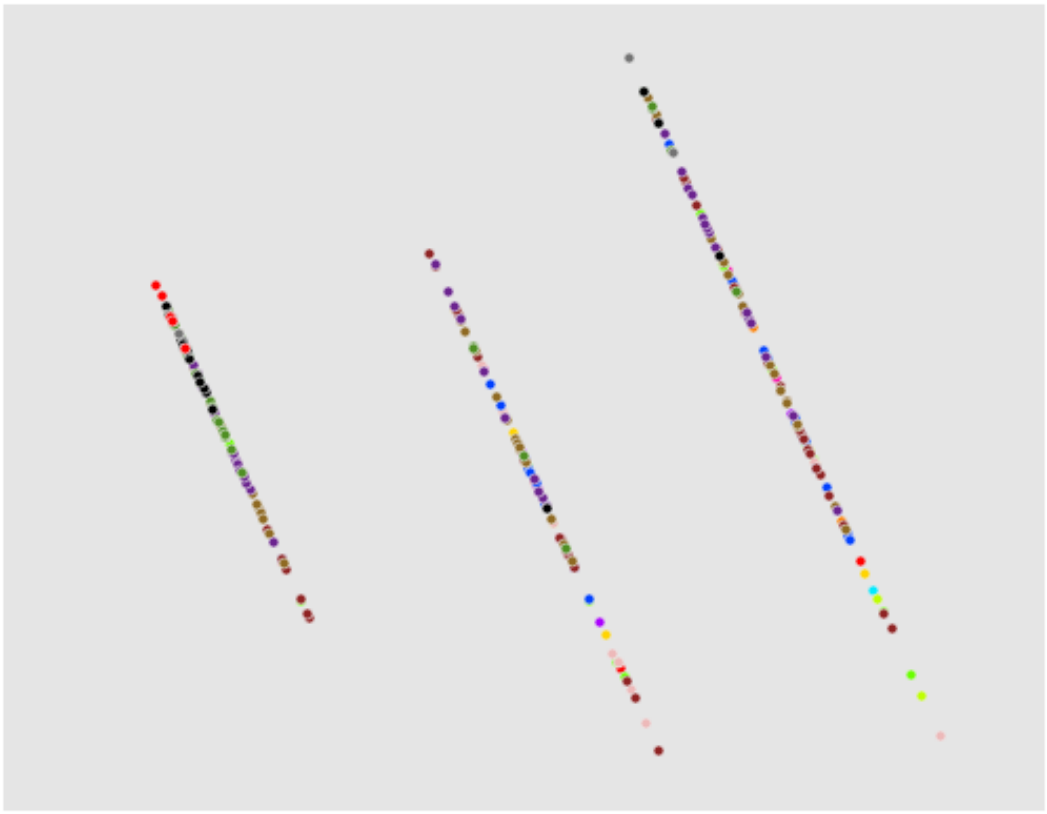
	Sex	Length	Diameter	Height	Whole weight	Shucked weight	\
0	0	0.455	0.365	0.095	0.5140	0.2245	
1	0	0.350	0.265	0.090	0.2255	0.0995	
2	1	0.530	0.420	0.135	0.6770	0.2565	
3	0	0.440	0.365	0.125	0.5160	0.2155	
4	2	0.330	0.255	0.080	0.2050	0.0895	

	Viscera weight	Shell weight	labels
0	0.1010	0.150	15
1	0.0485	0.070	7
2	0.1415	0.210	9
3	0.1140	0.155	10
4	0.0395	0.055	7

8.3 Visualize

```
In [71]: fig = plt.figure()
         ax = fig.add_subplot(111)
```

```
pca = PCA(2)
data = pca.fit_transform(df_train.iloc[:, :-1])
plot_classes(data, df_train.iloc[:, -1], ax)
ax.set_xticks([])
ax.set_yticks([])
plt.show()
```



9 Contraceptive Method Choice Data Set

This dataset is a subset of the 1987 National Indonesia Contraceptive Prevalence Survey. The samples are married women who were either not pregnant or do not know if they were at the time of interview. The problem is to predict the current contraceptive method choice (no use, long-term methods, or short-term methods) of a woman based on her demographic and socio-economic characteristics.

9.1 Variables

1. Wife's age (numerical)
2. Wife's education (categorical) 1=low, 2, 3, 4=high

3. Husband's education (categorical) 1=low, 2, 3, 4=high
4. Number of children ever born (numerical)
5. Wife's religion (binary) 0=Non-Islam, 1=Islam
6. Wife's now working? (binary) 0=Yes, 1=No
7. Husband's occupation (categorical) 1, 2, 3, 4
8. Standard-of-living index (categorical) 1=low, 2, 3, 4=high
9. Media exposure (binary) 0=Good, 1=Not good
10. Contraceptive method used (class attribute) 1=No-use, 2=Long-term, 3=Short-term

9.2 Processing Data

```
In [59]: df = pd.read_csv(data_path + 'contraceptive/raw.csv')
         random.seed(123)
         indexes = [i for i in range(df.shape[0])]
         random.shuffle(indexes)
         df_train = df.iloc[indexes[:300], :]
         df_train.to_csv(data_path + 'contraceptive/train.csv', index=False)
         df_test = df.iloc[indexes[300:600], :] #labels == 21 is not precent in train set, but i
         df_test.to_csv(data_path + 'contraceptive/test.csv', index=False)
         df.head()
```

Out[59]:

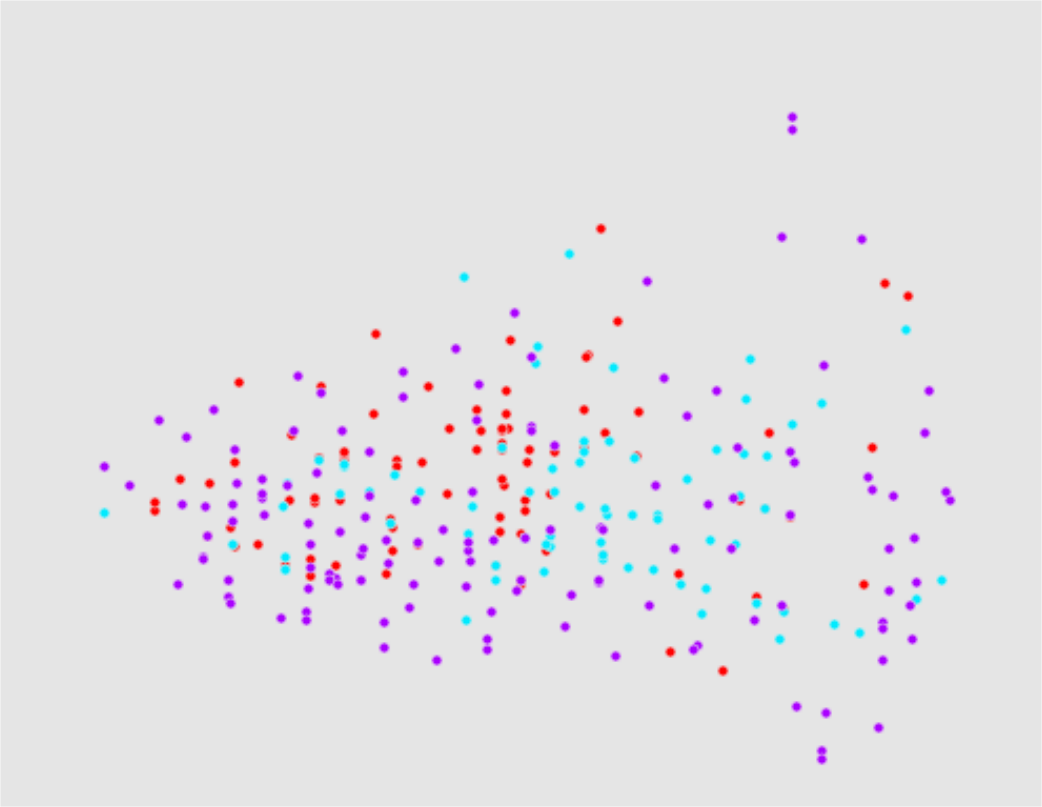
	Wife's age	Wife's education	Husband's education	\
0	24	2	3	
1	45	1	3	
2	43	2	3	
3	42	3	2	
4	36	3	3	

	Number of children ever born	Wife's religion	Wife's now working?	\
0	3	1	1	
1	10	1	1	
2	7	1	1	
3	9	1	1	
4	8	1	1	

	Husband's occupation	Standard-of-living	Media exposure	labels
0	2	3	0	1
1	3	4	0	1
2	3	4	0	1
3	3	3	0	1
4	3	2	0	1

9.3 Visualization

```
In [60]: fig = plt.figure()
         ax = fig.add_subplot(111)
         pca = PCA(2)
         data = pca.fit_transform(df_train.iloc[:, :-1])
         plot_classes(data, df_train.iloc[:, -1], ax)
         ax.set_xticks([])
         ax.set_yticks([])
         plt.show()
```



10 Ecoli Data Set

10.1 Variables

1. Sequence Name: Accession number for the SWISS-PROT database
2. mcg: McGeoch's method for signal sequence recognition.
3. gvh: von Heijne's method for signal sequence recognition.

4. lip: von Heijne's Signal Peptidase II consensus sequence score. Binary attribute.
5. chg: Presence of charge on N-terminus of predicted lipoproteins. Binary attribute.
6. aac: score of discriminant analysis of the amino acid content of outer membrane and periplasmic proteins.
7. alm1: score of the ALOM membrane spanning region prediction program.
8. alm2: score of ALOM program after excluding putative cleavable signal regions from the sequence.

10.2 Processing Data

```
In [61]: f = open(data_path + 'ecoli/raw.csv', 'r')
         df_list = []
         for i, line in enumerate(f):
             x = line.split(' ')
             x = list(filter(lambda a: a != '', x))
             x[-1] = x[-1][:-1]
             if i == 0:
                 col = x
             else:
                 df_list.append(x)

         df = pd.DataFrame(df_list, columns=col)
         df = df.drop(['Sequence'], axis=1)
         def name_converter(label_string):
             if label_string == 'cp':
                 return 0
             elif label_string == 'im':
                 return 1
             elif label_string == 'pp':
                 return 2
             elif label_string == 'imU':
                 return 3
             elif label_string == 'om':
                 return 4
             elif label_string == 'omL':
                 return 5
             elif label_string == 'imS':
                 return 6
             elif label_string == 'imL':
                 return 7
         df['labels'] = df['labels'].apply(name_converter)
         random.seed(123)
         indexes = [i for i in range(df.shape[0])]
         random.shuffle(indexes)
         print(df.shape)
```

```
df_train = df.iloc[indexes[:200], :]
df_train.to_csv(data_path + 'ecoli/train.csv')
df_test = df.iloc[indexes[200:], :]
df_test.to_csv(data_path + 'ecoli/test.csv')
df.head()
```

```
(336, 8)
```

Out[61]:

	mcg	gvh	lip	chg	aac	alm1	alm2	labels
0	0.49	0.29	0.48	0.50	0.56	0.24	0.35	0
1	0.07	0.40	0.48	0.50	0.54	0.35	0.44	0
2	0.56	0.40	0.48	0.50	0.49	0.37	0.46	0
3	0.59	0.49	0.48	0.50	0.52	0.45	0.36	0
4	0.23	0.32	0.48	0.50	0.55	0.25	0.35	0

In [62]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
pca = PCA(2)
data = pca.fit_transform(df_train.iloc[:, :-1])
plot_classes(data, df_train.iloc[:, -1], ax)
ax.set_xticks([])
ax.set_yticks([])
plt.show()
```

11 Titanic

11.1 Variables

- Variable
- survival
- pclass
- sex
- Age
- sibsp # of siblings / spouses aboard the Titanic
- parch # of parents / children aboard the Titanic
- ticket
- fare
- cabin
- embarked

```
In [72]: df = pd.read_csv(data_path + 'Titanic/raw.csv')
         def embarked_converter(emb_string):
             if emb_string == 'S':
                 return 0
             elif emb_string == 'C':
                 return 1
             elif emb_string == 'Q':
                 return 2
         labels = df['Survived']
         df = df.drop(['PassengerId',
                       'Cabin',
                       'Name',
                       'Ticket',
                       'Survived'], axis=1)
         df['labels'] = labels
         df['Sex'] = df['Sex'].apply(lambda x: 0 if x == 'male' else 1)
         df['Embarked'] = df['Embarked'].apply(embarked_converter)
         df = df.iloc[[not i for i in df.isnull().sum(axis=1)], :]
         random.seed(123)
         indexes = [i for i in range(df.shape[0])]
         random.shuffle(indexes)
         print(df.shape)
```

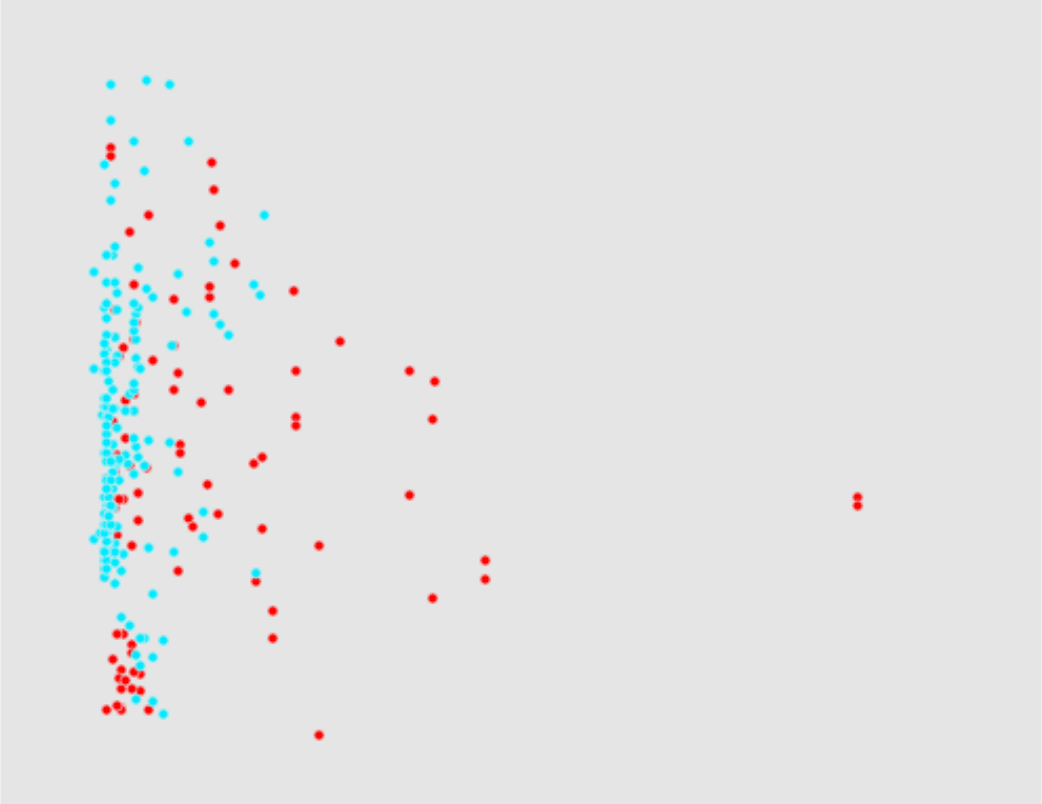
```
df_train = df.iloc[indexes[:300], :]
df_train.to_csv(data_path + 'Titanic/train.csv', index=False)
df_test = df.iloc[indexes[300:600], :]
df_train.to_csv(data_path + 'Titanic/test.csv', index=False)
df.head()
```

```
(712, 8)
```

Out[72]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	labels
0	3	0	22.0	1	0	7.2500	0.0	0
1	1	1	38.0	1	0	71.2833	1.0	1
2	3	1	26.0	0	0	7.9250	0.0	1
3	1	1	35.0	1	0	53.1000	0.0	1
4	3	0	35.0	0	0	8.0500	0.0	0

In [64]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
pca = PCA(2)
data = pca.fit_transform(df_train.iloc[:, :-1])
plot_classes(data, df_train.iloc[:, -1], ax)
ax.set_xticks([])
ax.set_yticks([])
plt.show()
```



12 McDonald's Nutritions

Predicting the food category given the nutritions within the meal.

```
\subsection{Variables}
```

```
In [77]: df = pd.read_csv(data_path + 'mcdonald/menu.csv')
         def converter(cat_string):
             uniques = np.unique(df['Category'].value_counts().index)
             for i in range(len(uniques)):
                 if cat_string == uniques[i]:
                     return i

         def serving_converter(serving_string):
             return float(serving_string.split(' ')[0])

         print(df['Category'].value_counts().index)
         df['Category'] = df['Category'].apply(converter)
         df['Serving Size'] = df['Serving Size'].apply(serving_converter)
         labels = df['Category']
         df = df.drop(['Category', 'Item'], axis=1)
         df -= df.min()
         df /= df.max()
         df['labels'] = labels
         random.seed(123)
         indexes = [i for i in range(df.shape[0])]
         random.shuffle(indexes)
         df_train = df.iloc[indexes[:150], :]
         df_train.to_csv(data_path + 'mcdonald/train.csv', index=False)
         df_test = df.iloc[indexes[150:], :]
         df_train.to_csv(data_path + 'mcdonald/test.csv', index=False)
         print(df.shape)
         df.head()
```

```
Index(['Coffee & Tea', 'Breakfast', 'Smoothies & Shakes', 'Beverages',
       'Chicken & Fish', 'Beef & Pork', 'Snacks & Sides', 'Desserts',
       'Salads'],
      dtype='object')
(260, 23)
```

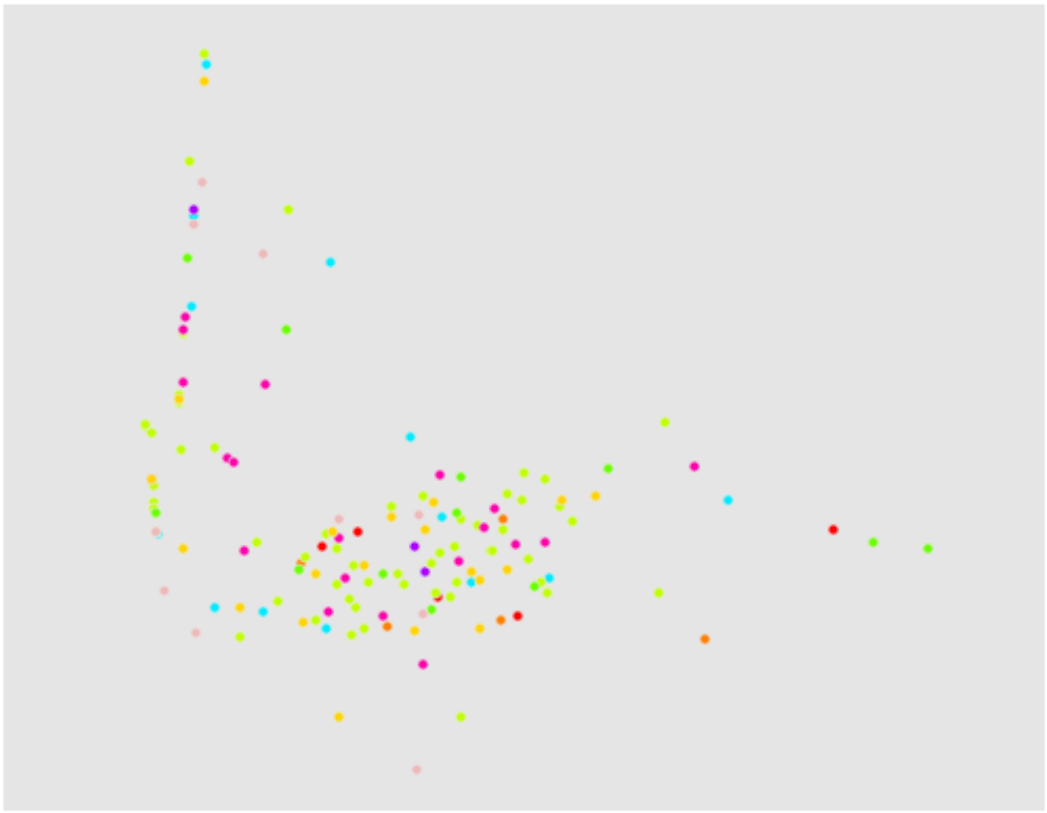
```
Out[77]:    Serving Size  Calories  Calories from Fat  Total Fat  \
         0      0.122581  0.159574           0.113208   0.110169
         1      0.122581  0.132979           0.066038   0.067797
         2      0.093548  0.196809           0.188679   0.194915
```

```
3      0.151613  0.239362           0.235849   0.237288
4      0.151613  0.212766           0.198113   0.194915

   Total Fat (% Daily Value)  Saturated Fat  Saturated Fat (% Daily Value)  \
0                   0.109890           0.25                        0.245098
1                   0.065934           0.15                        0.147059
2                   0.192308           0.40                        0.411765
3                   0.236264           0.50                        0.509804
4                   0.192308           0.40                        0.411765

   Trans Fat  Cholesterol  Cholesterol (% Daily Value)   ...    \
0        0.0     0.452174                     0.453125   ...
1        0.0     0.043478                     0.041667   ...
2        0.0     0.078261                     0.078125   ...
3        0.0     0.495652                     0.494792   ...
4        0.0     0.086957                     0.083333   ...

   Carbohydrates (% Daily Value)  Dietary Fiber  \
0                       0.212766       0.571429
1                       0.212766       0.571429
2                       0.212766       0.571429
3                       0.212766       0.571429
4                       0.212766       0.571429

   Dietary Fiber (% Daily Value)    Sugars   Protein  \
0                       0.607143  0.023438  0.195402
1                       0.607143  0.023438  0.206897
2                       0.607143  0.015625  0.160920
3                       0.607143  0.015625  0.241379
4                       0.607143  0.015625  0.241379

   Vitamin A (% Daily Value)  Vitamin C (% Daily Value)  \
0                   0.058824                        0.0
1                   0.035294                        0.0
2                   0.047059                        0.0
3                   0.088235                        0.0
4                   0.035294                        0.0

   Calcium (% Daily Value)  Iron (% Daily Value)  labels
0                 0.357143                 0.375       2
1                 0.357143                 0.200       2
2                 0.357143                 0.250       2
3                 0.428571                 0.375       2
4                 0.357143                 0.250       2

[5 rows x 23 columns]
```

In [79]:
```
fig = plt.figure()
```

```
    ax = fig.add_subplot(111)
    pca = PCA(2)
    data = pca.fit_transform(df.iloc[:, :-1])
    plot_classes(data, df_train.iloc[:, -1], ax)
    ax.set_xticks([])
    ax.set_yticks([])
    plt.show()
```

```
/usr/local/lib/python3.5/dist-packages/ipykernel_launcher.py:6: VisibleDeprecationWarning: boole
```



Bibliography

1. Lichman, M. *UCI Machine Learning Repository* 2013. <http://archive.ics.uci.edu/ml>.
2. Yeh, I.-C. & Lien, C.-h. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications* **36,** 2473–2480 (2009).
3. Dal Pozzolo, A., Caelen, O., Johnson, R. A. & Bontempi, G. *Calibrating Probability with Undersampling for Unbalanced Classification* in *Computational Intelligence, 2015 IEEE Symposium Series on* (2015), 159–166.
4. Mangasarian, O. L., Setiono, R. & Wolberg, W. Pattern recognition via linear programming: Theory and application to medical diagnosis. *Large-scale numerical optimization,* 22–31 (1990).
5. Bergstra, J. & Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* **13,** 281–305 (2012).
6. Kohavi, R. *et al. A study of cross-validation and bootstrap for accuracy estimation and model selection* in *Ijcai* **14** (1995), 1137–1145.
7. Arlot, S. & Celisse, A. A survey of cross-validation procedures for model selection. *Statist. Surv.* **4,** 40–79 (2010).
8. Ferguson, T. S. A Bayesian analysis of some nonparametric problems. *The annals of statistics,* 209–230 (1973).
9. Antoniak, C. E. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *The annals of statistics,* 1152–1174 (1974).
10. Escobar, M. D. & West, M. Bayesian density estimation and inference using mixtures. *Journal of the american statistical association* **90,** 577–588 (1995).
11. MacEachern, S. N. in *Practical nonparametric and semiparametric Bayesian statistics* 23–43 (Springer, 1998).
12. Müller, K.-R., Mika, S., R ä tsch, G., Tsuda, K. & Schölkopf, B. in *Handbook of Neural Network Signal Processing* (CRC Press, 2001).
13. Weston, J. *et al.* Semi-supervised protein classification using cluster kernels. *Bioinformatics* **21,** 3241 (2005).
14. Izquierdo-Verdiguier, E., Jenssen, R., Gómez-Chova, L. & Camps-Valls, G. Spectral clustering with the probabilistic cluster kernel. *Neurocomputing* **149,** 1299–1304 (2015).

15. Tuia, D. & Camps-Valls, G. Semisupervised Remote Sensing Image Classification With Cluster Kernels. *IEEE Geoscience and Remote Sensing Letters* **6,** 224–228. ISSN: 1545-598X (Apr. 2009).

16. Rasmussen, C. E. *The Infinite Gaussian Mixture Model.* in *NIPS* **12** (1999), 554–560.

17. Kamper, H. Gibbs sampling for fitting finite and infinite Gaussian mixture models (2013).

18. Richardson, S. & Green, P. J. On Bayesian Analysis of Mixtures with an Unknown Number of Components (with discussion). *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **59,** 731–792. ISSN: 1467-9868 (1997).

19. Dahl, D. B. Model-based clustering for expression data via a Dirichlet process mixture model. *Bayesian inference for gene expression and proteomics,* 201–218 (2006).

20. Vlachos, A., Korhonen, A. & Ghahramani, Z. *Unsupervised and constrained Dirichlet process mixture models for verb clustering* in *Proceedings of the workshop on geometrical models of natural language semantics* (2009), 74–82.

21. Vlachos, A., Ghahramani, Z. & Korhonen, A. *Dirichlet process mixture models for verb clustering* in *Proceedings of the ICML workshop on Prior Knowledge for Text and Language* (2008).

22. Da Silva, A. R. F. A Dirichlet process mixture model for brain {MRI} tissue classification. *Medical Image Analysis* **11,** 169–182. ISSN: 1361-8415 (2007).

23. Zhu, J., Chen, N. & Xing, E. P. *Infinite SVM: a Dirichlet process mixture of large-margin kernel machines* in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (2011), 617–624.

24. Xue, Y., Liao, X., Carin, L. & Krishnapuram, B. Multi-task learning for classification with dirichlet process priors. *Journal of Machine Learning Research* **8,** 35–63 (2007).

25. Hosmer Jr, D. W., Lemeshow, S. & Sturdivant, R. X. *Applied logistic regression* (John Wiley & Sons, 2013).

26. Menard, S. *Applied logistic regression analysis* **106** (Sage, 2002).

27. Reichart, R., Elidan, G. & Rappoport, A. *A Diverse Dirichlet Process Ensemble for Unsupervised Induction of Syntactic Categories.* in *COLING* (2012), 2307–2324.

28. Yu, S. X. & Shi, J. *Multiclass spectral clustering* in *Proceedings Ninth IEEE International Conference on Computer Vision* (Oct. 2003), 313–319 vol.1. doi:10.1109/ICCV.2003.1238361.

29. Duda, R. O., Hart, P. E. & Stork, D. G. *Pattern classification* (John Wiley & Sons, 2012).

30. Tukey, J. W. Exploratory data analysis (1977).

31. Tabachnick, B. G., Fidell, L. S. & Osterlind, S. J. Using multivariate statistics (2001).

32. Chapelle, O., Scholkopf, B. & Eds., A. Z. Semi-Supervised Learning (Chapelle, O. et al., Eds.; 2006) [Book reviews]. *IEEE Transactions on Neural Networks* **20,** 542–542. ISSN: 1045-9227 (Mar. 2009).

33. Jain, A. K. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters* **31.** Award winning papers from the 19th International Conference on Pattern Recognition (ICPR)19th International Conference in Pattern Recognition (ICPR), 651–666. ISSN: 0167-8655 (2010).

34. Shi, J. & Malik, J. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22,** 888–905. ISSN: 0162-8828 (Aug. 2000).

35. Steinbach, M., Karypis, G., Kumar, V., *et al. A comparison of document clustering techniques* in *KDD workshop on text mining* **400** (2000), 525–526.

36. Eisen, M. B., Spellman, P. T., Brown, P. O. & Botstein, D. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences* **95,** 14863–14868 (1998).

37. Bellman, R. Dynamic programming and Lagrange multipliers. *Proceedings of the National Academy of Sciences* **42,** 767–769 (1956).

38. Keogh, E. & Mueen, A. in *Encyclopedia of Machine Learning* (eds Sammut, C. & Webb, G. I.) 257–258 (Springer US, Boston, MA, 2010). ISBN: 978-0-387-30164-8. doi:10.1007/978-0-387-30164-8_192. <http://dx.doi.org/10.1007/978-0-387-30164-8_192>.

39. Wold, S., Esbensen, K. & Geladi, P. Principal component analysis. *Chemometrics and intelligent laboratory systems* **2,** 37–52 (1987).

40. Murphy, K. *Machine Learning: A Probabilistic Perspective* ISBN: 9780262018029 (MIT Press, 2012).

41. Aronszajn, N. Theory of Reproducing Kernels. *Transactions of the American Mathematical Society* **68,** 337–404. ISSN: 00029947 (1950).

42. Saitoh, S. & Sawano, Y. *Theory of reproducing kernels and applications* (Springer, 2016).

43. Scholkopf, B. *et al.* Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks* **10,** 1000–1017. ISSN: 1045-9227 (Sept. 1999).

44. Shawe-Taylor, J. & Cristianini, N. *Kernel methods for pattern analysis* (Cambridge university press, 2004).

45. Kavzoglu, T. & Colkesen, I. A kernel functions analysis for support vector machines for land cover classification. *International Journal of Applied Earth Observation and Geoinformation* **11,** 352–359. ISSN: 0303-2434 (2009).

46. Cristianini, N. & Shawe-Taylor, J. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods* ISBN: 9780521780193 (Cambridge University Press, 2000).

47. Suykens, J. & Vandewalle, J. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters* **9,** 293–300. ISSN: 1573-773X (1999).

48. Schiolkopf, B., Burges, C. & Vapnik, V. *Extracting support data for a given task* in *Proceedings, First International Conference on Knowledge Discovery & Data Mining. AAAI Press, Menlo Park, CA* (1995), 252–257.

49. Boser, B. E., Guyon, I. M. & Vapnik, V. N. *A training algorithm for optimal margin classifiers* in *Proceedings of the fifth annual workshop on Computational learning theory* (1992), 144–152.

50. Cortes, C. & Vapnik, V. Support-vector networks. *Machine learning* **20,** 273–297 (1995).

51. Saul, L. K. & Roweis, S. T. An introduction to locally linear embedding. *unpublished. Available at: http://www. cs. toronto. edu/˜ roweis/lle/publications. html* (2000).

52. Schölkopf, B., Smola, A. & Müller, K.-R. in *Artificial Neural Networks — ICANN'97: 7th International Conference Lausanne, Switzerland, October 8–10, 1997 Proceeedings* (eds Gerstner, W., Germond, A., Hasler, M. & Nicoud, J.-D.) 583–588 (Springer Berlin Heidelberg, Berlin, Heidelberg, 1997). ISBN: 978-3-540-69620-9. doi:10.1007/BFb0020217. <http://dx.doi.org/10.1007/BFb0020217>.

53. Xu, R. & Wunsch, D. Survey of clustering algorithms. *IEEE Transactions on neural networks* **16,** 645–678 (2005).

54. Backer, E. & Jain, A. K. A clustering performance measure based on fuzzy set decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 66–75 (1981).

55. Wan, X., YANG, J. & Chen, X. An Improved k-means Algorithm for Documents Clustering [J]. *Computer Engineering* **2,** 042 (2003).

56. Ray, S. & Turi, R. H. *Determination of number of clusters in k-means clustering and application in colour image segmentation* in *Proceedings of the 4th international conference on advances in pattern recognition and digital techniques* (1999), 137–143.

57. Ng, H., Ong, S., Foong, K., Goh, P. & Nowinski, W. *Medical image segmentation using k-means clustering and improved watershed algorithm* in *Image Analysis and Interpretation, 2006 IEEE Southwest Symposium on* (2006), 61–65.

58. Hussain, H. M., Benkrid, K., Seker, H. & Erdogan, A. T. *FPGA implementation of K-means algorithm for bioinformatics application: An accelerated approach to clustering Microarray data* in *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)* (June 2011), 248–255. doi:10.1109/AHS.2011.5963944.

59. Basu, S., Bilenko, M. & Mooney, R. J. *A probabilistic framework for semi-supervised clustering* in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (2004), 59–68.

60. Reddy, D. & Jana, P. K. Initialization for K-means Clustering using Voronoi Diagram. *Procedia Technology* **4,** 395–400. ISSN: 2212-0173 (2012).

61. Dhillon, I. S. *Co-clustering Documents and Words Using Bipartite Spectral Graph Partitioning* in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, San Francisco, California, 2001), 269–274. ISBN: 1-58113-391-X. doi:10.1145/502512.502550. <http://doi.acm.org/10.1145/502512.502550>.

62. Hu, S., Su, L., Liu, H., Wang, H. & Abdelzaher, T. F. SmartRoad: Smartphone-Based Crowd Sensing for Traffic Regulator Detection and Identification. *ACM Trans. Sen. Netw.* **11,** 55:1–55:27. ISSN: 1550-4859 (July 2015).

63. Von Luxburg, U. A tutorial on spectral clustering. *Statistics and Computing* **17,** 395–416. ISSN: 1573-1375 (2007).

64. Ng, A. Y., Jordan, M. I., Weiss, Y., *et al. On spectral clustering: Analysis and an algorithm* in *NIPS* **14** (2001), 849–856.

65. Chung, F. R. *Spectral graph theory* (American Mathematical Soc., 1997).

66. Mohar, B. in *Graph symmetry* 225–275 (Springer, 1997).

67. Dhillon, I. S., Guan, Y. & Kulis, B. *Kernel k-means: spectral clustering and normalized cuts* in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (2004), 551–556.

68. Kulis, B., Basu, S., Dhillon, I. & Mooney, R. Semi-supervised graph clustering: a kernel approach. *Machine Learning* **74,** 1–22. ISSN: 1573-0565 (2009).

69. Dhillon, I. S., Guan, Y. & Kulis, B. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence* **29** (2007).

70. Dhillon, I. S., Guan, Y. & Kulis, B. *A unified view of kernel k-means, spectral clustering and graph cuts* (Citeseer, 2004).

71. Fred, A. L. & Jain, A. K. Combining multiple clusterings using evidence accumulation. *IEEE transactions on pattern analysis and machine intelligence* **27,** 835–850 (2005).

72. Freund, Y. & Schapire, R. E. in *Computational Learning Theory: Second European Conference, EuroCOLT '95 Barcelona, Spain, March 13–15, 1995 Proceedings* (ed Vitányi, P.) 23–37 (Springer Berlin Heidelberg, Berlin, Heidelberg, 1995). ISBN: 978-3-540-49195-8. doi:10.1007/3-540-59119-2_166. <http://dx.doi.org/10.1007/3-540-59119-2_166>.

73. Chen, T. & Guestrin, C. *XGBoost: A Scalable Tree Boosting System* in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, San Francisco, California, USA, 2016), 785–794. ISBN: 978-1-4503-4232-2. doi:10.1145/2939672.2939785. <http://doi.acm.org/10.1145/2939672.2939785>.

74. Breiman, L. Bagging predictors. *Machine Learning* **24,** 123–140. ISSN: 1573-0565 (1996).

75. Vega-Pons, S. & Ruiz-Shulcloper, J. A survey of clustering ensemble algorithms. *International Journal of Pattern Recognition and Artificial Intelligence* **25,** 337–372 (2011).

76. Monti, S., Tamayo, P., Mesirov, J. & Golub, T. Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data. *Machine Learning* **52,** 91–118. ISSN: 1573-0565 (2003).

77. Mikalsen, K. Ø., Bianchi, F. M., Soguero-Ruiz, C. & Jenssen, R. Time Series Cluster Kernel for Learning Similarities between Multivariate Time Series with Missing Data. *arXiv preprint arXiv:1704.00794* (2017).

78. Belkin, M. & Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation* **15,** 1373–1396 (2003).

79. Hore, P., Hall, L. O. & Goldgof, D. B. A scalable framework for cluster ensembles. *Pattern Recognition* **42,** 676–688. ISSN: 0031-3203 (2009).

80. Ayad, H. G. & Kamel, M. S. Cumulative voting consensus method for partitions with variable number of clusters. *IEEE transactions on pattern analysis and machine intelligence* **30,** 160–173 (2008).

81. Ayad, H. G. & Kamel, M. S. On voting-based consensus of cluster ensembles. *Pattern Recognition* **43,** 1943–1953 (2010).

82. Li, Y., Yu, J., Hao, P. & Li, Z. *Clustering ensembles based on normalized edges* in *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2007), 664–671.

83. Strehl, A. & Ghosh, J. Cluster ensembles: a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research* **3,** 583–617 (2002).

84. Topchy, A., Jain, A. K. & Punch, W. *A mixture model for clustering ensembles* in *Proceedings of the 2004 SIAM International Conference on Data Mining* (2004), 379–390.

85. Frigyik, B. A., Kapila, A. & Gupta, M. R. *Introduction to the Dirichlet Distribution and Related Processes* tech. rep. 206 (2010).

86. Goldman, R. N. Polya's Urn Model and Computer Aided Geometric Design. *SIAM Journal on Algebraic Discrete Methods* **6,** 1–28 (1985).

87. Blackwell, D. & MacQueen, J. B. Ferguson distributions via Pólya urn schemes. *The annals of statistics,* 353–355 (1973).

88. Orbanz, P. Lecture notes on bayesian nonparametrics. Version: May 16, 2014.

89. Schmidt, M. N. & Morup, M. Nonparametric Bayesian modeling of complex networks: An introduction. *IEEE Signal Processing Magazine* **30,** 110–128 (2013).

90. Simpson, D., Rue, H., Riebler, A., Martins, T. G. & Sørbye, S. H. Penalising Model Component Complexity: A Principled, Practical Approach to Constructing Priors. *Statist. Sci.* **32,** 1–28 (Feb. 2017).

91. Diaconis, P. & Ylvisaker, D. Conjugate Priors for Exponential Families. *Ann. Statist.* **7,** 269–281 (Mar. 1979).

92. Fink, D. A compendium of conjugate priors. *See http://www. people. cornell. edu/pages/df36/CONJINTRnew% 20TEX. pdf,* 46 (1997).

93. Teh, Y. W. in *Encyclopedia of Machine Learning* (eds Sammut, C. & Webb, G. I.) 280–287 (Springer US, Boston, MA, 2010). ISBN: 978-0-387-30164-8. doi:10.1007/978-0-387-30164-8_219. <http://dx.doi.org/10.1007/978-0-387-30164-8_219>.

94. Gershman, S. J. & Blei, D. M. A tutorial on Bayesian nonparametric models. *Journal of Mathematical Psychology* **56,** 1–12 (2012).

95. Ross, S. M. *Introduction to Probability Models, Tenth Edition* ISBN: 978-0-12-375686-2 (Academic Press, Inc., Orlando, FL, USA, 2010).

96. Gilks, W. R., Richardson, S. & Spiegelhalter, D. J. Introducing markov chain monte carlo. *Markov chain Monte Carlo in practice* **1,** 19 (1996).

97. Andrieu, C., De Freitas, N., Doucet, A. & Jordan, M. I. An introduction to MCMC for machine learning. *Machine learning* **50,** 5–43 (2003).

98. Efron, B. Computers and the Theory of Statistics: Thinking the Unthinkable. *SIAM Review* **21,** 460–480 (1979).

99. B. Efron, R. T. Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy. *Statistical Science* **1,** 54–75. ISSN: 08834237 (1986).

100. Johnson, R. W. An Introduction to the Bootstrap. *Teaching Statistics* **23,** 49–54. ISSN: 1467-9639 (2001).

101. Cooke, R. M. *Experts in uncertainty: opinion and subjective probability in science* (Oxford University Press on Demand, 1991).

102. Kass, R. E. & Wasserman, L. The selection of prior distributions by formal rules. *Journal of the American Statistical Association* **91,** 1343–1370 (1996).

103. Neal, R. M. Markov chain sampling methods for Dirichlet process mixture models. *Journal of computational and graphical statistics* **9,** 249–265 (2000).

104. Damlen, P., Wakefield, J. & Walker, S. Gibbs sampling for Bayesian non-conjugate and hierarchical models by using auxiliary variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **61,** 331–344. ISSN: 1467-9868 (1999).

105. Papaspiliopoulos, O. & Roberts, G. O. Retrospective Markov Chain Monte Carlo Methods for Dirichlet Process Hierarchical Models. *Biometrika* **95,** 169–186. ISSN: 00063444 (2008).

106. Blei, D. M., Jordan, M. I., *et al.* Variational inference for Dirichlet process mixtures. *Bayesian analysis* **1,** 121–143 (2006).

107. Attias, H. *et al. A Variational Baysian Framework for Graphical Models.* in *NIPS* **12** (1999).

108. Kurihara, K., Welling, M. & Teh, Y. W. *Collapsed Variational Dirichlet Process Mixture Models.* in *IJCAI* **7** (2007), 2796–2801.

109. Chang, J. & Fisher III, J. W. in *Advances in Neural Information Processing Systems 26* (eds Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z. & Weinberger, K. Q.) 620–628 (Curran Associates, Inc., 2013). <`http://papers.nips.cc/paper/5162-parallel-sampling-of-dp-mixture-models-using-sub-cluster-splits.pdf`>.

110. Dahl, D. B. An improved merge-split sampler for conjugate Dirichlet process mixture models. *Technical R eport* **1,** 086 (2003).

111. Jain, S. & Neal, R. M. A split-merge Markov chain Monte Carlo procedure for the Dirichlet process mixture model. *Journal of Computational and Graphical Statistics* **13,** 158–182 (2004).

112. Jain, S. & Neal, R. M. Splitting and merging components of a nonconjugate Dirichlet process mixture model. *Bayesian Anal.* **2,** 445–472 (Sept. 2007).

113. Lovell, D., Adams, R. P. & Mansingka, V. *Parallel markov chain monte carlo for dirichlet process mixtures* in *Workshop on Big Learning, NIPS* (2012).

114. Williamson, S., Dubey, A. & Xing, E. P. *Parallel Markov Chain Monte Carlo for Nonparametric Mixture Models.* in *ICML (1)* (2013), 98–106.

115. Izquierdo-Verdiguier, E., Gómez-Chova, L., Bruzzone, L. & Camps-Valls, G. *Semisupervised nonlinear feature extraction for image classification* in *2012 IEEE International Geoscience and Remote Sensing Symposium* (July 2012), 1525–1528. doi:`10.1109/IGARSS.2012.6351244`.

116. Xuan, G., Zhang, W. & Chai, P. *EM algorithms of Gaussian mixture model and hidden Markov model* in *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)* **1** (2001), 145–148 vol.1. doi:`10.1109/ICIP.2001.958974`.

117. Bilmes, J. A. *et al.* A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. *International Computer Science Institute* **4,** 126 (1998).

118. Ho, Y. & Pepyne, D. Simple Explanation of the No-Free-Lunch Theorem and Its Implications. *Journal of Optimization Theory and Applications* **115,** 549–570. ISSN: 1573-2878 (2002).

119. Wolpert, D. H. & Macready, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* **1,** 67–82. ISSN: 1089-778X (Apr. 1997).

120. Leisch, F. Bagged clustering (1999).

121. Miller, J. W. & Harrison, M. T. Inconsistency of Pitman-Yor process mixtures for the number of components. *Journal of Machine Learning Research* **15,** 3333–3370 (2014).

122. Silberman, N., Hoiem, D., Kohli, P. & Fergus, R. in *Computer Vision – ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part V* (eds Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y. & Schmid, C.) 746–760 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2012). ISBN: 978-3-642-33715-4. doi:10.1007/978-3-642-33715-4_54. <http://dx.doi.org/10.1007/978-3-642-33715-4_54>.

123. Sklansky, J. Image Segmentation and Feature Extraction. *IEEE Transactions on Systems, Man, and Cybernetics* **8,** 237–247. ISSN: 0018-9472 (Apr. 1978).

124. Hoiem, D., Efros, A. A. & Hebert, M. *Geometric context from a single image* in *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1* **1** (Oct. 2005), 654–661 Vol. 1. doi:10.1109/ICCV.2005.107.

125. Estevez, P. A., Tesmer, M., Perez, C. A. & Zurada, J. M. Normalized Mutual Information Feature Selection. *IEEE Transactions on Neural Networks* **20,** 189–201. ISSN: 1045-9227 (Feb. 2009).

126. Chapelle, O., Weston, J. & Schölkopf, B. Cluster kernels for semi-supervised learning. *Advances in neural information processing systems,* 601–608 (2003).

127. Myhre, J. N. & Jenssen, R. *Mixture weight influence on kernel entropy component analysis and semi-supervised learning using the Lasso* in *2012 IEEE International Workshop on Machine Learning for Signal Processing* (Sept. 2012), 1–6. doi:10.1109/MLSP.2012.6349814.

128. Topchy, A., Minaei-Bidgoli, B., Jain, A. K. & Punch, W. F. *Adaptive clustering ensembles* in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* **1** (Aug. 2004), 272–275 Vol.1. doi:10.1109/ICPR.2004.1334105.

129. Elghazel, H. & Aussem, A. Unsupervised feature selection with ensemble learning. *Machine Learning* **98,** 157–180 (2015).

130. He, X., Cai, D. & Niyogi, P. *Laplacian score for feature selection* in *NIPS* **186** (2005), 189.

131. Das, S. *Filters, wrappers and a boosting-based hybrid for feature selection* in *ICML* **1** (2001), 74–81.

132. Weston, J. *et al. Feature selection for SVMs* in *Proceedings of the 13th International Conference on Neural Information Processing Systems* (2000), 647–653.