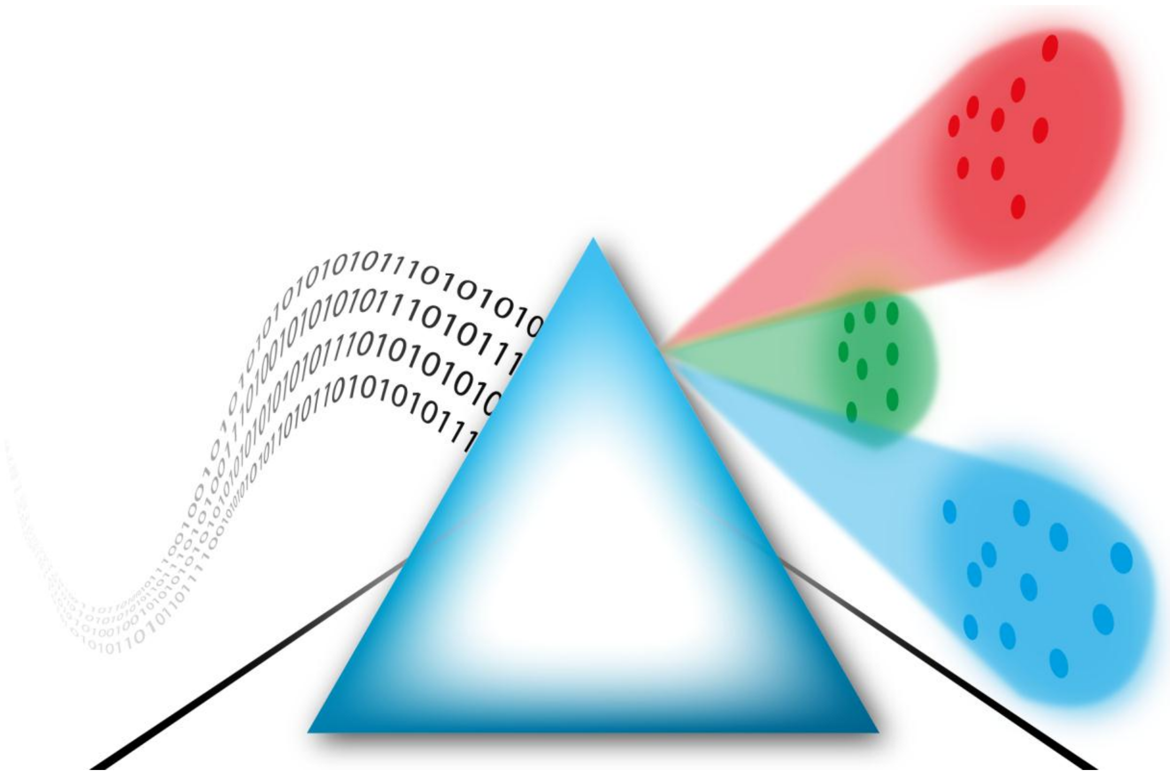
UNIVERSITY OF TROMSØ UIT

FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF PHYSICS AND TECHNOLOGY

Information Theoretic Learning with K Nearest Neighbors: A New Clustering Algorithm



Vidar V. Vikjord

FYS-3921 Master's Thesis in Electrical Engineering

June 2012

Abstract

The machine learning field based on information theory has received a lot of attention in recent years. Through kernel estimation of the probability density functions, methods developed with information theoretic measures are able to use all the statistical information available in the data, not just a finite number of moments. However, by using kernel estimation, the methods are dependent on choosing a suitable bandwidth parameter and have trouble dealing with data which vary on different scales.

In this thesis, the field of information theoretic learning has been explored using k-nearest neighbor estimates for the probability density functions instead. The developed estimators of the information theoretic measures was used in a clustering routine and compared with the traditional kernel estimators. Performing clustering on a range of datasets and comparing the performance, the new method proved to provide superior results without the need of tuning any parameters. The performance difference was found to be especially large when clustering datasets where groups were on different scales.

Acknowledgments

First and foremost, I want to thank by advisor, Robert Jenssen for all his help and guidance. All the discussions we have had on machine learning subjects have really been a huge help and your knowledge and enthusiasm an inspiration.

I would also like to thank my fellow students, my K Nearest Neighbors here in the office, for the good company and all the discussions we've had. Getting to air my problems, receive immediate debugging help or just having a distraction has really made the whole process of writing more enjoyable. Thank you. You know who you are. All K of you.

Lastly, a huge thanks to my wonderful girlfriend and great family. Working on this thesis, I know I have been a bit absent. I look forward to spending more time with all of you again.

Vidar V. Vikjord
June 2012

Contents

Preface

0.1 Notation Used

The notational conventions described in Table 1 is used throughout this text.

Table 1: Notation used throughout this text

$\boldsymbol{X}$, X	Random variable (multi- and univariate).
$\boldsymbol{x}_i$, x_i	Realization of random variable (multi- and univariate).
$\mathbb{R}^d$	Feature space of d dimensional dataset.
N	Number of datapoints in set.
$(\omega_1, ..., \omega_C),\ C$	Class labels and number of classes in total.
$\kappa(\cdot, \cdot)$	Kernel function.
$V_{k_{\omega_i}}(\boldsymbol{x})$	Hypervolume spanned from $\boldsymbol{x}$ to the k-th nearest neighbor in class ω_i. For the one-class case, this ω_i is omitted.
F	Feature space for mapped data.
$H_S(X)$	Shannon Entropy.
$H_\alpha(X)$	Renyi's α Entropy.
$G = (\boldsymbol{V}, \boldsymbol{E})$	Graph with vertex set $\boldsymbol{V}$ and edges $\boldsymbol{E}$.
M	Manifold of data embedded in $\mathbb{R}^d$.
$(\lambda_i, \boldsymbol{v}_i)$	Eigenvalue, eigenvector pair.

0.2 Abbreviations

PDF Probability Density Function

PMF Probability Mass Function

KNN K Nearest Neighbor

RPKH Reproducing Kernel Hilbert space

ML	Maximum Likelihood
SVM	Support Vector Machine
PCA	Principal Component Analysis
IT	Information Theory
ITL	Information Theoretic Learning
ITC	Information Theoretic Clustering
CIP	Cross Information Potential

Chapter 1

Introduction

In recent years, a new direction in machine learning based on Information Theory (IT) has received a lot of attention. This emerging field has been able to extend many of the techniques in the established theory by replacing old measures with information theoretic alternatives. Examples of this is for instance new ways of performing dimensionality reduction, classification, signal denoising and clustering. See e.g. the discussions in Príncipe [45] and Jenssen et al. [31].

The breakthrough leading to these developments within Information Theoretic Learning (ITL), came from the realization that one specific information theoretic measure, Renyi's quadratic entropy [46], lends itself to be very easily estimated directly from data. Up until this point, a lot of the established techniques were limited to only consider the first and second order statistics of a dataset. Letting entropy, and later other IT measures, instead control the machine learning algorithms allowed them to consider *all* the statistical information contained in the data, since these measures are functions of the underlying Probability Density Functions (PDFs) [17, 22, 21, 30].

Estimation of the PDFs in this setting have primarily been done via a technique called Parzen windowing [42]. This method uses special functions which map the d-dimensional data vectors to the real line according to some non-linear transformation to produce an estimate. How this transformation should be done vary from problem to problem and because of this, the functions have a bandwidth parameter which helps control the behavior. With this introduced parameter choice, all of the methods using Parzen windowing have a limitation in that the results they produce are highly dependent on choosing the correct bandwidth.

In certain settings, this means that another step is required by the algorithms to do cross-validation searching for the best choice. While this increases the complexity of the implementations, it is generally accepted as

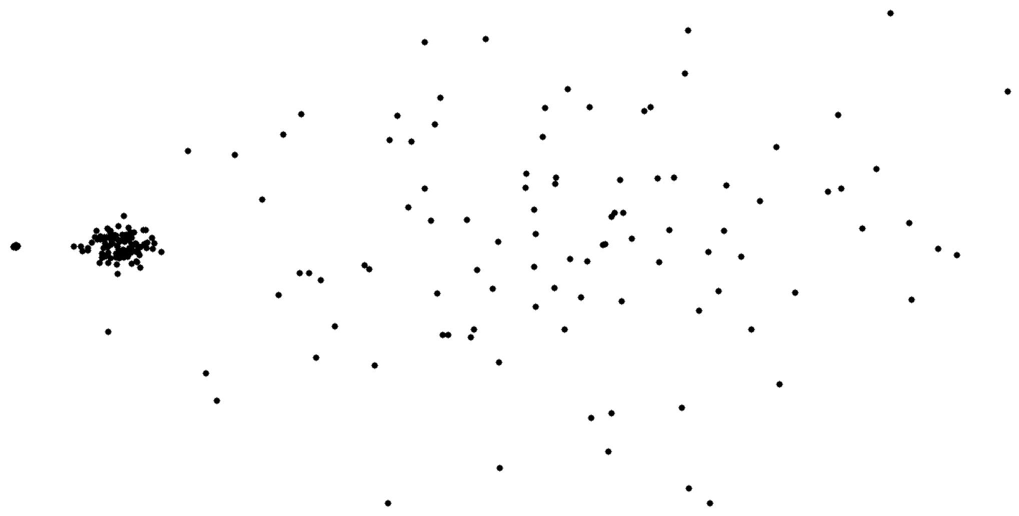


Figure 1.1: Example of dataset with three clusters on different scales.

a necessity when using these methods. If no prior information about the datapoints is known however, which is typically the case when performing clustering, the cross-validation step is not possible to do. In these cases, the methods rely on using heuristic techniques to decide on a bandwidth parameter, but these approximations often produce poor results [53].

Another problem with using a fixed bandwidth, is that when presented data varying on different scales, there does not exist one specific choice which is suitable for all the datapoints. An example of such a dataset is seen in Figure 1.1. This seemingly easy structure, consisting of three artificially created groups on different scales, proves very difficult for the Parzen window technique to handle directly. There does exist techniques for letting the bandwidth parameter adapt locally, see e.g. [51, 43], however these have not seen much use in ITL thus far.

A different method of estimating the PDF of a dataset is by the K Nearest Neighbor (KNN) method [54, 9]. This an estimation technique derived from the same basic principal as the Parzen window method, but does the estimation in a fundamentally different way. As will be seen in Chapter 2, the KNN approach is more adaptive to the local structure of the data and with that better suited to handle data on different scales.

So far in ITL, only the Parzen method has been used to estimate the PDFs which enter into the information theoretic measures. The IT techniques which makes use of the Parzen estimate, then naturally inherit the problems the estimate suffers from. This means that many of the machine learning techniques extended to use IT considerations, are also sensitive to correct parameter choice and do not handle data on different scales well.

Performing ITL with KNN-estimates have so far received little attention and only recently have some articles explored the topic [33, 55, 11]. Given the estimates original connection with the Parzen window estimate and its ability to adapt to different scales and structures in the data, it is interesting to investigate if the inherited problems of ITL methods can be avoided by instead using KNN estimates.

In this thesis a new method of obtaining the required information theoretic measures will be explored using the KNN approach. Using this method, new estimates of the IT measures will be derived and shown to be more adaptive to the local variations in the data. These estimates will then be explored in the setting of a heuristic clustering routine similar to one previously developed which used Parzens method [21]. Adapting the algorithm to use the KNN estimates, it will also prove to produce reliable results without tuning any parameters.

Through comparison with the Parzen estimate, exploring clustering on many real and synthetic datasets, it is shown that the new KNN method consistently outperforms the old estimation technique. Especially when clustering datasets where groups are on different scales, the new method provide vastly better results.

1.1 Kernel and KNN Methods

For reasons which will become apparent in Section 2.3, the Parzen window estimate can for certain choices of windowing function be called a kernel method. Kernel methods are a family of machine learning algorithm which implicitly map the input to an unknown feature space where inner products are calculated by the use of Mercer kernels [39]. As the most common choice of windowing function in the Parzen estimate is a Mercer kernel, the method is in this thesis called a kernel method.

KNN methods on the other hand, are not connected with any such mapping to another space. Instead, by KNN method, it is meant any machine learning technique which intrinsically evaluate a restricted neighborhood in any point of interest.

The theme of this thesis will be to explore the similarity and differences of Kernel and KNN methods, with the goal of developing a new KNN approach to ITL.

1.2 Structure of Thesis

As the main focus of the thesis is to explore a KNN approach to learning in a field dominated by kernel methods, it is natural discuss some of the fundamental similarities and differences between these two paradigms. Chapter 2 will discuss the Parzen windowing and KNN approach to estimating PDFs and look at some of the properties of each of the estimates. Next the term kernel method will be more precisely defined and an example of a method investigated. The final section of Chapter 2 will explore methods in machine learning connected to KNN-considerations.

Chapter 3 will introduce the field of Information Theoretic Learning (ITL). The intuition behind the theory is discussed and the various IT measures are presented. The topic of clustering with IT principals is then introduced and a cost function used by several algorithms is presented.

In Chapter 4, the new KNN approach to performing Information Theoretic Clustering will be derived. Here the estimate of the cost function will be presented along this an algorithm for optimization it. The results chapter will highlight some of the new methods properties and compare it to a previously developed algorithm which uses Parzen window estimates. Finally, the conclusions are presented in Chapter 6

Chapter 2

Background Theory

2.1 Introduction

In this chapter, some of the relevant background information will be introduced and reviewed. As the topic of the thesis is to explore a new KNN learning technique in a field where kernel methods are dominating, the focus will naturally fall on highlighting similarities and differences between these two approaches. It is then helpful to investigate the places in the established theory where the two paradigms appear.

Firstly, their uses and properties in non-parametric density estimation will be examined. Here, examples will be provided on a two dimensional synthetic dataset to give some intuition on the results. Then, the estimates will be used in a practical task of classification in a Bayesian setup with examples further highlighting some of their features.

Next, some theory only related to the use of a kernel is introduced with The *Kernel Trick* being defined and used to do non-linear dimensionality reduction with *Kernel*-PCA. The subject of dimensionality reduction is also explored in the KNN-setting with Laplacian Eigenmaps. This is a graph based method which is also shown performing clustering.

2.2 Probability Density Estimation

This section will deal with the different ways of estimating what underlying probability model a dataset is generated from. Given N samples, $\{\boldsymbol{x}_1,\boldsymbol{x}_2, \ldots \boldsymbol{x}_N\}$, in d dimensions from some process, the task is to use this known data to gain generalized knowledge about the system, i.e. do *learning*. With this information, typical machine learning and pattern recognition tasks such as classification, prediction, dimensionality reduction and clustering can be done.

The literature on this topic typically make a distinction between two main approaches for performing the estimation; *parametric* and *non-parametric* methods. The first one assumes a model with a set of parameters and then tries to estimate these parameters. The latter assumes no model and lets the given data infer information about the model more directly. Since this thesis largely revolves around non-parametric techniques, the theoretical introduction to parametric methods is left an appendix (see Appendix A). However, some of the results of parametric inference is illustrated in the section on classification herein.

For illustration throughout this discussion, a dataset, X, generated from a bivariate normal mixture model is assumed:

$$X \sim P_1 N(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + P_2 N(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \tag{2.1}$$

where $\boldsymbol{\mu_{1/2}}$ are the mean vectors and $\boldsymbol{\Sigma}_{1/2}$ the covariances. P_1 and P_2 are the prior probabilities for each of the mixtures.

2.2.1 Nonparametric Methods

Parametric estimation techniques can give very good results if the model assumed for the data is correct. Often however, a simple underlying model (whose parameters can be estimated) is not obtainable. When this is the case, no amount of advanced estimation can help the methods escape the fundamental problem; a wrong model. Real world data can indeed be hard to approximate with an analytical model; the data set could be multimodal, have different distributions along the different dimensions and come from a phenomenon with complicated dependencies between the components. Non-parametric methods strong suit is that they assume no model and instead do inference based purely on the given dataset.

The basic idea used to build up the nonparametric methods comes from the fact that the probability of a random variable $\boldsymbol{x}$ falling inside a region R, is given by

$$P(\boldsymbol{x} \in R) = \int_R p(\boldsymbol{x'}) d\boldsymbol{x'} \tag{2.2}$$

From this it can be seen that $P(\boldsymbol{x} \in R)$ is an averaged version of $p(\boldsymbol{x})$. Now, using (2.2) the other way around, a smoothed estimate of $p(\boldsymbol{x})$ can be obtained by estimating $P(\boldsymbol{x} \in R)$.

Given a dataset of N points, $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$ defined on $\mathbb{R}^d$, the probability of k points falling inside region R is governed by the binomial law as a function of k by

$$P(k_{\boldsymbol{x}} \in R) = \binom{n}{k} P^k (1-P)^{n-k}, \; P = P(\boldsymbol{x} \in R) \tag{2.3}$$

with the expected value of k being

$$E(k) = NP \tag{2.4}$$

From this, a reasonable estimate of $P(\boldsymbol{x} \in R)$ would be k/N.

Now, if $p(\boldsymbol{x})$ is assumed to be continuous and the region R is made so small that $p(\boldsymbol{x})$ does not vary by much within it, the integral in (2.2) can be approximated by a simple product

$$\int_R p(\boldsymbol{x'}) d\boldsymbol{x'} \approx p(\boldsymbol{x})V \tag{2.5}$$

with V being the volume enclosed by R.

By combining (2.2), (2.3) and (2.5), the following estimate of $p(\boldsymbol{x})$ is found

$$p(\boldsymbol{x}) \approx \frac{k/N}{V} \tag{2.6}$$

with k being the number of datapoints (from a total of N) falling inside a region around $\boldsymbol{x}$ with volume V. Equation (2.6) is the general form of the nonparametric estimator of $p(\boldsymbol{x})$ which will be built upon in the two extensions to come.

2.2.2 Parzen Windowing

The Parzen window estimate of a Probability Density Function (PDF) is based on using (2.6) with the volume V kept constant. The probability is estimated by evaluating the number of points, k, inside the fixed volume at different places in $\mathbb{R}^d$. Fixing the volume in this manner will turn out to make the estimate very sensitive to which volume is chosen for the given dataset, and is the fundamental difference between kernel and KNN estimates in this regard.

Parzen windowing is put in the domain of kernel methods. This comes from the choice of function used in the end counting the datapoints; a kernel function. Unfortunately, this will not be completely clear until the kernel trick and Mercers theorem is presented in the next section.

To introduce the Parzen windowing approach, firstly a mathematical function to count the number of datapoints falling inside R is defined. This function is used to determine k in equation (2.6) for any $\boldsymbol{x} \in \mathbb{R}^d$.

The so-called windowing function is defined as

$$\kappa_{\text{hist}}(\boldsymbol{u}) = \begin{cases} 1, & |u_j| \leq 1/2, \quad j = 1, 2, ..., d \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

and constitutes a hypercube around the origin. Note that $\boldsymbol{u}$ here can be replaced by the difference between two vectors in $\mathbb{R}^d$; $\boldsymbol{u} = \boldsymbol{x}_i - \boldsymbol{x}_j$. As such, $\kappa_{\text{hist}}()$ can also be thought of as a function measuring the similarity or closeness of two vectors. In this case, either they are inside each others hyper volume, or they are not.

By using this, the number of points falling inside a hypercube with sides of length h (and thus a volume of h^d) around $\boldsymbol{x}$ can be counted as

$$k(\boldsymbol{x}) = \sum_{i=1}^{N} \kappa_{\text{hist}} \left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h} \right) \tag{2.8}$$

By inserting (2.8) into (2.6), the histogram estimate of $p(\boldsymbol{x})$ is obtained

$$\hat{p}_{\text{hist}}(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{V} \kappa_{\text{hist}} \left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h} \right) \tag{2.9}$$

The estimate above immediately hints of a multitude of possible solutions, by different choices of counting function $\kappa(\cdot)$, as noted by Parzen in his original paper [42].

Since $\kappa_{\text{hist}}(\cdot)$ is discontinuous, the estimate of the probability density found by using this function will also be discontinuous. If it is instead replaced by some other, smooth, function $\kappa(\cdot)$, the resulting estimate will itself be smooth [54]. Parzen suggested several choices of this kernel function $\kappa(\cdot)$ and discussed which properties it would need to fulfill for the estimator to be valid. These properties will be discussed below.

The most common choice of $\kappa(\cdot)$ in the d-dimensional space is the Gaussian kernel

$$\kappa_{\text{Gauss}}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp\left\{ -\frac{(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})}{2} \right\} \tag{2.10}$$

Using a centered kernel ($\boldsymbol{\mu} = \boldsymbol{0}$) with spherical symmetry in space ($\boldsymbol{\Sigma} = \sigma \cdot \boldsymbol{I}$), the probability estimate reduces to

$$\hat{p}_{\text{Gauss}}(\boldsymbol{x}; \sigma) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{(2\pi)^{l/2} \sigma^d} \exp\left\{ -\frac{(\boldsymbol{x} - \boldsymbol{x}_i)^T (\boldsymbol{x} - \boldsymbol{x}_i)}{2\sigma^2} \right\} \tag{2.11}$$

with σ replacing h as a width parameter (the standard deviation of the Gaussian kernel).

Note, κ_{Gauss} is actually a Mercer kernel, meaning that using it is related to calculating inner product in some unknown feature space. Because of this, any machine learning algorithm which makes use of the Parzen PDF estimate in (2.11), can be called a kernel method. Details on this will be discussed in Section 2.3.

Properties

It was noted with the introduction of equation (2.2) that the nonparametric methods yield a space averaged estimation of the true probability density. Because of this, if the estimator given by (2.6) is to converge to the true probability density, the hypervolume V would need to tend to zero. This however leads to problems as the number of datapoints is finite; if V is made smaller while the number of datapoints are kept constant, each region R (of volume V) could be made to enclose no datapoints and the estimate would be useless. Moreover, if one datapoint would coincide with an arbitrarily small region, the estimate would diverge to infinity - an equally useless estimate. For these reasons, whenever a finite number of datapoints is assumed, the estimate will have to do some averaging to obtain useful estimates.

This averaging causes the probability density estimate to always deviate from the true underlying density as long as the number of points evaluated is limited (as it is in any real life application). If however this restriction is lifted, some of the probability estimates underlying properties can be more thoroughly investigated. Given a dataset $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$ in $\mathbb{R}^d$ drawn independently from $p(\boldsymbol{x})$ where N is allowed to go to infinity, the Parzen windows' convergence in mean square sense is evaluated. The mean square convergence of $\hat{p}_n(\boldsymbol{x})$ (where the subscript N is there to highlight the dependency on the number of data points) is defined as

$$\lim_{N\to\infty} \bar{p}_N(\boldsymbol{x}) = p(\boldsymbol{x}) \tag{2.12}$$

$$\lim_{N\to\infty} \sigma_N^2(\boldsymbol{x}) = 0 \tag{2.13}$$

where $\bar{p}_N(\boldsymbol{x})$ refers to the expected value of the estimate in $\boldsymbol{x}$ and $\sigma_N^2(\boldsymbol{x})$ is the variance of $\hat{p}(\boldsymbol{x})$ in $\boldsymbol{x}$.

To prove this convergence, some conditions has to be placed on the unknown density $p(\boldsymbol{x})$, $\kappa(\boldsymbol{x})$ and the window width σ. These conditions, as noted in [9], are

$$\sup_{\boldsymbol{u}} \kappa(\boldsymbol{u}) < \infty \tag{2.14}$$

$$\lim_{||\boldsymbol{u}||\to\infty} \kappa(\boldsymbol{u}) \prod_{i=1}^{d} u_i = 0 \tag{2.15}$$

$$\lim_{N\to\infty} V_N = 0 \tag{2.16}$$

$$\lim_{N\to\infty} NV_N = \infty \tag{2.17}$$

In addition, $\kappa(\boldsymbol{x})$ has to be continuous, non-negative and integrate to 1, i.e. itself be a PDF.

Considering the convergence of the mean value of the estimate, the expected value is investigated

$$\begin{aligned}\bar{p}(\boldsymbol{x}) &= E[p_N(\boldsymbol{x})]\\ &= \frac{1}{N}\sum_{i=1}^{N} E\left[\frac{1}{V_N}\kappa\left(\frac{\boldsymbol{x}-\boldsymbol{x_i}}{\sigma_N}\right)\right]\\ &= \int \frac{1}{V_N}\kappa\left(\frac{\boldsymbol{x}-\boldsymbol{v}}{\sigma_N}\right)p(\boldsymbol{v})d\boldsymbol{v}\\ &= \int \delta_N(\boldsymbol{x}-\boldsymbol{v})p(\boldsymbol{v})d\boldsymbol{v},\end{aligned}$$

where $\delta(\boldsymbol{x}-\boldsymbol{v}) = \frac{1}{V_N}\kappa\left(\frac{\boldsymbol{x}-\boldsymbol{v}}{\sigma_N}\right)$. Here, the definition of the expected value and the assumption of independence between each data sample has been used.

The final integral above can be recognized as the convolution between the true density and a smoothing function. By the condition given in (2.16), the smoothing windowing function $\delta(\cdot)$ becomes the Dirac delta function as N approaches infinity. The convolution between a function and the delta is known to yield the function itself, and thus the expected value converges to the true value for all $\boldsymbol{x}$ in $\mathbb{R}^d$.

In the derivations above it was shown that the mean of the estimate could fit the underlying distribution if the width σ_N (and consequently V_N) could approach 0. However, in so doing the estimate will exhibit a higher variance. Under the assumption of infinitely many datapoints, a smooth $p(\boldsymbol{x})$ and (2.14) – (2.17), the variance of the estimate is evaluated in the following manner

$$\begin{aligned}\mathrm{Var}(\hat{p}_N(\boldsymbol{x})) &= \sum_{i=1}^{N} E\left[\left(\frac{1}{NV_N}\kappa\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{\sigma_N}\right) - \frac{1}{N}\bar{p}_N(\boldsymbol{x})\right)^2\right] \\ &= NE\left[\frac{1}{N^2V_N^2}\kappa^2\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{\sigma_N}\right)\right] - \frac{1}{N}\bar{p}_N^2(\boldsymbol{x}) \\ &= \frac{1}{NV_N}\int \frac{1}{V_N}\kappa^2\left(\frac{\boldsymbol{x}-\boldsymbol{v}}{\sigma_N}\right)p(\boldsymbol{v})d\boldsymbol{v} - \frac{1}{N}\bar{p}_N^2(\boldsymbol{x})\end{aligned}$$

Now, dropping the last term and recognizing the integrand as $\kappa(\cdot)\delta_N(\boldsymbol{x} - \boldsymbol{v})p(\boldsymbol{v})$, a bound on the variance can be found to be

$$\mathrm{Var}(\hat{p}_N(\boldsymbol{x})) \leq \frac{\sup(\kappa(\cdot))\bar{p}_N(\boldsymbol{x})}{NV_N} \tag{2.18}$$

From this it can be seen that the variance is minimized by having the smoothing function $\kappa(\cdot)$ have a large volume. However, for the estimate to converge to the true distribution, the kernel should approach the Dirac delta corresponding to a volume approaching zero. In this artificial setting of infinitely many datapoints, (2.18) could be made small by invoking condition (2.17) and for instance letting the volume grow slower than $1/N$. Any real life scenario would however lead to having to make a trade off between estimate error and variance.

Silverman [53] has explored the optimal choice of σ in the case of a Gaussian smoothing kernel to find

$$\sigma_{\mathrm{S}} = \hat{\sigma}_{\mathrm{M}}\left\{4N^{-1}(2d+1)^{-1}\right\}^{\frac{1}{d+4}} \tag{2.19}$$

where $\hat{\sigma}_{\mathrm{M}}$ is the mean value of the Maximum Likelihood (ML) standard deviation estimate obtained from the data[1]. It is given as

$$\hat{\sigma}_M = d^{-1}\sum_{i=1}^{d}\hat{\Sigma}_{i,i} \tag{2.20}$$

where $\hat{\Sigma}_{i,i}$ is the element in row i and column i of the ML-covariance matrix estimate from the data.

A visualization of the Parzen window probability density estimate of model (2.1) for two different choices of σ is shown in Figure 2.1. In this figure it it is evident that the smallest σ is causing the estimate to be a poor one, with small Gaussian peaks around each datapoint. While the other choice of σ smooths out the datapoints sufficiently to recreate the underlying bivariate Gaussian mixture.

[1]See Appendix A for an introduction to the ML estimates of the covariance matrix.)

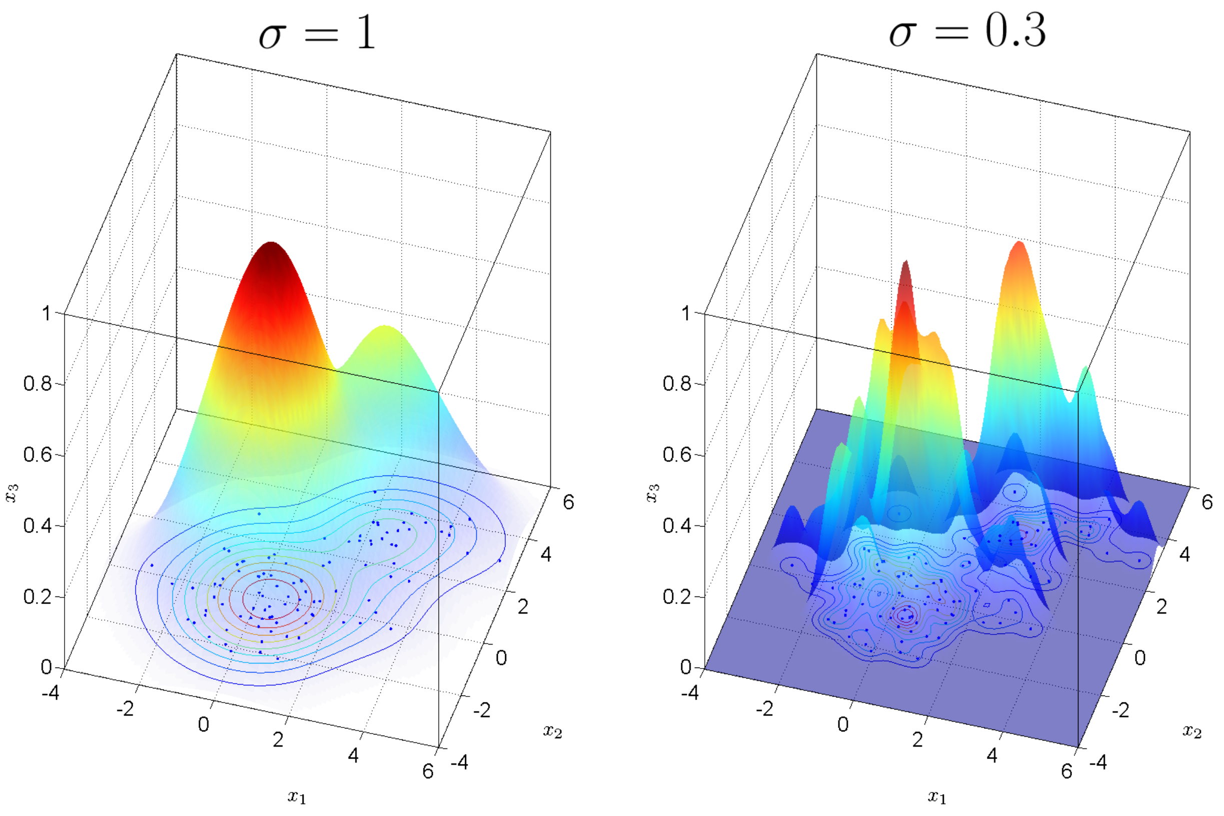


Figure 2.1: Probability density estimate using Parzen Windowing with different σ on synthetic dataset.

2.2.3 K Nearest Neighbors

The K Nearest Neighbor probability density estimation is, like the Parzen Window approach, based on (2.6). However, instead of keeping the hypervolume V constant and varying k, k is now kept constant and V is changed so that it encloses k points. With this in mind, the following estimate is defined

$$\hat{p}_{\text{kNN}}(\boldsymbol{x}; k) = \frac{k}{NV_k(\boldsymbol{x})} \tag{2.21}$$

with $V_k(\boldsymbol{x})$ being a function which measures the hypervolume of the structure in $\mathbb{R}^d$ which encloses the k nearest neighbors of $\boldsymbol{x}$.

With an Euclidean distance (the 2-norm), this results in the volume becoming a hyper-sphere. For a hyper-sphere in d dimensions, the volume is given as

$$V(\boldsymbol{x}) = \frac{\pi^{d/2}}{\Gamma(d/2+1)} ||\boldsymbol{x}||_2^d \tag{2.22}$$

where $\Gamma(\cdot)$ is the gamma function and $||\cdot||_2$ is the Euclidean norm.

Properties

In the estimation given in (2.21), the volume which encloses k points depends on the distance metric. Choosing different metrics leads to different formulas for the volume and it might also change which points are considered the k nearest as was noted in the original article on the subject [35]. A visualization of the three most common metrics (the 1-, 2- and ∞-norm) is given in Figure 2.2. It could be noted that even though the same point has been found to be the fifth closest to the $\times$, the areas (or hypervolumes in higher dimensions) are different. The shaded regions correspond to areas where, if one or more datapoints were enclosed, would cause a different set of points to be evaluated as the k (in this case 5) nearest.

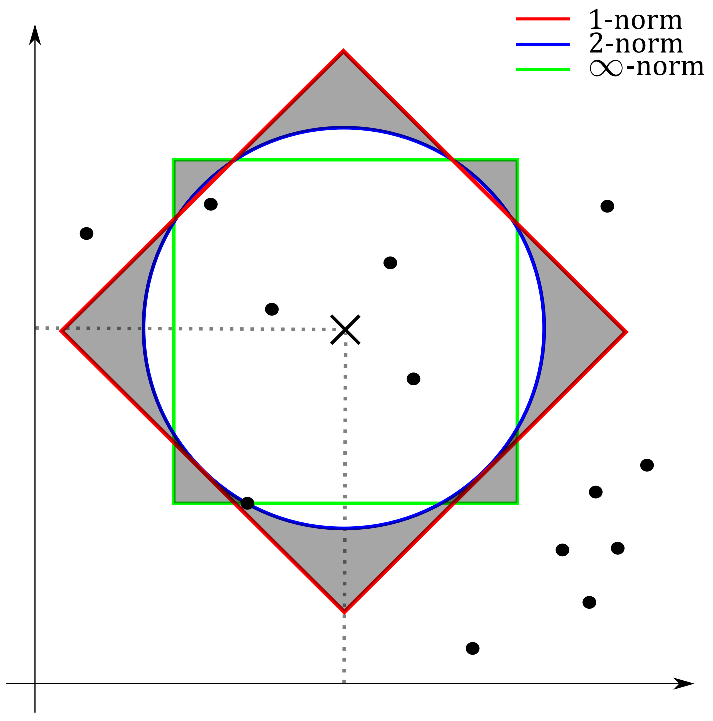


Figure 2.2: The $k = 5$ nearest neighbors for different distance metrics.

The estimator $\hat{p}_{\text{kNN}}$ has been proven to be asymptotically consistent under the assumptions

$$\lim_{N\to\infty} k_N = \infty \tag{2.23}$$

$$\lim_{N\to\infty} k_N/N = 0 \tag{2.24}$$

in the original article [35].

The estimate is itself continuous, however, its derivative is not. This comes from the fact that as the designated k nearest points change, the estimate could, over an arbitrarily small step, change from growing to shrinking (and vice versa) whenever the k-th nearest neighbor changes. This phenomenon also causes the estimate to be somewhat spiky for k and $N \ll \infty$.

Using gradient descent optimization on a KNN surface is also impossible. This is because the gradient of the surface never goes to zero. Moving a delta in a direction from any point in space, the movement is either towards or away from the k-nearest neighbor (if not the k nearest neighbor point changes), and thereby never zero.

Using the synthetic dataset and an estimate with $k = 5$, the results visualized in Figure 2.3 is produced. Here the estimates erratic behavior is seen quite clearly.

Comparison

Two techniques for performing nonparametric density estimation was discussed in the two preceding sections. It was seen that they both stem from the probability approximation given in (2.6) and they are both consistent estimates as $N \to \infty$ under mild assumptions. In any real life applications where a limited number of datapoints are used, the two estimates do however produce slightly different results. This can be seen if Figure 2.1 is compared with Figure 2.3. A better comparison of the two is given in Figure 2.4. It is here seen that the Parzen windowing technique produce a smoother estimate, and the true bimodal form of the underlying distribution is uncovered.

It is also interesting to note that the Parzen window estimate uses all the points in the dataset (N evaluations) to estimate the PDF in $\boldsymbol{x}$, while the KNN method only rely on *one* point, the k-nearest neighbor. This does not mean that the complexity of using the KNN estimate is any less that the Parzen method however. As the distance have to be calculated to all the points in the set, before the k-th neighbor can be found, this leaves N operations for this method also. Actually, the KNN method could be said to have a higher complexity, as a sorting of the distances is also needed.

An even more important difference between the two estimates, is the behaviors when presented with data on very different scales. In Figure 2.5,

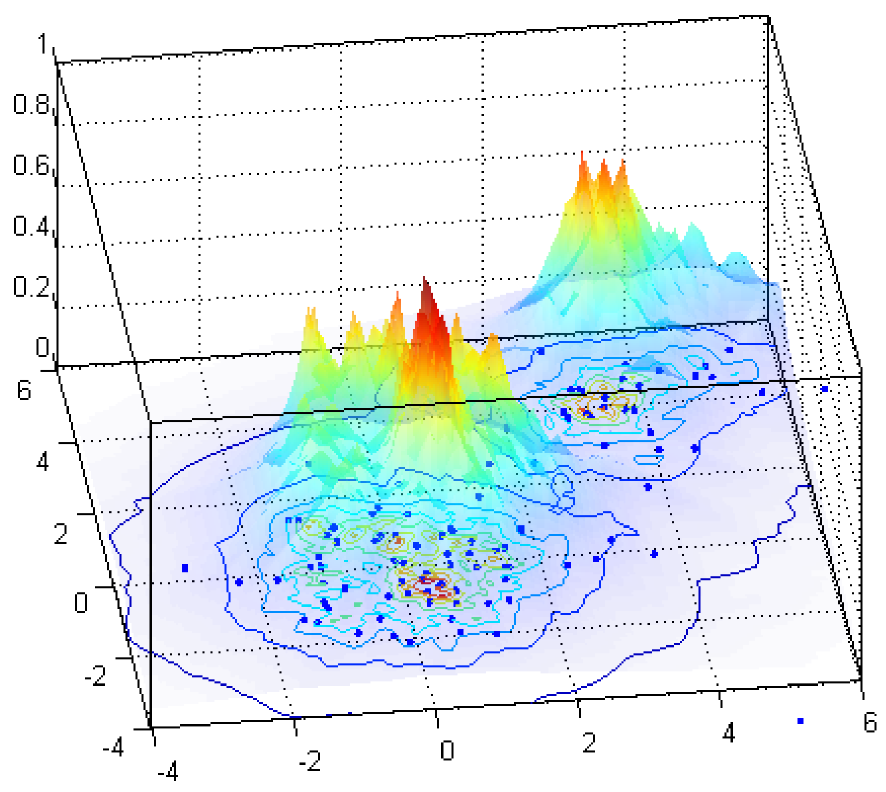
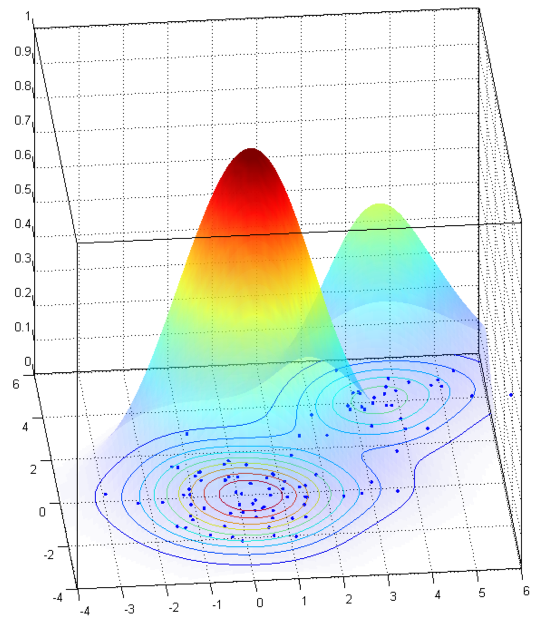


Figure 2.3: The $k = 5$ nearest neighbors PDF estimate for the synthetic dataset.



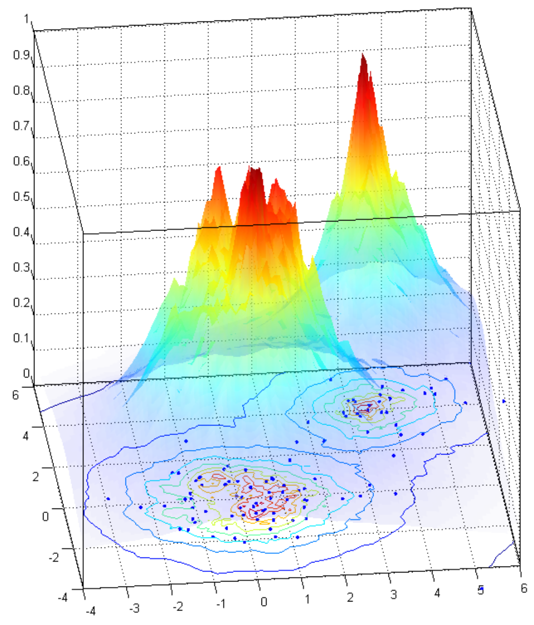


Figure 2.4: Probability density estimate with Parzen windowing to the left, and K-Nearest Neighbor to the right.

200 datapoints drawn from

$$\boldsymbol{X} \sim 0.5N\left(\begin{bmatrix}0\\0\end{bmatrix}, 0.01 \times \boldsymbol{I}\right) + 0.5N\left(\begin{bmatrix}20\\20\end{bmatrix}, 100 \times \boldsymbol{I}\right) \tag{2.25}$$

is attempted estimated by the two non-parametric methods. The parzen window method here uses Silverman's approximation (2.19) to choose a bandwidth. It is in (a) seen that this choice of bandwidth over-smooths the dense area. This causes the estimate to be very high in large regions completely void of datapoints near the dense area, which is clearly wrong. Using the KNN estimate, the PDF surface falls off much more quickly when moving out of the dense region, something which better reflect the true underlying densities[2].

2.2.4 Classification

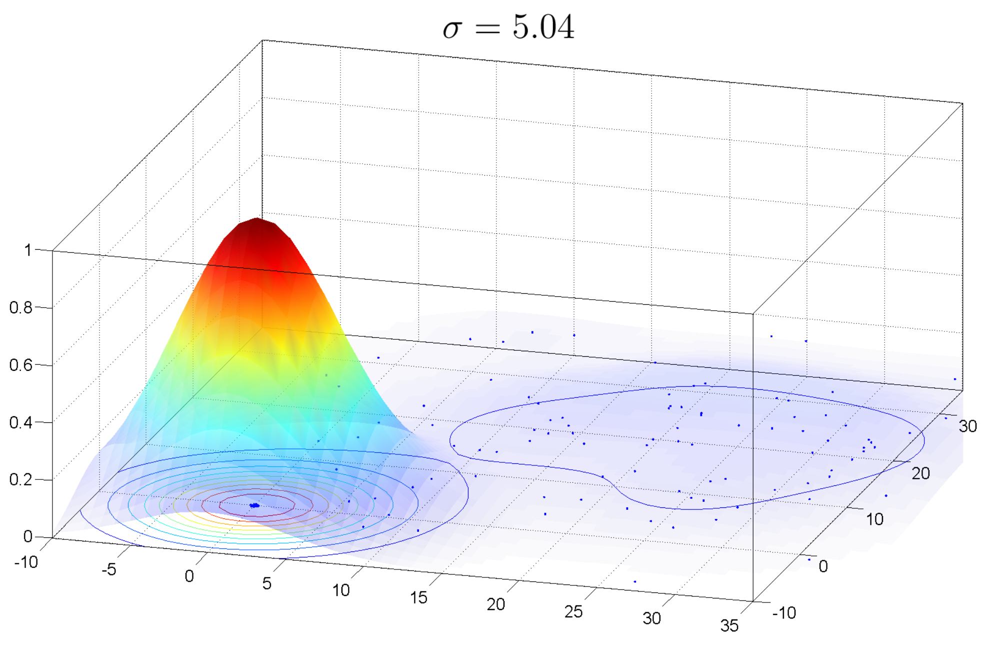
In this section, one of the more common uses of the probability density is explored, namely classification in the Bayesian setting. As with many machine learning tasks, it minimizes a cost function, here related to the possible misclassifications of the data, in order to reach a solution. This solution turns out to be dependent on the which probability density estimate is obtained from the training data. With that, it is well suited to explore the estimates developed in the previous section further. The introduction to the basic Bayesian classifier setup, is left as an appendix (B). This is done to keep the text somewhat more focused on the goal of investigating the differences between the PDF-estimates.

The setup is to evaluate the created decision boundary given data from two Gaussian distributions with the same covariance structure and different mean vectors; $p_1(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$ and $p_2(\boldsymbol{\mu}_2, \boldsymbol{\Sigma})$. This data is the training set for which the class of each datapoint is known. The training data is then used to create a decision rule for classifying any new datapoints. More details regarding this can be found in the appendix, but the end decision rule in the two class case is stated here; an unknown $\boldsymbol{x}$ is classified to according to

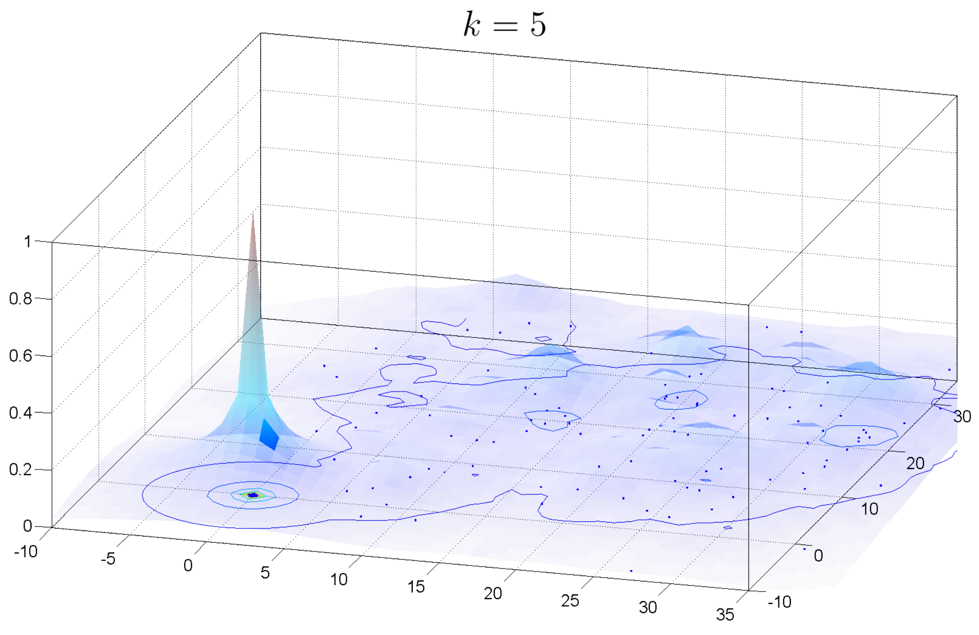
$$\boldsymbol{x} \rightarrow \omega_1(\omega_2) \quad \text{if} \quad \frac{p(\boldsymbol{x}|\omega_1)}{p(\boldsymbol{x}|\omega_2)} \geq (<) \frac{P(\omega_2)}{P(\omega_1)} \frac{[L_{2,1} - L_{2,2}]}{[L_{1,2} - L_{1,1}]} \tag{2.26}$$

where the condition for the alternative classification, $\boldsymbol{x} \rightarrow \omega_2$, is given in the parenthesis. The L-values refer to user set costs of classifying/misclassifying. As explained in the appendix, the prior probabilities $P(\omega_1)$ and $P(\omega_2)$ are

[2]Some smoothing was applied to the KNN estimate to get the visualization seen in the figure. See Appendix D for more details.



(a) Parzen



(b) KNN

Figure 2.5: Parzen and KNN estimate of data drawn from Gaussian mixture with very different covariance scale. The KNN is seen to adapt more to the data and not create an unreasonable estimate around the dense datapoints.

estimated by simply counting the number of occurrences from each class and dividing by the total number of points.

Parametric maximum likelihood

Using the parametric approach of Appendix A, it is assumed that both $p(\boldsymbol{x}|\omega_1)$ and $p(\boldsymbol{x}|\omega_2)$ are Gaussian distributions; $\boldsymbol{X}_1 \sim N(\boldsymbol{\mu_1}, \boldsymbol{\Sigma_1})$ and $\boldsymbol{X}_2 \sim N(\boldsymbol{\mu_2}, \boldsymbol{\Sigma_2})$. This method states that the parameters could be estimated by Maximum Likelihood estimates given in (A.5) – (A.6). Using these estimates yields

$$\hat{p}(\boldsymbol{x}|\omega_1) = N(\hat{\boldsymbol{\mu}}_1, \hat{\boldsymbol{\Sigma}}_1) \qquad \hat{p}(\boldsymbol{x}|\omega_2) = N(\hat{\boldsymbol{\mu}}_2, \hat{\boldsymbol{\Sigma}}_2)$$

These estimated densities can then be inserted directly into (2.26) to get the classification rule

$$\boldsymbol{x} \to \omega_1(\omega_2) \text{ if } \frac{|\hat{\boldsymbol{\Sigma}}_1|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\hat{\boldsymbol{\mu}}_1)'\hat{\boldsymbol{\Sigma}}_1^{-1}(\boldsymbol{x}-\hat{\boldsymbol{\mu}}_1)\right\}}{|\hat{\boldsymbol{\Sigma}}_2|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\hat{\boldsymbol{\mu}}_2)'\hat{\boldsymbol{\Sigma}}_2^{-1}(\boldsymbol{x}-\hat{\boldsymbol{\mu}}_2)\right\}} \geq (<) \frac{\hat{P}(\omega_2)}{\hat{P}(\omega_1)} \frac{[L_{2,1}-L_{2,2}]}{[L_{1,2}-L_{1,1}]} \quad (2.27)$$

Parzen windowing

If the non-parametric estimation method called Parzen windowing is used, no assumptions has to be placed on the data. Instead, so-called kernel functions are placed over each known datapoint from the distribution and summed up to give an estimate. In 2.2.2 the probability density estimate using this technique with a Gaussian kernel was shown, (2.11), to be

$$\hat{p}_{\text{Gauss}}(\boldsymbol{x}; h) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{(2\pi)^{d/2}\sigma^d} \exp\left\{-\frac{(\boldsymbol{x}-\boldsymbol{x}_i)^T(\boldsymbol{x}-\boldsymbol{x}_i)}{2\sigma^2}\right\}$$

If the same spherical kernel is used to estimate both $p(\boldsymbol{x}|\omega_1)$ and $p(\boldsymbol{x}|\omega_2)$, giving the estimates $\hat{p}(\boldsymbol{x}|\omega_1)$ and $\hat{p}(\boldsymbol{x}|\omega_2)$, these can be used directly to get the decision boundary based on the training data

$$x \to \omega_1(\omega_2) \text{ if } \frac{\frac{1}{N_1}\sum_{i=1}^{N_1} \exp\{\frac{-(\boldsymbol{x}-\boldsymbol{x}_i)'(\boldsymbol{x}-\boldsymbol{x}_i)}{2h^2}\}}{\frac{1}{N_2}\sum_{j=1}^{N_2} \exp\{\frac{-(\boldsymbol{x}-\boldsymbol{x}_j)'(\boldsymbol{x}-\boldsymbol{x}_j)}{2h^2}\}} \geq (<) \frac{\hat{P}(\omega_2)}{\hat{P}(\omega_1)} \frac{[L_{2,1}-L_{2,2}]}{[L_{1,2}-L_{1,1}]} \quad (2.28)$$

where the common terms in the fraction on the left side, $1/\left((2\pi)^{d/2}\sigma^d\right)$, have been divided out.

K Nearest Neighbor

With the KNN probability density estimation, it was seen in 2.2.3 that the volume was left free to vary to at each point to encapsulate k points. The probability density estimate then became

$$\hat{p}(\boldsymbol{x}) = \frac{k}{NV(\boldsymbol{x})}$$

where the dependency on the volume is emphasized.

Adopting this probability density estimate and using the Bayesian classification rule, the classification of a new point $\boldsymbol{x}$ is done by evaluating

$$x \to \omega_1(\omega_2) \quad \text{if} \quad \frac{\frac{k}{N_1 V_{k_{\omega_1}}(\boldsymbol{x})}}{\frac{k}{N_2 V_{k_{\omega_2}}(\boldsymbol{x})}} = \frac{N_2 V_{k_{\omega_2}}(\boldsymbol{x})}{N_1 V_{k_{\omega_1}}(\boldsymbol{x})} \geq (<) \frac{\hat{P}(\omega_2)}{\hat{P}(\omega_1)} \frac{[L_{2,1} - L_{2,2}]}{[L_{1,2} - L_{1,1}]} \tag{2.29}$$

where $V_{k_{\omega_1}}(\boldsymbol{x})$ and $V_{k_{\omega_2}}(\boldsymbol{x})$ is the hyper volume with center in $\boldsymbol{x}$ enclosing k points from ω_1 and ω_2 respectively.

Compar

2.2.5 Examples

In this section, some decision boundaries produced from using the different plug in estimators of the probability density discussed above are shown. The synthetic dataset used is a bivariate Gaussian mixture of $N = 100$ points. The dataset, $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{100}\} \in \mathbb{R}^2$, has its first 70 elements drawn as

$$\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{70}\} \sim N\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)$$

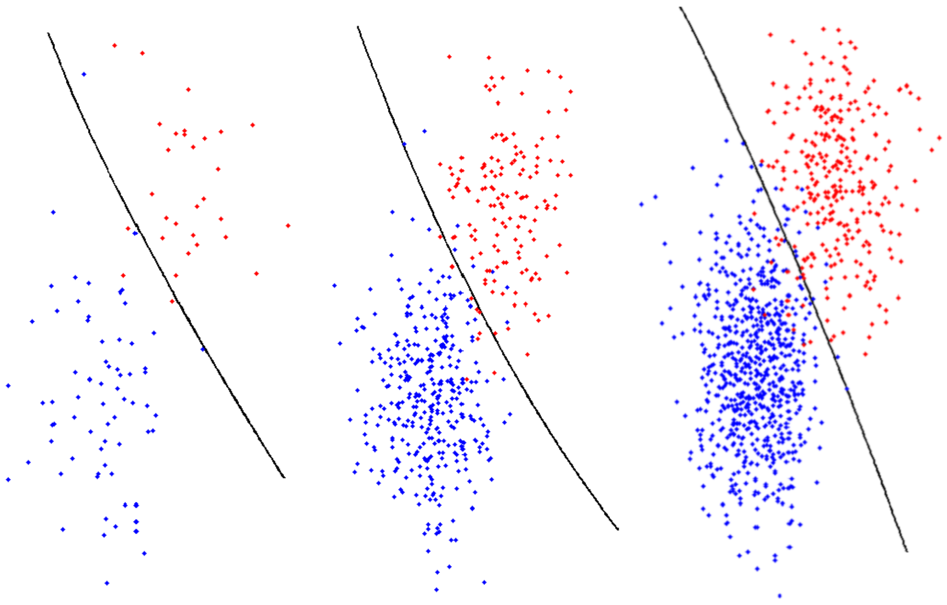
while the last 30 are drawn as

$$\{\boldsymbol{x}_{71}, \boldsymbol{x}_{72}, ..., \boldsymbol{x}_{100}\} \sim N\left(\begin{bmatrix}2.5\\2.5\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)$$

Here, $P(\omega_1) = 0.7$ and $P(\omega_2) = 0.3$ has been assumed, and the dataset has been created so that the prior probabilities will be exact, i.e. no stochastic evaluation has been done in choosing which of the classes to create data from.

In the proceeding figures, some intuition about the behavior of the different plug in estimators is sought and the effect of varying their parameters is demonstrated.

For the parametric density estimate, the correct model (described above) is selected, and the estimates described in Appendix A are used to obtain



(a) $N = 100$ (b) $N = 500$ (c) $N = 1000$

Figure 2.6: Decision boundaries with a parametric PDF estimate for different training set sizes.

the parameters $\hat{P}(\omega_1)$, $\hat{P}(\omega_2)$, $\hat{\boldsymbol{\mu}}_1$, $\hat{\boldsymbol{\mu}}_2$, $\hat{\boldsymbol{\Sigma}}_1$ and $\hat{\boldsymbol{\Sigma}}_2$. As these estimates are unbiased, it is expected that as the number of datapoints grow, the estimates converge to the true parameters. The effect of this is seen in Figure 2.6, where the number of datapoints in the synthetic set is allowed to grow (from 100 to 1000). With the number of available datapoints increasing, the estimates gets closer to the true parameters. Moreover, the fact that the mixture models are drawn with the same underlying covariance becomes clearer as the decision boundary can be seen to shift towards a more straight line. This is known to be the optimal classification rule in the case of a bivariate mixture of two Gaussians with equal covariance [54].

In light of the assumptions which lead to (B.10) (namely that the two classes needed to be drawn with the same covariance structure), this is expected; with more datapoints, the estimates $\hat{\boldsymbol{\Sigma}}_1$ and $\hat{\boldsymbol{\Sigma}}_2$ both converge to the true covariance $\boldsymbol{\Sigma}$[3] and the line is straightened.

With the Parzen window estimate, no explicit data model has to be assumed and only the data itself and the width parameter σ governs the prob-

[3]Equal to $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

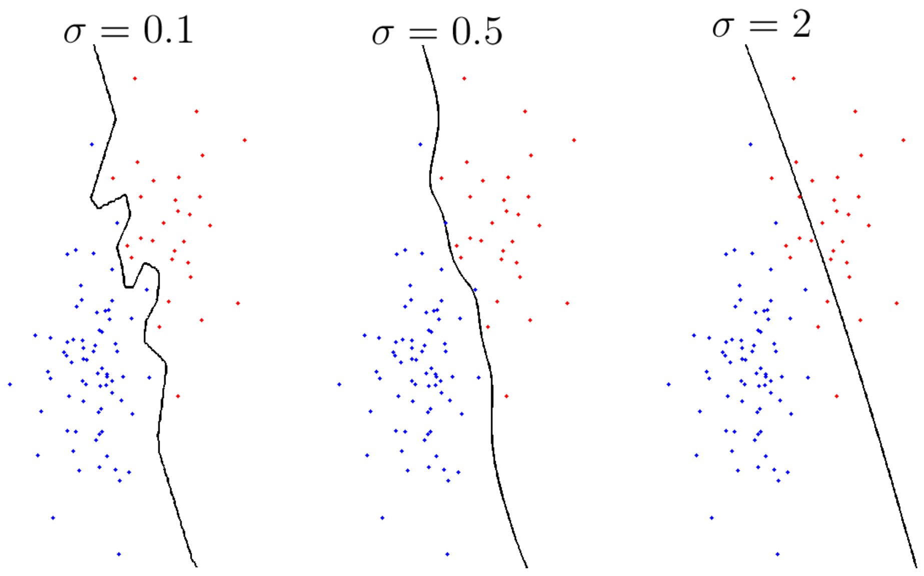


Figure 2.7: Decision boundary with a Parzen window PDF estimate for different kernel sizes.

ability estimates. As such, different choices of σ was tried on the synthetic dataset to demonstrate the different possible behaviors. Letting σ take on the values of 0.1, 0.5 and 2, produced the decision boundaries seen in Figure 2.7. It is here seen that a small choice (corresponding to a narrow kernel and consequently little averaging) yields a decision line with high variance, prone to over-fitting. Increasing σ, the line gets smoother as more and more averaging over the data is performed.

It should be noted that in the parametric setup of Bayes Decision theory, the discrimination lines are limited to straight, closed ellipses or hyperbolas. When replacing the multivariate Gaussian models with non-parametric estimates, the lines can take on any form depending on the dataset used for training. This typically makes non parametric estimates better classifiers for irregular data.

As with Parzen windowing, the K Nearest Neighbor estimate requires no assumptions of the underlying model the data has been generated from, only the data itself and the parameter k is relevant in determining the decision boundary. In Figure 2.8, k is varied from 1, to 3 and 10.

Choosing $k = 1$ leads to a decision boundary with a very high variance and the dataset being over-fitted. Even a region close to the mean of the red class will be classified incorrectly because of a single outlier from the blue class. With increasing k, the variance is reduced, but the decision line can

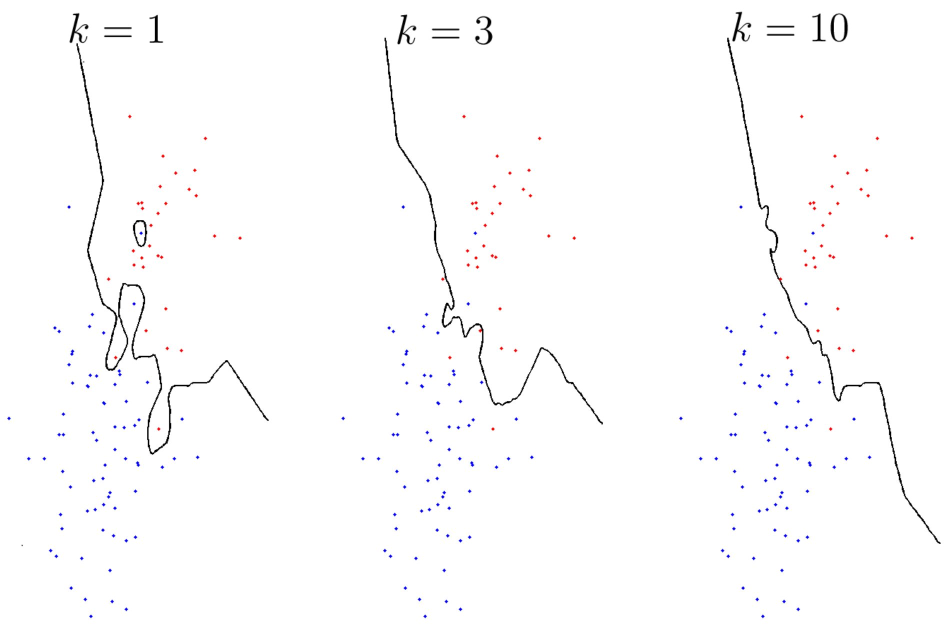


Figure 2.8: Decision boundary with a K Nearest Neighbor PDF estimate for different choices of k.

still be seen to be quite dependent on the data.

Lastly in this section, the convergence properties of the different decision boundaries are investigated. As all the probability density estimators are unbiased and converge to the correct distribution under the assumptions stated in their derivations, it should be possible to obtain close to the same classification line for each of the methods.

Recall from 2.2 that the important behavior of the Parzen window estimate was that $V_n \to 0$, while at the same time $nV_n \to \infty$ (with some restrictions placed on the kernel function $\kappa(\cdot)$). For the K Nearest Neighbor estimate it was noted that $k_n \to \infty$ and $k_n/n \to 0$ as $N \to \infty$ proved the estimator to be consistent. The parametric method discussed in Appendix A estimates, under the assumptions of a Gaussian mixture, its parameters ($\boldsymbol{\mu}_1$,$\boldsymbol{\mu}_2$,$\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$) correctly as $N \to \infty$.

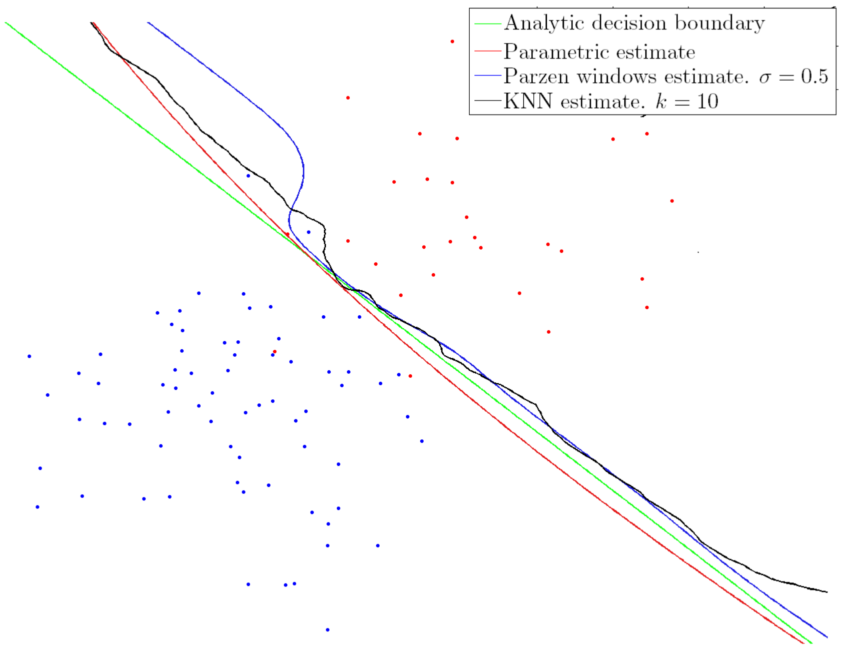
First 100 datapoints are generated from the bivariate Gaussian mixture and the decision line is estimated using the methods of the previous section. Then, 100 000 datapoints[4] are generated from the same model and the same methods are used to produce new decision boundaries. As N is increased, V_n and k_n of the parametric methods are also changed to follow the pattern

[4]Simulating N at ∞

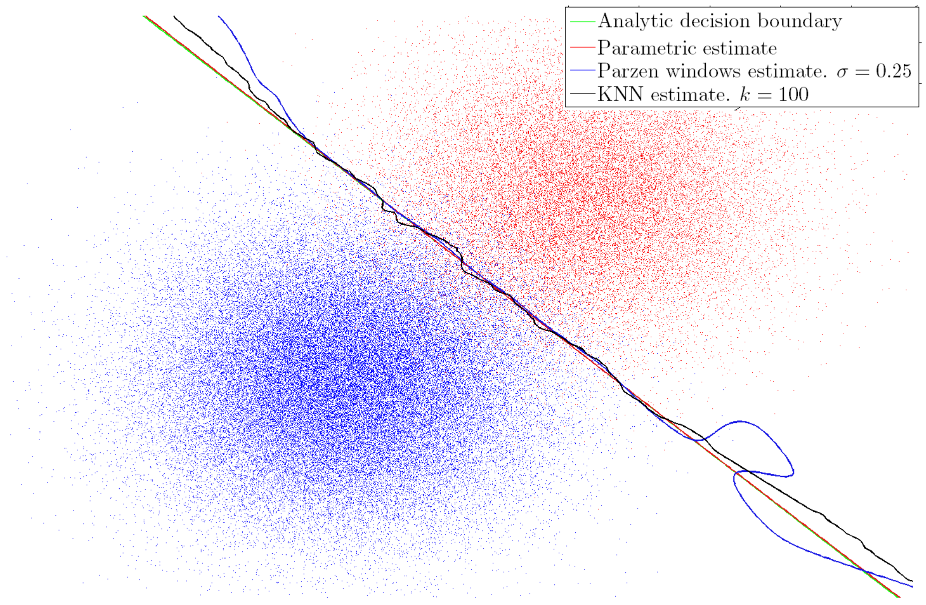
required for consistent estimators.

The resulting decision boundaries are shown in Figure 2.9. In this figure, it could be noted at all the methods converge towards the analytical solution as N grows. With $N = 100\ 000$, the parametric methods estimates is off by only about 0.1%. The non parametric methods are also seen moving closer to the analytical solution. However, they have problems when moving out of the region populated with datapoints. Indeed, especially the Parzen window estimate exhibits high variance a small distance away from the datapoints. This, of course, has to do with there being very few points close by relative to the shrinking kernel ($V_n \to 0$) size. The KNN estimate can be seen to perform somewhat better under these conditions. This could be due to an overall better choice of its parameter k. It could also be that this method is more adaptive in regions of few datapoints (increasing its volume until k datapoints are enclosed no matter what)[5]. However, discussing the behavior of the non parametric approaches outside of the regions populated by data can be futile. The take away from this example should instead be the convergence towards the same, optimal, desicion bondary (in the region of interest) for all estimators, given enough data points.

[5]As producing the data rich decision boundaries was computational expensive, not a lot of different choices for the parameters was tried.



(a) $N = 100$ datapoints.



(b) $N = 100\ 000$ datapoints.

Figure 2.9: Decision boundaries made using the different methods of probability density estimation with for varying N.

2.3 Kernel Methods

In this section, some of the fundamentals related to kernel methods will be introduced. It will be shown how a simple idea of a Mercer kernel function, which implicitly calculate inner products in some unknown space, can be used extend the functionality for many of the established algorithms in the field. Using inner products in a unknown space between datapoints, these methods turn out to be inherently different from similar KNN-approaches. This is due to the fact that all the datapoints are implicitly mapped to a new domain, where inner products can be calculated. KNN methods are only interested in some local neighborhood of each datapoint in the original domain.

In the field of machine learning, many of the well established and well understood methods were, for a long time, only so called linear methods. Techniques and methods such has Principal Component Analysis (PCA), Support Vector Machines (SVMs), Ridge Regression, Fisher Discriminant Analysis (FDA) and Canonical Correlation Analysis (CCA) could here be mentioned [40]. The fact that these methods are regarded as linear, refers to that they all have intrinsic connections with linear hyper-surfaces and perform poorly when presented data with prominent non-linear structure.

The method for making these techniques non-linear, involves the so called *kernel trick*, where the data is mapped to a new (possibly much higher dimensional) feature space F where the linear methods can be used to produce non-linear results in the original space. In this chapter, the theoretic basis which guarantees the validity of this mapping idea is discussed and an example of an application is shown.

In Appendix C, the reader will find another example involving classification where the Support Vector Machine (SVM) has been made non-linear by the methods discussed in this section. To keep the theory more succinct, it has been omitted from the main text.

2.3.1 The Kernel Trick

The basis for all the techniques being made non-linear is based on performing a non-linear mapping, via a mapping function here denoted $\Phi(\cdot)$, to a new space where the problem (hopefully) reduces to one which a linear method can solve. Given a dataset $\boldsymbol{X}$ of N points, $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\} \in \mathbb{R}^d$, the nonlinear mapping to the feature space F is defined as

$$\begin{aligned} \Phi: \quad & X && \rightarrow F \\ & \boldsymbol{x} && \mapsto \Phi(\boldsymbol{x}) \end{aligned} \tag{2.30}$$

with $F \in \mathbb{R}^m$, $m \geq d$.

Now, a general mapping of the data to a higher (possibly infinite) dimensional space, would in most cases lead to problems in choice of mapping and in handling the data once the mapping has been done. Fortunately, in certain feature spaces (with corresponding mappings $\Phi(\cdot)$), it is possible to compute inner products, and it is this knowledge the kernel trick is based around. A kernel function $\kappa(\cdot,\cdot)$ is defined as a function which maps data from $\mathbb{R}^d$ to the unknown feature space F and calculates inner products there

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle \tag{2.31}$$

Mercers theorem states the requirements on $\kappa(\cdot,\cdot)$ which guarantee that it calculates inner product in some unknown space [39].

Theorem 1 Mercers's Theorem: *Given an* $\boldsymbol{x} \in \mathbb{R}^d$ *and a mapping* Φ *such that*

$$\boldsymbol{x} \mapsto \Phi(\boldsymbol{x}) \in F$$

with an equivalent the inner product definition

$$\langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{z}) \rangle = \kappa(\boldsymbol{x}, \boldsymbol{z}) \tag{2.32}$$

where $\langle \cdot,\cdot \rangle$ *defines an inner product in* F. *Then,* $\kappa(\boldsymbol{x}, \boldsymbol{z})$ *is a symmetric continuous function satisfying*

$$\int_C \int_C \kappa(\boldsymbol{x}, \boldsymbol{z}) g(\boldsymbol{x}) g(\boldsymbol{z}) d\boldsymbol{x} d\boldsymbol{z} \geq 0 \tag{2.33}$$

for all $\boldsymbol{x}, \boldsymbol{z} \in C \subset \mathbb{R}^d$ *and for any function* $g(\cdot)$ *such that*

$$\int_C g^2(\boldsymbol{x}) d\boldsymbol{x} < \infty \tag{2.34}$$

Now, the opposite is always true; given a function $\kappa(\cdot,\cdot)$ *satisfying (2.33) and (2.34), there always exists some space where* $\kappa(\cdot,\cdot)$ *defines an inner product.*

It could here be noted that for a kernel obaying the requirements of Theorem 1, the corresponding feature space F is a so-called Hilbert space[6]. This is a result of *Moore-Aronszajn's Theorem* [45], which states that all valid choices of $\kappa(\cdot,\cdot)$ (with respect to (2.33) and (2.34)) reproduce a Hilbert space. Because of this, the feature space that machine learning algorithms implicitly use when the kernel trick is applied is often called a Reproducing Kernel Hilbert space (RPKH) [54].

[6]A complete linear space equipped with an inner product operations [54].

To reiterate, according to the theory of RPKHs and their corresponding kernel functions, it is possible to calculate inner products in some unknown high dimensional space. This means that [40][48]: *every (linear) algorithm that only uses scalar products can implicitly be executed in F by using kernels, i.e., one can very elegantly construct a nonlinear version of a linear algorithm.* This is also true for algorithms which operates on similarity measures, which produce positive definite matrices[7]. In the next section, this simple philosophy is applied to a linear algorithm to produce its nonlinear kernel counterpart.

Given a dataset $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$ of vectors $\mathbb{R}^d$, and a kernel function $\kappa(\cdot,\cdot)$, calculating inner products in F, the kernel matrix $\boldsymbol{K}$ for the dataset is defined as all the cross inner products. $\boldsymbol{K}$ will then be an $N \times N$ matrix where each element $K_{i,j}$ is given by

$$K_{i,j} := \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) \tag{2.35}$$

Assuming a valid kernel function is given, $\boldsymbol{K}$ will be a positive semidefinite symmetric Gramian matrix [48].

Many kernels can be chosen for which the requirements of Theorem 1 is fulfilled. In the field of machine learning and pattern recognition, the most commonly used is the Gaussian kernel [54]

$$\kappa_\sigma(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{z}||^2}{2\sigma^2}\right) \tag{2.36}$$

where σ is a width parameter (equivalent to that in 2.2.2) which can be varied.

The name *kernel*, refers to a class of functions which are used in all these methods, either to measure similarity between points, estimate densities or do an implicit mapping of the data to a new domain. Many more kernels could be mentioned, but as only (2.36) will be used in this text, the reader is instead referred to [54] for examples.

With the theory here presented, it can now be understood why the Parzen window estimate (2.11) in Section 2.2.2 could be called a kernel method. Approximating the density in $\boldsymbol{x}$, can be viewed as the mean of the inner products between $\boldsymbol{x}$ and the other datapoints in some unknown space F.

2.3.2 Kernel PCA

The goal of Principal Component Analysis (PCA) is to find a new, more meaningful way to express a given dataset $X : \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\} \in \mathbb{R}^d$. This

[7]The definition of a positive definite matrix $\boldsymbol{K}$ is: $\forall\ \boldsymbol{x} \neq \boldsymbol{0},\ \boldsymbol{x}^T\boldsymbol{K}\boldsymbol{x} > 0$.

is done by finding a new set of orthonormal bases with corresponding weight parameters $\{(\boldsymbol{v}_1, \lambda_1), (\boldsymbol{v}_2, \lambda_2), ..., (\boldsymbol{v}_d, \lambda_d)\}$ which span $\mathbb{R}^d$. With PCA, the goal is to capture as much of the variance of the data along the first principal components $\boldsymbol{v}_i, i = 1, 2, ..., d$. In this new basis, it is then easy to do meaningful dimensionality reduction by simply projecting onto a chosen number of basis vectors[8] to obtain a new representation of the data whilst keeping as much of the variance as possible [57][54]. This choice of basis construction is equivalent to minimizing the projection error in the sum squared sense [9] as the variance is a measure of sum squared variation. Often this projection error can in fact help denoise the data, as typical interference on a signal is non-correlated and as such, described by the less important (end) part of the basis set [40].

Finding this new basis which best describes the variance of the dataset comes from solving the eigenvalue problem on the sample covariance matrix $\boldsymbol{C}$ of the data. If a centered dataset X is provided such that $\sum_{i=1}^{N} \boldsymbol{x}_i = \boldsymbol{0}$, the sample covariance matrix is given by

$$\boldsymbol{C} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T \tag{2.37}$$

The eigenvalue problem for $\boldsymbol{C}$ is defined as

$$\lambda \boldsymbol{v} = \boldsymbol{C} \lambda \tag{2.38}$$

where inserting the definition for $\boldsymbol{C}$ yields

$$\lambda \boldsymbol{v} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T \boldsymbol{v} = \frac{1}{N} \sum_{i=1}^{N} \langle \boldsymbol{x}_i, \boldsymbol{v} \rangle \, \boldsymbol{x}_i \tag{2.39}$$

From this it can be seen that all solutions of $\boldsymbol{v}$ must lie in the span of $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N$[9].

To do PCA in the feature space, the problem now becomes one of solving the eigenvalue problem on the sample correlation matrix of the mapped data $\Phi(\boldsymbol{x}_i),\ i = 1, 2, ..., N$. An expression for the correlation matrix in F is obtained by performing the (unknown) mapping $\Phi(\cdot)$ on each datapoint (where again, centered feature space data has been assumed)

$$\boldsymbol{C}_F = \frac{1}{N} \sum_{i=1}^{N} \Phi(\boldsymbol{x}_i) \Phi(\boldsymbol{x}_i)^T \tag{2.40}$$

[8]This is especially easy as the set of basis vectors is orthonormal.

[9]A corollary of this is that if $N<d$, $\boldsymbol{C}$ will be singular.

which gives the eigenvalue problem

$$\lambda \boldsymbol{V} = \frac{1}{N}\sum_{i=1}^{N}\Phi(\boldsymbol{x}_i)\Phi(\boldsymbol{x}_i)^T\boldsymbol{V} = \frac{1}{N}\sum_{i=1}^{N}\left\langle \Phi(\boldsymbol{x}_i), \boldsymbol{V}\right\rangle \Phi(\boldsymbol{x}_i) \tag{2.41}$$

As for regular PCA, it is here seen that any $\boldsymbol{V}$ must lie in the span of $\Phi(\boldsymbol{x}_1), \Phi(\boldsymbol{x}_2), ..., \Phi(\boldsymbol{x}_N)$. This has useful consequences in this context. One of them being that

$$\lambda\Phi(\boldsymbol{x}_k)\boldsymbol{V} = \Phi(\boldsymbol{x}_k)\boldsymbol{C}_F\boldsymbol{V} \ \forall\ k = 1, 2, ..., N \tag{2.42}$$

Next, any $\boldsymbol{V}$ can be expressed with the given mapped data as a basis, i.e. there exists a set of scalars $\alpha_1, \alpha_2, ..., \alpha_N$ such that

$$\boldsymbol{V} = \sum_{i=1}^{N}\alpha_i\Phi(\boldsymbol{x}_i) \tag{2.43}$$

Going from (2.41), it is observed that

$$\boldsymbol{V} = \frac{1}{\lambda N}\sum_{i=1}^{N}\left\langle \Phi(\boldsymbol{x}_i), \boldsymbol{V}\right\rangle \Phi(\boldsymbol{x}_i) = \sum_{i=1}^{N}\alpha_i\Phi(\boldsymbol{x}_i) \tag{2.44}$$

which means that the coefficients on the mapped data points which span an eigenvector of $\boldsymbol{C}_F$ can be found by evaluating some inner product in F. Also note that the scalar coefficients from this inner product and the term $1/\lambda N$ is absorbed in the α's.

Inserting (2.44) into (2.42) and using the fact that $\langle\cdot,\cdot\rangle$ is a scalar (i.e. it can be moved freely with regards to matrices and vectors), the following expression is obtained

$$\lambda\sum_{i=1}^{N}\alpha_i\Phi(\boldsymbol{x}_k)^T\Phi(\boldsymbol{x}_i) = \Phi(\boldsymbol{x}_k)^T\frac{1}{N}\sum_{i=1}^{N}\Phi(\boldsymbol{x}_i)\Phi(\boldsymbol{x}_i)^T\sum_{j=1}^{N}\alpha_j\Phi(\boldsymbol{x}_j) \tag{2.45}$$

where k can take on the values $1, 2, ..., N$.

Evaluating the left hand side of (2.45) for all k, the set of equations obtained can be written as

$$\lambda\sum_{i=1}^{N}\alpha_i\Phi(\boldsymbol{x}_k)^T\Phi(\boldsymbol{x}_i) \overset{\forall k}{=} \lambda\boldsymbol{K}\boldsymbol{\alpha} \tag{2.46}$$

where $\boldsymbol{K}$ is $N \times N$ matrix of the inner products terms and $\boldsymbol{\alpha}$ is a $N \times 1$ column vector of the α_i terms:

$$\boldsymbol{K} = \begin{bmatrix} \Phi(\boldsymbol{x}_1)^T\Phi(\boldsymbol{x}_1) & \Phi(\boldsymbol{x}_1)^T\Phi(\boldsymbol{x}_2) & \cdots & \Phi(\boldsymbol{x}_1)^T\Phi(\boldsymbol{x}_N) \\ \Phi(\boldsymbol{x}_2)^T\Phi(\boldsymbol{x}_1) & & & \\ & \vdots & \ddots & \vdots \\ \Phi(\boldsymbol{x}_N)^T\Phi(\boldsymbol{x}_1) & & \cdots & \Phi(\boldsymbol{x}_N)^T\Phi(\boldsymbol{x}_N) \end{bmatrix}, \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix}$$

With these definitions, the right hand side can be evaulated as

$$\begin{aligned} \Phi(\boldsymbol{x}_k)^T \frac{1}{N}\sum_{i=1}^{N}\Phi(\boldsymbol{x}_i)\Phi(\boldsymbol{x}_i)^T\sum_{j=1}^{N}\alpha_j\Phi(\boldsymbol{x}_j) &= \\ \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}\Phi(\boldsymbol{x}_k)^T\Phi(\boldsymbol{x}_i)\Phi(\boldsymbol{x}_i)^T\Phi(\boldsymbol{x}_j)\alpha_j &= \\ \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N}\langle\Phi(\boldsymbol{x}_k),\Phi(\boldsymbol{x}_i)\rangle\langle\Phi(\boldsymbol{x}_i),\Phi(\boldsymbol{x}_j)\rangle\alpha_j &\overset{\forall k}{=} \frac{1}{N}\boldsymbol{K}^2\boldsymbol{\alpha} \end{aligned} \tag{2.47}$$

Now, from (2.46) and (2.47), (2.45) can be reformulated as

$$N\lambda\boldsymbol{\alpha} = \lambda_F\boldsymbol{\alpha} = \boldsymbol{K}\boldsymbol{\alpha} \tag{2.48}$$

where $\lambda_F = N\lambda$ is an eigenvalue of the kernel matrix $\boldsymbol{K}$. Solving the eigenvalue problem for the kernel matrix in (2.48) yields the α-coefficients sought in the expression for $\boldsymbol{V}$, (2.44), which is related to the original datapoints projection onto one of the principal axes in F.

As $\boldsymbol{V}$ is a principal axes of the feature space F for a given $\boldsymbol{\alpha}_k$ (the k-th eigenvector of (2.48)), it is normalized to be of length 1:

$$\begin{aligned} \boldsymbol{V}^T\boldsymbol{V} &= \left(\sum_{i=1}^{N}\alpha_{i,k}\Phi(\boldsymbol{x}_i)\right)^T\left(\sum_{j=1}^{N}\alpha_{j,k}\Phi(\boldsymbol{x}_j)\right) \\ &= (\boldsymbol{\alpha}_k)^T\boldsymbol{K}\boldsymbol{\alpha}_k = \lambda_k\underbrace{(\boldsymbol{\alpha}_k)^T\boldsymbol{\alpha}_k}_{=1} \\ &= \lambda_k \end{aligned}$$

From which it follows that the principal axis k in F is defined as

$$\boldsymbol{V} = \frac{1}{\sqrt{\lambda_k}}\sum_{i=1}^{N}\alpha_{i,k}\Phi(\boldsymbol{x}_i) \tag{2.49}$$

A new point $\boldsymbol{x} \in \mathbb{R}^d$, its image in F has a projection onto the k-th principal axes defined as

$$\begin{aligned}(\boldsymbol{V}_k)^T \Phi(\boldsymbol{x}) &= \left(\frac{1}{\sqrt{\lambda_k}} \sum_{i=1}^{N} \alpha_{i,k} \Phi(\boldsymbol{x}_i) \right)^T \Phi(\boldsymbol{x}) \\ &= \frac{1}{\sqrt{\lambda_k}} \sum_{i=1}^{N} \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}) \rangle \\ &= \frac{1}{\sqrt{\lambda_k}} \sum_{i=1}^{N} k(\boldsymbol{x}_i, \boldsymbol{x}) \end{aligned} \tag{2.50}$$

Given that the projection on the principal axes of F for the data points used in construction of $\boldsymbol{K}$ is sought, the following result follow

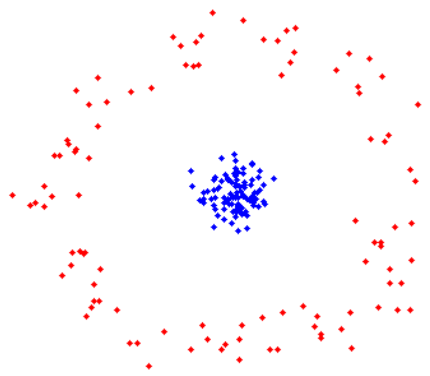
$$\begin{aligned}\boldsymbol{V}_k^T \Phi(\boldsymbol{x}_j) &= \frac{1}{\sqrt{\lambda_k}} \sum_{i=1}^{N} \alpha_{i,k} \Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}_j) \\ &\overset{\forall j}{=} \frac{1}{\sqrt{\lambda_k}} \sum_{i=1}^{N} \alpha_{i,k} \boldsymbol{K} \\ &= \frac{1}{\sqrt{\lambda_k}} \boldsymbol{\alpha}_k^T \boldsymbol{\alpha}_k \lambda_k \boldsymbol{\alpha}_k^T \\ &= \sqrt{\lambda_k} \boldsymbol{\alpha}_k^T \end{aligned} \tag{2.51}$$

From which it is seen that the projection of the original data onto a principal axes of F is equivalent to a scaled version of the given eigenvector of $\boldsymbol{K}$.

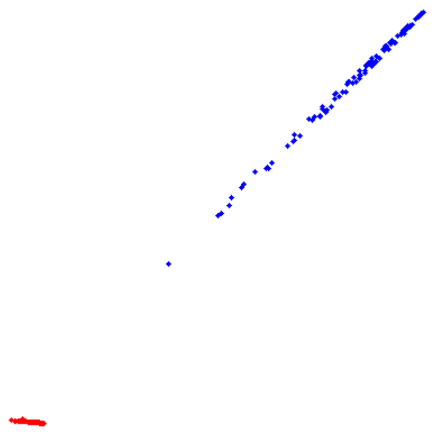
Examples

Given datasets of circular, or spherical symmetry, regular PCA will perform poorly due to the covariance matrix being invariant of spacial rotation [57]. Applying the kernel trick and performing PCA in the feature domain F instead can improve the results. In Figure 2.10, two dataset exhibiting rotational symmetry around the origin is given, along with their projections onto the principal axes of F. The coloring applied here is for visualization only, and all the datapoints are treated equally by the Kernel PCA algorithm.

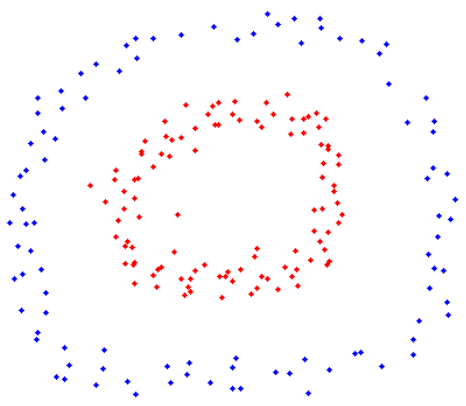
In this case, obviously, no dimensionality reduction is done. However, the data is transformed to a new representation where it becomes linearly separable. Note that linear separability also holds if dimensionality reduction was done and the data projected onto the horizontal axis only (the first principal component).



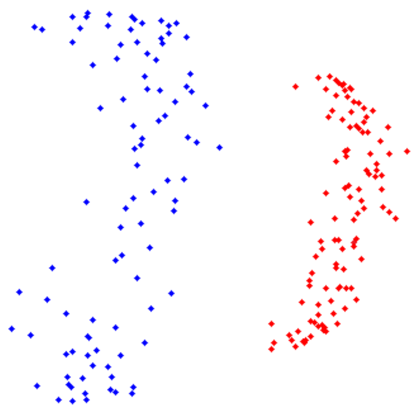
(a) Set 1: Original data



(b) Set 1: Mapping onto principal axes of F



(c) Set 2: Original data



(d) Set 2: Mapping onto principal axes of F

Figure 2.10: Toy datasets with mapping to principal axes of F. It is observed that the data becomes linearly separable upon transformation.

2.4 KNN Methods

This section will discuss methods more exclusive to the K Nearest Neighbor approach. As mentioned in the introduction, this approach is focused on the local structure around each datapoint and is with that more adaptive to changes in the data on a global scale.

The section will introduce some mathematical constructions for evaluating local neighborhoods and show practical examples of the use of them. It will be shown how these measures can be used to do non-linear dimensionality reduction and clustering.

2.4.1 Laplacian Eigenmaps

Using KNN-considerations, dimensionality reduction and clustering is here investigated. This will be done in the form of analysis of a so-called *Laplacian matrix*. The idea of this technique is that, given a dataset in $\mathbb{R}^d$, it is assumed that this dataset lies on a smooth manifold, M, embedded in $\mathbb{R}^d$: $M \subset \mathbb{R}^d$ with a dimension $m < d$. A new representation of the data with local neighborhood properties preserved is sought such that the geometric structure of M in $\mathbb{R}^d$ is unraveled and disregarded, thereby performing dimensionality reduction. By indirectly evaluating this new representation, via the *Laplacian matrix*, it is also possible to do non-linear clustering by minimizing some function which defines the cost of splitting the dataset.

To achieve this non-linear dimensionality reduction and clustering, a non-directed weighted graph will be constructed based on the dataset given. The properties of this graph can be expressed in terms of matrices, and upon analysis of these, the goals are reached. As it is most common to construct these graphs evaluating the nearest neighbors of the datapoints given, one could argue that these algorithms fall under the category of methods related to k-nearest neighbors and are different from the kernel methods discussed previously.

Constructing the Graph

For a dataset of N points, $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\} \in \mathbb{R}^d$, let $G = (\boldsymbol{V}, \boldsymbol{E})$ define a undirected graph constructed by the set. Each data point correspond to a vertex in the graph G, giving a set of N vertices $V = \{v_1, ..., v_N\}$. Furthermore, define all *connected* points to have edges, stored in $\boldsymbol{E}$, between them. These edges can then be weighted with regards to some similarity measure between the connected vertices. A common similarity measure is the Gaussian kernel, which in evaluating vertex i and j is defined as

$$w(i,j) = \exp\left\{-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}\right\} \tag{2.52}$$

Setting up the connection between the vertices of the graph should be done in such a way that neighborhoods relationships are captured. In [36], three possible choices are suggested

1. **The ϵ-neighborhood graph**: Given some metric measuring similarity in $\mathbb{R}^d$, all points with a pairwise distance lower than a choosen ϵ are said to be neighbors and an edge between their vertices is set up.

2. **The k-nearest neighbor graph:** In this setup, the goal is to connect vertex v_i with v_j if it is one of its k nearest neighbors. A problem with this is that it leads to a directed graph, as v_j not necessarily have v_i as one of its k nearest neighbors. One way of dealing with this, is to construct the graph such that there is an edge between v_i and v_j, e_{ij}, if one of the vertices has the other as one of its k nearest neighbors. Another way, is to only insert an edge if both vertices has the other among is k nearest neighbors.

3. **The fully connected graph:** Here, all the vertices are connected with edges regardless of their neighborhood properties. The weighting of the edges applied afterwards is used to capture the similarities between each vertex. This method is generally not recommended, as connecting each vertex leads to more demanding computations (without added value) later in the process [36].

Next, the connections between points are weighted with a similarity metric such that points which are pairwise close to each other in $\mathbb{R}^d$ are assigned a high weight to their edge relative to pairwise points further away from each other. This weighting between the N points can then be represented in a $N \times N$ matrix $\boldsymbol{W}$. The elements of which, $w(i,j),\ i,j = 1,2,...,N$, correspond to the weight on the edges e_{ij}, with $w(i,j) = 0$ if v_i and v_j are not connected. Note that since the similarity measure should be a metric, $\boldsymbol{W}$ will be symmetric.

It is interesting to note that (2.52) is exactly the kernel function $\kappa_\sigma(\boldsymbol{x}_i, \boldsymbol{x}_j)$ used in 2.3. With this choice of kernel and using method 3 for constructing the graph, the matrix $\boldsymbol{W}$ becomes exactly the $\boldsymbol{K}$ matrix of the previous chapter; implicitly expressing inner products in some other space. The merging of the two methods in this limiting case comes as no surprise as they are intrinsically related. However, method 3 above does not fit with this thesis

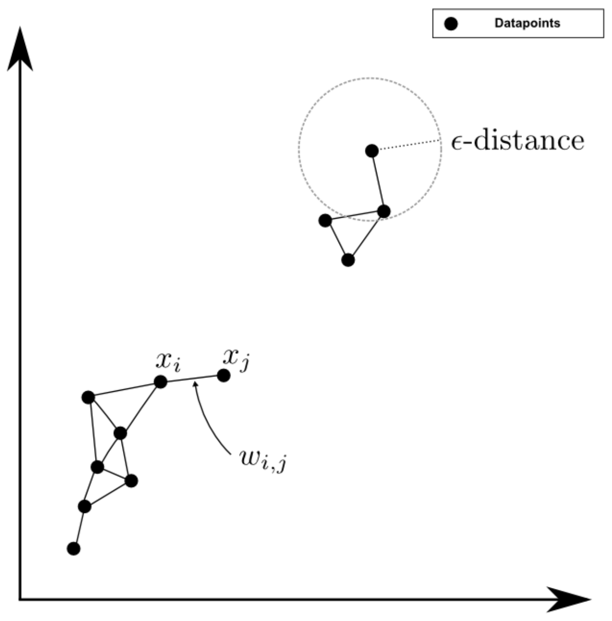


Figure 2.11: Example of a non-connected graph produced by ϵ-distance setup.

definition of a KNN-consideration and is mentioned only to highlight this connection to the kernel methods.

For illustration, a graph constructed from datapoints in $\mathbb{R}^2$ with the ϵ-*neighborhood* approach from above is shown in Figure 2.11. The result of this is an unconnected graph where only points within a distance ϵ are connected and weighted by some $w(i,j)$. In this figure, the equidistant ϵ from a point is seen to be a circle, meaning that an Euclidean 2-norm has been used in this example (see 2.2.3 for more details).

The Laplacian Matrix

In order to introduce the Laplacian matrix, some preliminary notation is required. Given a set of vertices V, connected with edges E weighted by $\boldsymbol{W}$, a vertex v_i's degree d_i is the sum of the outbound weights

$$d_i = \sum_{j=1}^{N} w(i,j) \tag{2.53}$$

where the weight $w(i,j)$ is zero for j such that v_i and v_j are not connected. With this, it is possible do define the diagonal matrix $\boldsymbol{D}$ ($N \times N$), having the degree of each vertex along its diagonal and zeros elsewhere.

The unnormalized graph Laplacian $\boldsymbol{L}$ is now defined as

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W} \tag{2.54}$$

The literature on this subject also provide two alternative definitions of Laplacian matrices, the so-called normalized graph Laplacians [36]

$$L_{\text{sym}} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2} = I - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2} \tag{2.55}$$

$$L_{\text{rw}} = \boldsymbol{D}^{-1}\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{W} \tag{2.56}$$

The normalization here comes from the multiplicative dependency of the degree holding matrix $\boldsymbol{D}$. In Luxburg [36], it is noted that for algorithms relying on evaluating a Laplacian matrix, the normalized versions outperform the unnormalized one. For simplicity however, this chapter will derive its algorithms using the unnormalized Laplacian.

Dimensionality Reduction

An important property of the matrix $\boldsymbol{L}$ which makes it suitable for dimensionality reduction on the underlying dataset is the fact that given a vector $\boldsymbol{f} \in \mathbb{R}^N$ (same dimensionality as number of datapoints), the following holds

$$\begin{aligned}
\boldsymbol{f}'\boldsymbol{L}\boldsymbol{f} = \boldsymbol{f}'\boldsymbol{D}\boldsymbol{f} - \boldsymbol{f}'\boldsymbol{W}\boldsymbol{f} &= \sum_{i=1}^{N} d_i f_i^2 - \sum_{i,j=1}^{N} w(i,j)(f_i - f_j)^2 \\
&= \frac{1}{2}\left(\sum_{i=1}^{N} d_i f_i^2 - 2\sum_{i,j=1}^{N} f_i f_j w(i,j) + \sum_{j=1}^{N} d_j f_j^2\right) \\
&= \frac{1}{2}\sum_{i,j=1}^{N} w(i,j)(f_i - f_j)^2
\end{aligned} \tag{2.57}$$

Further more, it can be shown that the smallest eigenvalue of $\boldsymbol{L}$ is 0, and that this eigenvalues multiplicity is equal to the number of connected components in the graph.

Now, imagining that $\boldsymbol{f}$ is a 1-dimensional representation of the N datapoints (originally expressed in $\mathbb{R}^d$), it is seen in (2.57) that if neighborhoods should be preserved when reducing the dimensionality, $\boldsymbol{f}$ should be chosen such that the difference $f_i - f_j$ is small when $w(i,j)$ is large.

If $\boldsymbol{f}$ is set to be the eigenvector of $\boldsymbol{L}$ corresponding to the smallest eigenvector ($\lambda_1 = 0$), the following holds

$$\begin{aligned}
\lambda_1 \boldsymbol{f} &= \boldsymbol{L}\boldsymbol{f} \\
0 &= \boldsymbol{f}^T\boldsymbol{L}\boldsymbol{f} = \frac{1}{2}\sum_{i,j=1}^{N} w(i,j)(f_i - f_j)^2
\end{aligned}$$

Now, since by definition all $w(i,j)$ are non-negative, the only way for this equation to be zero, is $\boldsymbol{f}$ to be a constant vector. This solution, although trivial, hints of other (meaningful) solutions, evaluating eigenvectors of $\boldsymbol{L}$.

In Belkin and Niyogi [5] it was proposed that finding an optimal embedding of the data in a lower dimension could be solved by

$$\begin{gathered}\arg\min_{\boldsymbol{f}} \boldsymbol{f}^T \boldsymbol{L} \boldsymbol{f} \\ \text{such that } \boldsymbol{f}^T \boldsymbol{D} \boldsymbol{f} = 1\end{gathered} \tag{2.58}$$

Here, the constraints removes an arbitrary scaling of the embedded points. This minimization problem can be solved using Lagrange multipliers, which gives the eigenvalue/eigenvector problem

$$\boldsymbol{L}\boldsymbol{f} = \lambda \boldsymbol{D}\boldsymbol{f} \tag{2.59}$$

where the solutions lie in the eigenvectors of $\boldsymbol{D}^{-1}\boldsymbol{L}$. From this approach to the problem, it is easy to see the role the normalization of $\boldsymbol{L}$ plays; it is constraints imposed on the solution spaced with regards to scaling. In the solution above, the eigenvectors of $\boldsymbol{L}_{\text{rw}}$ is sought. Had the constraint been $\boldsymbol{f}^T \boldsymbol{D}^{1/2} \boldsymbol{D}^{1/2} \boldsymbol{f} = 1$ or $\boldsymbol{f}^T \boldsymbol{f} = 1$ instead, the optimization would reduce to evaluating the eigenvectors of the different Laplacian matrices.

If a given dataset of N points were to consist of q distinct, unconnected regions, the $\boldsymbol{L}$ matrix (of size $N \times N$) could, without loss of generality, be constructed as

$$\boldsymbol{L}_{1-q} = \begin{bmatrix} \boldsymbol{L}_1 & & & \\ & \boldsymbol{L}_2 & & \\ & & \ddots & \\ & & & \boldsymbol{L}_q \end{bmatrix}$$

where the elements outside the matrix blocks along the diagonal are all zero. As every $\boldsymbol{L}_1, \boldsymbol{L}_2, ..., \boldsymbol{L}_q$ are them selfs valid Laplacians, living in disjoint subspaces of $\mathbb{R}^N$, they all lead to eigenvalues of 0 for $\boldsymbol{L}_{1-q}$: $\lambda_1 = \lambda_2 = ... = \lambda_q = 0$. The corresponding eigenvectors will then be indicator vectors in $\mathbb{R}^N$ taking some constant value for all connected vertices forming the relevant sub-laplacian, and zero for all other vertices. This fact hints of possible applications in clustering, which will be looked at in the next section.

Examples Given a dataset originally placed in $\mathbb{R}^3$, the graph is constructed by calculating all the distances between the points and creating edges between those within a distance epsilon from each other. These edges are then

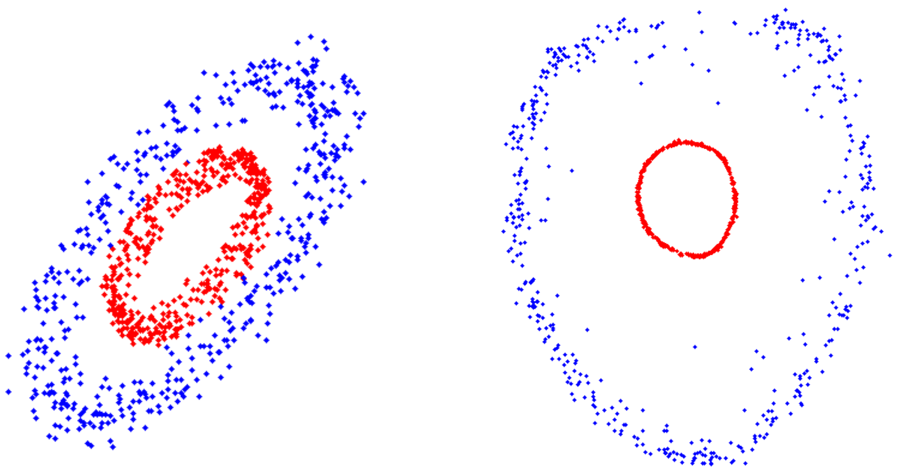

(a) Original data in $\mathbb{R}^3$ (b) Optimal embedding onto $\mathbb{R}^2$

Figure 2.12: Original dataset as two skewed cylinders in $\mathbb{R}^3$ and its optimal embedding onto $\mathbb{R}^2$ using the non trivial solutions of the laplacian $\boldsymbol{L}$ ($\epsilon = 1$, $\sigma = 5$). Colors are for visualization only.

weighted using the function

$$w(i,j) = \exp\left(\frac{-d_{i,j}^2}{\sigma^2}\right) \tag{2.60}$$

This creates the sparse affinity matrix $\boldsymbol{W}$ from which $\boldsymbol{D}$ and $\boldsymbol{L}$ is constructed. The eigenvectors of $\boldsymbol{L}$ then gives an embedding of the data onto M. Creating two rings, with added gaussian noise, placed skewed in $\mathbb{R}^3$, yields the results visualized in Figure 2.12.

A slightly harder test example is shown in Figure 2.13, where a non-linearity has been introduced by bending the skew cylinders in space. The result of evaluating the eigenvectors of $\boldsymbol{L}_{\text{rw}}$ is seen to separate the two structures nicely, although the intuitive form in recovered in Figure 2.12 is not seen. Nevertheless, using this method for dimensionality reduction would greatly help subsequent algorithms handle the data.

Spectral Clustering using Graph Cuts

As was noted in the previous section, the eigenvectors of the Laplacians *can* give knowledge about possible clusters in the data directly. However, for this to apply, the user must choose the correct parameters (like the ϵ-distance) for meaningful clusters to appear in the affinity matrix . A way around this

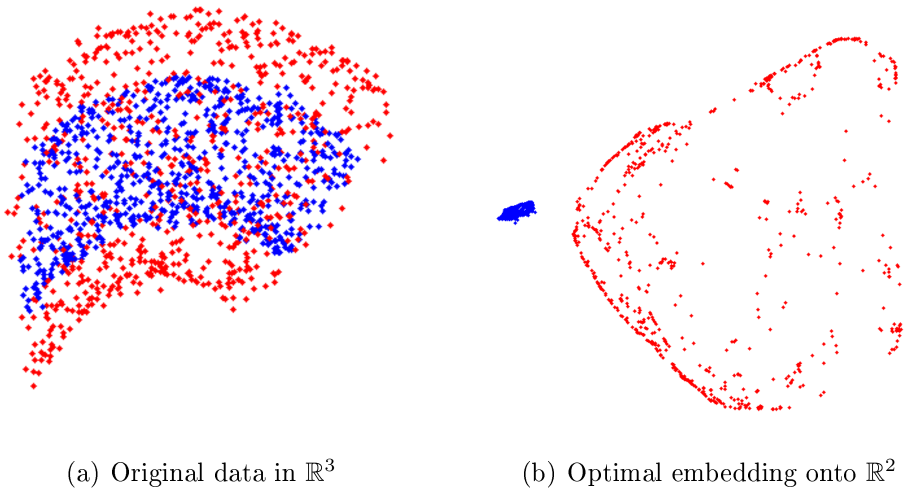

(a) Original data in $\mathbb{R}^3$ (b) Optimal embedding onto $\mathbb{R}^2$

Figure 2.13: Original dataset as two skewed and bent cylinders in $\mathbb{R}^3$ and its optimal embedding onto $\mathbb{R}^2$ using the non trivial solutions of the normalized Laplaican $\boldsymbol{D}^{-1}\boldsymbol{L}$ ($\epsilon = 1.5$, $\sigma = 0.1$). Colors are for visualization only.

problem is to define an operation on the graph called a cut, and try to split the graph into pieces in a way which minimizes some cost function related to the splitting. It is assumed that the graph being worked on is connected so that none of the non-trivial eigenvectors (of the smallest eigenvalues) are just indicator vectors for some of the disjoint parts of the graph.

To derive the Spectral Clustering algorithm, some notation is required. Assuming a graph $G = (\boldsymbol{V}, \boldsymbol{E})$ is constructed with an associated Laplacian $\boldsymbol{L}$, an affinity matrix $\boldsymbol{W}$ and a diagonal matrix $\boldsymbol{D}$ containing the degree of each vertex, it is possible to split the graph into two sub-graphs A and B such that

$$A \cup B = G \quad \text{and} \quad A \cap B = \emptyset$$

The degree of a vertex is given in (2.53) and gives the relevance of that vertex. Next, given a subgraph A, its relevance can be measured by its *volume*. The volume of sub-graph A is defined as

$$Vol(A) = \sum_{i:v_i \in A} D(i,i) = \sum_{\substack{i\,:\,v_i \in A \\ j \in v_i}} w(i,j) \tag{2.61}$$

As proposed in [58], one way to split G into two sub-graphs A and B, is to try to minimize the *cut*, defined as

$$cut(A,B) = \sum_{i:v_i \in A,\ j:v_j \in B} w(i,j) \tag{2.62}$$

Minimizing this function would correspond to assigning the vertices into two groups such that the sum of the weights on the edges running between the two are minimized.

This choice of cost function is, as noted in [54], however not optimal as it tends to cut off single lone vertices; if a vertex only has one edge from itself to another vertex, the cost of making it the only vertex in B while the rest of the graph is assigned to A would, using (2.62) be very small. A remedy for this was suggested in [52], where the cut cost is normalized with respect to the volumes of each of the resulting sub-graphs. The *normalized cut* is defined as

$$Ncut(A,B) = \frac{cut(A,B)}{Vol(A)} + \frac{cut(A,B)}{Vol(B)} \tag{2.63}$$

Minimizing (2.63) would not tend to cut off lone vertices, as one of the resulting sub-graphs would then have a very small volume, making the cost of that particular cut high. Another choice of graph separation cost function is the *RatioCut* [18], which also normalizes with respect to sub-graph sizes. For simplicity, it will not be covered in this thesis.

So solve the task of minimizing the normalized cut, it is assumed that labels for the vertices in each sub-graph is assigned as follows

$$y_i = \begin{cases} \frac{1}{Vol(A)} & \text{if } v_i \in A \\ -\frac{1}{Vol(B)} & \text{if } v_i \in B \end{cases} \tag{2.64}$$

which, given a set of N vertices, results in a vector $\boldsymbol{y} \in \mathbb{R}^N$.

Now, by this definition of indicator values for each of the sub-graphs, the following result follows

$$\begin{aligned} \boldsymbol{y}^T \boldsymbol{W} \boldsymbol{y} &= \frac{1}{2} \sum_i \sum_j (y_i - y_j)^2 w(i,j) \\ &= \frac{1}{2} \sum_{i:v_i \in A} \sum_{j:v_j \in B} \left(\frac{1}{Vol(A)} + \frac{1}{Vol(B)} \right)^2 cut(A,B) \\ &\propto \left(\frac{1}{Vol(A)} + \frac{1}{Vol(B)} \right)^2 cut(A,B) \end{aligned} \tag{2.65}$$

where the proportionality comes from removing the multiplicity of the summation term with regards to vertices in each sub-graph.

With the definition in (2.64), the following equation also holds

$$\begin{aligned}\boldsymbol{y}^T\boldsymbol{D}\boldsymbol{y} &= \sum_{i:v_i\in A} y_i D(i,i) + \sum_{j:v_j\in B} y_j D(j,j) \\ &= \frac{1}{Vol(A)^2}Vol(A) + \frac{1}{Vol(B)^2}Vol(B) \\ &= \frac{1}{Vol(A)} + \frac{1}{Vol(B)} \qquad (2.66)\end{aligned}$$

Combining (2.65) and (2.66), it is seen that (2.63) can be replaced by

$$J = \frac{\boldsymbol{y}^T\boldsymbol{L}\boldsymbol{y}}{\boldsymbol{y}^T\boldsymbol{D}\boldsymbol{y}} \qquad (2.67)$$

and minimized.

By the definition of (2.64), the y_i elements are only allowed to take on two very specific values[10]. In order to solve the optimization, the problem is relaxed such that $\boldsymbol{y}$ can vary over all real values constrained by

$$\boldsymbol{y}^T\boldsymbol{D}\boldsymbol{y} = \mathbf{1} \qquad (2.68)$$

where $\mathbf{1}$ is a column vector of length N with only 1's. With this, (2.67) can then by optimized by introducing Lagrange multipliers on the constraints, obtaining the eigenvalue, eigenvector equation

$$\boldsymbol{L}\boldsymbol{y} = \lambda\boldsymbol{D}\boldsymbol{y} \qquad (2.69)$$

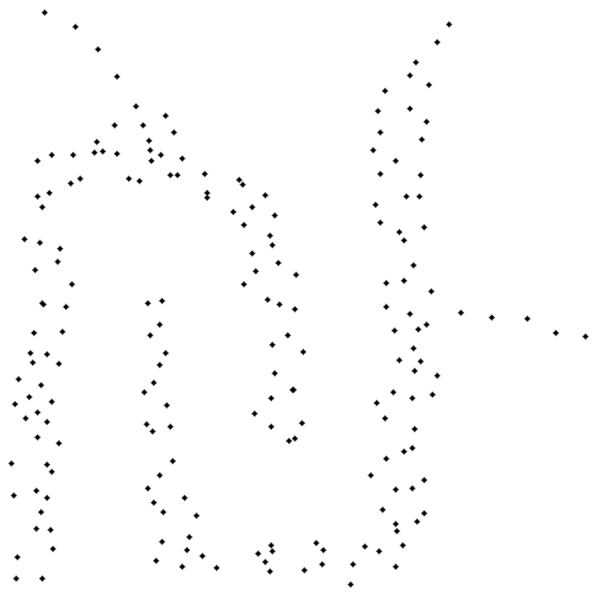
Since a minimum cost is sought, the eigenvector $\boldsymbol{y_i}$ of $\boldsymbol{D}^{-1}\boldsymbol{L}$ corresponding to the smallest eigenvalue λ_i should be chosen. However, if the graph is connected, it will have an eigenvalue λ_1 of 0 corresponding to a constant vector $\boldsymbol{y}_1$. Evaluation of $\boldsymbol{y}_1$ to obtain a cut is then impossible and instead the eigenvector associated with the second smallest eigenvalue is used. To avoid the ambiguity when it is unclear whether the graph is connected or not, is it generally constructed so that it is [54].

The cluster labels can now be set by thresholding the given eigenvector. Since it should mimic the true definition of $\boldsymbol{y}$, given in (2.64), a common choice here is clustering vertex v_i to A if $y_i \leq 0$ and to B otherwise.

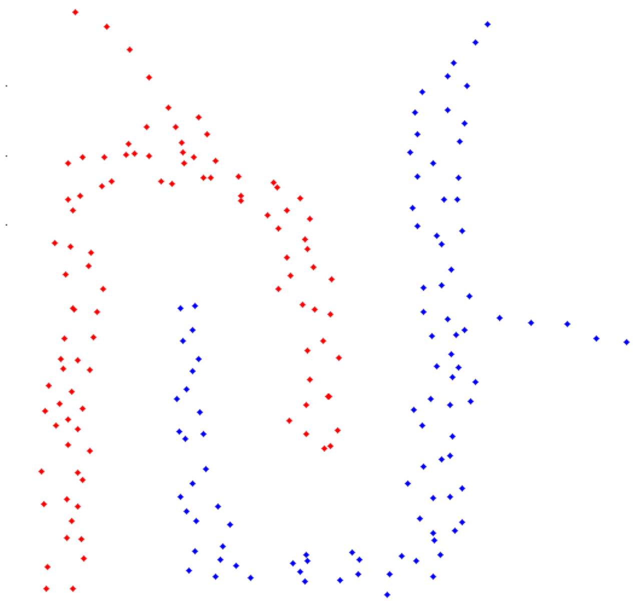
[10]These values also vary based on which cut is performed.

Example

A synthetic dataset of somewhat complicated structure is created in 2 dimensions to test the Laplacian method of clustering. The dataset can be seen in Figure 2.14(a). The distance matrix $\boldsymbol{D}$ is constructed using the Euclidean norm, and the connected nodes are determined using the ϵ-method with a distance threshold of 0.1. The weight on the edges between each node is calculated using (2.52) with a width parameter σ of 0.3. The Laplacian matrix $\boldsymbol{L}$ is then calculated and its eigenvalues and eigenvectors found. Thresholding out class assignments based on the elements along the second smallest eigenvector is done simply evaluating the sign. The result is the class labeling shown by different colors in Figure 2.14(b), which is very satisfactory.



(a) Original unlabeled data



(b) Result of graph based spectral clustering

Figure 2.14: Spectral clustering applied to non-linear dataset ($\epsilon = 0.1$ and $\sigma = 0.3$).

Chapter 3

Information Theoretic Learning

In the previous sections, some of the machine learning applications involving estimating the probability densities or connections between datapoints were given an overview. Over the last 10 years, a new direction which also rely on the probability densities has been explored; namely the field of information theoretic learning.

This new approach uses what is called the entropy and divergence measures to capture information about a dataset beyond the first and second order moments, which most novel techniques rely on. Being a functional of the PDFs themselfs, it relays information about *all* the statistics of the process [10]. A basic example of why this can be worth while is shown in Figure 3.1, where three probability densities with the same mean and the same variance is plotted. Clearly the three distributions look very different, but this remains unknown to the first and second order statistics.

Up until very recently, ITL was done exclusively by means of kernel methods, with the Parzen window estimate (discussed in 2.2.1) of the PDF playing an important role. In this chapter, it will be shown why this has been the case and how the Parzen estimate naturally shows up. The next chapter will then look at a state-of-the-art clustering routine where this estimate is used and extend the theory so that the adaptive KNN-approaches can be used instead.

The term entropy was first used by Shannon in 1948 discussing optimal communication over transmission lines [49]. In the original paper, Shannon developed his theory to evaluate transmission potential in presence of noise, also taking into account the statistical properties of the ensemble of messages being communicated. With that, he spawned of a whole new field of research still active to this day ranging from engineering of cables to design of adaptive machine learning systems. This wide range of uses for information theoretic considerations was apparent to Shannon from the outset and

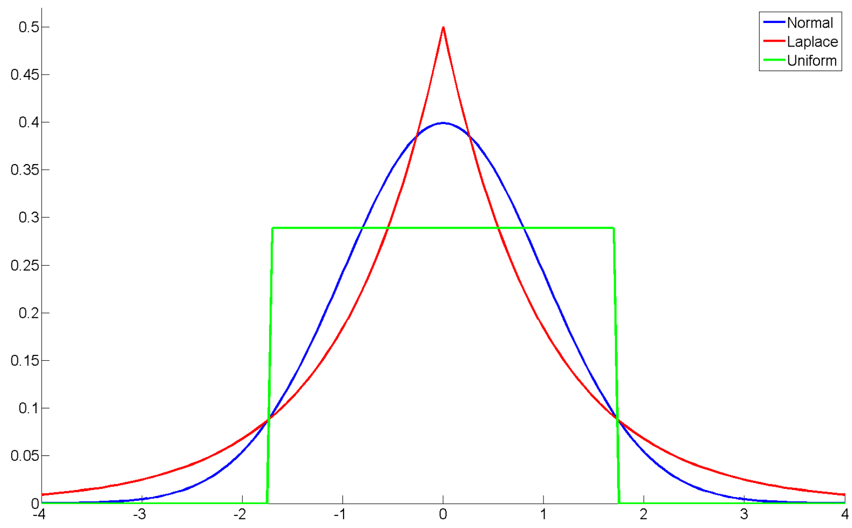


Figure 3.1: Three distributions with same mean and variance, yet very different shapes.

a quote by Sir Arthur Stanley Eddington from the book *The Mathematics of Communications* [50] of 1949 is here apt:

> Suppose that we were asked to arrange the following in two categories - distance, mass, electric force, entropy beauty, melody.
>
> I think there are the strongest grounds for placing entropy alongside beauty and melody, and not with the first three. Entropy is only found when parts are viewed in association, and it is by viewing and hearing the parts in association that beauty and melody are discerned. All three are features of arrangement. It is a pregnant thought that one of these three associates should be able to figure as a commonplace quantity of science. The reason why this stranger can pass itself off among the aborigines of the physical world is that it is able to speak their language, viz., the language of arithmetic.

It is interesting to note how well this quotation about entropy fits into our modern theory of machine learning and pattern recognition, talking about comparing sets of structures and extracting universal knowledge. The quote could serve as motivation moving forward in investigating and developing the theory.

In what follows, an introduction to the background is given. Firstly the Shannon entropy is introduced before moving on to the more applicable Renyi entropy. With this, the basic way of estimating the entropy is investigated before the different distance measures in information theoretic learning is given a review.

3.1 Entropy

Shannons original definition of entropy is given on the form

$$H_S(\boldsymbol{X}) = -\int p(\boldsymbol{x}) \log p(\boldsymbol{x}) d\boldsymbol{x} \tag{3.1}$$

in the continuous case and

$$H_S(\boldsymbol{X}) = -\sum_k p(\boldsymbol{x}_k) \log p(\boldsymbol{x}_k) \tag{3.2}$$

in the discrete case [49].

The quantity

$$I_k = -\log(p(\boldsymbol{x}_k)) \; = \; 1/\log(p(\boldsymbol{x}_k)) \tag{3.3}$$

refers to the information obtained from a specific event occurring as first described by Hartley [19] and later refined by Shannon in his seminal article of 1948 [49]. It should be noted that this *information* is completely described by the PDF of X in the form of inverse logarithmic proportionality.

In the case where $p(\boldsymbol{x}_k)$ becomes 1 (only one event in the probability space), the information of that event occurring becomes zero - seeing the event happen can not serve to explain any phenomenon as it always happens. The same balancing happens in the opposite case of an improbable event carrying much information. The entropy can with this be understood as a weighted mean of the information of a process with the weighting being done according to the probability of each event actually happening. The entropy acts as a descriptor of the Probability Mass Function (PMF), or the PDF, in such a way that a balance is struck between the information an event carries and the probability of that event occurring [45]. It captures the fact that not all random processes are equally random with the degree of uncertainty fully described as a function of the PDF.

Having introduced the Shannon entropy and discussed the intuition behind it, it looks like a promising descriptor of data to build upon within machine learning. However, the task of actually estimating the entropy from real world data is altogether nontrivial in its current form. Here, the extension done to the theory by Alfred Renyi in the mid 1950s proves very helpful in managing to do straight forward estimation of entropy from an unknown dataset. With this, one can then train algorithms (do learning) to solve various tasks with ITL-principles as the foundation.

3.2 Renyi's Entropy

Renyi set out to find the most general form of an information measure that preserved the property of additivity for information about independent events, which he describes as the most important aspect of entropy [46]. This means that the joint observation of two independent events should yield as much information as the sum of them combined

$$I(\boldsymbol{x}_i \cap \boldsymbol{x}_j) \overset{\boldsymbol{x}_i \text{ind.} \boldsymbol{x}_j}{=} I(\boldsymbol{x}_i) + I(\boldsymbol{x}_j) \tag{3.4}$$

He noted, as some authors before him had [3], that the summation and integration in (3.1) and (3.2), could be understood as a linear weighted mean of the quantities $1/\log(p(x_k)) = I_k$. Extending the definition of taking the mean value by applying an invertible, monotonic function, $g(x)$, a new entropy estimate can be written as

$$H_{g()}(\boldsymbol{X}) = g^{-1}\left(\int p(\boldsymbol{x}) g(I(\boldsymbol{x})) d\boldsymbol{x}\right) \tag{3.5}$$

where $I(\boldsymbol{x})$ is the information contained in event $\boldsymbol{x}$ as defined by Shannon. In these discussions, only the continuous case will be considered. The same theory holds for discrete variables also (with sums taking the place of integrals), but is here omitted for simplicity.

The restriction of additivity for independent events turn out to restrict the number of valid functions, $g(\boldsymbol{x})$, substantially [45], leaving only

$$g(\boldsymbol{x}) = c\boldsymbol{x} \tag{3.6}$$

$$g(\boldsymbol{x}) = c\, 2^{(1-\alpha)\boldsymbol{x}} \tag{3.7}$$

with $\alpha > 0$ and $\alpha \neq 1$.

Applying the former (for any $c \neq 0$), (3.6), into (3.5) returns the ordinary mean leaving the original Shannon entropy. By using the latter however, the following entropy estimate is obtained

$$H_\alpha(\boldsymbol{X}) = \frac{1}{1-\alpha} \log\left(\int p^\alpha(\boldsymbol{x}) d\boldsymbol{x}\right) \tag{3.8}$$

This new formulation leaves the logarithm outside of the integral and lends itself to being more flexible through different choices of the parameter α. Indeed, in the limiting case where α approaches 1, it is easily verified (through l'Hôpitals rule) that $\lim_{\alpha \to 1} H_\alpha(X) = H_S(X)$, the original Shannon entropy [45].

3.2.1 Renyi's Quadratic Entropy

One particularly useful choice of α, namely $\alpha = 2$, gives the quadratic entropy measure

$$H_2(\boldsymbol{X}) = -\log \left(\int_{\boldsymbol{X}} p^2(\boldsymbol{x}) d\boldsymbol{x} \right) \tag{3.9}$$

This parameter choice has been the most prevalent in applied uses of information theoretic learning in recent years. The reason for this is the elegant solution obtained when noticing that $\int_{\boldsymbol{X}} p^2(\boldsymbol{x}) d\boldsymbol{x}$ (or equivalently $\sum_{k=1}^{N} p^2(\boldsymbol{x}_k)$) is the expected value of $p(\boldsymbol{x})$ taken over $p(\boldsymbol{x})$

$$E_{p(\boldsymbol{x})}[p(\boldsymbol{x})] = \int_{\boldsymbol{X}} p^2(\boldsymbol{x}) d\boldsymbol{x} \tag{3.10}$$

This, of course, is just the first order moment of $p(\boldsymbol{x})$, a quantity that can be easily estimated from the sample mean. Given N realizations of $p(\boldsymbol{x})$, the expected value (and thus the integral) can be estimated by

$$E_{p(\boldsymbol{x})}[p(\boldsymbol{x})] \approx \frac{1}{N} \sum_{k=1}^{N} p(\boldsymbol{x}_k) \tag{3.11}$$

This holds for both continuous and discrete $\boldsymbol{X}$. Now, all that remains is inserting the estimate into the negative logarithm of (3.9) to have the entropy measure sought.

It could here be worth commenting the link to the technique called Monte Carlo integration. This theory states that given an integral over the product of two functions $f(\boldsymbol{x})$ and $h(\boldsymbol{x})$, the integral can be estimated by simulating data according to $h(\boldsymbol{x})$ applying $f(\boldsymbol{x})$ and taking the mean. This gives [47]

$$\int h(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} = E_{h(\boldsymbol{x})}[f(\boldsymbol{x})] \approx \frac{1}{N} \sum_{k=1}^{N} f(\boldsymbol{x}'_k) \tag{3.12}$$

where $X' = \{\boldsymbol{x}'_1, \boldsymbol{x}'_2, ..., \boldsymbol{x}'_N\}$ is generated from $h(\boldsymbol{x})$. Setting $f(\boldsymbol{x}) = h(\boldsymbol{x}) = p(\boldsymbol{x})$, this is exactly the situation of (3.11).

Estimation of $\boldsymbol{H_2}$

As was seen in the previous section, using Renyi's quadratic entropy results in a easy form of the estimate where the mean value of $p(\boldsymbol{x})$ is needed. To achieve this from actual data, the underlying unknown probability density needs to be estimated. To this end, one can draw on the theory examined in Section 2.2. As mentioned there, real world problems of estimating PDFs are often tackled using non-parametric approaches [45]. This comes from their strength of not introducing any model bias and being able to handle complex structures along the different dimensions.

Recall from Section 2.2.1 that the probability density in a point $\boldsymbol{x}$ could be written as the mean contribution from all datapoints weighted according to distance by some kernel function. Pursuing this intuition, gives the (non-parametric) estimate

$$\hat{p}(\boldsymbol{x}) = \frac{1}{N\sigma}\sum_{k=1}^{N}\kappa\left(\frac{\boldsymbol{x}-\boldsymbol{x}_k}{\sigma}\right) \tag{3.13}$$

with σ being a bandwidth parameter and the kernel function κ adhering to the requirements of section 2.2.1.

Using a Gaussian kernel, $G_\sigma(\cdot)$, (on the from given in (2.10)), gives the classic Parzen window estimate used extensively in practice to realize entropy estimates from real data. This will be the standard method of estimating probability densities moving forward. However, in the next chapter, the theory will be extended to the k nearest neighbor case.

The reason this Gaussian kernel estimator has been so popular, comes from the nice properties the kernel exhibits under convolution with itself. Firstly, the arguments of the kernel reduces to the difference, and the variance of each of the kernels adds together. This yields the estimate

$$\begin{aligned}
\hat{H}_2(X) &= -\log\int_{-\infty}^{\infty}\left(\frac{1}{N}\sum_{i=1}^{N}G_{\sigma^2}(\boldsymbol{x}-\boldsymbol{x}_i)\right)^2 d\boldsymbol{x} \\
&= -\log\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\int_{-\infty}^{\infty}G_{\sigma^2}(\boldsymbol{x}-\boldsymbol{x}_i)G_{\sigma^2}(\boldsymbol{x}-\boldsymbol{x}_j)d\boldsymbol{x} \\
&= -\log\left(\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}G_{2\sigma^2}(\boldsymbol{x}_i-\boldsymbol{x}_j)\right)
\end{aligned} \tag{3.14}$$

This is a very nice result, as only the N known datapoints enter into the estimation. With only evaluations across the datapoints needed, the estimate reduces to something which can be done in $\mathcal{O}(N^2)$ time regardless of the dimensionality of the data. This is a highly attractive property, as it circumvents the problem of populating a high dimensional space with testing points in order to do estimation [45]. In scenarios with large amounts of data however, this complexity can prove too time consuming and steps have to be taken to avoid too many cross datapoint evaluations.

It could also be noted that the double sum in (3.14), can be evaluated in matrix- and vector notation as

$$\hat{H}_2(X) = -\log\left\{\frac{1}{N^2}\mathbf{1}^T\boldsymbol{K}\mathbf{1}\right\} \tag{3.15}$$

where $\mathbf{1}$ is a $(N \times 1)$-dimensional vector of ones. Further more, if a valid Mercer kernel function is used to estimate the PDF, $\boldsymbol{K}$ $(N \times N)$ will be a positive semidefinite matrix of cross point kernel evaluations corresponding to inner product calculations in F.

The elements of $\boldsymbol{K}$ implicitly give information about the entropy in the dataset. Eigendecomposing this matrix can then help do e.g. dimensionality reduction and denoising [26], classification [27] or clustering [16].

3.3 Divergence

So far, only informational theoretic measures related to a single probability density has been given a review. An equally important subject, is how to construct some metric to measure similarity (or dissimilarity) between two PDFs. Ever since Mahalanobis introduced the concept of distance between probability densities [38], the topic has been extensively expanded and is today a cornerstone in information theoretic learning. As such, it is in the literature also referred to as information divergence, information gain and relative entropy.

Given two PDFs, $p(x)$ and $q(x)$, the Kullback-Liebler (KL) divergence between them is defined as [32]

$$D_{KL}(p||q) = \sum_x p(\boldsymbol{x})\log\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \tag{3.16}$$

$$D_{KL}(p||q) = \int_X p(\boldsymbol{x})\log\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}dx \tag{3.17}$$

This is a measure of the dissimilarity between between p and q adhering to some, but not all, of the postulates of a metric; it is non-negative and 0 if and only if $p = q$. However, it is not symmetric and does not fulfill the triangle inequality and as such it is called a divergence measure.

Even though (3.16) – (3.17) is not symmetric ($D_{KL}(p||q) \neq D_{KL}(q||p)$), it can easily be made so by instead using the J *divergence* (introduced by Jeffrey) as mentioned by Príncipe [45] citing [8]. One version of it is given as follows

$$D_J(p||q) = \sqrt{\frac{1}{2}(D_{KL}(p||q))^2 + \frac{1}{2}(D_{KL}(q||p))^2} \tag{3.18}$$

As was discussed in the previous section, entropy is related to the uncertainty in a process. In a similar fashion, the divergence is related to the gain in information on $p(\boldsymbol{x})$ by observing $q(\boldsymbol{x})$. Given an event A with probability $q(\boldsymbol{x}_i)$, which changes to $p(\boldsymbol{x}_j)$ upon observing B. From this, the uncertainty changes according to the information gained from observing B. This gain is $\log 1/q(\boldsymbol{x}_i) - \log 1/p(\boldsymbol{x}_j) = \log p(\boldsymbol{x}_j)/q(\boldsymbol{x}_i)$, a quantity recognized in $D_{KL}(p||q)$. So, the KL-divergence is the weighted mean (according to $p(\boldsymbol{x})$) of the gain in information (or decrease in uncertainty) when observing an event. This intuition help explain the other names of divergence mentioned above.

3.3.1 Renyi's Divergence

Now, similar to the section about entropy, a scalar function over probabilities has been introduced and, similar still, it could be noted that evaluation of this function can be difficult in practice. In the case of entropy, Renyi's more general definition lead to promising results and his general divergence measure is investigated here.

Renyi's α divergence between $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ is defined as [1]

$$D_\alpha(p||q) = \frac{1}{\alpha - 1} \log \int_X p(\boldsymbol{x}) \left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \right)^{\alpha - 1} d\boldsymbol{x} \tag{3.19}$$

with $\alpha > 0$ and $\alpha \neq 1$.

Here, the connection with the Renyi's entropy is clear. The logarithm now appear outside of the integral and the free parameter α has been introduced. Interesting properties of this divergence include [45]

1. $D_\alpha(p||q) \geq 0, \ \forall p(\boldsymbol{x}), q(\boldsymbol{x})$
2. $D_\alpha(p||q) = 0$ if and only if $p(\boldsymbol{x}) = q(\boldsymbol{x}) \ \forall \ x \in \mathbb{R}^d$
3. $\lim_{\alpha \to 1} D_\alpha(p||q) = D_{KL}(p||q)$

[1]For simplicity, only the continuous case will be treated hereinafter. Changes needed to accommodate discrete random variables should be obvious where applicable.

1 and 2 states the same metric conditions which the KL-divergence follows. 3 states that the KL-divergence is the limiting case when α tends to 1, much in the same way as Shannon entropy was the limiting case of Renyi's entropy for $\alpha \rightarrow 1$.

Estimation of D_α

As with Renyi's entropy, the divergence could be understood as the expected value of $\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}$ under $p(\boldsymbol{x})$. From this, the measure could be estimated non-parametrically via parzen windowing. Given N points generated from $p(\boldsymbol{x})$, $\{\boldsymbol{x}_p(1), ..., \boldsymbol{x}_p(N)\}$, and M points from $q(\boldsymbol{x})$, $\{\boldsymbol{x}_q(1), ..., \boldsymbol{x}_q(M)\}$, the estimate becomes

$$\begin{aligned} D_\alpha(p||q) &= \frac{1}{\alpha-1}\log E_p\left[\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)^{\alpha-1}\right] \\ &\approx \frac{1}{\alpha-1}\log\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\hat{p}(\boldsymbol{x}_i)}{\hat{q}(\boldsymbol{x}_i)}\right)^{\alpha-1} \\ &= \frac{1}{\alpha-1}\log\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\frac{1}{N}\sum_{j=1}^{N}G_{\sigma^2}(\boldsymbol{x}_p(i)-\boldsymbol{x}_p(j))}{\frac{1}{M}\sum_{j=1}^{M}G_{\sigma^2}(\boldsymbol{x}_p(i)-\boldsymbol{x}_g(j))}\right)^{\alpha-1} \\ &= \hat{D}_\alpha(p||q) \end{aligned} \tag{3.20}$$

In employing the Parzen windowing, the estimate reduces to cross-datapoint evaluations leading to a complexity bounded above by $\mathcal{O}(N^2)$ (or $\mathcal{O}(NM)$ for $M > N$) as with the Renyi entropy estimate.

3.3.2 Cauchy-Schwartz Divergence

There exists many more divergence measures, such as the Bhattacharyya [6], the Chernoff [7] and the Hellinger [4] distance as cited by Príncipe [45], which can be used to evaluate similarity between PDFs. It is beyond the scope of this thesis to give a review of these. However, another based on Euclidean distance is worth investigating. It is defined as

$$D^*_{ED}(p||q) = \int \sqrt{(p(\boldsymbol{x}) - q(\boldsymbol{x}))^2} d\boldsymbol{x} \tag{3.21}$$

This functional obeys all the properties of a metric and is thus a distance measure (not just a divergence measure).

For use in machine learning applications (dealing with cost functions to be optimized), the square root term is often omitted, giving

$$\begin{aligned} D_{ED}(p||q) &= \int (p(\boldsymbol{x}) - q(\boldsymbol{x}))^2 d\boldsymbol{x} \\ &= \int p^2(\boldsymbol{x})d\boldsymbol{x} + \int q^2(\boldsymbol{x})d\boldsymbol{x} - 2\int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x} \end{aligned} \tag{3.22}$$

This geometrical interpretation of PDF-divergence/distance leads to the final measure, the Cauchy-Schwartz divergence which will be used in the applications to come. Its origin is in the Cauchy-Schwartz inequality

$$\sqrt{\int p^2(\boldsymbol{x})d\boldsymbol{x} \int q^2(\boldsymbol{x})d\boldsymbol{x}} \geq \int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x} \tag{3.23}$$

Here equality holds only if $p(\boldsymbol{x}) = q(\boldsymbol{x})^2$.

From this, Jenssen et al. defined a new divergence measure inspired by the inequality as

$$D_{CS}(p||q) = -\log \frac{\int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x}}{\sqrt{\int p^2(\boldsymbol{x})d\boldsymbol{x} \int q^2(\boldsymbol{x})d\boldsymbol{x}}} \tag{3.24}$$

where the larger factor have been moved to the other side of the inequality.

Because of the CS-inequality, the argument of the logarithm is always in the interval $[0, 1]$. This causes $D_{CS} \geq 0 \; \forall \; \boldsymbol{x}$ with equality only when $p(\boldsymbol{x}) = q(\boldsymbol{x})$. The measure is also symmetric, but it does not fulfill the triangle inequality, makes it a divergence measure instead of a metric.

Decomposing the factors of the logarithm yields

$$D_{CS}(p||q) = 0.5 \log \int p^2(\boldsymbol{x})d\boldsymbol{x} + 0.5 \log \int q^2(\boldsymbol{x})d\boldsymbol{x} - \log \int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x} \tag{3.25}$$

which is recognized as the terms in (3.22) divided by 2 with the logarithm applied. From this, it is understood that the CS divergence is closely related to the Euclidean distance divergence D_{ED} which directly measures differences between densities.

It should also be noted that this measure is particularly suitable for measureing the seperation of two datasets. This comes from the two different information theoretic considerations which enter into the equation; in order

[2] In general, this is true up to two constant factors but since $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ have to integrate to unity, these constants have to be 1.

to maximize the CS-divergence, the between set entropy should be small, while each set in itself should contain as much information as possible. A parallel to the normalized cut in 2.4.1 is here natural to draw. In that case, a good seperation between two sets was described as something which had large distance across the sets while keeping each set itself compact. For the CS-divergence case, the same thing is measured by small overlap in the individual PDFs (distance), while each of the sets carry as much information as possible (compactness).

Estimation of D_{CS}

Having decomposed the divergence measure into the three terms of (3.25), the first two integral evaluations is recognized as the Renyi's quadratic entropy of $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ from the previous section.

$$D_{CS}(p||q) = -0.5H_2(\boldsymbol{X}_p) - 0.5H_2(\boldsymbol{X}_q) - \log \int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x} \tag{3.26}$$

The estimation of these can be done as discussed in the previous section.

With this, the only term which needs a closer look is the integral over both densities, called the Cross Information Potential (CIP) [45]. It is interesting to note that this term is the only one of the three involving both densities. Because of this, it is the only one which actually relay anything about the difference between the two. The two other terms merely normalize the measure with respect to the carried information, the entropy.

To estimate the integral $\int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x}$, the two most common approaches are

1. Insert a Parzen window estimate with a Gaussian kernel for each density and use the properties of the Gaussian under convolution.

2. Exploit the fact that the datasets are realizations from each of the distributions and do Monte Carlo integration or mean value approximation.

Both methods will be investigated.

Assume N_p and N_q points are generated from p and q, giving the sets $\mathcal{P} = \{\boldsymbol{x}_i''\}_{i=1}^{N_p}$ and $\mathcal{Q} = \left\{\boldsymbol{x}_j'\right\}_{j=1}^{N_q}$ respectively. For the first method, the Parzen window estimates of the densities are inserted directly and the convolution properties of the kernel (as explained by Jenssen et al. [21], citing [44]) used

$$\begin{aligned}\int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x} &\approx \int \hat{p}(\boldsymbol{x})\hat{q}(\boldsymbol{x})d\boldsymbol{x} \\ &= \int \frac{1}{N_p}\sum_{i=1}^{N_p} G_{\sigma^2}(\boldsymbol{x}-\boldsymbol{x}_i)\frac{1}{N_p}\sum_{j=1}^{N_q} G_{\sigma^2}(\boldsymbol{x}-\boldsymbol{x}_j)d\boldsymbol{x} \\ &= \frac{1}{N_pN_q}\sum_{i=1}^{N_p}\sum_{j=1}^{N_q} G_{2\sigma^2}(\boldsymbol{x}_i-\boldsymbol{x}_j) \qquad (3.27)\end{aligned}$$

Again it is seen that the estimate reduces to a sum over all cross-datapoint differences weighted by some kernel as with the estimation of the Renyi quadratic entropy. This time however, all of the cross terms are between data from different distributions.

As with the estimate of $H_2(\boldsymbol{X})$, this quantity can be expressed in vector notation as $\mathbf{1}_{N_p}^T\boldsymbol{K}\mathbf{1}_{N_q}$ where $\boldsymbol{K}$ is a $(N_p \times N_q)$ matrix of the kernel evaluations and $\mathbf{1}_{N_x}$ is a column vector with ones of length N_x.

If instead the second approach is used, it is noted that both the datasets given are realizations from the underlying distributions p and q. Recall from Section 3.2.1 that using Monte Carlo integration [47], one of the sets will be inserted directly into the function for the other, and the mean taken. Choosing to insert $\mathcal{Q}$ into $p(\boldsymbol{x})$, gives the mean value approximation

$$\int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x} \approx \frac{1}{N_q}\sum_{j=1}^{N_q} p(\boldsymbol{x}'_j)$$

Notice that this can be regarded as as approximation to the expected value of q on p

$$E_{p(\boldsymbol{x})}[q(\boldsymbol{x})] \approx \frac{1}{N_q}\sum_{j=1}^{N_q} p(\boldsymbol{x}'_j) \qquad (3.28)$$

It should also be noted that instead choosing to insert $\mathcal{P}$ into $q(\boldsymbol{x})$ is also an option (by the commutative property of functions under integration). In that case, the opposite expectation would be evaluated (p on q).

Now, if $p(\boldsymbol{x})$ had been a known function, the estimation would have been done. It is not however, and has to be estimated using the other dataset $\mathcal{P}$. Using the Parzen window estimate with a Gaussian kernel $G_{\sigma'}(\cdot)$ gives

$$\int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x} \approx \frac{1}{N_qN_p}\sum_{i=1}^{N_p}\sum_{j=1}^{N_q} G_{\sigma'}(\boldsymbol{x}''_j-\boldsymbol{x}'_i) \qquad (3.29)$$

With this, the same result as with convolving two Gaussian kernels is reached. The width parameters σ and σ' are not necessarily equal, but these are a free parameters in this setting. Choosing $\sigma' = \sigma\sqrt{2}$, would yield the exact same estimate.

Reaching the estimate above by two different methods is good, albeit a bit unnecessary. However, something was gained in going by method number two. In this method, a Gaussian kernel for estimation was chosen, but the theory is consistent for any kernel choice which is asymptotically correct in estimating $p(\boldsymbol{x})$. Indeed, it is consistent for *any* asymptotically correct estimator of $p(\boldsymbol{x})$ (not necessarily a kernel). This fact will be crucial in developing a KNN-take on Information Theoretic Learning in the next chapter.

3.4 Clustering with IT-principles

Clustering is an unsupervised way of grouping data given some measure of similarity. In clustering, the goal is to organize unlabeled data into its natural groups such that points within each group is more similar to each other relative to datapoints in the other groups. In this setup of machine learning, there are no known data labels which could help guide our algorithms, nor are there some specific similarity measures which are guaranteed to capture separation of all possible datasets.

Traditionally, the measures of similarity have been linear in nature, only taking into account the first and second order statistics of the clusters produced (see Gokcay and Príncipe [17] and the references therein). The famous K-means algorithm is an example of an algorithm with such a measure, seeking to minimize the distance variance in each cluster [37]. This methods only works well if the data structures are hyperspherically distributed and will break down if presented data with prominent non-linear features.

In recent years the topic of clustering has been explored within the setting of Information Theoretic Learning (ITL). As seen in this chapter, this theory is equipped with measures for both across-cluster (divergence) and within-cluster (entropy) considerations. With that, an extension of the traditional clustering theory can be explored where Information Theoretic considerations replacing finite-order statistics.

Clustering with IT was first done by Watanabe [56] as cited in [17]. The major limitations in the early work of Information Theoretic Clustering (ITC) was that parametric methods were used for estimating the PDFs. Because of this, a lot of prior knowledge was required about the data if a significant model bias was to be avoided. Gokcay and Príncipe [17] circumvented this problem in their landmark article by estimating the required IT-measures with the non-parametric kernel methods discussed in the previous sections.

The first article using non-parametric kernel estimates applied a hierarchical optimization routine which tried to reduces the number of clusters (generated from a naïve preclustering routine) according to the Cross Information Potential (CIP) of the set [17]. The CIP was estimated as in (3.27) and the algorithm provided promising results. However, the complexity, T, of the optimization ($\mathcal{O}(N^3) < T < \mathcal{O}(N^4)$), rendered the algorithm very slow or unusable for problems of some size.

In later work, the clustering was done hierarchically in a fashion which tried to minimize the gain in Renyi's quadratic entropy when a datapoint was assigned a cluster [20]. When reducing the number of clusters, the CIP was again used to evaluate which cluster was to be removed. As opposed to the original algorithm, two information theoretic measures are here used to do the optimization, one evaluating within cluster entropy and one evaluating between cluster divergence. This made the optimization easier and reduced the complexity. The run-time was further improved by performing stochastic subsampling of the dataset in [29].

Extended development in this field lead Jenssen et al. [22, 21] to a clustering routine which used the Cauchy-Schwartz divergence measure to do ITC. This method produced state-of-the-art results by estimating the divergence measure by a kernel method (as in 3.3.2) and hierarchically optimizing it. However, as with any other kernel-method, the quality of the results proved highly dependent on a correct choice of the bandwidth parameter σ.

3.4.1 Kernel ITC

This section will introduction the cost function which the most recent research has focused on to do ITC. Here, its kernel estimate will also be reviewed. In the next chapter a new, KNN-approach is proposed for doing the optimization.

As was seen previously, the Cauchy-Schwartz divergence provided a normalized measure of distance between two PDFs inspired by the the inequality with the same name. It was also shown how the quantity could be estimated in a non-parametric fashion by only cross-datapoint evaluations in a kernel function very similar to the Renyi's quadratic entropy. This divergence has been the subject of much research in recent years with the focus on clustering [22, 21, 29, 31, 23].

Cost function and estimation

The machine learning task of clustering is based on optimizing a cost function which relays information about the separation of the groups found. It was seen in Section 3.3.2 that the Cauchy-Schwartz divergence measure could be suitable to that end, as it measures distance between two clusters while taking into account the information contained in each cluster.

To repeat (3.24), the CS-divergence is defined as

$$D_{CS}(p||q) = -\log \frac{\int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x}}{\sqrt{\int p^2(\boldsymbol{x})d\boldsymbol{x} \int q^2(\boldsymbol{x})d\boldsymbol{x}}}$$

The nominator is know as the Cross Information Potential (CIP) and measures the distance between p and q. The denominator is recognized as Renyi's quadratic entropy of p and q and measures the information contained in each of the PDFs.

This lead Jenssen et al. [21] to define a cost function based on the divergence as

$$J_{CS}(p,q) = \frac{\int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x}}{\sqrt{\int p^2(\boldsymbol{x})d\boldsymbol{x} \int q^2(\boldsymbol{x})d\boldsymbol{x}}} \tag{3.30}$$

where the monotonically increasing logarithmic function has been dropped and the minus sign removed. This means that to maximize D_{CS}, one could instead minimize J_{CS}.

Now, using the non-parametric estimates for $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$

$$\begin{aligned} \hat{p}(\boldsymbol{x}) &= \tfrac{1}{N_p} \textstyle\sum_{i=1}^{N_p} \kappa_{\sigma^2}(\boldsymbol{x}, \boldsymbol{x}_i) \\ \hat{q}(\boldsymbol{x}) &= \tfrac{1}{N_q} \textstyle\sum_{j=1}^{N_q} \kappa_{\sigma^2}(\boldsymbol{x}, \boldsymbol{x}_j) \end{aligned}$$

where κ_{σ^2} is a spherical Gaussian kernel with $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}$, the cost function can be estimated. By inserting the estimates above into (3.30) and using the convolution property of Gaussian kernels, the following estimate is obtained

$$\hat{J}_{CS}(p,q) = \frac{\frac{1}{N_p N_q} \sum_{i,j}^{N_p,N_q} \kappa_{2\sigma^2}(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\sqrt{\frac{1}{N_p^2} \sum_{i,j}^{N_p,N_p} \kappa_{2\sigma^2}(\boldsymbol{x}_i, \boldsymbol{x}_j) \frac{1}{N_q^2} \sum_{i,j}^{N_q,N_q} \kappa_{2\sigma^2}(\boldsymbol{x}_i, \boldsymbol{x}_j)}} \tag{3.31}$$

An interesting property of the Cauchy-Schwartz divergence and cost function is the normalizing interaction between the numerator and denominator. In (3.31), the terms N_p and N_q are canceled out. Indeed, any linear terms in the kernel function is also removed[3].

[3]Like the $((2\pi)^{d/2}\sigma^d)^{-1}$ terms of the spherical Gaussian kernel.

Multiple clusters To extend the estimate of the cost function to handle a set of M clusters, $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_M\}$, the divergence between every combination of cluster has to be evaluated. This is in [21, 29, 22] done by extending the cost function as

$$\hat{J}(\mathcal{C}) = \frac{\frac{1}{2}\sum_{i,j}^{N,N} M_{i,j}\kappa_{2\sigma^2}(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\sqrt{\prod_{m=1}^{M}\sum_{i,j}^{N,N} M_{C_{m_{i,j}}}\kappa_{2\sigma^2}(\boldsymbol{x}_i, \boldsymbol{x}_j)}} \tag{3.32}$$

where $M_{i,j}$ is one if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are in different clusters and zero otherwise. $M_{C_{m_{i,j}}}$ is one if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ both are in cluster m.

Optimization Optimization of this cost function has been the subject of much research. A heuristic optimization was proposed in [21]. One involving Lagrange optimization was investigated in [25, 24] while another based on spectral clustering explored in [28]. For a complete review on these optimization techniques, the reader it referred to [23]. A discussion on the divergence measures' connection to graph theory and Mercer kernels can be found in [31].

Chapter 4

New Information Theoretic Clustering Algorithm with KNN

In this chapter, the new K Nearest Neighbor (KNN) approach to Information Theoretic Learning (ITL) will be introduced in the form of a new clustering algorithm. This algorithm will in the results chapter be shown to provide state-of-the-art results without the need of any parameter tuning and vastly outperform traditional kernel methods when working on data spread over a wide range of scales.

4.1 Cost Function

In the previous chapter, several IT measures was introduced. It was seen that the Cauchy-Schwartz divergence D_{CS} provided a very intuitive measure of cluster separation by both measuring the similarity between two clusters *and* the information contained in each separate cluster. The measure was closely related to Renyi's quadratic entropy and could with that easily be estimated from data.

For these reasons, it is a natural choice to build upon for a KNN-approach to Information Theoretic Clustering. Recall from section 3.4.1 that the Cauchy-Schwartz cost of separating two clusters p and q, is given as

$$J_{CS}(p,q) = \frac{\int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x}}{\sqrt{\int p^2(\boldsymbol{x})d\boldsymbol{x} \int q^2(\boldsymbol{x})d\boldsymbol{x}}} \tag{4.1}$$

4.2 KNN Estimation

In order to do Information Theoretic Learning in a KNN-setting, an estimate of (4.1) building on KNN principles is needed. The previous work in this field drew on Parzen window kernel estimates of the PDFs and used the convolution properties of the Gaussian kernel to do the estimation; method number one of section 3.3.2. This method can not be applied in this setting. Instead, an estimate based on the mean approximation of the integrals is built upon; method two mentioned in the section.

The estimation of J_{CS} can be split up into the estimation of each of the integrals. Evaluating the integrals in the denominator first, it is known from previous theory that

$$\int p^2(\boldsymbol{x})d\boldsymbol{x} = E_{p(\boldsymbol{x})}[p(\boldsymbol{x})] \approx \frac{1}{N_p}\sum_{i=1}^{N_p} p(\boldsymbol{x}_i) \tag{4.2}$$

Now, since $p()$ is not known, it has to be estimated, something which will here be done by the KNN method.

Recall from Section 2.2.3 that given a dataset of N_p points assumed to be generated from p, the PDF in a point $\boldsymbol{x}$ could be estimated non-parametrically as

$$\hat{p}_{\text{kNN}}(\boldsymbol{x}; k) = \frac{k}{NV_k(\boldsymbol{x})}$$

Where $V_k(\boldsymbol{x})$ calculates the hyper volume spanned from $\boldsymbol{x}$ to the k-th nearest neighbor in the dataset.

The estimate can now be inserted directly into (4.2) to give a non-parametric KNN estimate of the integral as

$$\int p^2(\boldsymbol{x})d\boldsymbol{x} \approx \frac{1}{N_p}\sum_{i=1}^{N_p} p(\boldsymbol{x}_i) \approx \frac{k}{N_p^2}\sum_{i=1}^{N_p} \frac{1}{V_k(\boldsymbol{x}_i)}$$

Estimation of the other integral in the denominator of (4.1) can be done in the same way, replacing p with q.

This means that the original integral can be estimated by evaluating the mean value of the inverse hyper volume spanned from each datapoint out to its k-th nearest neighbor in the dataset. It is a fundamentally different way of obtaining the estimation and has not been seen before in any of the literature studied on the subject.

For the numerator of (4.1), the same approach as above is used. However, here the integral is over two different PDFs. This leads to the expected value evaluating points between the two datasets

$$\int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x} = E_{p(\boldsymbol{x})}[q(\boldsymbol{x})] \approx \frac{1}{N_q}\sum_{i=1}^{N_q} p(\boldsymbol{x}_i)$$

where the $\boldsymbol{x}_i$'s are from q.

Next, the KNN estimate of p is used with the datapoints from q. This leads to

$$\int p(\boldsymbol{x})q(\boldsymbol{x})d\boldsymbol{x} \approx \frac{1}{N_q}\sum_{i=1}^{N_q} p(\boldsymbol{x}_i) \approx \frac{1}{N_q}\sum_{i=1}^{N_q} \frac{k}{N_p V_{k_p}(\boldsymbol{x}_i)} = \frac{k}{N_q N_p}\sum_{i=1}^{N_q} \frac{1}{V_{k_p}(\boldsymbol{x}_i)}$$

Where $V_{k_p}(\boldsymbol{x}_i)$ means that the hyper volume spanned from $\boldsymbol{x}_i$ (belonging to q) to the k-th nearest neighbor in p is evaluated.

In general $E_{p(\boldsymbol{x})}[q(\boldsymbol{x})] \neq E_{q(\boldsymbol{x})}[p(\boldsymbol{x})]$ when performing the estimation this way. The reason for this is that no two set of points from p and q necessarily both are each others k-th nearest neighbor. Another reason for this difference, is the formula's sensitivity for dataset size. Imagine one point from p lying a unit distance from ten overlapping points of q, with these being the only points in each of the sets. Evaluating the integral by the expected value both ways with $k = 1$ yields

$$\begin{aligned} E_{p(\boldsymbol{x})}[q(\boldsymbol{x})] &\approx \frac{k}{N_p N_q}\sum_{i=1}^{N_q} \frac{1}{V_{k_p}(\boldsymbol{x}_i)} \\ E_{q(\boldsymbol{x})}[p(\boldsymbol{x})] &\approx \frac{k}{N_p N_q}\sum_{i=1}^{N_p} \frac{1}{V_{k_q}(\boldsymbol{x}_i)} \end{aligned}$$

In the first case, 10 evaluations of $1/V(1)$ would be summed up, while in the other case only 1 term would enter the summation. Since all the other values are unchanged, this leads to two potentially very different estimates of the integral.

With no obvious way of choosing which of the above estimates to use for the integral in the numerator, the choice has been made to do both estimations and take the mean. This makes the end estimate of the cost function and the CS-Divergence symmetric[1].

The end KNN estimate of $J_{CS}(p, q)$ becomes

$$\hat{J}_{CS}(p,q) = \frac{\frac{1}{2}\left(\frac{k}{N_q N_p}\sum_{i=1}^{N_q} \frac{1}{V_{k_p}(\boldsymbol{x}_i)} + \frac{k}{N_q N_p}\sum_{i=1}^{N_p} \frac{1}{V_{k_q}(\boldsymbol{x}_i)}\right)}{\sqrt{\left(\frac{k}{N_p^2}\sum_{i=1}^{N_p} \frac{1}{V_{k_q}(\boldsymbol{x}_i)}\right)\left(\frac{k}{N_q^2}\sum_{i=1}^{N_q} \frac{1}{V_{k_p}(\boldsymbol{x}_i)}\right)}} \tag{4.3}$$

[1]Similar to Jeffreys way of making divergence measures symmetric.

This can be easily extended to an estimator of the Cauchy-Schwartz divergence by applying the logarithm and switching the sign

$$\hat{D}_{CS}(p||q) = -\log \hat{J}_{CS}(p,q) \tag{4.4}$$

4.2.1 Multiple clusters

Since a divergence measure in itself is a measure between *two* PDFs, extending it to apply to a higher number of functions has to be done by some heuristic. There are here several choices which have been used in related work previously and they all revolve around calculating all the combinations of divergence between the different functions. The choice presented here have used a method inspired by the work of Jenssen et al. [30].

Given M clusters $\mathcal{C} = \{\mathcal{C}_1, ..., \mathcal{C}_M\}$ with corresponding PDFs, $\{p_{\mathcal{C}_1}, ..., p_{\mathcal{C}_M}\}$, the Cauchy-Schwartz divergence of the set is defined as

$$\hat{D}_{CS}(\mathcal{C}) = -\log \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} \frac{1}{C_M} \hat{D}_{CS}(p_{\mathcal{C}_i}||p_{\mathcal{C}_j}) \tag{4.5}$$

Here, C_M is the number of ways M clusters can be arranged in groups of two

$$C_M = \binom{M}{2} = \frac{M(M-1)}{2} \tag{4.6}$$

This term helps normalize the divergence measure when different number of clusters is used, something which proves useful when trying to estimate the true number of clusters in a given dataset (more on this in Section 5.1.2).

Wanting to maximize (4.5), one could equivalently minimize the extended cost function

$$\hat{J}_{CS}(\mathcal{C}) = \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} \frac{1}{C_M} \hat{J}_{CS}(p_{\mathcal{C}_i}, p_{\mathcal{C}_j}) \tag{4.7}$$

where each of the C_M inner terms are estimated according to (4.3).

4.3 Optimization Scheme

Previous work on ITC with the CS-divergence measure has produced several optimization methods for the cost function [21, 24, 28]. The optimization technique presented here is inspired by the heuristic algorithm of [21] which in [23] proved to provide the best and most reliable results of the explored alternatives.

To do the optimization, the dataset is first *seeded* with K_{init} clusters of N_{init} points. An unlabeled point is chosen at random and then that cluster is grown to contain a total of N_{init} points. This is done in an iterative fashion such that each point added to the cluster is the one currently the closest to any of the points in the cluster already. By obtaining the pre-clustering this way, instead of using the $N_{init} - 1$ nearest neighbors of the initial point, provides initial clusters which more closely follow the natural distribution of the data [17].

Having seeded $K_{\text{init}} \times N_{\text{init}}$ points, the remaining unlabeled points are iteratively assigned a cluster. This is done by choosing the unlabeled point closest to some labeled point and finding which label assignment would minimize $J_{CS}(\mathcal{C})$ (thereby maximizing the divergence).

With all the points assigned one of K_{init} clusters and K_{init} being much higher than the number of clusters sought, the task now is to gradually reduce the number of clusters until the user set end cluster number K_{end} is reached. To do this, each of clusters currently in the set is tried removed one at the time and the cost function is evaluated with each of the clusters removed. Recording which of the cluster removals provided the best cost, a *worst* cluster is found. The points of this cluster then have their labels removed and the remaining clusters are regrown to absorb the points of the removed clusters. This growing is done iteratively in a fashion minimizing J_{CS} as before. All the points are now again labeled, but the number of clusters are reduced by one.

This procedure is repeated until the predefined end number of clusters is reached. For completeness, the above description of the algorithm is restated in Algorithm 1 and a visualization of the algorithm running is provided in the next section.

4.3.1 Example of algorithmic steps

In Figure 4.1, each of the steps the clustering algorithm takes is visualized. In (a), the random initial seed step is seen. Here, 8 clusters each of 10 points is seeded, before in (b) the clusters are seen after the iterative growing. This has been done for illustration purposes only and is in general not recommended as each initial cluster is too small. Next, in (c), the worst cluster found has been removed and in (d) the clusters have again been regrown to give each datapoint a label. This is repeated in (e) to (h) and in (h), the final result of 4 clusters is seen.

As is evident, the algorithm has produced a good clustering of the data, which a linear method could not do.

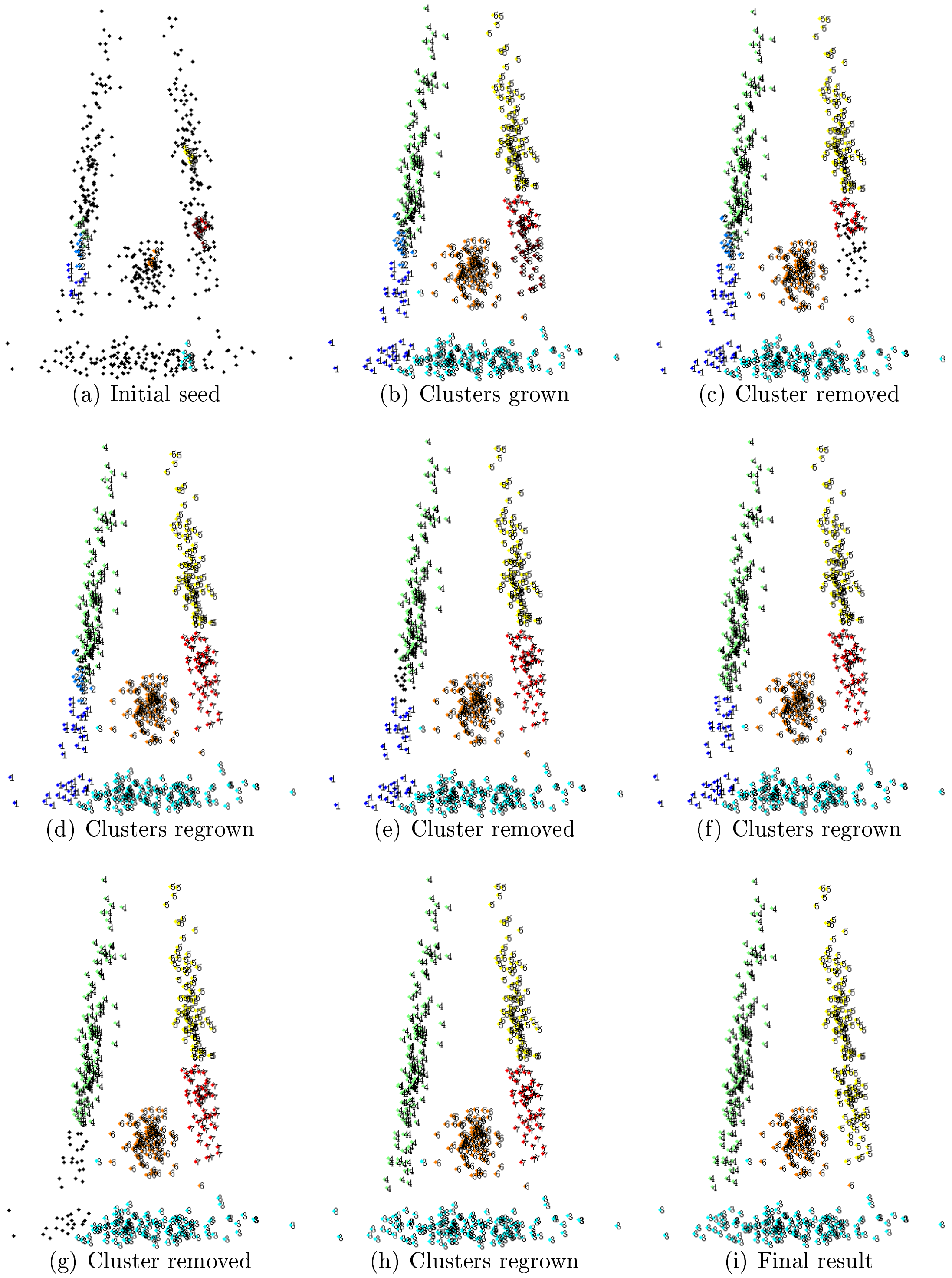

(a) Initial seed (b) Clusters grown (c) Cluster removed
(d) Clusters regrown (e) Cluster removed (f) Clusters regrown
(g) Cluster removed (h) Clusters regrown (i) Final result

Figure 4.1: Visualization of the algorithm running.

Algorithm 1 Pseudocode for clustering with KNN ITL. All cluster assignments (line 3 and 10) are done in order of which point is closest to already clustered point.

```
1: procedure KNN Clustering($K_{\text{init}}, N_{\text{init}}, C_{\text{end}}$)
2:     Seed $K_{\text{init}}$ clusters of $N_{\text{init}}$ points
3:     Iterativly cluster unlabeled points minimizing $\hat{J}_{CS}$
4:     $K_{\text{curr}} = K_{\text{init}}$                        ▷ Set current cluster number
5:     while $K_{\text{curr}} > K_{\text{end}}$ do
6:         $\min_{\mathcal{C}_r} \hat{J}_{CS}(\mathcal{C} \setminus \mathcal{C}_r)$                ▷ Find cluster to remove
7:         $\mathcal{C} \setminus \mathcal{C}_r$
8:         $K_{\text{curr}} = K_{\text{curr}} - 1$
9:         for $i = 1 : N_{\mathcal{C}_r}$ do          ▷ Reassign points in removed cluster
10:            Cluster $\boldsymbol{x}_i : \min_{\mathcal{C}_x} \hat{J}_{CS}(\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_x + \boldsymbol{x}_i, ...\mathcal{C}_{C_{\text{curr}}})$
11:        end for
12:    end while
13:    Return $\mathcal{C}$
14: end procedure
```

4.3.2 Choice of seeding parameters

The choice of K_{init} and N_{init} can be somewhat important for the algorithm to work. Experiments have shown that the algorithm performs best when getting a chance to remove at least 5 clusters before K_{end} is reached. That way, it has some flexibility to adjust to the dataset over several re-clustering steps. At the same time, each initial cluster should not be made up of very few points, as the Information Theory measures being used will tend to be more unreliable.

This creates a trade-off between K_{init} and N_{init}, connected to the parameter K_{end}. When having an end number of clusters of 2 or 3, setting K_{init} to $10-12$ and letting N_{init} be of such a size that 80% of the dataset is included in the initial seeding has been found to be a stable choice.

4.4 Cluster Assignment Voting

As the algorithm is based on random initialization of the K_{init} clusters, it will typically produce different clustering results on each run. Sometimes, the algorithm will iterate towards a good solution, while other times it could get trapped in a local minimum. To find the best clustering result without knowledge of the true labels, a natural choice would be to use the result which provided the highest end divergence. However, although divergence and accuracy is correlated (see Figure 5.3), the relationship is subject to random fluctuations.

Because of this, a voting scheme among the best clustering results is proposed. It is based on the idea that each of the clustering results which gave the highest divergence measures all have provided usable results. Choosing to use only one of them, although easy, can be quite limiting. If instead the different results are allowed to each cast a vote regarding the final clustering assignment of a data point, the end result will be more stable and based on more data.

The way the voting is done is visualized in Figure 4.2. Since the algorithm starts out with K_{init} clusters, the end label names for each of the clusters will typically be different. To solve this, one of the results chosen to participate in the voting has its labels mapped down to run from 1 to K_{end}. Each of the other results also voting then have their labels mapped down to the same range in a manner which minimizes the distance to the first result mapped down. This is stage two of the scheme seen in the figure.

The different label vectors then do a voting for the end cluster assignment of each data point. As can be seen in the example in the figure, as long as the different results have not committed errors which overlap, the end result can turn out to be better that any one result alone [13, 14].

This phenomenon was often times seen on real datasets as well. Running the algorithm on the benchmark set Wine and performing voting among the three best results, the accuracies shown in Table 4.1 was obtained.

Run	1	2	3	Voted
Accuracy	0.865	0.944	0.955	0.961

Table 4.1: Result on voting with 3 best results from the Wine dataset. The voted accuracy is higher than any one of its individual constituents.

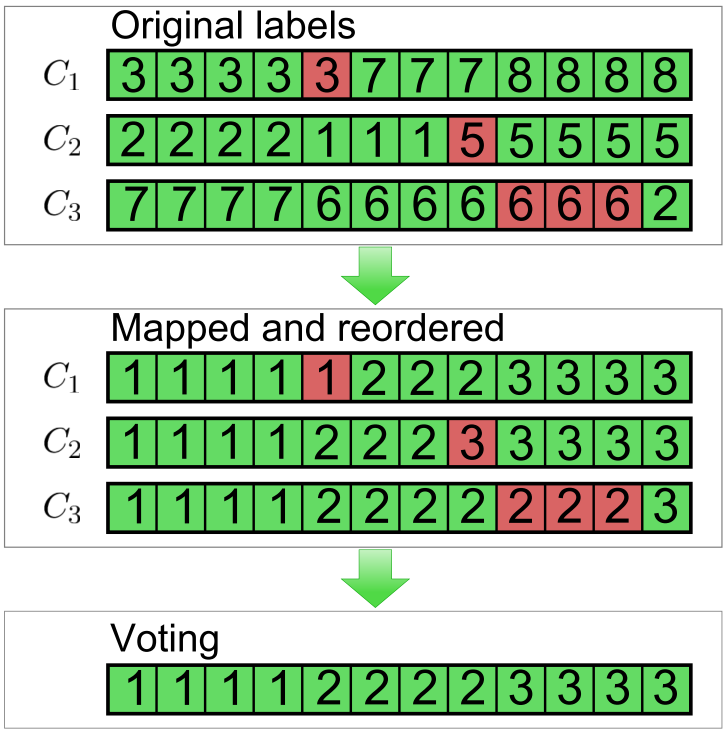


Figure 4.2: Algorithmic scheme for label assignment according to multiple clustering results' voting. Red boxes indicate errors.

4.5 High Dimensional Data

Trying to cluster images typically results in handling feature vectors of many dimensions. Using each pixel in the image as a feature, the dimensionality of the problem quickly grows. Even for a small image of size 20×20, the feature space would be 400-dimensional.

Recall that the formula for calculating the hyper volume of a sphere in d dimensions is given as

$$V(\boldsymbol{x}) = \frac{\pi^{d/2}}{\Gamma(d/2+1)}||\boldsymbol{x}||_2^d$$

If d is sufficiently large, any computer will run into numerical problems handling the extremely large numbers produced to do the calculation. Especially the Gamma function grows rapidly as a function of the dimension[2]. The distances calculated in a high dimensional space can also become quite large, and taking these large numbers to the power of a large d, will produce instabilities in this term also.

The end result is that the calculation either breaks down (tries to evaluated $\frac{\infty}{\infty}$) or calculates the hyper volume as 0. To circumvent this, it was found that for high dimensional data, using only the distance $||\boldsymbol{x}||_2$ instead of the hyper volume (i.e. the distance out to the k-th nearest neighbor instead of the hypervolume), provided reliable results.

Mathematically, the clustering is then no longer done according to maximizing a divergence measure since the PDF estimates are no longer asymptotically correct. But, the optimization is closely related to the original problem and proves to perform well on the images tried in the results chapter.

4.6 Choice of K

The final algorithm presented in this paper has no parameters which need tuning, as opposed to the kernel methods requiring a very specific bandwidth parameter to function properly. This is one of the key advantages of the new method. However, this does not mean that the flexibility is gone, but rather that one parameter choice seems to perform well on all problems tested. The choices presented here are results of countless tests on various datasets and does not necessarily have a strict mathematical justification.

Initially, the same k was used in both the numerator and denominator of (4.3). Here $k = 1$ (the nearest neighbor) produced good results and

[2]Breaking down using double point precision at around $\Gamma(175)$.

nothing was gained in increasing this choice for k. However, sometimes the optimization algorithm turned out to let *one* cluster grow out of proportion relative to the other clusters. The end result then was typically one cluster dominating the entire data set with $K_{\text{end}} - 1$ small clusters scattered around. From the intuition that the denominator is related to the compactness of each cluster, the choice for k was here tried changed (while keeping $k = 1$ in the numerator). In the end, it turned out that letting this k be equal to the cluster size was a better choice for keeping the clusters compact. This means that in the denominator, each probability is calculated by the inverse of the hyper volume spanned to the point in the cluster furthest away. As a cluster is growing, i.e. new points are incorporated along the border of the cluster, the effect of choosing this k is that more points, of lack of a better word, 'feel' the growth, thereby tending to make the clusters more compact.

With this, the algorithm proved to be more reliable, and all the results presented in the next chapter are based on optimizing the cost function using the nearest neighbor in the numerator and the farthest neighbor in the denominator.

Another solution was attempted to solve the compactness-problem involving introducing a non-linear scaling parameter on the hyper volume. This also helped in keeping the clusters compact, but upon implementing the solution with the different ks, the scaling was obsolete. Nevertheless, it is included in Appendix D as it proved to be an efficient way of smoothing the KNN-PDF estimate.

4.7 Complexity

In previous work on ITC using kernel estimates, the bound on the complexity has for the most part been governed by the need to calculate the distance between each dataset. Any algorithm where this is required typically has an upper bound on complexity given by $\mathcal{O}(N^2)$. This complexity can prove difficult if the data set given is of some size.

The new algorithm presented in this chapter also needs to calculate the distances between each of the data points, so it is at least bounded by $\mathcal{O}(N^2)$. However, since it needs to know the ordering of the neighbors for each data point, a sorting is also required. Given a distance matrix $\boldsymbol{D}$ of size $N \times N$, each of the N columns have to be sorted to find the ordering to the neighboring points. Assuming each sorting can be done in $\mathcal{O}(N \log N)$ time, the overall complexity of this method would become $\mathcal{O}(N^2 \log N)$. This is a very slight increase compared to the kernel methods.

Chapter 5

Results

In this chapter, various results from clustering with the new algorithm is presented. The algorithm has been applied to a wide range of problems and is shown to produce state-of-the-art results without any parameter adjustments.

Where relevant, the new algorithm has been compared to the kernel based method presented in [21]. This is done to keep the comparisons drawn more relevant with regards to comparing KNN and kernel methods. The reliability of the kernel method has proved highly sensitive to the bandwidth parameter σ. Because of this, whenever a comparison between the two methods is done, the kernel method is allowed to do its calculations with two different heuristic approximations to the bandwidth choice; σ_M and σ_S. Here, σ_M is the mean of the standard deviation along each of the dimensions in the data, while σ_S is Silverman's rule-of-thumb approximation [53]. See Section 2.2.2 for more details.

The chapter begins with examples on two synthetic datasets highlighting some of the fundamental properties of the algorithm. Next, a comparative study between the new KNN method and the old kernel method is examined on a suit of standard benchmark datasets. After that, the algorithm is studied working in high dimensional spaces before clustering on a multi spectral satellite image to do cloud screening is investigated. Finally, some datasets with features on different scales are studied before the results are summarized.

5.1 Synthetic Data

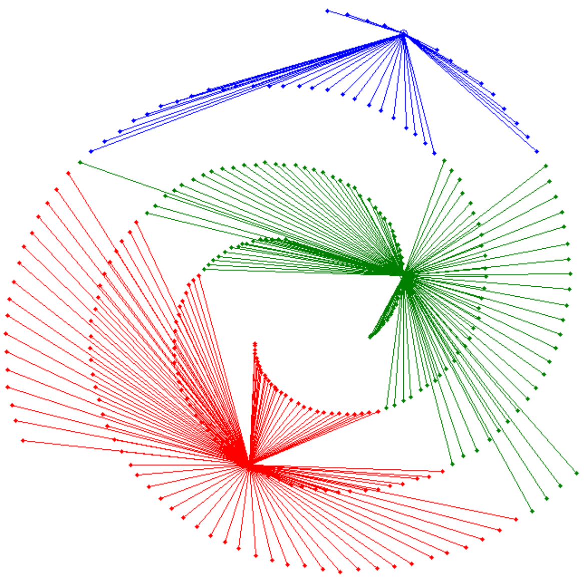
5.1.1 Non-linear

In Figure 5.1, the new algorithm is compared to the K-means algorithm on a dataset which can not be separated by straight lines. As is evident, the K-means performs very poorly, while the new algorithm is fully capable of separating datasets with non-linear boundaries. It should be noted that the kernel method is also capable of handling this dataset if a suitable bandwidth parameter is provided.

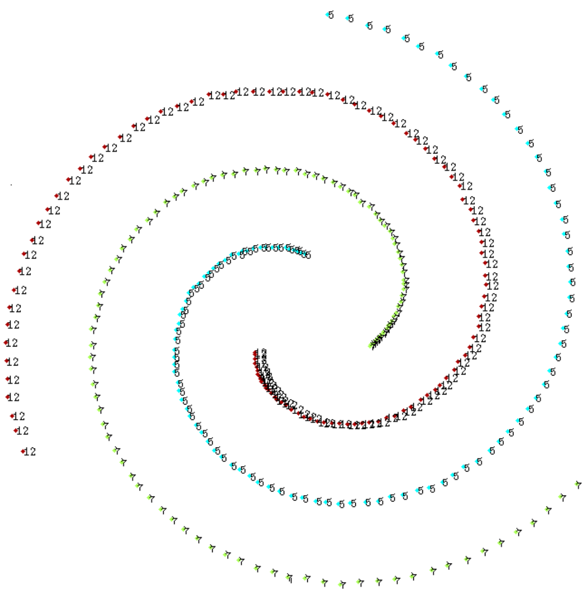
5.1.2 Estimation of number of clusters

As was seen in the previous chapter, the cost function constructed for this algorithm is normalized with regards to number of cluster combinations in each step. This normalization makes the estimated cost (and also the estimated divergence) invariant to the number of clusters the algorithm is currently working on. Because of this, it is in some cases possible to find the true number of clusters in the data by simply evaluating the divergence as a function of number of clusters.

This is done for datapoints from four well separated Gaussian distributions in Figure 5.2. Here a clear peak is seen when the algorithm hits four clusters. This information could be used to make a decision about the true number of clusters in an unknown set. However, the divergence measure proved a bit more erratic when the same approach was attempted on harder datasets. In general it did not produce reliable results when attempted on the benchmark sets analyzed in the next section.

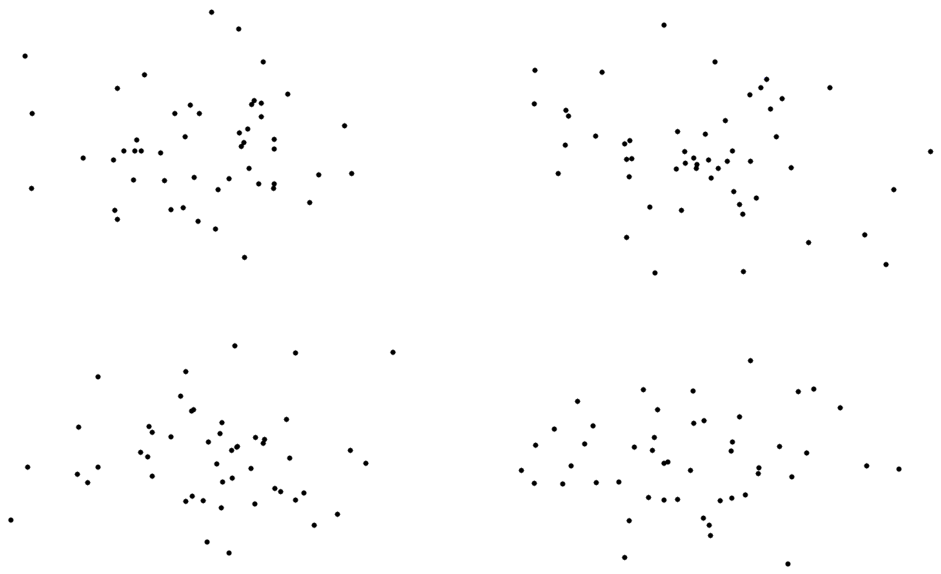


(a) K Means

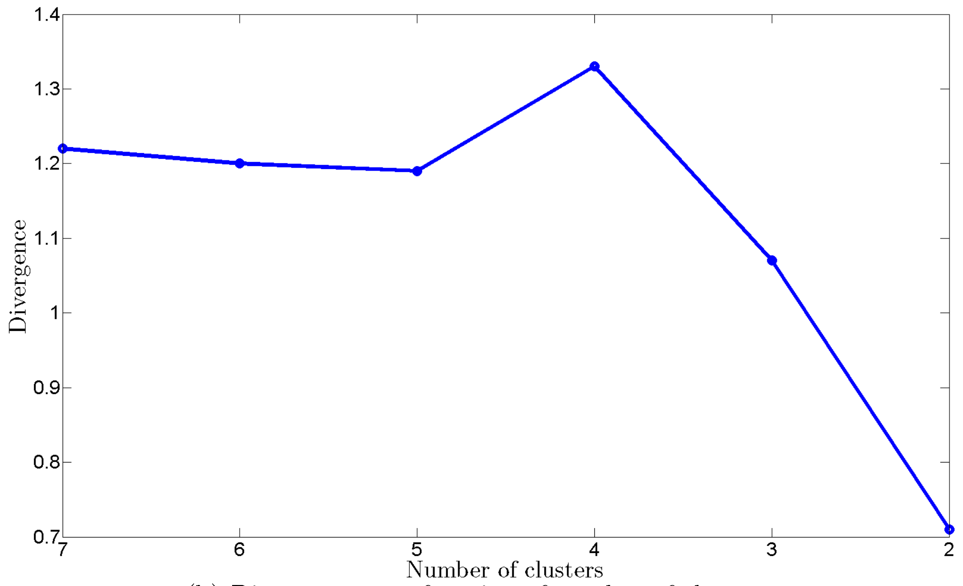


(b) KNN ITC

Figure 5.1: Linear K Means clustering (a) and ITL Clustering with KNN approach (b). The K Means algorithm fails to cluster the 3 spirals while the information theoretic method performs perfectly.



(a) 4 well separated Gaussians.



(b) Divergence as a function of number of clusters.

Figure 5.2: Divergence as a function of number of clusters in optimization algorithm. A clear peak is seen in the divergence when the correct number of clusters is hit.

5.2 Benchmark Sets

In this section, the results from testing on several standard benchmark sets[1] are presented. Since the true cluster information is available for these known sets, the accuracy can be calculated for each of the clustering results. For the new KNN-method, the best result and the result of letting the highest divergence clusterings vote on the final grouping is presented. This is compared with the best result from the kernel approach after sweeping over a wide range of bandwidth parameters, as well as using the two heuristic methods to choose a bandwidth.

A more thorough analysis will be done of the Wine dataset before the the remaining test sets have their results presented. All the data analyzed here has been linearly normalized to have its features in the range $[-1, 1]$ and the algorithms have been given the true number of clusters in the set. The new KNN method has been run with random initialization 50 times on each dataset and the results with the top 10% divergence has voted on the label for each datapoint. For the kernel method, a sweep over a range of bandwidths has been done and the algorithm ran 50 times on each. In addition, the algorithm was run 50 times for each of the bandwidths found using the two bandwidth approximations.

For both methods, K_{init} has been set to 10, while N_{init} chosen such that 80% of the dataset is initially seeded.

5.2.1 Wine

The Wine data is a set of 178 feature vectors in 13 dimensions. Each of the features come from a chemical analysis of wines grown in the same region of Italy. The wine analyzed originating from three different cultivars, thereby producing a dataset where 3 clusters is natural to assume.

In Figure 5.3, the linear relationship between the accuracy of the clustering and the Cauchy-Schwartz divergence is seen. This is a very helpful result for choosing which of the solutions to go for. For any clustering done on an unknown dataset, the accuracy of each the results is not available. This result shows that the highest divergence terms should be considered for a good solution.

Figure 5.4 shows the fundamental problem with kernel methods. Here the clustering is done 50 times for each of the bandwidths in the sweep and the accuracy result analyzed. The figure shows that the kernel method provides stable results only over a limited range of bandwidths. It is also seen that the

[1]Obtained from University of California, Irvine, Machine Learning Repository [12].

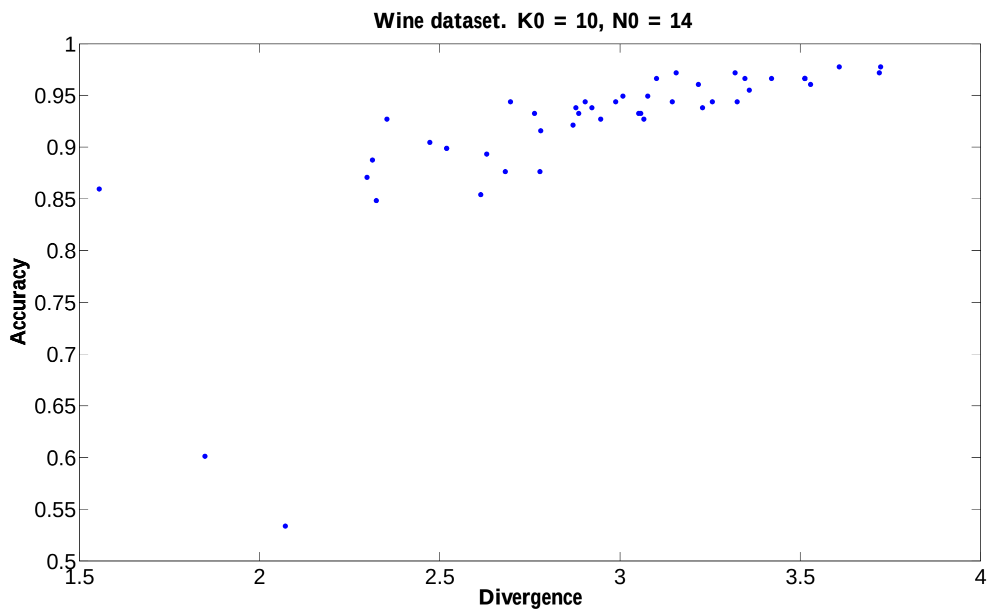


Figure 5.3: Accuracy plotted as a function of CS-divergence. The linear relationship help justify the reason for letting the highest divergence results vote on which cluster assignment is correct.

best accuracy provided by the algorithm comes from a choice of σ where the results overall is unstable. This means that less confidence should be placed on this best result, as it could be a fluke.

In Table 5.1 the accuracy of performing clustering on the dataset is given. It is here seen that the new KNN algorithm performs as good as the best result of the kernel method. As discussed earlier however, having the kernel method actually select this solution is not trivial when the labels of the dataset is unknown. It is a huge advantage that the new algorithm returns this accuracy without needing any prior knowledge.

The weakness of having to select some heuristic approximation to the bandwidth for the kernel method is clearly seen in Table 5.1. Neither σ_M nor σ_S come close to the best result of the KNN method.

KNN		Kernel	
Best	0.978	Best	0.978
Voted	0.978	$\sigma_M = 0.403$	0.923 ± 0.017
		$\sigma_S = 0.266$	0.867 ± 0.058

Table 5.1: The various accuracies of the two methods clustering the Wine dataset.

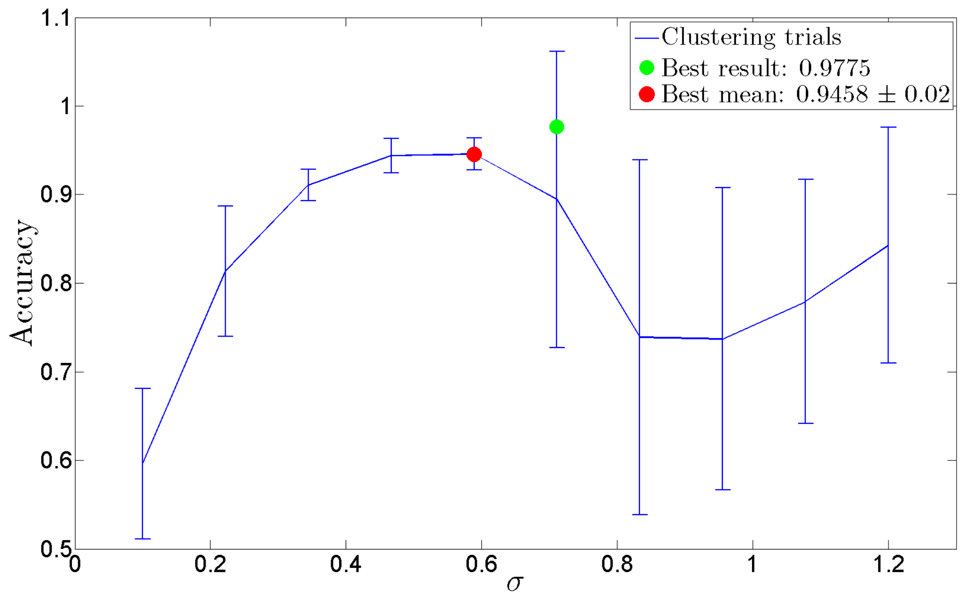


Figure 5.4: Clustering accuracy for kernel method using different kernel widths σ on the Wine dataset. The algorithm is shown to be stable only within a narrow range of bandwidths.

For completeness, the confusion matrix for the voted clustering result with the new method is given in Table 5.2. It is here seen that only four errors are committed, which is comparable to the best results reported for the dataset.

	C1	C2	C3
True 1	58	1	0
True 2	0	68	3
True 3	0	0	48

Table 5.2: Confusion matrix for clustered Wine data with new KNN method.

5.2.2 Iris

The Iris benchmark is a 4-dimensional dataset based on length measurements on the flower of an Iris plant. The 150 sets of measurements are from 3 different types of plant, making the sought number of clusters in the set 3.

The clustering tests were done as with Wine. The new KNN algorithm was run without any parameter adjusting and highest divergence results voted for the final clustering. For the kernel method, the heuristic methods of estimating a bandwidth parameter was used while also reporting the best accuracy obtained from performing a sweep of values.

The results are reported in Table 5.3. Again it is seen that the new algorithm performs as well as the best result reported from the kernel method. At the same time, the KNN method vastly outperforms the kernel method when the heuristically bandwidths are used.

KNN		Kernel	
Best	0.973	Best	0.974
Voted	0.967	$\sigma_M = 0.514$	0.573 ± 0.117
		$\sigma_S = 0.248$	0.920 ± 0.029

Table 5.3: The various accuracies of the two methods clustering the Iris dataset.

5.2.3 Wisconsin breast cancer

The Wisconsin breast cancer dataset consist of 683 10-dimensional features extracted from a digitized image of a breast mass. For each of the features, the task is to evaluate if the mass examined is malignant or benign. This means that the clustering routine should look for 2 groups in the set.

The result of using both the methods discussed in this text is shown in Table 5.4. Here it is evident that both bandwidth σ_M and σ_S has missed their marked and produced poor results. The parameter free new method however, has given very satisfactory results comparable to the overall best found when sweeping a wide range of bandwidths.

KNN		Kernel	
Best	0.958	Best	0.974
Voted	0.955	$\sigma_M = 0.561$	0.695 ± 0.090
		$\sigma_S = 0.313$	0.696 ± 0.057

Table 5.4: The various accuracies of the two methods clustering the Wisconsin breast cancer dataset.

5.2.4 Pima

The Pima dataset is based on 8 physiological measurements on females of Pima Indian heritage living near Phoenix, Arizona, USA. A total of 768 females, all over the age of 21, was examinded. The goal is to evaluate whether the data recorded could help predict if a patient would test positive for diabetes or not.

The results of the clustering is shown in Table 5.5. It is seen that the KNN method outperforms the kernel method for both the heuristic choices of the bandwidth parameter. It is also intereseting that the accuracy is increased by about 1.5% when letting the top 5 clustering results (in term of CS-Divergence) vote on each datapoints' cluster assignment.

This dataset is by far the hardest of the benchmark sets with the best clustering obtained only being correct for 70% of the data. However, this result is comparable to previous work done on the dataset. In [34], a wide range of classifiers was compared on this dataset, and the best result was found to be a classification accuracy of 77.3%. This accuracy is a result of training a classifier using the known labels, so the task is overall easier. Getting a clustering accuracy this close to the best ever reported is good, as no prior knowledge was used.

KNN		Kernel	
Best	0.688	Best	0.710
Voted	0.703	$\sigma_M = 0.323$	0.656 ± 0.018
		$\sigma_S = 0.162$	0.660 ± 0.023

Table 5.5: The various accuracies of the two methods clustering the Pima dataset.

5.3 High Dimensional Data

To illustrate that the new algorithm also can perform clustering in high dimensional spaces, two datasets of images were chosen. Recall from the previous chapter that when the dimensionality of the data increases, calculating the hypervolume can lead to numerical problems for a computer working with finite precision. To work around this, the distance to the k nearest neighbor was used in the optimization, instead of the hypervolume spanned by the distance.

As will be seen in the next two examples, optimizing this related cost function also leads to meaningful clusters.

5.3.1 Frey faces

Out of an movie sequence with 1965 images of a face, 300 frames were drawn at random and tried clustered into two groups. Each image is of dimension 28×20, making the feature vectors of dimension 560; enough for the hypervolume calculation to break down.

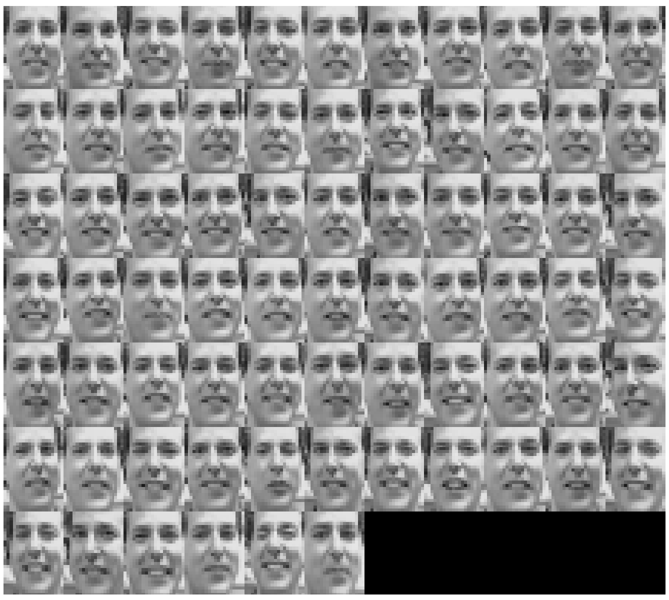
The result of executing the KNN IT clustering starting with $K_{\text{init}} = 8$ and seeding 80% of the dataset is seen in Figure 5.5. Here the 560 long feature vectors have been reshaped back into images and displayed.

From the figure, it appears that the algorithm as found two natural clusters in the data. One where the face is smiling and one where it is not.

5.3.2 Handwritten digits

Using the US Postal Service dataset for handwritten digits, 30 images of the number zero and 30 images of the number one was drawn at random. Each image is of dimension 16×16, making each feature vector 256-dimensional.

The result of clustering this dataset (with the same starting parameters as in the previous section), is shown in Figure 5.6. The clustering algorithm is here seen to perform without error. However, it should be noted that these two digits look quite different, making the clustering task easier. The result was more unstable when trying other, more similar looking digits.


(a) Cluster 1: Smiling.

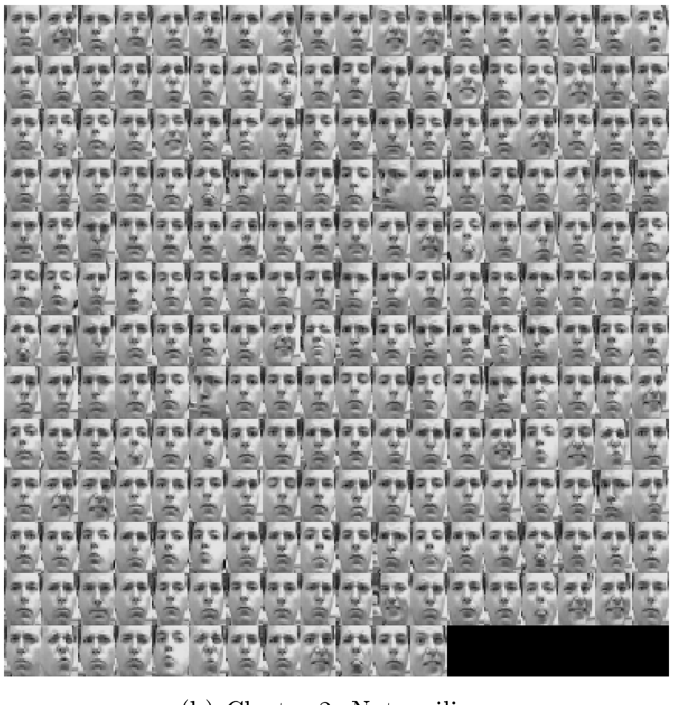
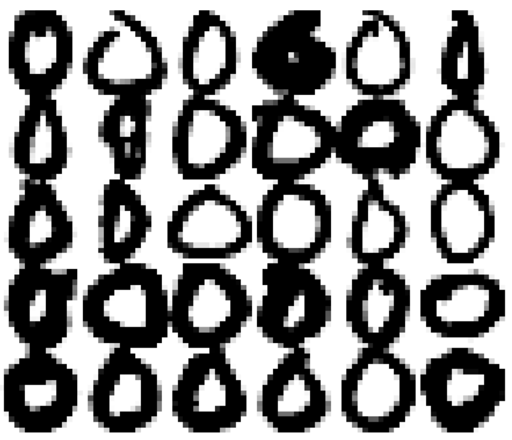
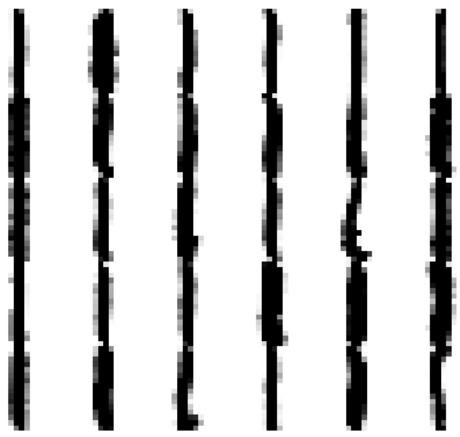

(b) Cluster 2: Not smiling.

Figure 5.5: Clustering of 300 randomly drawn Frey-faces. Cluster 1 appears to always be smiling, while the other cluster does not.



(a) Cluster 1: All zeros.



(b) Cluster 2: All ones.

Figure 5.6: Clustering of randomly drawn zeros and ones from the USPS dataset. The result is without errors.

5.4 Cloud Screening

In this section, the algorithm is tried on a multispectral satellite image to do cloud screening. This is an important task in remote sensing when working in frequency ranges where energy is absorbed by the presence of water droplets in the air. In [16], Gómez-Chova et al. attempted to pre-process land-cover images using Kernel PCA and Kernel ECA before performing cloud screening [26]. The images are taken from the MERIS instrument on board the Environmental Satellite (ENVISAT)[2]. Six physically inspired features obtained from the MERIS bands was used [15].

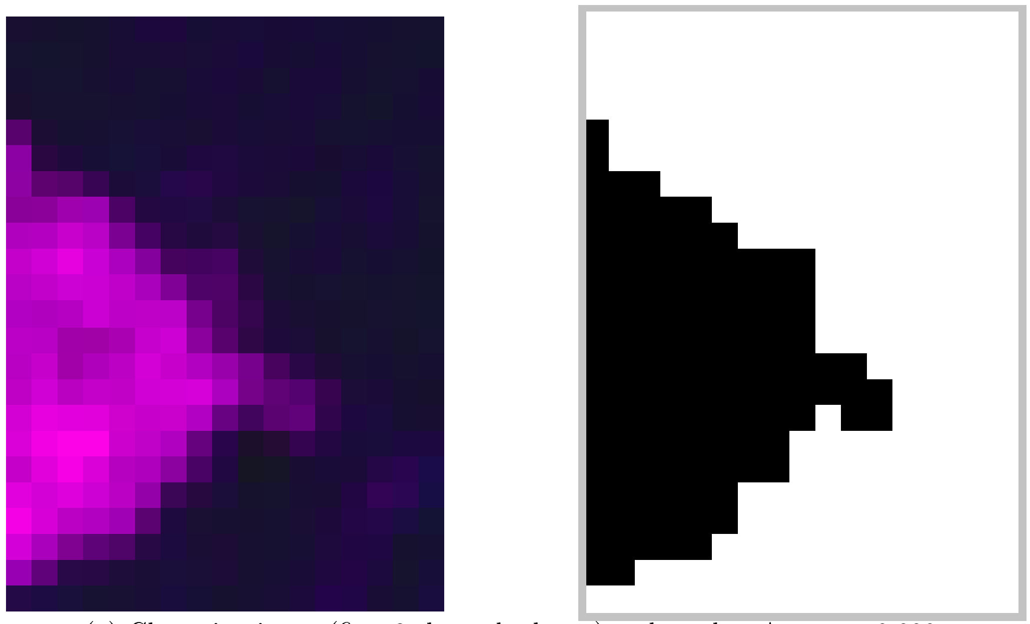
The new KNN ITC algorithm is here used to find two clusters in the original image. In theory, the entire image could have been clustered directly, but since it has a resolution of $\approx 1000 \times 1000$, approximately 500 billion distances would have to be calculated. Instead, the known labels of the image is used to draw 150 pixels of clouds and 150 pixels of land cover at random, similar to [16]. This reduced dataset is then clustered into two groups using the new KNN algorithm. With the clustering results, a 1-NN classifier trained and used to classify the rest of the pixels in the image.

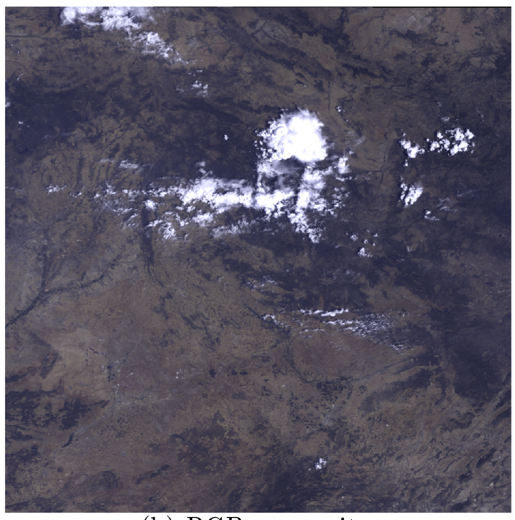
The result of this classification routine is seen in Figure 5.7. In (a), a small section of the original image has been cropped out and clustered directly. This is done to illustrate that the clustering algorithm applied directly produce reliable results.

In (c), the result of using 300 clustered points in a 1-NN classifier is shown. The accuracy of 0.989 is comparable to the best results reported in the original article of 0.9941 [16]. In the article, a lot of other clustering attempts was also made, and they all obtained worse results than the KNN ITC. For instance, running a kernel k-means clustering gave an accuracy of 0.9622. Lastly, it should be noted that the best result reported in [16] was obtained from performing cross validation when sweeping the bandwidth parameter over 5 orders of magnitude to find the best one. The KNN algorithm presented in this thesis worked without any parameter tuning.

This section is concluded with some words on the merits of training the classifier with the clustering result and performing cloud screening. The classifiers used in the original article was trained based on "ground truth" labels from manual inspection by humans. As the density of clouds fall along a continuous gradient, it is natural to assume that the labels set by humans are such that only very dense clouds have been marked. This causes any classifier trained with the labels to focus on assigning the dense areas as

[2]Images provided by Luis Gómez-Chova from The Image Processing Laboratory (IPL), Universitat de València, Spain.


(a) Clustering input (first 3 channels shown) and results. Accuracy 0.990.


(b) RGB composite.

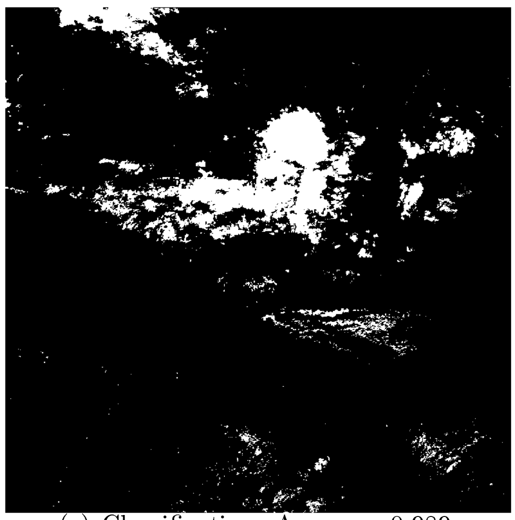

(c) Classification. Accuracy 0.989.

Figure 5.7: Classification of clouds with training from clustering results. Image is taken over Spain (BR-2003-07-14).

cloud regions. Pixels falling somewhere between dense clouds and clear sky tend to not be represented in the training set.

If instead the training set is a result of clustering randomly drawn pixels into two groups, it is likely that information exists in the training set regarding semi-transparent cloud regions. Also, in having performed the clustering, a decision has been made regarding if these pixels are more similar to dense clouds, or clear sky (any signature from typical land covers in the region).

In Figure 5.8, the ground truth data is compared with the classification result when training has been done using the clustering results. Examining the figure, it is interesting to note that the regions where the classification differ from the truth map tend to look like typical cloud formations. This is especially seen in the lower right corner of (b) where maybe a cloud is forming. It is therefore possible that more is gained from knowing less when examining classification tasks of this nature. It would have been interesting to examine further if some of the errors made by the new algorithm could in fact be forming clouds.

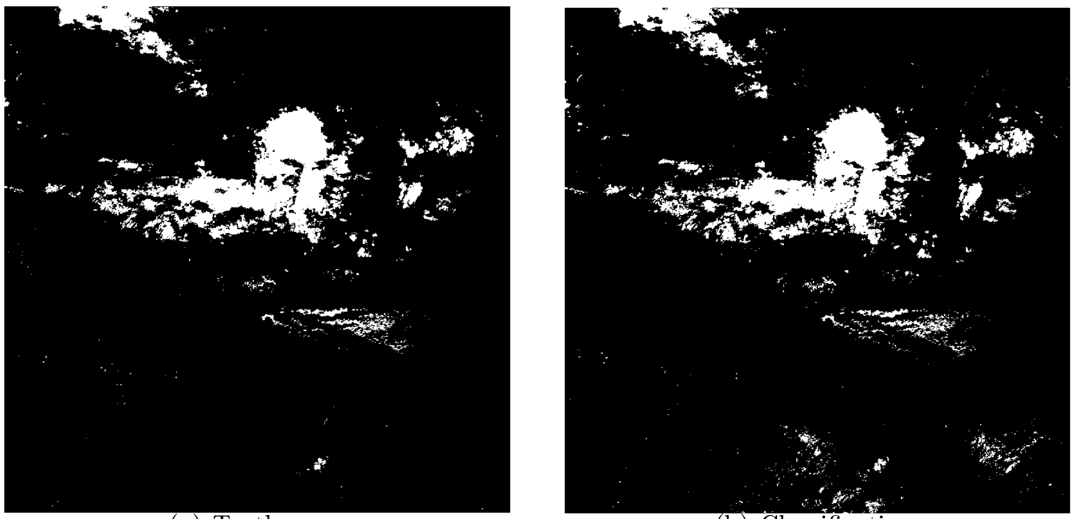

(a) Truth map. (b) Classification.

Figure 5.8: Comparison of truth map and classification with clustered training data.

5.5 Data on Different Scales

This last section of the results chapter will focus on datasets where the scales vary. Traditional kernel methods typically struggle to do anything useful in these cases, as no single bandwidth parameter can capture all of the structure in the data. As will be seen, the adaptive nature of the PDF estimation in the new KNN-approach help the optimization such that meaningful clusters are obtained.

Two synthetic datasets will first be examined to highlight the problem. Then, a simple example from the real world with data on different scales is presented. In all the cases looked at, the new method is shown to handle the clustering far better than the kernel approach.

5.5.1 Gaussian distributions

In the first example, 300 points are generated from 3 Gaussian distributions, 100 points from each. These distributions are created such that the covariance matrices are scaled to be very different, with the most spread class having 5 orders of magnitude higher variance than the most compact class. The data is generated as follows

$$\boldsymbol{X}_1 \sim N\left(\begin{bmatrix}0\\0\end{bmatrix}, 10^{-2}\boldsymbol{I}\right), \quad \boldsymbol{X}_2 \sim N\left(\begin{bmatrix}4\\0\end{bmatrix}, 10^{0}\boldsymbol{I}\right), \quad \boldsymbol{X}_3 \sim N\left(\begin{bmatrix}20\\0\end{bmatrix}, 10^{2}\boldsymbol{I}\right)$$

This means that there does not exist some specific bandwidth which the kernel-ITC algorithm can use to capture the structure of the data completely. For the kernel method, the bandwidth was chosen to be equal to the variance of the middle class. As before, the KNN method does not need any parameter adjustment.

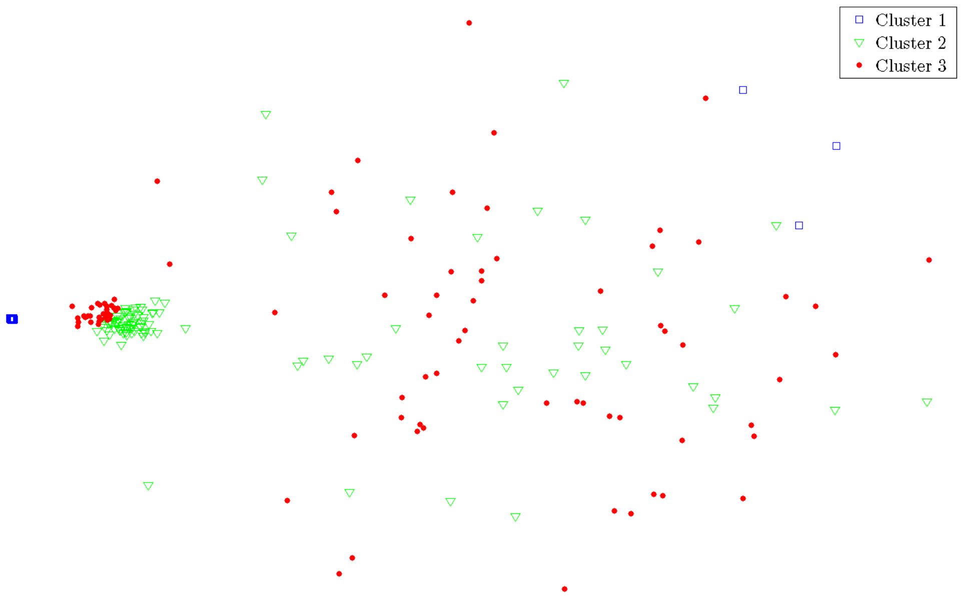
A typical clustering result is shown in Figure 5.9 where the kernel method obtains an accuracy[3] of 0.76. Notice that the kernel method has problems clustering the most spread class. It has even done some errors in the middle cluster, even though the bandwidth is set to be equal to this clusters' variance. The KNN method is seen to perform without any mistakes and has an accuracy of 1.

5.5.2 Color distributions

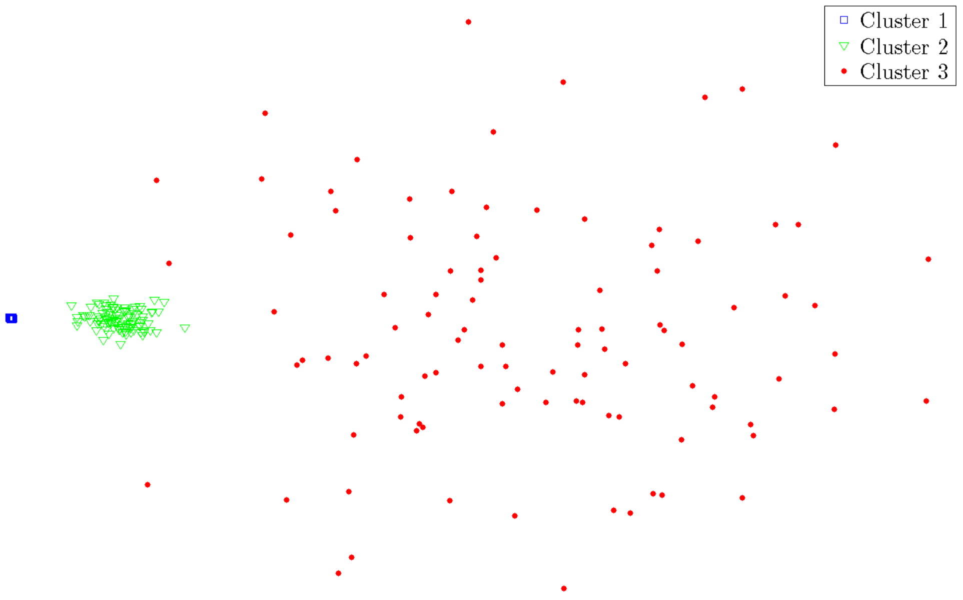
The dataset presented here is based on an RGB image of two parts, where one part is monochromatic, while the other has a huge variance in the colors. The image constructed is shown in Figure 5.10(a).

Clustering this image proves hard for a kernel method, as the color of one of the clusters is spread, while the other is very compact. This is seen

[3]Running this clustering several times, the accuracy of the kernel method never exceeded 0.80.



(a) Kernel: Accuracy 0.76



(b) KNN: Accuracy 1

Figure 5.9: Clustering on mixture of Gaussians on different scales. The fixed kernel size in (a) causes poor performance, while the KNN method performs perfectly.

in Figure 5.10(b). For each pixel, the three color channels as well as the pixels' position in the image is used as features for the clustering algorithms. Each feature is normalized to be in the range $[-1, 1]$ and the clustering is attempted with $K_{\text{init}} = 6$ and N_{init} such that 80% of the datapoints are included in the initial seeding.

Assuming no knowledge about the process which generated the image, the kernel methods have to estimate the bandwidth to use. As previously done in this chapter, the kernel method is allowed to try to cluster the data using two different estimates of the bandwidth; σ_M and σ_S. The KNN method clusters without any parameters needing to be specified.

The clustering results can be seen in (c). The kernel method fails completely when using Silverman's-rule-of-thumb, σ_S, as the bandwidth of the kernel. It does better when instead the mean variance along each dimension, σ_M, is used. However the result is by far the best for the KNN clustering with only 6 pixels (out of 400) being assigned to the wrong cluster. For comparison, σ_M produced 31 errors and σ_S around 200.

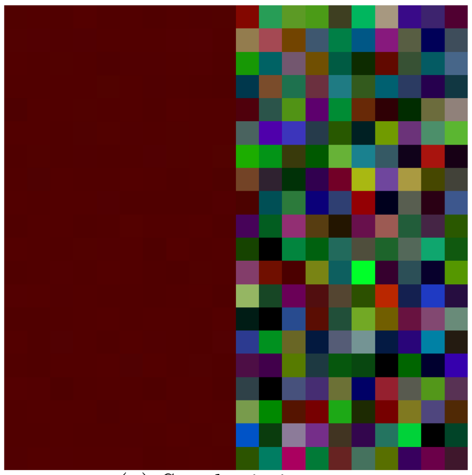
5.5.3 Texture and object

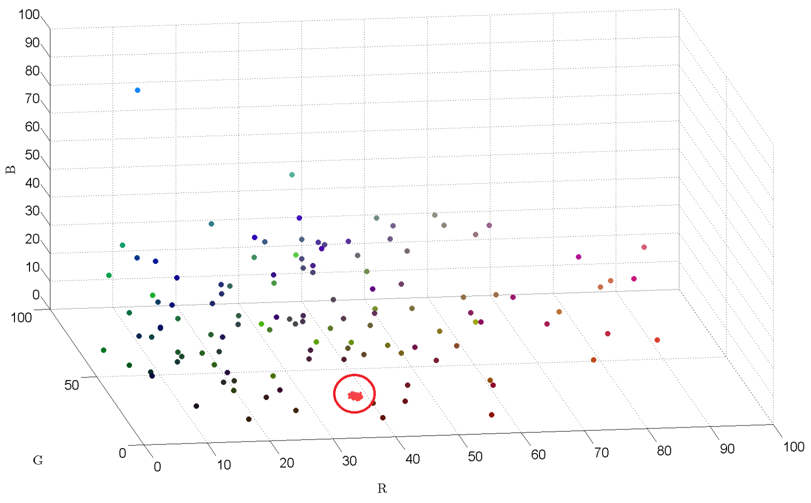
The experiments with data on different scales is concluded with a real life example. In Figure 5.11(a), a scene where a monochromatic gray object is lying on a rough texture is seen. The color of the texture is varying greatly compared to the object. This can be seen in (c), where the RGB value of each pixel of the downsized image constitutes a point in $\mathbb{R}^3$.

This clustering task is recognized as one where the two clusters are on different scales. The position of each pixel is included as features and the dataset normalized to be in the range $[-1, 1]$ before the clustering is carried out with the two competing algorithms.

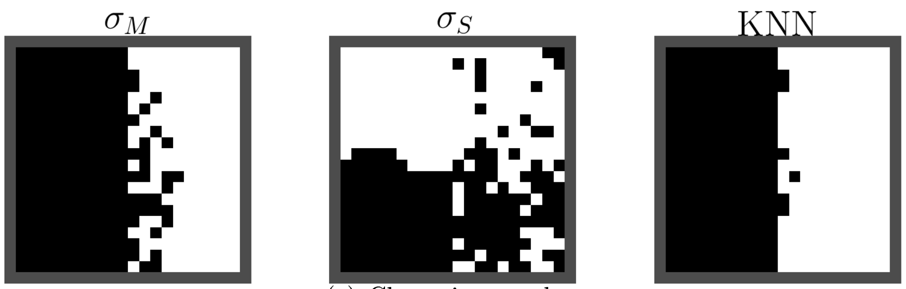
The kernel method is allowed to do the clustering with the two approximations to the bandwidth seen previously, σ_M and σ_S, while the KNN method is as always run without specifying any parameters.

It is in (d) seen that the kernel clustering fails to separate out the object for both choices of bandwidth. The KNN approach however, returns a very satisfactory result where the object in the image has been found. One pixel to the right of the object has been clustered to the wrong group. This is likely due to the downsized image (b) having an extra bright/gray pixel in that area as well as it being close to the object.


(a) Synthetic image.



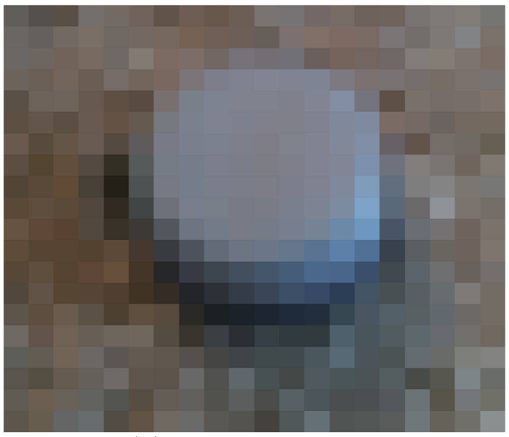
(b) Color spread. Left side of image marked.

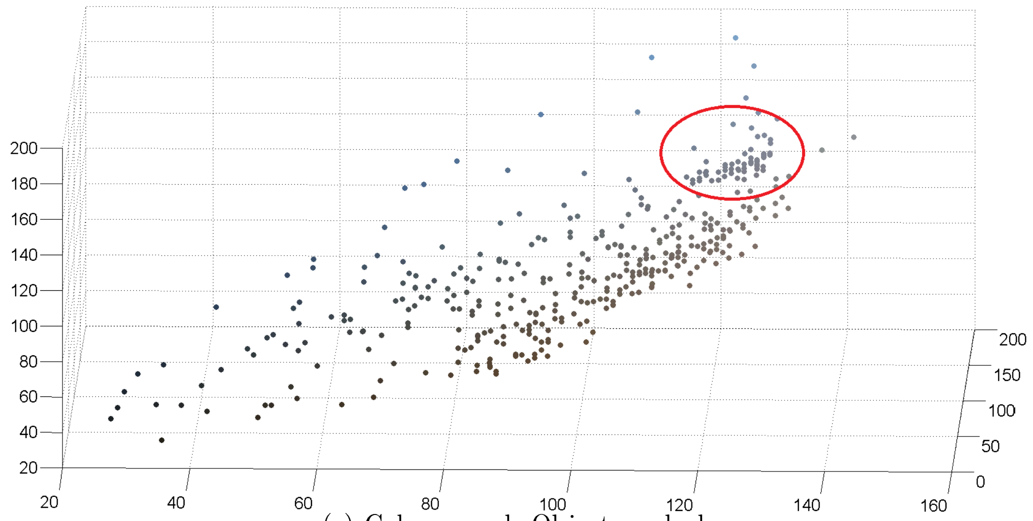


(c) Clustering results.

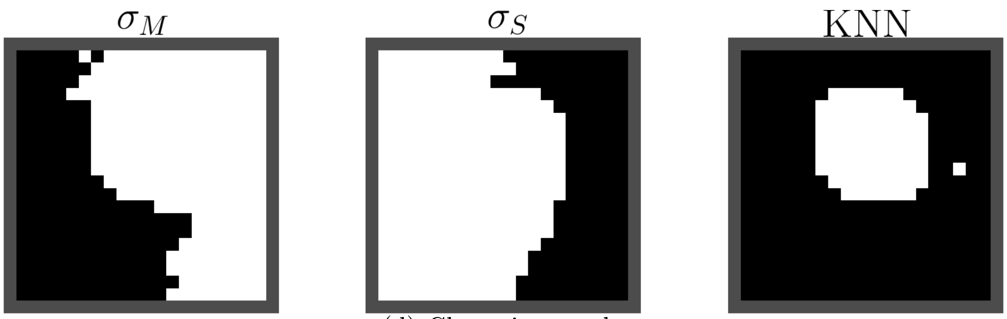
Figure 5.10: Clustering of synthetic image with different color spreads. The KNN algorithm outperforms the kernel method for both choices of bandwidth parameter.

(a) Original image.


(b) Downsized image.



(c) Color spread. Object marked.



(d) Clustering results.

Figure 5.11: Clustering of image with object and texture. Both bandwidths of the kernel method fails to cluster the object.

5.6 Summary of Results

In this chapter, results of the new KNN approach to ITC has been investigated. It was seen that the optimization routine was able to handle datasets of non-linear structure and that the cost function could in some cases be used to estimate how many clusters a dataset is made of.

Running the algorithm on the benchmark datasets, it consistently outperformed the kernel method. Here it is interesting to note that the clustering accuracy when letting the results with the highest CS-divergence vote, was always on par with the best accuracy reported. This makes the new algorithm capable of finding a good cluster result in an unknown dataset by evaluation of the divergence and performing the voting scheme to obtain the final cluster labels.

For the high dimensional data, the related optimization task from evaluating distance out to the k nearest neighbor, instead of using the hypervolume formula, was shown. This proved to produce meaningful clusters of the images looked at. For the Frey-faces dataset, the clustering seemed to separated based on the facial expression into smiling and non-smiling images. The handwritten digits got clustered as expected with all the ones in one cluster and all the zeros in another.

When training a cloud cover classifier based on the clustering results from the new algorithm, the accuracy was seen to be close to that of a recent research paper. It was noted that this research paper used a kernel approach where a sweep of different bandwidth parameters had to be tried with cross validation. This makes the overall training process more complex than with the new KNN method.

Further more, the training of the classifier based on the clustering result revealed some new structures in the image. It was noted that these resembled clouds and it was speculated whether something new had been learned by letting a clustering result dictate the classification.

The fundamental problem of clustering on datasets where the clusters are on different scales was tackled next. Here, the kernel method was seen to be helpless in trying to cluster all the datapoints correctly with one set bandwidth parameter. The new KNN algorithm performed far better, producing error free, or close to error free, clusters.

Lastly it should again be stressed that all the results produced by the KNN algorithm in this chapter was done without any parameters being tuned (except for the optimization algorithms' parameters $K_{\text{init}}, N_{\text{init}}$ and K_{end}, but the clustering results proved quite insensitive to these). Regardless of which dataset the algorithm worked on, it was simply set to run multiple times (as it could run into a local optima) and the end result was proved to be good and reliable. Avoiding the whole uncertainty of whether or not the current parameter choice is the absolute best choice proved really helpful in reducing the complexity of the testing.

Chapter 6

Conclusion

This thesis has explored the field of Information Theoretic Learning (ITL) from the perspective of using K Nearest Neighbor (KNN) estimates to do learning. Specifically, a new clustering algorithm has been introduced which uses these principles instead of the traditional kernel estimates.

Using this new clustering algorithm has been shown to consistently produce results comparable to the best of previous research. These results are obtained without the tuning of of any parameters. This is a major advantage, as the task of clustering a dataset typically means that no known labels, which could be used to find usable parameters, are available.

Another advantage of performing Information Theoretic Clustering (ITC) with a KNN approach is the estimators robustness when handling data where the clusters are on different scales. Several examples of datasets where the traditional kernel method breaks down has been examined. In all of these cases, the KNN approach has provided very good results.

Further work could be to investigate if the CS-divergence estimate of the final clustering could be made more stable. The positive correlation between the divergence and the accuracy should be kept, while removing the phenomenon of some local minima solutions reporting high divergences. If this is managed, the voting among the top results would generally be done with selected constituents of higher quality.

It could also be interesting to do more research on using the divergence measure as a means of selecting the number of clusters in an unknown dataset. The experiments done on this showed promise, but on harder datasets the results were often inconclusive.

Appendix A

Parametric Density Estimation

To do any sort of parametric estimation, one has to assume a model for the data. Given a d-dimensional stochastic vector $\boldsymbol{X}$ generated from a probability density function $p(\boldsymbol{x};\boldsymbol{\theta})$ with q unknown parameters $\{\theta_1, \theta_2, ..., \theta_q\}$ in $\boldsymbol{\theta}$. This (parametric) model is completely described by the unknown $\boldsymbol{\theta}$ parameters, which have to be estimated.

The most common method for estimating the models parameters, is the Maximum Likelihood (ML) method. Given a dataset with N realizations of the stochastic variable $\boldsymbol{X}$, $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$, the joint probability function can be constructed, if statistically independent realizations are assumed, by

$$p(X;\boldsymbol{\theta}) = p(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N;\boldsymbol{\theta}) = \prod_{k=1}^{N} p(\boldsymbol{x}_k;\boldsymbol{\theta}) \tag{A.1}$$

For a given dataset (A.1), called the likelihood function, is strictly dependent of $\boldsymbol{\theta}$. The maximum likelihood method involves estimating $\boldsymbol{\theta}$ so that the likelihood function is maximized

$$\hat{\boldsymbol{\theta}}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p(X;\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{k=1}^{N} p(\boldsymbol{x}_{\boldsymbol{k}};\boldsymbol{\theta}) \tag{A.2}$$

Finding a maximum of this function is, in the cases where the underlying model is well-behaved (differentiable and unimodal), often done by evaluating for which $\boldsymbol{\theta}$ the gradient is $\mathbf{0}$:

$$\hat{\boldsymbol{\theta}}_{\mathrm{ML}} : \frac{\partial \prod_{k=1}^{N} p(\boldsymbol{x}_k;\hat{\boldsymbol{\theta}}_{\mathrm{ML}})}{\partial \boldsymbol{\theta}} = \mathbf{0} \tag{A.3}$$

To make the calculations easier, the monotonically increasing logarithmic function can be applied to (A.3) without changing the position of $\hat{\boldsymbol{\theta}}_{\mathrm{ML}}$. By doing this, the product of all the realizations of $\boldsymbol{X}$ instead becomes a sum, and a new expression for $\hat{\boldsymbol{\theta}}_{\mathrm{ML}}$ is found

$$\sum_{k=1}^{N} \frac{\partial \ln p(\boldsymbol{x}_k; \hat{\boldsymbol{\theta}}_{\mathrm{ML}})}{\partial \boldsymbol{\theta}} = \sum_{k=1}^{N} \frac{1}{p(\boldsymbol{x}_k; \hat{\boldsymbol{\theta}}_{\mathrm{ML}})} \frac{\partial p(\boldsymbol{x}_k; \hat{\boldsymbol{\theta}}_{\mathrm{ML}})}{\partial \boldsymbol{\theta}} \overset{!}{=} \mathbf{0} \tag{A.4}$$

For the case of estimating the parameters of a single Gaussian distribution, completely described by $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, it can be shown [57] that the maximum likelihood estimators for $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ given N datapoints from the distribution are

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i = \bar{\boldsymbol{x}} \tag{A.5}$$

$$\hat{\boldsymbol{\Sigma}}_N = \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T \tag{A.6}$$

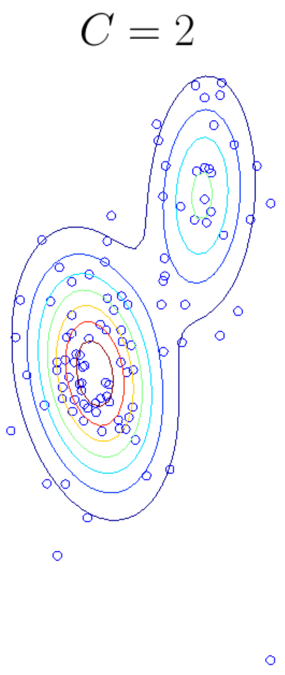
where the subscript N in $\boldsymbol{\Sigma}_N$ refers to the fact that the estimator divides by N. By evaluating the expectation of these estimators, it can be shown that $\hat{\boldsymbol{\Sigma}_N}$ is only asymptotically unbiased, as

$$E[\hat{\boldsymbol{\Sigma}_N}] = \frac{N-1}{N} \boldsymbol{\Sigma} = \boldsymbol{\Sigma} - \frac{1}{N} \boldsymbol{\Sigma}$$

Because of this, it is common to divide by $N-1$ instead of N in (A.6) to get an unbiased estimate of $\boldsymbol{\Sigma}$. Having unbiased estimates of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ means that as the number of datapoints in the estimate grows, the estimates converge to the true parameters [57] [2]. This is also true for asymptotically correct estimates, but for small N here, an error will be done in the estimation.

The solution found by evaluating (A.4), often yield good results if the correct underlying model for the data is assumed and this model has the nice properties of being differentiable and unimodal (as in the case leading to (A.5) – (A.6)). If these assumptions are not fulfilled, different numerical and/or iterative methods (like the EM-algorithm) could be deployed to still obtain a solution. There are however no guarantees that the solution found is the optimal one; it could be a local maximum (for which an example is given in Figure A.1) of $\boldsymbol{\theta}$ (or even a minimum), it could be an inflection point or the solution could lie in the boundaries of the solution space.

The classic example when discussing parametric models is the task of estimating the probability density function given data drawn from a Gaussian mixture model. The distribution of a random variable X generated such a



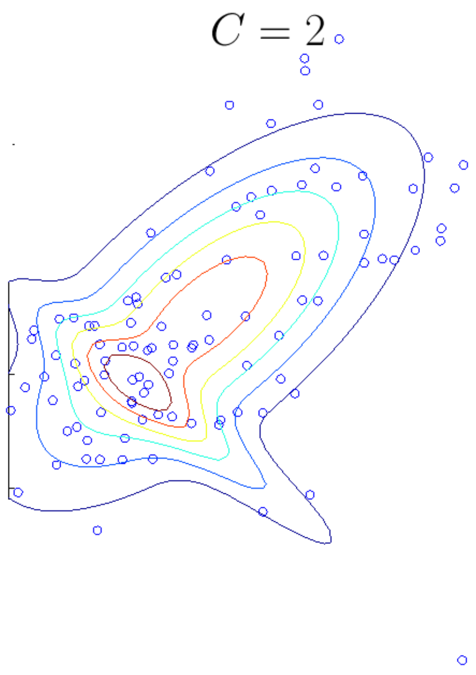


Figure A.1: EM-algorithm which has converged. To the left a correct solution is found. The on on the right has found a local maximum.

model of C classes ($\{\omega_1, ..., \omega_C\}$) is given by

$$X \sim \sum_{i=1}^{C} P(\omega_i) N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \tag{A.7}$$

where $P(\omega_i)$ is the probability of class i and $\sum_{i=1}^{C} P(\omega_i) = 1$.

Assuming X is generated by the model above, there are approximately $C^2/2 + (d+1)C$ parameters to estimated. The quality of the solution is also highly dependent on assuming the correct C before proceeding with the estimation. The standard way of finding the parameters of this model is the EM-algorithm. To illustrate some of the problems with parametric approximations, a synthetic dataset of 100 points is generated from a mixture of 3 classes given by

$$\boldsymbol{X} \sim 0.4N\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right) + 0.3N\left(\begin{bmatrix}3\\3\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right) + 0.3N\left(\begin{bmatrix}6\\0\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)$$

Next, the EM-algorithm is applied in order to estimate all the parameters the model above requires. The number of classes C thought to be correct generally have to be given to the algorithm. If C is not chosen correctly, the model is likely to either overfit or underfit the dataset.

Using the synthetic dataset, the EM-algorithm is run three times, with different guesses for C. The result of fitting these different models is visualized in Figure A.2. It is in this figure clearly seen that the data is underfitted if $C = 2$ and overfitted if $C = 6$. Using the correct $C = 3$ however, yields satisfactory results. In closing, it should be noted that there exists well established techniques for choice of reasonable a C, such as the Akaike or Bayesian Information Criterion [1].

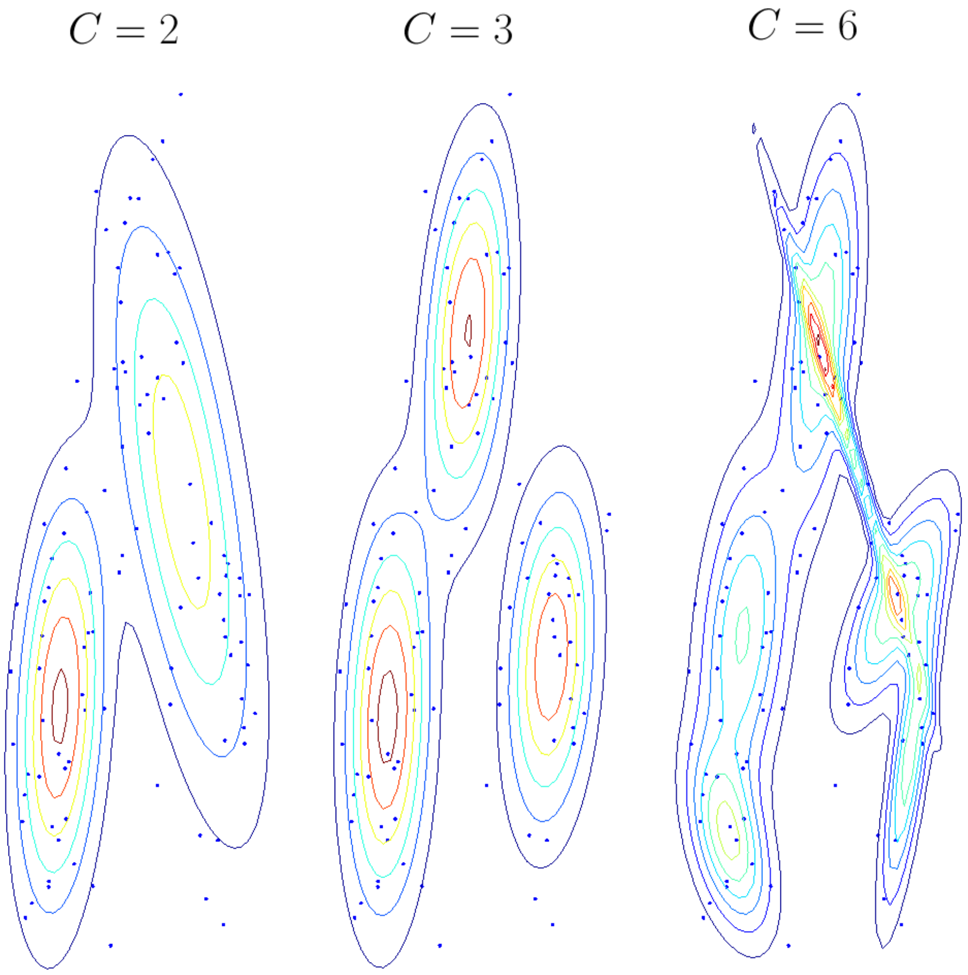


Figure A.2: EM-algorithm with different guesses for C.

Appendix B

Bayes Decision Theory

In Bayes Decision Theory, the task is to assign a unknown vector $\boldsymbol{x} \in \mathbb{R}^d$ to one of C classes, $\{\omega_1, \omega_2, ..., \omega_C\}$, in a meaningful way. A reasonable choice of this assignment is to let $\boldsymbol{x}$ be assign to the class which is most probable. That is, given $\boldsymbol{x}$, assign it class label ω_i if $P(\omega_i|\boldsymbol{x}) > P(\omega_j|\boldsymbol{x}) \ \forall \ j \neq i$. In this expression, the probability $p(\omega_j|\boldsymbol{x}),\ j = 1, ..., C$ is not readily available and to obtain it, Bayes rule, given in (B.1), is used.

Given C classes with known distributions $p(x|\omega_i),\ i = 1, ..., C$ and known prior probabilities $P(\omega_i),\ i = 1, ..., C$, it is possible to estimate the probability of an unknown $\boldsymbol{x}$ coming from class ω_i by using Bayes formula, given by

$$P(\omega_i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\omega_i)P(\omega_i)}{p(\boldsymbol{x})} \tag{B.1}$$

with

$$p(\boldsymbol{x}) = \sum_{i=1}^{C} p(\boldsymbol{x}|\omega_i)P(\omega_i)$$

Now with the assumptions stated above, a new datapoint $\boldsymbol{x}$ can be assigned to a class i with the rule

$$\boldsymbol{x} \rightarrow \omega_i \text{ if } P(\omega_i)p(\boldsymbol{x}|\omega_i) > P(\omega_j)p(\boldsymbol{x}|\omega_j) \ \forall \ j \neq i \tag{B.2}$$

It is noted that the denominator in (B.1) has been dropped as it is equal for all classes and plays no role in the evaluation of the most probably class.

It can be shown that the decision boundaries given by (B.2) minimizes the probability of error P_e [54]. For a random vector $\boldsymbol{x} \in \mathbb{R}^d$, if the region where it is assign to the class $\omega_i \in \{\omega_1, ..., \omega_C\}$ is denoted R_i, then the total probability of error is given by

$$P_e = \sum_{i=1}^{C} \int_{R_i} \left(\sum_{\substack{j \,\in\, \{1,\,\dots,\,C\} \\ j \,\neq\, i}} p(\boldsymbol{x}|\omega_j) P(\omega_j) \right) d\boldsymbol{x} \tag{B.3}$$

In this formula, the probabilities of all misclassifications of $\boldsymbol{x}$ possible have been summed up to give the total probability of misclassification.

With this, the theory of generating decision boundaries could be said to be complete, as (B.2) can be shown to minimize (B.3) [54], [9]. However, in many real life applications the different classes are not deemed equally important. A classic example of this can be drawn from medicine, where a tumor classifier has to weight the two outcomes (malignant or benign) differently; the consequences of classifying a malignant one as benign are far worse than the other way around. To let the classifier take into account this consideration, different weights on each of the possible classifications is introduced and stored in a loss matrix $\boldsymbol{\Lambda}$. This matrix is defined as

$$\boldsymbol{\Lambda} = \begin{bmatrix} \lambda_{1,1} & \lambda_{1,2} & \dots & \lambda_{1,C-1} & \lambda_{1,C} \\ \lambda_{2,1} & \lambda_{2,2} & \dots & \lambda_{2,C-1} & \lambda_{2,C} \\ \vdots & & \ddots & & \vdots \\ \lambda_{C-1,1} & \lambda_{C-1,2} & \dots & \lambda_{C-1,C-1} & \lambda_{C-1,C} \\ \lambda_{C,1} & \lambda_{C,2} & \dots & \lambda_{C,C-1} & \lambda_{C,C} \end{bmatrix} \tag{B.4}$$

where $\lambda_{i,j}$ gives the weight of misclassifying something from class ω_i as ω_j. Note here that the diagonal of the matrix corresponds to the correct classifications, and these parameters are usually set to zero.

These parameters could be said to represent the relative risk of the different misclassification the system can make. Given an unknown $\boldsymbol{x}$ to be labeled, the system now tries to minimize the average risk (instead of the average classification error) with the risk defined as

$$r = \sum_{i=1}^{C} \int_{R_i} \left(\sum_{\substack{j \,\in\, \{1,\,\dots,\,M\} \\ j \,\neq\, i}} \lambda_{j,i} p(\boldsymbol{x}|\omega_j) P(\omega_j) \right) d\boldsymbol{x} \tag{B.5}$$

The decision boundaries which minimize the risk, is similar to the ones which minimize the average error[1]. The only thing that has changed is that now, each term is scaled with its corresponding λ-value and the minimum

[1]In fact, minimizing the average error can be thought of as minimizing the average risk with all elements of Λ set equal to some arbitrary constant.

(risk) is sought rather than the maximum probable class. With this in mind, the classifier given below is obtained.

$$\boldsymbol{x} \to \omega_i \text{ if } \sum_{k=1}^{C} \lambda_{i,k} P(\omega_i) p(\boldsymbol{x}|\omega_i) < \sum_{k=1}^{C} \lambda_{i,k} P(\omega_j) p(\boldsymbol{x}|\omega_j) \ \forall\ j \neq i \tag{B.6}$$

To gain some intuition on how this classifier works, the best way is to simplify the dimensionality of $\boldsymbol{x}$ to 1 ($\boldsymbol{x} \in \mathbb{R}$) and limit the problem to the two class case. The decision boundary ends up as

$$\begin{aligned} \boldsymbol{x} \to \omega_1 \quad \text{if} \quad & \lambda_{1,1} p(\boldsymbol{x}|\omega_1) P(\omega_1) + \lambda_{2,1} p(\boldsymbol{x}|\omega_2) P(\omega_2) \leq \\ & \lambda_{1,2} p(\boldsymbol{x}|\omega_1) P(\omega_1) + \lambda_{2,2} p(\boldsymbol{x}|\omega_2) P(\omega_2) & \text{(B.7)} \\ \boldsymbol{x} \to \omega_2 \quad \text{if} \quad & \lambda_{1,1} p(\boldsymbol{x}|\omega_1) P(\omega_1) + \lambda_{2,1} p(\boldsymbol{x}|\omega_2) P(\omega_2) > \\ & \lambda_{1,2} p(\boldsymbol{x}|\omega_1) P(\omega_1) + \lambda_{2,2} p(\boldsymbol{x}|\omega_2) P(\omega_2) & \text{(B.8)} \end{aligned}$$

Evaluating the inequality (B.7)

$$\begin{aligned} \lambda_{1,1} p(\boldsymbol{x}|\omega_1) P(\omega_1) + \quad & \leq \quad \lambda_{1,2} p(\boldsymbol{x}|\omega_1) P(\omega_1) + \\ \lambda_{2,1} p(\boldsymbol{x}|\omega_2) P(\omega_2) \quad & \qquad \lambda_{2,2} p(\boldsymbol{x}|\omega_2) P(\omega_2) \\ \frac{p(\boldsymbol{x}|\omega_1)}{p(\boldsymbol{x}|\omega_2)} \quad & \geq \quad \frac{P(\omega_2)}{P(\omega_1)} \frac{[\lambda_{2,1} - \lambda_{2,2}]}{[\lambda_{1,2} - \lambda_{1,1}]} \\ \frac{p(\boldsymbol{x}|\omega_1)}{p(\boldsymbol{x}|\omega_2)} \frac{P(\omega_1)}{P(\omega_2)} \frac{[\lambda_{1,2} - \lambda_{1,1}]}{[\lambda_{2,1} - \lambda_{2,2}]} \quad & \geq \quad 1 & \text{(B.9)} \end{aligned}$$

where the assumption that $\lambda_{1,2} > \lambda_{1,1}$ and $\lambda_{2,1} > \lambda_{2,2}$ has been used (changing the direction of the inequality sign).

The same result, with opposite inequality direction, is obtained evaluating (B.8). Classification can now be done based on whether the fraction on the left evaluates to larger than or smaller than 1 (larger than 1 corresponds to ω_1). From inspection of the fraction in (B.9), it is seen that the classification is done evaluation density-, prior probability- and risk-ratios between the two classes.

Example

Assuming $\lambda_{1,1} = \lambda_{2,2} = 0$ (no penalty for correct classification) as is customary, some of the different cases which can occur in this classification scheme is visualized in Figure B.1. In this figure, the distributions of each of the classes is

$$\boldsymbol{x}|\omega_1 \sim N(-2, 1) \quad , \quad \boldsymbol{x}|\omega_2 \sim N(2, 1)$$

In the top figure, the prior probabilities for each of the classes are the same ($= 0.5$), and the risk coefficients $\lambda_{1,2}$ and $\lambda_{2,1}$ are the same constant c. Because of this, and the fact that the two distributions have the same standard deviation, the decision boundary gets placed directly between the mean values of each of the classes.

In the middle figure, the prior probabilities are changed, with $P(\omega_1) > P(\omega_2)$. This is reflected in the height of the densities drawn and it is seen that the decision boundary is shifted to the right, reflecting the fact that $\boldsymbol{x}$ being from ω_1 is more probable. In these two cases, the risk coefficients have been the same, effectively making this classifier one that minimizes the mean classification error. This can be understood by observing that the decision boundary have been placed in such a way that it minimizes the integrals given in (B.3) (here visualized by the green and red colored areas under each distribution).

In the bottom plot of Figure B.1, misclassifying ω_1 as ω_2 is deemed worse than misclassifying ω_2 as ω_1. This is reflected in the fact that $\lambda_{1,2}$ is now higher than $\lambda_{2,1}$ and the decision boundary is shifted to the left. Note now that the probability of error is no longer minimized. Instead, the overall risk is. When this happens, the decision boundary is not placed at $\boldsymbol{x_d} : P(\omega_1)p(\boldsymbol{x_d}|\omega_1) = P(\omega_2)p(\boldsymbol{x_d}|\omega_2)$ as is the case in the two other plots (which minimize P_e).

In general, when the data is bivariate normal and still consist of two classes with equal variance (or in general variance and covariance), simplifications (such as using the logarithmic function) can be applied to (B.9) to obtain an analytical expression for the optimal decision boundary. Doing this and simplifying the expression somewhat, it can be shown [54] that decision line $g(\boldsymbol{x})$ is given by

$$g(\boldsymbol{x}) = \boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{x}_0) \tag{B.10}$$

where

$$\begin{aligned} \boldsymbol{w} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\ \boldsymbol{x}_0 &= \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) - \log\left(\frac{P(\omega_1)}{P(\omega_2)}\right)\frac{\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2}{||\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2||^2_{\boldsymbol{\Sigma}^{-1}}} \end{aligned}$$

and $||\cdot||^2_{\boldsymbol{\Sigma}^{-1}}$ denotes the squared *Mahalanobis-distance* [38]. The important point here is that from (B.10), it is seen that the decision boundary is linear. By the derivations which leads to this simple form, the linearity is understood from the fact that both classes have the same covariance structure.

In the derivations above, the densities of the different classes, $p(\boldsymbol{x}|\omega_i)$, $i = 1, 2, ..., C$, and their prior probabilities $P(\omega_i)$, $i = 1, 2, ..., C$, has been assumed to be known. In most real life applications this is not the case

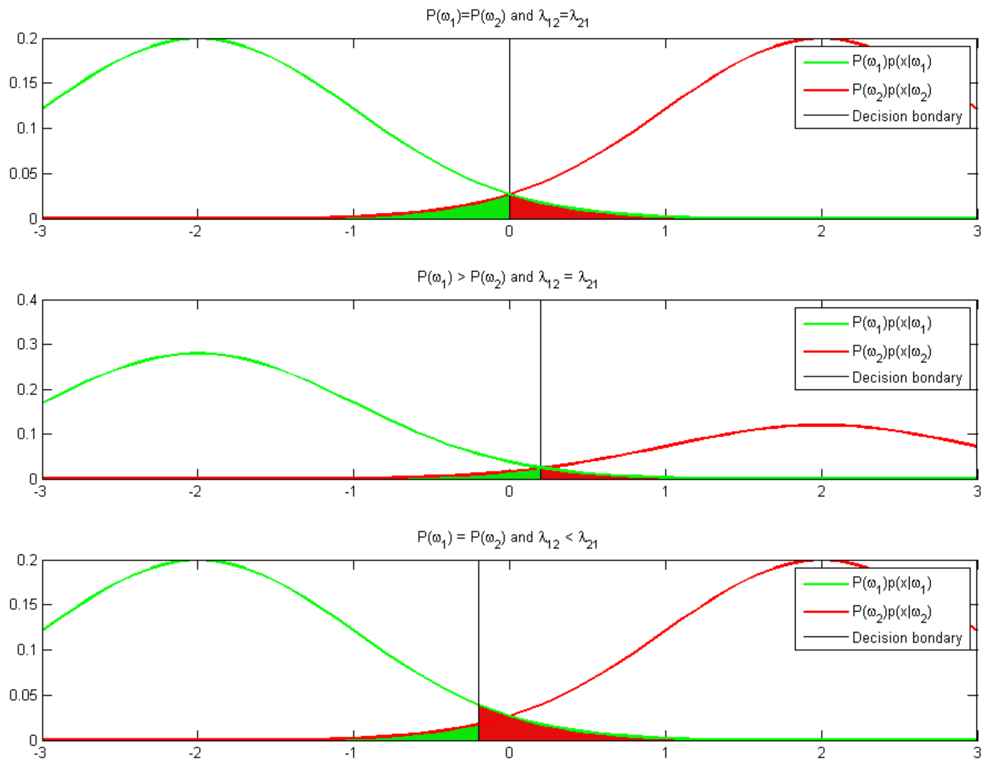


Figure B.1: Decision boundary for 3 different cases of the Bayes Classifier.

however and estimation methods have to be invoked. The density estimators discussed in 2.2 and Appendix A can here be used in order to approximate the densities in (B.9) and do Bayesian classification.

Estimation of the priors

If $N_1 + N_2 = N$ points, $\{\boldsymbol{x}_1, \boldsymbol{x}_1, ..., \boldsymbol{x}_{N_1}, \boldsymbol{x}_{N_1+1}, ..., \boldsymbol{x}_{N_1+N_2}\}$ are given. The first N_1 points known to be generated from $p(\boldsymbol{x}|\omega_1)$, while the last N_2 points comes from $p(\boldsymbol{x}|\omega_2)$, then the prior probabilities can be estimated with the number of training examples directly by

$$\hat{P}(\omega_1) = N_1/N \qquad \hat{P}(\omega_2) = N_2/N \tag{B.11}$$

Appendix C

Kernel SVM

Given a linearly separable set of N feature vectors from two classes, with corresponding class labels $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_N, y_N)\}$ ($y \in \{-1, 1\}$), there exist infinitely many decision boundaries which perform equally well (zero errors) on the training set. However, each of them does not necessarily generalize equally good to new data. Obtaining a classifier by minimizing a cost function on the form

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} (y_i \cdot \text{sign}(\boldsymbol{w}^T \boldsymbol{x} + b) - 1) \tag{C.1}$$

where $\boldsymbol{w}$ is the normal vector to the decision boundary, usually leads to this ambiguity[1].

Support Vector Machines aim to solve this problem by introducing a term called the *margin*. It is defined as the length from the decision boundary to the closest points in each class. An illustration of a typical linear SVM is shown in Figure C.1.

As the distance from any point $\boldsymbol{x}$ to a hyperplane is given by

$$d = \frac{|\boldsymbol{w}^T \boldsymbol{x} + b|}{||\boldsymbol{w}||}, \tag{C.2}$$

the distance to the closest points of the set can be set to 1 by varying $\boldsymbol{w}$. As such the margin can be maximized by minimizing $||\boldsymbol{\omega}||$ while still classifying the training set correctly. From this idea, the following optimization problem arise

$$\min J(\boldsymbol{\omega}) = \frac{1}{2} ||\boldsymbol{w}||^2 \tag{C.3}$$

$$\text{such that } y_i(\boldsymbol{\omega}^T \boldsymbol{x_i} + b) \geq 1, \ \forall \ i = 1, 2, ..., N \tag{C.4}$$

[1]The sign(x) function is defined as -1 for $x \leq 0$ and 1 otherwise.

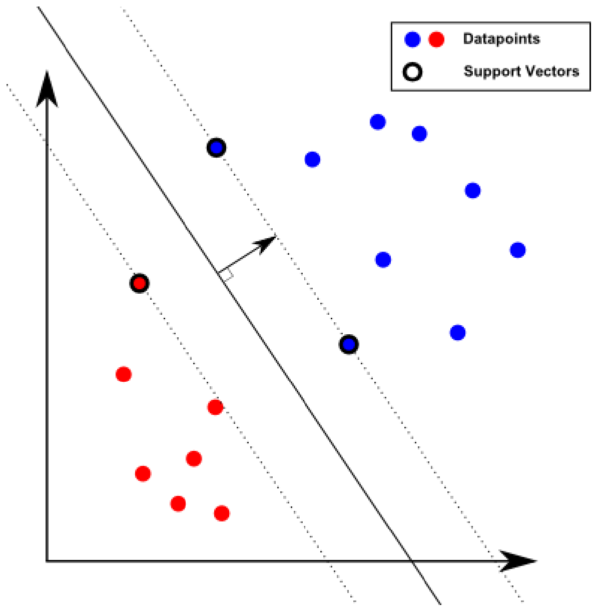


Figure C.1: Support vectors and margin of a SVM for a given training set.

Now, for the Support Vector Machine to also be able to handle a non-separable training set, each datapoint is given a slack parameter ξ_i, $i = 1, 2, ..., N$. This introduced slack is to make the optimization handle points which fall within the margin, or even on the wrong side of the hyperplane and can be thought of as an regularization to avoid the SVM over-fitting the data.

With this added set of parameters, the optimization problem now becomes

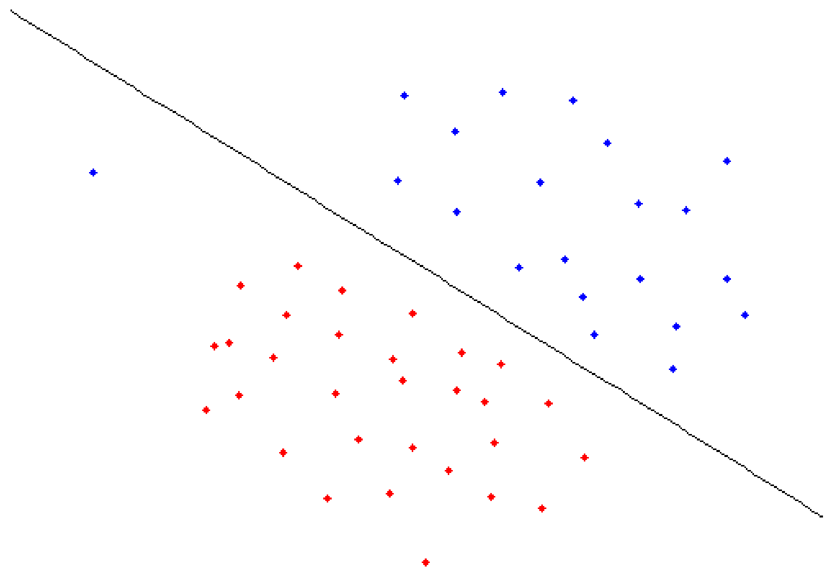
$$\min \quad J(\boldsymbol{\omega}, \boldsymbol{\xi}) = \frac{1}{2}||\boldsymbol{w}||^2 + R\sum_{i=1}^{N} \xi_i \tag{C.5}$$

$$\text{such that} \quad y_i(\boldsymbol{\omega}^T \boldsymbol{x_i} + b) \geq 1 - \xi_i, \quad \forall \;\; i = 1, 2, ..., N \tag{C.6}$$

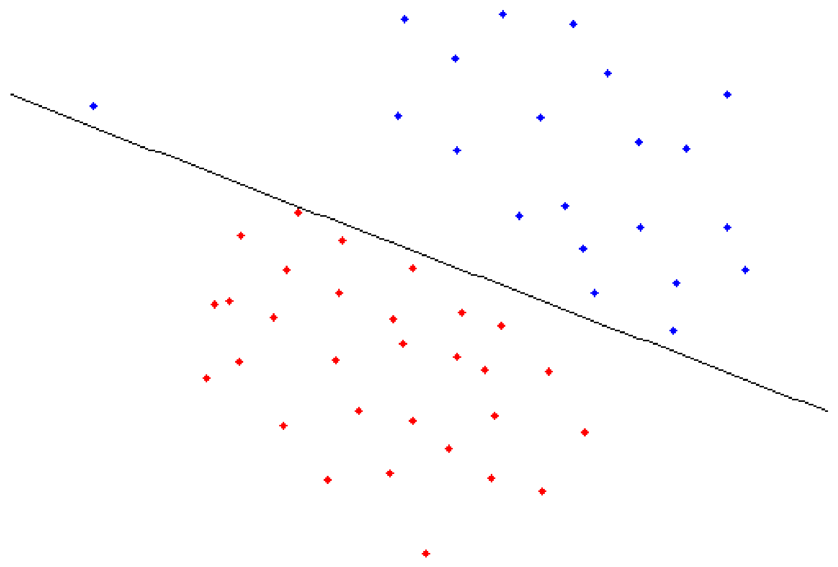
$$\xi_i \geq 0, \quad \forall \;\; i = 1, 2, ..., N \tag{C.7}$$

where R is a parameter chosen by the user to determine the degree of regularization. An example of a linear SVM applied with different choices of R is shown in Figure C.2.

As it turns out, this formulation belongs to a class of convex optimization problems and can be solved using the Wolfe dual representation of the Lagrangian [54]. With this reformulation, the following associated problem



(a) $R = 1$



(b) $R = 100$

Figure C.2: Decision boundaries from a linear Support Vector Machine for different choices of R. Notice C.2(a) does not make error free predictions on the training set due to an outlier, but could be said to generalize better.

is obtained

$$\max \quad \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \underbrace{\boldsymbol{x}_i^T \boldsymbol{x}_j}_{\text{An inner product!}} \right) \tag{C.8}$$

$$\text{such that} \quad 0 \le \lambda_i \le R, \quad \forall \;\; i = 1, 2, ..., N \tag{C.9}$$

$$\sum_{i=1}^{N} \lambda_i = 0 \tag{C.10}$$

Assuming all required parameters are obtained from the optimization, classification of a new point $\boldsymbol{x}$ using an SVM is done by evaluating

$$\boldsymbol{x} \;\rightarrow\; \omega_1(\omega_2) \text{ if } \boldsymbol{\omega}^T \boldsymbol{x} + b \ge (<)\ 0 \tag{C.11}$$

As both classification and the training of the parameters (seen in (C.8) – (C.10)) is done using inner products, it is known that a non-linear version of the algorithm can be obtained by applying the kernel trick and replacing all inner products with a kernel function. The linear training and the classification is then effectively done in the higher dimensional feature space F, producing non-linear results in the original space.

Example

An example of a non-linear decision boundary produced by using the *kernel trick* with the ordinary Support Vector Machine is shown in Figure C.3. Here a highly irregular training set[2] can be seen to be fitted nicely using a Gaussian kernel (similar to that given in (2.36)) to calculate inner products in F.

[2]Taken from the online Stanford course in Machine Learning [41].

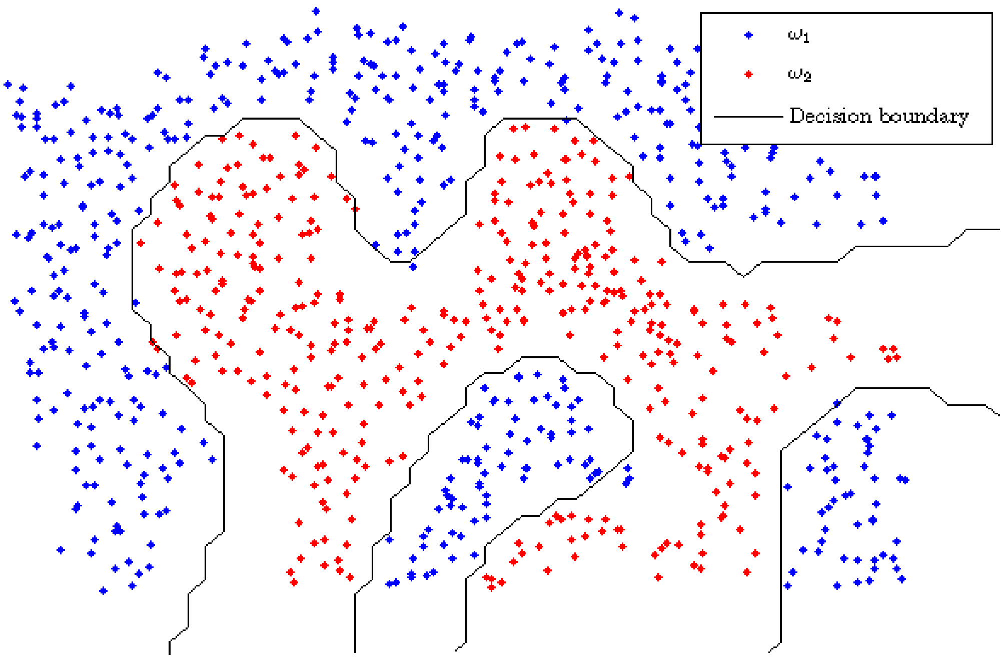


Figure C.3: Non-linear SVM decision boundary using the kernel trick ($\sigma = 0.1, C = 1$).

Appendix D

Smoothing of KNN PDF Estimate by Non-linear Scaling

The idea of a non-linear scaling on the PDF estimate, was originally an experiment for trying to avoid clusters growing out of control. Scaling was done by introducing a free parameter α which the hyper volume in the estimate was taken to the power of. This produces a function related to the KNN PDF estimate given as

$$\tilde{p}(\boldsymbol{x}; k, \alpha) = \frac{k}{NV_k^\alpha(\boldsymbol{x})} \tag{D.1}$$

The effect of scaling the inverse hyper volume by different α's between zero and one is seen in Figure D.1. Notice that as α decreases, the sensitivity on the distance also decreases. This results in the PDF estimate becoming more smooth, as can be seen in Figure D.2.

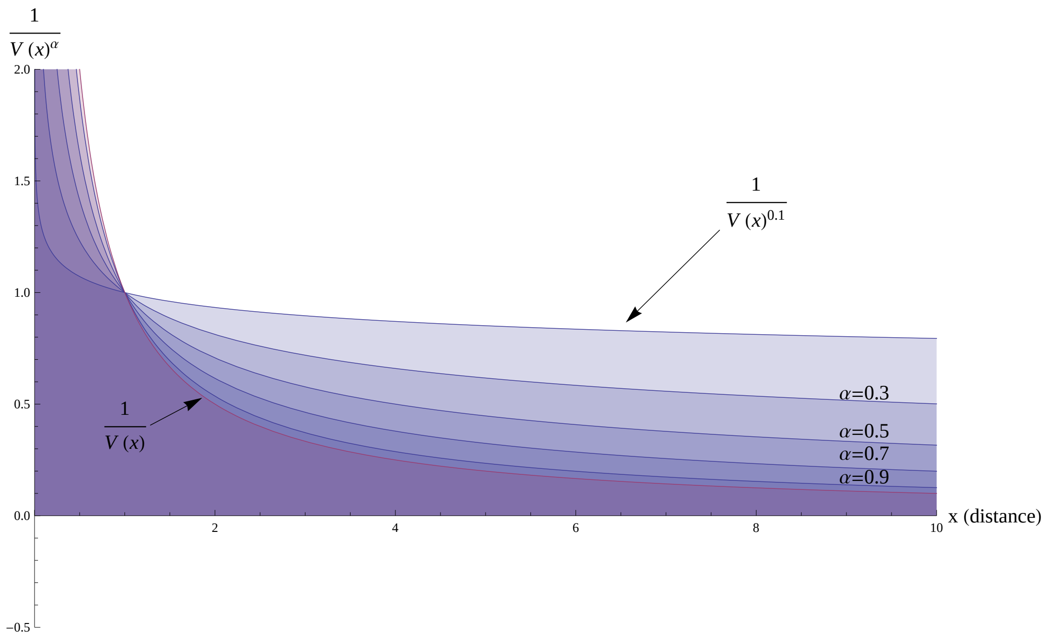


Figure D.1: Effect of scaling inverse hyper volume by different α's.

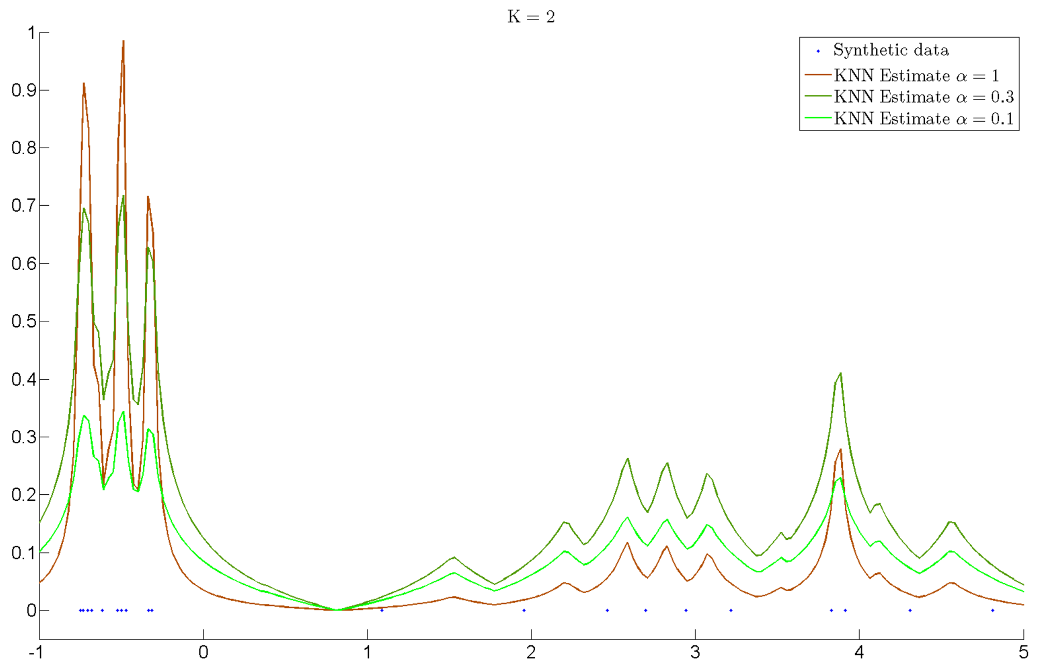


Figure D.2: Effect of scaling seen on PDF-estimate.

List of Figures

List of Tables

Bibliography

[1] H. Akaike. A new look at the statistical model indentification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[2] T. W. Anderson. *An Introduction to Multivariate Statistical Analysis.* John Wiley, 2003.

[3] G. A. Barnard. The theory of information. *Journal of the Royal Statistical Society. Series B (Methodological)*, 13(1):pp. 46–64, 1951. ISSN 00359246.

[4] A. Basu and B. Lindsay. Minimum disparity estimation in the continuous case: efficiency, distributions, robustness. *Ann. Inst. Statist. Math.*, 46(3):683–705, 1994. ISSN 0167-7152. doi: 10.1016/0167-7152(95) 00019-4.

[5] Belkin and Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396, 2003.

[6] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–109, 1943.

[7] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):pp. 493–507, 1952. ISSN 00034851.

[8] Thomas M. Cover and Joy A. Thomas. *Elements of information theory.* Wiley-Interscience, New York, NY, USA, 1991. ISBN 0-471-06259-6.

[9] Duda, O. Richard, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition).* Wiley-Interscience, 2000. ISBN 0471056693.

[10] Deniz Erdogmus, Jose C. Príncipe, and Kenneth E. Hild, II. Beyond second-order statistics for learning: A pairwise interaction model for

entropy estimation. *Natural Computing*, 1(1):85–108, May 2002. ISSN 1567-7818.

[11] Lev Faivishevsky and Jacob Goldberger. Nonparametric information theoretic clustering algorithm. In *ICML*, pages 351–358, 2010.

[12] A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL http://archive.ics.uci.edu/ml.

[13] A.L.N. Fred and A.K. Jain. Combining multiple clusterings using evidence accumulation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(6):835–850, 2005.

[14] Y. Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995.

[15] Luis Gómez-Chova, Gustavo Camps-Valls, Javier Calpe-Maravilla, Luis Guanter, and Jose Moreno. Cloud-screening algorithm for envisat/meris multispectral images. *IEEE T. Geoscience and Remote Sensing*, 45(12-2):4105–4118, 2007.

[16] Luis Gómez-Chova, Robert Jenssen, and Gustavo Camps-Valls. Kernel entropy component analysis in remote sensing data clustering. In *IGARSS*, pages 3728–3731, 2011.

[17] Erhan Gokcay and Jose C. Príncipe. Information theoretic clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(2):158–171, February 2002. ISSN 0162-8828.

[18] L. Hagen and A. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design*, 11(9): 1074–1085, 1992.

[19] R. Hartley. Transmission of information. *Bell Syst. Tech. J.*, 7:535, 1928.

[20] R. Jenssen, II Hild, K.E., D. Erdogmus, J.C. Principe, and T. Eltoft. Clustering using renyi's entropy. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 1, pages 523 – 528 vol.1, july 2003. doi: 10.1109/IJCNN.2003.1223401.

[21] R. Jenssen, J. Principe, and T. Eltoft. Cauchy-schwarz pdf divergence measure for non-parametric clustering. *Proc. IEEE Norway Section Int'l. Symposium on Signal Processing*, CD-ROM:–, 2003.

[22] R. Jenssen, J.C. Principe, and T. Eltoft. Information cut and information forces for clustering. In *Neural Networks for Signal Processing, 2003. NNSP'03. 2003 IEEE 13th Workshop on*, pages 459 – 468, sept. 2003. doi: 10.1109/NNSP.2003.1318045.

[23] R. Jenssen, T. Eltoft, and J.C. Principe. Information theoretic clustering: a unifying review of three recent algorithms. In *Signal Processing Symposium, 2004. NORSIG 2004. Proceedings of the 6th Nordic*, pages 292 –295, june 2004.

[24] R. Jenssen, D. Erdogmus, II K. E. Hild, J. C. Principe, and T. Eltoft. Cauchy-schwarz distance for fuzzy clustering using parzen kernel annealing. Submitted to IEEE Transactions on Pattern Analysis and Machine Inteligence, 2004.

[25] R. Jenssen, D. Erdogmus, K.E. Hild, J.C. Principe, and T. Eltoft. Information cut for clustering using a gradient descent approach. *Pattern recognition*, 40(3):796–806, 2007.

[26] Robert Jenssen. Kernel entropy component analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(5):847–860, 2010.

[27] Robert Jenssen. Kernel entropy component analysis: New theory and semi-supervised learning. In *IEEE Workshop on Machine Learning for Signal Processing*, pages 1–6, 2011. doi: 10.1109/MLSP.2011.6064626.

[28] Robert Jenssen, Torbjørn Eltoft, and Jose C. Principe. Information theoretic spectral clustering. In *In Proceedings of International Joint Conference on Neural Networks*, pages 111–116, 2004.

[29] Robert Jenssen, Deniz Erdogmus, Kenneth Hild, Jose Principe, and Torbjørn Eltoft. Optimizing the cauchy-schwarz pdf distance for information theoretic, non-parametric clustering. In Anand Rangarajan, Baba Vemuri, and Alan Yuille, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, volume 3757 of *Lecture Notes in Computer Science*, pages 34–45. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-30287-2.

[30] Robert Jenssen, Deniz Erdogmus, Jose Principe, and Torbjørn Eltoft. The laplacian pdf distance: A cost function for clustering in a kernel feature space. In *in Advances in Neural Information Processing Systems 17*, pages 625–632. MIT Press, 2005.

[31] Robert Jenssen, Jose C. Principe, Deniz Erdogmus, and Torbjørn Eltoft. The cauchy-schwarz divergence and parzen windowing: Connections to graph theory and mercer kernels, 2006.

[32] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):pp. 79–86, 1951. ISSN 00034851.

[33] Nikolai Leonenko, Luc Pronzato, and Vippal Savani. Estimation of entropies and divergences via nearest neighbors. In *Tatra Mt. Math. Publ.*, volume 39, pages 265–273, Smolenice, Slovaquie, 2008.

[34] Tjen-Sien Lim, Wei-Yin Loh, and W. Cohen. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms, 2000.

[35] D.O. Loftsgaarden and C. P. Quesenberry. A nonparametric estimate of a multivariate density function. *Ann. Math. Stat.*, 36:1049–1051, 1965.

[36] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, December 2007. ISSN 0960-3174. doi: 10.1007/s11222-007-9033-z.

[37] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[38] P. C. Mahalanobis. On the generalised distance in statistics. In *Proceedings National Institute of Science, India*, volume 2, pages 49–55, April 1936.

[39] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Roayal Society*, A 209 (441-458):415–446, 1909.

[40] K. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12:181–201, 2001.

[41] A. Ng, 2011. URL `https://www.coursera.org/course/ml`.

[42] E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33(3):1065–1076, 1961.

[43] D.N. Politis. Adaptive bandwidth choice. *Journal of Nonparametric Statistics*, 15(4-5):517–533, 2003.

[44] J. Príncipe, D. Xu, and J. Fisher. *Information Theoretic Learning*, chapter 7. John Wiley and Sons, 2000.

[45] J.C. Príncipe. *Information Theoretic Learning: Renyi's Entropy and Kernel Perspectives*. Information Science and Statistics. Springer, 2010. ISBN 9781441915696.

[46] A. Renyi. On measures of entropy and information. *Proc. of the 4th Berkely Symp. Math. Statist. Prob.*, 1:–, 1960.

[47] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 0387212396.

[48] B. Schölkopf, A. Smola, and K. Müller. Nonlinear component analysis as a kernel eigenvalue problem. Technical report, Max-Planck-Institut für biologische Kybernetik, 1996.

[49] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423 and 623–656, 1948.

[50] C. Shannon and W. Weaver. *The Mathematical Theory of Communication*, volume 1. University of Illinois Press, 1949.

[51] S.J. Sheather and M.C. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, -:683–690, 1991.

[52] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[53] B. W. Silverman. Density estimation for statistics and data analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22: 888–905, 2000.

[54] Koutroumbas Theodoridis. *Pattern Recognition*. Academic Press, 2009.

[55] Qing Wang, Sanjeev R. Kulkarni, and Sergio Verdú. Divergence estimation for multidimensional densities via k-nearest-neighbor distances. *IEEE Trans. Inf. Theor.*, 55(5):2392–2405, May 2009. ISSN 0018-9448.

[56] Satosi Watanabe. *Pattern recognition: human and mechanical.* John Wiley & Sons, Inc., New York, NY, USA, 1985. ISBN 0-471-80815-6.

[57] R. A. Johnson & D. W. Wichern. *Applied Multivariate Statistical Analysis.* Pearson Int., 2007.

[58] Z. L. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theroy and its applications to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15 (11): 1101–1113, 1993.

UNIVERSITY OF TROMSØ