Master's Thesis Report

Applications and Implementation of Kernel Principal Component Analysis to Specific Data Sets

Daniel Olsson
daolsson@kth.se

January 28, 2011

Abstract

Kernel Principal Component Analysis (KPCA) is a dimension reduction method that is closely related to Principal Component Analysis (PCA). This report gives an overview of kernel PCA and presents an implementation of the method in MATLAB. The implemented method is tested in a transductive setting on two data bases: Iris data and sugar data. Two methods for labeling data points are considered, the nearest neighbor method and kernel regression, together with some possible improvements of the methods.

Contents

1 Introduction

Data mining is concerned with finding meaningful patterns in large sets of data. The swift development in computation power that has taken place in recent years has increased the amount of available data in an avalanche-like fashion (consider for example the Human Genome Project, in which the whole human DNA is stored in a database). The task of analyzing these gigantic amounts of data can appear overwhelming. Naturally, there is a need for tools that can aid the analyst in identifying the interesting features of the data.

In the past decades, a group of methods known as dimension reduction methods have gained wide-spread attention. As the name implies, dimension reduction methods seek to solve this problem by reducing the number of dimensions of the data, while retaining most of the information in the data set. By removing redundant dimensions, it becomes easier to recognize patterns or tendencies in the remaining data. Some examples of dimension reduction methods are Support Vector Machines and Random Projections (e.g. [11] and [2, 3]).

In this report, one particular dimension reduction method is considered, namely Kernel Principal Component Analysis (KPCA). The method is implemented in MATLAB and applied to data sets. In addition, different ways of modifying kernel PCA in order to improve the results for these particular data sets will be considered.

The outline of the report is as follows: Section 2 provides the description of kernel PCA and considers other techniques that are used in the application of the method. Section 3 contains the numerical experiments that were performed on two databases. The results are commented along with a discussion in section 4. Central parts of the MATLAB-source code that was used in order to perform kernel PCA can be found in the appendix.

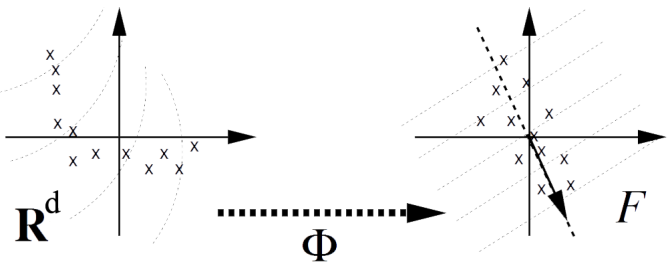

Figure 1: Data points in original space mapped into feature space by the mapping Φ (von Luxburg 2006). Although a cluster of points may be scattered in a nonlinear pattern in the original space (left), they might form a pattern that is linear in the high dimensional feature space (right).

2 Kernel PCA

Kernel PCA uses the same basic idea as the statistical method Principal Component Analysis (PCA), namely that it seeks to project the set of data onto a low-dimensional subspace that captures the highest possible amount of variance in the data. However, while PCA performs a linear separation of the data in the original space (referred to as R^d), kernel PCA embeds the data into a high dimensional space, called the feature space (referred to as F), by a mapping function Φ and performs a linear separation in that space instead. In this way also nonlinear separations of data are made possible.

Thus, while kernel PCA looks for linear features in feature space, these features correspond to nonlinear features in the original lower dimensional space R^d, as shown in Figure 1. The data in the feature space is projected onto a low-dimensional subspace, spanned by the eigenvectors that capture most of the variance.

One important fact is that it is not necessary to know the mapping Φ nor the feature space F explicitly in order to perform kernel PCA. Instead computations are performed on the inner product of pairs of points which are stored in a kernel matrix. The procedure of working with the data in feature space without knowing the mapping Φ nor F is known as "the kernel trick" and is a central part of the kernel PCA method.

The advantage of kernel PCA over standard (linear) PCA is illustrated in Figure 2. In the original space (left), the two types of data points cannot be linearly separated. However, by extracting nonlinear features, the points can be sorted according to their class (right).

For a further description of kernel PCA, see [13] and [14, Chapter 6].

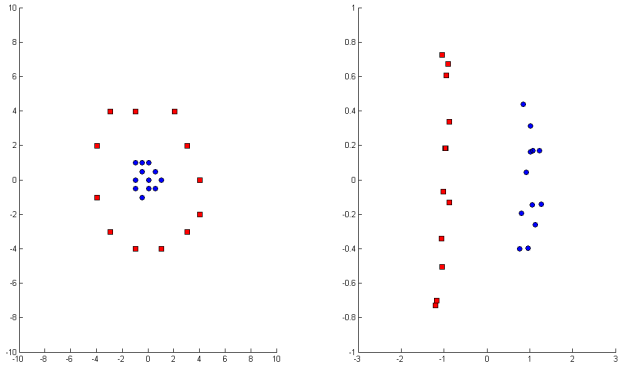

Figure 2: An illustration of kernel PCA. In the left plot, the two circles of points (red and blue) cannot be separated linearly from each other (for example by linear PCA). Kernel PCA embedds the points into a high dimensional space and then projects them onto a subspace spanned by eigenvectors, resulting in the plot to the right. The two classes of points in the plot to the right can be linearly separated from each other.

2.1 Mathematical formulation

Let the set of n data points be represented by

$$X = \{x_1, \ldots, x_n\}, \tag{1}$$

where $x_i \in R^d$, $i = 1, \ldots, n$. It is assumed that the data in X is centered in the original space R^d, so that

$$\sum_{i=1}^{n} x_i = 0. \tag{2}$$

This condition can be met by using the formula

$$\tilde{x}_k = x_k - \frac{1}{n} \sum_{i=1}^{n} x_i, \tag{3}$$

where $\tilde{x}_k$ is the k:th data point in the new, centered set.

The following is a description of how kernel PCA is performed on a data set X, consisting of n data points.

Let Φ be a mapping from the original space R^d of the data vectors, into a inner product space F, so that

$$\Phi : R^d \to F, \tag{4}$$

where the space F is referred to as the feature space. Thus, $\Phi(x_i)$ represents the image of the data vector $x_i \in R^d$ in feature space.

Let the kernel matrix K be a symmetric and positive semidefinite $n \times n$-matrix, with its elements defined by the inner product of all pairs of points $\Phi(x_i)$ and $\Phi(x_j)$ in feature space so that

$$K_{ij} \triangleq (\Phi(x_i) \cdot \Phi(x_j)), \quad i, j = 1, \ldots, n. \tag{5}$$

Interestingly, it is not necessary to known the values of $\Phi(x_i)$ and $\Phi(x_j)$ explicitly in order to compute the kernel matrix K. In other words, in order to map the data points into F, the mapping Φ is not required to be known. Instead, the elements of K can be calculated by computing the pairwise relation of all points $x_i, x_j \in X$ in the original space R^d.

This is motivated by Mercer's Theorem (see below) and done through a kernel function $k(x_i, x_j)$, so that

$$K_{ij} = k(x_i, x_j), \quad i, j = 1, \ldots, n. \tag{6}$$

This way the features space F and the mapping Φ are implicitly defined by the kernel function that is used. This is known as "the kernel trick" and is the perhaps most central part of the kernel PCA method.

There are a variety of kernel functions that can be used for kernel PCA. In this report the Gaussian radial basis function (RBF) has been used (equation (12)).

The data points represented in the kernel matrix K are assumed to be centered in feature space, in the sense that

$$\sum_{i=1}^{n} \Phi(x_i) = 0. \tag{7}$$

For a noncentered set, this centering can be achieved by the inner product computation

$$(\tilde{K})_{ij} = ((\Phi(x_i) - \overline{\Phi}) \cdot (\Phi(x_j) - \overline{\Phi})) \tag{8}$$

where $\tilde{K}$ is the centered kernel matrix and $\overline{\Phi}$ is the center point in the feature space, given by

$$\overline{\Phi} = \frac{1}{n} \sum_{i=1}^{n} \Phi(x_i). \tag{9}$$

A derivation of the centering in feature space is found in the appendix.

The eigenvectors of the centered kernel matrix $\tilde{K}$ can be obtained by solving the equation

$$\tilde{K}V = \Lambda V, \tag{10}$$

where the columns of V represent the normalized eigenvectors and Λ is a matrix in which the diagonal consist of the corresponding eigenvalues and all other elements in Λ are zero. The eigenvectors in V are assumed to be ordered according to the size of their corresponding eigenvalues, in descending order. Also the eigenvalues in Λ are sorted in descending order.

The representation of the low-dimensional points $\{\chi_1, \ldots, \chi_n\}$, with $\chi_i \in R^l$, $i = 1, \ldots, n$, can be obtained by projecting the kernel matrix $\tilde{K}$ onto the l eigenvectors that have the largest corresponding eigenvalues, by using the formula

$$\chi_i = \tilde{K}_i V^l, \tag{11}$$

where $\tilde{K}_i$ is the i:th row of K, and V^l is the matrix consisting of the l first eigenvector columns in V.

The kernel matrix is defined as (5) and thus contains the inner product of each pair of points x_i, x_j in feature space F. However, the value of K_{ij} is determined by some kernel function $k(x_i, x_j)$. This is a consequence of Mercer's Theorem, which states that any symmetric, positive semidefinite matrix can be regarded as a kernel matrix, that is as an inner product matrix in some space, which in the case of kernel PCA is the feature space.

Theorem 2.1 (Mercer's Theorem, [13])**.** *If k is a continuous kernel of a positive integral operator, then it is possible to construct a mapping into a space where k acts as a dot product.*

The implication of Mercer's theorem for kernel PCA is that the inner product $(\Phi(x_i) \cdot \Phi(x_j))$ can be substituted by the kernel function $k(x_i, x_j)$ that we choose. There are many different kernel functions available, for example the Gaussian radial basis function [1]

$$K_{ij} = k(x_i, x_j) = \exp(-\sigma \|x_i - x_j\|^2), \tag{12}$$

where σ is a parameter. As can be seen from equation (12), K_{ij} tends to go toward zero if the points x_i and x_j are far away from each other in R^d, while K_{ij} is close to one as the two points are near each other in R^d.

As seen from the definition in equation (5), K_{ij} is the inner product of $\Phi(x_i)$ and $\Phi(x_j)$. Thus, the following relationship between the points x_i, x_j in the original space R^d and their images $\Phi(x_i)$, $\Phi(x_j)$ in the feature space is valid:

If $K_{ij} \approx 0$, i.e. x_i and x_j are distant from each other in R^d, then $\Phi(x_i)$ and $\Phi(x_j)$ are orthogonal in feature space. Likewise, if $K_{ij} \approx 1$, i.e. x_i and x_j are near each other in R^d, then $\Phi(x_i)$ and $\Phi(x_j)$ are parallel (and if $\Phi(x_i)$ and $\Phi(x_j)$ are assumed to be normal vectors in F, the two points are equal to each other).

The consequence of this is that data points from a high-dimensional data set X that are geometrically close to each other in the space R^d, will be "almost" parallel in feature space. Furthermore, if the points in feature space are normal vectors in the sense that

$$\|\Phi(x_i)\| = 1, \quad i = 1, \ldots, n, \tag{13}$$

then points that are almost parallel will be be close to each other, as they are located on the hypersphere of radius one in the feature space.

[1] An alternative formulation of the radial basis function is $k(x, y) = \exp(-\frac{\|x-y\|^2)}{2\sigma^2})$, used in for example [13, 12]. In this report, the definition in (12) is consistently used.

2.2 The transduction problem

The transduction problem is the problem of completing the labeling of a partially labeled set. Many of the problems that are considered in this report have been posed in this form. The idea is to use the data and known labels of a training set in order to label the data in the test set. Naturally, a large training set tends to increase the classification performance, as there are more features that can be extracted. It is possible to compute a kernel matrix based on the test points relation to the training points, and then project it onto a low dimensional space. That is one way in which a test set consisting of points of different classes can potentially be separated, and thus labeled, according to class.

Let the first n_{tr} data points in X belong to the training set

$$X_{tr} = \{x_1, \ldots, x_{n_{tr}}\} \subset X \tag{14}$$

and the remaining n_{tt} of the data points in X belong to the test set

$$X_{tt} = \{x_{n_{tr}+1}, \ldots, x_{n_{tr}+n_{tt}}\} \subset X, \tag{15}$$

so that $n_{tr} + n_{tt} = n$. Furthermore, let each data point $x_i \in X$ belong to a class $c_i \in R$ and assume that the class of each point in X_{tr} is known, and stored in the label set

$$y = \{y_1, \ldots, y_{n_{tr}}\}. \tag{16}$$

Thus, the class of the i:th data point in the training set is given by the element $y_i \in R$ (in the case of binary classification, $y_i \in \{+1, -1\}$).

The $n_{tr} \times n_{tt}$-test kernel matrix K_{tt} is formed by computing the relation between the points in the test set X_{tt} and the training set X_{tr} with the kernel function

$$K_{ij} = k(x_i, x_j) = \exp(-\sigma \|x_i - x_j\|^2), \tag{17}$$

where $x_i \in X_{tr}$ and $x_j \in X_{tt}$, for $i = 1, \ldots, n_{tr}$ and $j = 1, \ldots, n_{tt}$.

The parameter σ determines how much the kernel function shall generalize when it determines the relation between points. If σ is very large then the function becomes a "look-up table", in the sense that only test points that are identical to any of the training points are accepted as belonging to the same class as that of the training points. Likewise, if σ is near zero, then a test point need only to be slightly similar to the training points in order to be accepted.

The low dimensional projection of the test points

$$\chi_{tt} = \{\chi_1, \ldots, \chi_{n_{tt}}\} \tag{18}$$

is obtained by projecting the test kernel matrix onto the l first eigenvectors that are extracted from the training kernel matrix. The test kernel matrix is assumed to be centered in the same sense as equation (7). However, as the test kernel matrix is not a square matrix, the formula for centering in feature space is slightly different (see the appendix for details).

The resulting projection is given by

$$\chi_{tt} = K_{tt} V^l, \tag{19}$$

in which the projected test points $\chi_i \in R^l$, for $i = 1, \ldots, n_{tt}$ and V^l is the matrix consisting of the l first eigenvector columns in V (see equation (11)). Note that V is the eigenvectors of the training kernel matrix. Thus equation (19) projects the test kernel matrix onto the subspace that is spanned by some of the eigenvectors of the training kernel matrix. Recall that V^l are the eigenvectors that contain the largest amount of variance (information) of the training data.

Another way of interpreting Equation (11) is to view the eigenvectors V^l as determining what features of the data that will be looked at in the test set.

2.3 The nearest neighbor method

The nearest neighbor method can be used in transduction problems, i.e. in the classification of a partially labeled set [4]. The idea is to use reference points with known labels in addition to the test points, and to assign the same label to a test point as that of its geometrically nearest reference point. In other words, each test points will receive the same class as its nearest reference point.

The reference points are randomly selected from the same data pool as the test set and are therefore expected to be distributed evenly among the test points, if the size of the two sets is large. Note that the class of the reference points must be known, so we cannot allow the same points into both the reference set and the test set. Aside from that, any points with known class can be used in the reference set.

As we already have a training set in which the class of all points is known, it is possible to use the training points both for the training set and the reference set. This has been done in the nearest neighbor experiments of this report; the same points were used both in the reference set and in the training set (i.e. $X_{ref} = X_{tr}$). Also note that no test points should be included in the reference set, as the class of the test points are unknown and are to be determined using the nearest neighbor method.

Let $X_{ref} \subset X$ be the set of n_{ref} reference points, where each point belongs to the class given by the corresponding element in the label vector y_{ref}. Furthermore, let K_{ref} be a $n_{ref} \times n_{tr}$-kernel matrix based on X_{ref} and X_{tr}, through the kernel function $k(x, y)$, $x \in X_{ref}$ and $y \in X_{tr}$. The points in feature space are projected onto the eigenvectors V^l by the formula

$$\chi_{ref} = K_{ref} V^l, \tag{20}$$

where χ_{ref} is the low-dimensional projection of the reference points (compare to Equation (19)). The test points and reference points in feature space are projected onto the same eigenvectors (see equation (19)), so the test points and reference points will be found in the same l-dimensional subspace.

The test points and reference points are assumed to be centered in original space, in the sense that

$$\frac{1}{n_{tt} + n_{ref}} \left(\sum_{i=1}^{n_{tt}} x_{tt,i} + \sum_{j=1}^{n_{ref}} x_{ref,j} \right) = 0. \tag{21}$$

In order to apply the nearest neighbor method, the projected points χ_{tt} and χ_{ref}, and the label set y_{ref} are required. The geometric distance between all test points and reference points can be expressed by a distance matrix D, with its elements given by

$$D_{ij} = d(\chi_i, \chi_j) = \|\chi_i - \chi_j\|_l, \tag{22}$$

where $\chi_i \in \chi_{tt}$ and $\chi_j \in \chi_{ref}$.[2] Note that $\chi_i \in R^l$, in which l is the number of eigenvectors that are extracted from the training kernel matrix (see equation (11)). The reference point that is nearest to the i:th test point is then given by the index of the smallest element in the i:th row vector of the matrix D. In other words, the index j_i^* of the reference point that is nearest to the i:th test point is given by

$$j_i^* = \underset{j}{\operatorname{argmin}}\ D_{ij},\ j \in \{1, \ldots, n_{ref}\} \tag{23}$$

for $i = 1, \ldots, n_{tt}$.[3] In this way, the i:th test point x_i in X_{tt} receives the same label as y_{ref,j_i^*}, indicating that it belongs to the same class as the j_i^*:th point in the reference set X_{ref}. By performing this procedure on each test point, the whole test set is labeled.

The nearest neighbor method can be extended into taking into account more than one nearest point. The method, referred to as the "k-nearest neighbor method", uses a classification function based on the k nearest points to determine the label of each test point.

2.4 Kernel regression

As described in [12], kernel regression can be used to form a classification function for the labeling of a partially labeled set. This is done by using a weight-vector

$$c = \{c_1, \ldots, c_{n_{tr}}\} \tag{24}$$

to fit a training kernel matrix K_{tr} to the corresponding set of labels $y = \{y_1, \ldots, y_{n_{tr}}\}$. Thus, the classification function is trained on data from the training set in order to be able to predict the labels of the data in the test set. In this section, binary classification is considered, meaning that each label can be either +1 or -1, that is to say, $y_i \in \{+1, -1\}$, for $i = 1, \ldots, n_{tr}$. The task is to find a vector c such that,

$$(n_{tr}\gamma I_{n_{tr}} + K_{tr})c = y, \tag{25}$$

where $I_{n_{tr}}$ is a $n_{tr} \times n_{tr}$ identity matrix, γ is a parameter. The term $n_{tr}\gamma I_{n_{tr}}$ in Equation (25) is used to make the problem well-posed, meaning that there exists an unique solution to the problem and that there is a continous relation

[2] Definition: $\|x\|_p \triangleq (\sum_{i=1}^{d} |x_i|^p)^{1/p}$, where $p \geq 0$.

[3] Definition: $\underset{x}{\operatorname{argmin}}\ f(x) \triangleq \{x | \forall y : f(x) \leq f(y)\}$

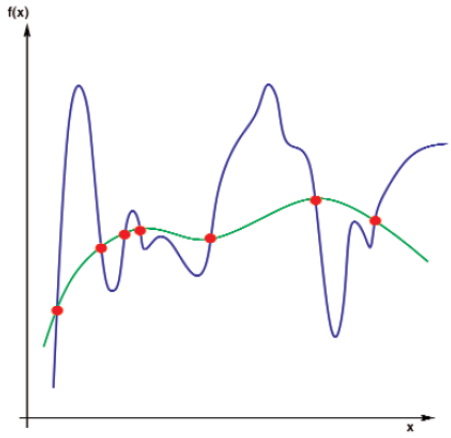

Figure 3: An illustration of kernel regression (Poggio and Smale 2003). Kernel regression fits a function (green curve) to the set of training points (red points), and then uses the function to determine the class of each test points (located along the blue curve). The parameter γ controls the smoothness and the uniqueness of the solution.

between the input data and the solution. Together with σ, the parameter γ controls the generalization of the predictions. More specifically, γ determines the smoothness and the uniqueness of the solution (see Figure 3).

A "large" value of $m\gamma$ is said to yield a good condition number, so that minor variations in the test set do not affect the outcome of the labeling. The method is derived from Tikhonov regularization and is further described in [12].

The weight vector c can then be used to form a classification function $f : R^{n_{tr}} \rightarrow R$ that is defined as

$$f(x) \triangleq \sum_{i=1}^{n_{tr}} c_i K_{tt,x_i}(x), \tag{26}$$

in which $K_{tt,x_i}(x)$ is the row vector in the test kernel matrix that corresponds to the test point x_i. For each test point x_i, the classification function gives either a positive or negative value. If the value is positive, then x_i belongs to the class +1, and if the value is negative, then x_i belongs to the class -1. In other words, the classification function labels the test point x_i as

$$\text{sgn}(f(x_i)) = \begin{cases} +1 & \text{if} \quad f(x_i) \geq 0 \\ -1 & \text{if} \quad f(x_i) < 0. \end{cases} \tag{27}$$

As has been seen, kernel regression introduces the additional parameter γ. There are several techniques for finding a suitable value of γ. The most commonly used is through k-fold cross-validation, which has also been the approach of this report.

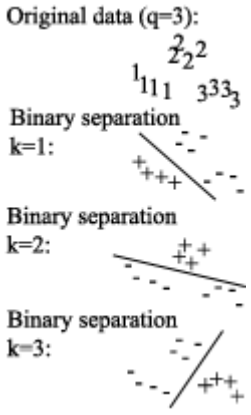

Figure 4: Multiclass classification using repeated binary classification. The numbers 1, 2 and 3 represent test points that belong to that respective class. The symbol "+" represents a point belonging to the class +1 and the symbol "-" represents a point that belongs to the class -1. The lines represent the separation of test points (according to class) that results from for example kernel regression.

2.5 Multiclass classification

All the data in the experiments have been multiclass data, in the sense that data points of more than two classes have been involved. Previous experiments have suggested that kernel PCA is more efficient at separating binary class problems than multiclass-problems. Because of this, a variant of the skewed binary classification tree (BCT) approach has been taken for the kernel PCA method [8]. The BCT can be used to classify a multi-class problem, using repeated binary classifications.

Consider the previously described transductive problem in which we have a training set X_{tr} consisting of n_{tr} points and a test set X_{tt} consisting of n_{tt} points. The labels $Y^{tr} = \{y_1^{tr}, \ldots, y_{n_{tr}}^{tr}\}$ of the points in the training set are known. The task is to determine the labels $Y^{tt} = \{y_1^{tt}, \ldots, y_{n_{tt}}^{tt}\}$ of the points in the test set. Each point in the two sets belong to one of q classes ($q > 2$). In the method described below, two matrices $Z^{tr} \in R^{n_{tr} \times q}$ and $Z^{tt} \in R^{n_{tt} \times q}$ are introduced. Each element in Z^{tr} and Z^{tt} can take the value $+1$ or -1. The i:th column vector of Z^{tr} and Z^{tt} represent the binary labeling of the training set and test set in the case when the i:th class is separated from the other $q-1$ classes.

In order to perform binary classification on the multi-class classification problem, the following approach was taken:

1. Let $k = 1$.
2. For each data point $x_i \in X_{tr}$: If $y_i^{tr} = k$, then let $Z_{ik}^{tr} = +1$. Otherwise, let $Z_{ik}^{tr} = -1$.

3. Use kernel regression or the nearest method to assign binary labels (Z^{tt}_{jk}, $j = 1, \ldots, n_{tt}$) to the testing set, using the new labels Z^{tr}_{ik}, $i = 1, \ldots, n_{tr}$ as input for the labeling method.

4. If $k < q$, then increase k by 1, and return to Step 2. Otherwise, continue to the next step.

5. For each data point $x_j \in X_{tr}$: Iterate through the classes $k = 1, \ldots, q$. For the first k for which $Z^{tr}_{jk} = 1$, let $y^{tt}_j = k$ and go to the next data point. If no such k exists, then let $y^{tt}_j = q$.

In Step 5, the binary label information in Z^{tt} is transformed into the original multi-class labeling. This is done by the majority voting procedure (see e.g. [10]). The approach taken in the experiments of this article differs from the standard skewed BCT in that the points that have been separated in one iteration k are not removed before the following iteration. This is due to evaluation purposes of the classification methods that were used; a way of seeing if any point is assigned more than one label. The advantage of using the original BCT is that the computation time is shorter.

The following is a summary of the method: Assume that we have q classes of data. In the first iteration, treat the first class as class $+1$ and the remaining classes as class -1. Label the test points by, for example, kernel regression and store this information about which data points are determined to belong to class one. In the second iteration, treat the second class as $+1$ and the remaining classes as -1. Again perform kernel regression to determine what test points are belonging to class 2, i.e. the points that have obtained the label $+1$, and store the information. Repeat this procedure for all q classes, until all test points have been given a label.

3 Numerical experiments

3.1 Introduction

It is interesting to see how the kernel PCA performs when applied to a data base. In order to evaluate performance of the kernel PCA method, two different approaches have been taken. The first approach has been to let the method predict the classes of test points (given a set of training points), and then look at the prediction accuracy, i.e. the percentage of test points that were correctly labeled by the method. The second approach has been to plot the points and look at how well the points belonging to different classes have been separated (the points have been colored according to class). The method was implemented in MATLAB and applied to two databases: Iris data and Sugar data. The MATLAB-code can be found in the appendix.

When applying kernel PCA to a data set, the choice of the parameters σ and γ (kernel regression) are very important. The common procedure is to find good values for these parameters by testing the performance of many different values of σ and γ on randomly chosen subsets of the data and then using the values that performed best. This method for determining the parameters is known as k-fold Cross Validation.

3.2 Kernel PCA applied to Iris data

Table 1 presents result of tests in which the nearest neighbor method and kernel regression are tested on an Iris database.[4] The Iris data consist of $n = 150$ four-dimensional data points, divided into three equally-sized classes. The n_{tr} data points in the training set were randomly selected from the Iris data. The remaining points were then used as test data. In Table 1, each prediction accuracy-value is an average of 10 sequential tests, in which the training set and test set were randomly selected anew each time.

$\frac{n_{tr}}{n}$ (%)	$\frac{n_{tt}}{n}$ (%)	$\#Eigenvecs.$	σ	γ	KR (%)	NN (%)
10	90	5	0.4	1.0000	82.7	92.7
20	80	5	0.4	0.0005	96.8	94.5
20	80	10	0.4	0.0002	94.5	93.1
40	60	5	0.3	0.0001	95.1	94.6
40	60	10	0.3	0.0004	97.6	95.4
40	60	25	0.4	0.0001	95.2	95.3
60	40	10	0.3	0.0005	97.8	96.2
60	40	5	0.3	0.0001	95.3	95.5
90	10	10	0.3	0.0001	96.7	96.7

Table 1: Prediction accuracy of kernel regression (KR) and the nearest neighbor method (NN) applied to Iris data.

[4] Available online: `http://archive.ics.uci.edu/ml/`

As can be seen in Table 1, the nearest neighbor method and kernel regression perform about as well when a good value for the parameter γ is found (in this test $\gamma << 1$ gives a good classification performance). The performance of the nearest neighbor method is only dependent on σ. Note that $\frac{n_{tr}}{n}$ and $\frac{n_{tt}}{n}$ indicate the percentage of training points and test points, respectively.

3.3 Kernel PCA applied to sugar data

The sugar database consists of 681 eleven-dimensional data points. The sugar data consists of four classes: Allose, Galactose, Glucose and Mannose. In the tests on which the results of Table 3.3 is based, σ had a constant value ($\sigma = 0.001$). The value of σ was found by testing a range of different values. Ten eigenvectors were extracted from the kernel matrix in each test. The percentage of training points in the experiment was 10%, 20% and 40%, respectively. In the table, the prediction accuracies presented on each row is the computed average of 10 tests.

$\frac{n_{tr}}{n}$ (%)	$\frac{n_{tt}}{n}$ (%)	γ	KR (%)	NN (%)
10	90	0.00027	73.2	64.2
20	80	0.00006	78.6	71.4
40	60	0.00002	79.8	75.5
		Average:	77.2	70.3

Table 2: Prediction accuracy of kernel PCA applied to sugar data.

In this test, the kernel regression method yields higher classification accuracies than the nearest neighbor method. However, as the percentage of training points increased, the performance of the nearest neighbor method improved to levels close to that of kernel regression.

Most of the predictions error in the numerical experiments are due to missclassifications of points belonging to the classes Allose and Mannose. Experiments were performed in which these two classes were treated separately. Using KPCA and kernel regression and 20% of the data as training points, an average prediction accuracy of 94.0% was achieved for the two classes. The value of the parameters were $\sigma = 0.15$ and $\gamma = 0.000006$ and each result was the average prediction accuracy of 20 tests. Using 80% and 99% training points, the prediction accuracy increased to 97.5% and 98.3%, respectively.

3.4 Using a modified labeling function

As described in section 2.4, the binary labels can be determined in kernel regression by the sign of the elements of the classification function $f(x)$, so that the i:th point x_i is labeled as $\text{sgn}(f(x_i))$. A possible way of improving the classification performance of certain data is to use a modified version of equation (27), namely the function

$$\text{sgn}(f(x) + \delta), \quad \delta \in R. \tag{28}$$

$\frac{n_{tr}}{n}(\%)$	KR ($\delta = 0$)	KR ($\delta = -1/3$)
20	77.0	83.1
40	79.5	85.6
60	81.6	86.9

Table 3: Prediction accuracy of kernel regression, using the modified labeling function.

The parameter δ can be used to shift the vector of label values $f(x)$ in one direction, increasing the likelihood of a $+1$ labeling ($\delta > 0$) or -1 labeling ($\delta < 0$).

It was found that $\delta = -1/3$ gave a noticeable increase of prediction accuracy for the sugar data (see Table 3.4). In these tests, the parameters were set to $\sigma = 0.0006$ and $\gamma = 0.000006$.

It is interesting to see how kernel PCA performs compared to the linear PCA method. In order to perform standard PCA the kernel function

$$k(x, y) = (x^T \cdot y) \tag{29}$$

was used, where the points x and y are represented as row vectors. The results in Table 3.4 suggests that for the sugar data, kernel PCA has a somewhat higher prediction accuracy than standard PCA. The function in (28), with $\delta = -1/3$, notably increases the performance of both methods.

$\frac{n_{tr}}{n}(\%)$	δ	PCA (%)	KPCA (%)
20	0	67.4	76.7
20	-1/3	77.3	83.3

Table 4: A comparison of PCA and KPCA, applied to sugar data.

Results presented in this section are generally the computed average of 20 or 25 tests (see the appendix for detailed numerical results).

3.5 Using label-based weights

One way of improving the separation of data points belonging to different classes is to modify the kernel matrix by using the label information that is available. There are different approaches for doing this, see for example [9]. In the tests that are described in this section, a factor, proportional to the difference in label values, was inserted into the kernel function. Two different cases are considered. In the first, the labels of the whole data set are known and used for modifying the kernel matrix. In the second case, only the training set is assumed to be known and used to modify the kernel matrix.

The RBF kernel function, that was given in equation (12), is modified by inserting a weight μ_{ij}, so that the elements of the kernel matrix are given by

$$K_{ij} = \exp(-\mu_{ij}\sigma\|x_i - x_j\|^2), \tag{30}$$

where the weight μ_{ij} is determined by

$$\mu_{ij} = \mu(y_i, y_j) = (y_i - y_j)^2. \tag{31}$$

Note that the function will then take the values

$$\mu_{ij} = \begin{cases} 0 & \text{if} \quad y_i = y_j \\ > 0 & \text{if} \quad y_i \neq y_j. \end{cases} \tag{32}$$

This way, if two points have the same label, then their inner-product in F will be one ($\mu_{ij} = 0$), meaning that the vectors representing the two points are parallel. On the other hand, if the labels are different, then the inner product between the two points in feature space will decrease or remain the same (depending on the labels), which is equivalent of the angle between the vectors of the two points becoming more right-angled.

Two different tests are performed. In one, only the label information from the training data points are used to modify the training kernel matrix, while in the other case the label information from all data points are used to modify both the training kernel matrix and test kernel matrix.

In the case in which the test kernel matrix is modified, it is computed as

$$K_{ij}^{tt} = \exp(-\mu_{ij}\sigma\|x_i^{tt} - x_j^{tr}\|^2), \tag{33}$$

for $i = 1, \ldots, n_{tt}$ and $j = 1, \ldots, n_{tr}$.

The figures presented in this section show the projection of the test data points onto the three eigenvectors that capture the largest amount of the variance, i.e. with the largest corresponding eigenvalues.

Figure 5 shows the separation of sugar data using the weights based on label-information that was described above for both the training kernel matrix and test kernel matrix. Likewise, Figure 6 shows the separation of the Iris data. This technique separates the data points very well, in the sense that the data points of each class are clustered together and the points of each class is distinctively separated from other points.

Note that the kernel becomes more discriminatory for larger values of σ, resulting in an increased distance between the clusters of points.

Previously the transduction problem has been considered, in which the labels of the training set are known and those of the test set are unknown. However, in the results of Figure 5 and Figure 6, label information for both the training and test set has been used to compute the training kernel matrix and test kernel matrix. Thus, these are not to be considered transduction problems.

Yet, the method of using weights based on label-information can be used in a transductive setting as well, i.e. where the label of only a part of the data is known and the task is to label the remaining data points. This can be done by using the labels of the training set to compute the training kernel matrix. Figure 7 shows the Iris data when the weights are only applied to the training kernel matrix. Figure 8 shows the impact of weights on the Sugar data. On the left side of the figure, the weights are used.

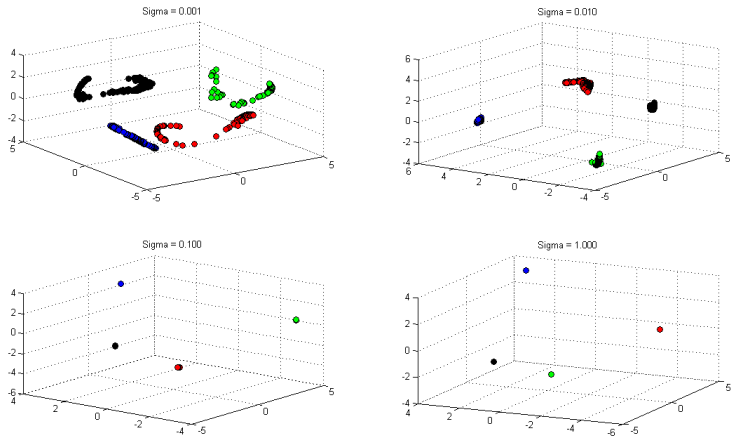

Figure 5: Test points of label-based weights in which labels of all data points are used, applied to Sugar data. The sugar classes in the plot are Allose (red), Galactose (green), Glucose (blue) and Mannose (black). The four plots show the clustering of points for different of σ. In all four cases the separations are good enough to linearly separate the points according to class, and for $\sigma = 0.1$ and $\sigma = 1$ the separations are nearly perfect.

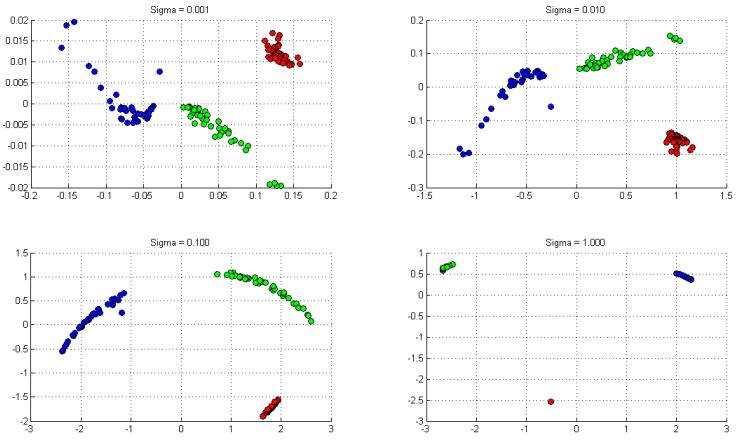

Figure 6: Test points of label-based weights in which labels of all data points are used, applied to Iris data. The plots show the separations of classes for different values of σ. In all cases it is possible to linearly separate the three classes from each other. For $\sigma = 1$ the separation is nearly perfect, in the sense that the points of each class are highly concentrated in clusters.

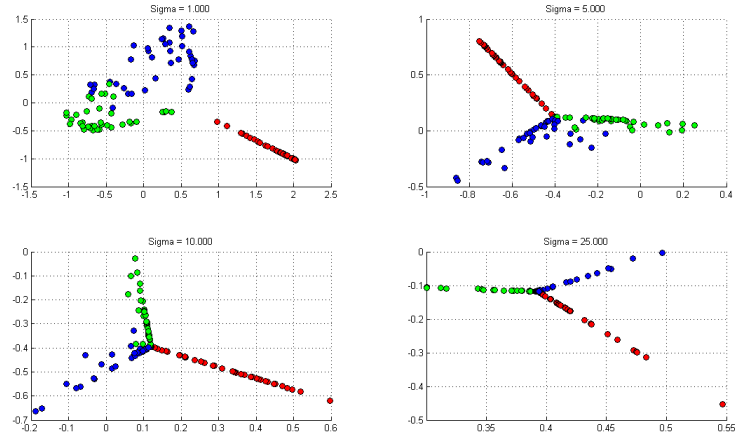

Figure 7: The figure shows test points of Iris data, when label-based weights are applied to the training kernel matrix (only the labels of the training points are used). The four plots present the resulting separation when different values of σ are used. Increased σ-values resulted in better separations.

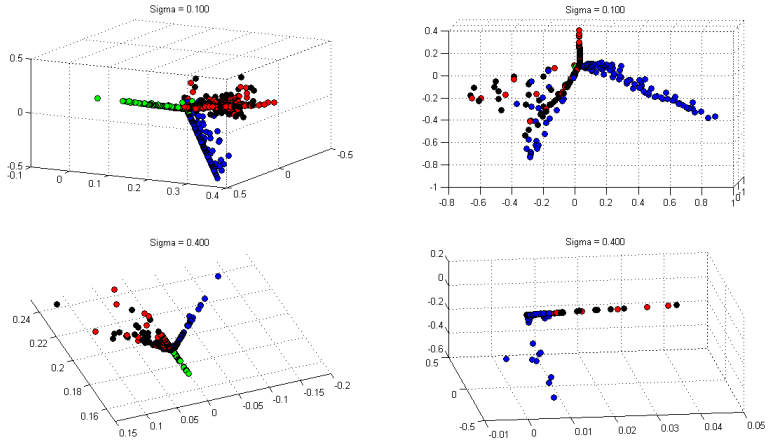

Figure 8: Test points of Sugar data, with label-based weights (left) and without weights (right). Only the labels of the training points are used in the weights. The four plots show the separation of the sugar classes for different values of σ. The sugar classes are Allose (red), Galactose (green), Glucose (blue) and Mannose (black). As can be seen from the plots, Allose and Mannose are most difficult to separate (and interestingly, chemically most alike).

4 Discussion

From the experiments it appears that for the used databases, kernel regression performs better in terms of prediction accuracy than the nearest neighbor method. However, using kernel regression comes with the cost of introducing an additional parameter γ.

The highest average prediction accuracies that were achieved using 20% of the set as training points were 97.5% for the Iris data and 83% for the Sugar data. Using 60% of the data as training data, the prediction accuracy for the Sugar data increased to around 87%.

The comparisons between kernel PCA and standard PCA showed that kernel PCA consistently gave higher prediction accuracies for the databases mentioned.

The random projection method was tested for the Iris data and Sugar data and compared to kernel PCA. In transduction problems kernel PCA outperformed random projections. This result seems reasonable, as random projection only reduces the available information into fewer dimensions, without extracting patterns out of it. Similar tests have been made on other databases before, yielding results in which kernel PCA outperforms RP (see for example [5]). The experiments carried out on the data of this project seem to suggest the same conclusion. However, it is possible that random projections can be used to decrease the computation time when performing kernel PCA on very large problems, as suggested in [1].

Most of the incorrect label-predictions of the sugar database were due to misclassifications of the sugar types Allose and Mannose. However, when treated separately, the prediction accuracy for the two sugar types was high (94.0% to 98.3%). Thus, if Allose and Mannose could first be separated from the other sugar types and then dealt with separately, the overall prediction accuracy of the sugar data could probably be significantly increased. While it has not been tried in this report, it would be an interesting topic to investigate in a future study.

5 Acknowledgments

This project was performed between August and December, 2010, at University of Florida, Gainesville, as a part of the author's Master's thesis. The author wishes to thank his advisor Prof. Pando Georgiev at the Industrial and Systems Engineering Department (ISE), University of Florida, and his examiner Prof. Ulf Jönsson at the Royal Institute of Technology (KTH) in Stockholm, for their help and supervision throughout the research project and for the opportunity to write the report at University of Florida.

A Centering in feature space

In this section the formula for centering the kernel matrix will be derived. The derivation in this section is based on [13]. The centered kernel matrix $\tilde{K}$ is to be expressed in terms of the noncentered kernel matrix K.

A set of n centered data points in feature space can be written as

$$\tilde{\Phi}(x_i) \triangleq \Phi(x_i) - \frac{1}{n}\sum_{k=1}^{n}\Phi(x_k), \tag{34}$$

for $i = 1, \ldots, n$.

The centered kernel matrix is then defined as

$$\tilde{K}_{ij} = (\tilde{\Phi}(x_i) \cdot \tilde{\Phi}(x_j)). \tag{35}$$

By inserting (34) into (35), the centered kernel matrix can be expressed in terms of $\Phi(x_i)$:

$$\begin{aligned}\tilde{K}_{ij} &= (\Phi(x_i) - \frac{1}{n}\sum_{k=1}^{n}\Phi(x_k)) \cdot (\Phi(x_j) - \frac{1}{n}\sum_{l=1}^{n}\Phi(x_l)) \\ &= \Phi(x_i)\cdot\Phi(x_j) - \Phi(x_i)\cdot\frac{1}{n}\sum_{l=1}^{n}\Phi(x_l) - \frac{1}{n}(\sum_{k=1}^{n}\Phi(x_k))\cdot\Phi(x_j) \\ &+ \frac{1}{n^2}\sum_{k=1}^{n}\Phi(x_k)\sum_{l=1}^{n}\Phi(x_l).\end{aligned}$$

Rewriting the equation above in matrix form gives

$$\begin{aligned}\tilde{K}_{ij} &= K_{ij} - \frac{1}{n}\sum_{k=1}^{n}1_{ik}K_{kj} - \frac{1}{n}\sum_{l=1}^{n}K_{il}1_{lj} + \frac{1}{n^2}\sum_{k,l=1}^{n}1_{ik}K_{kl}1_{lj} \\ &= (K - 1_n K - K1_n + 1_n K 1_n)_{ij},\end{aligned}$$

where 1_{ij} represents the element of an $n \times n$ matrix in which each element equals one, and 1_n is an $n \times n$ matrix in which each element equals $1/n$. Thus, the expression for centering the noncentered kernel matrix is

$$\tilde{K} = K - 1_n K - K1_n + 1_n K 1_n. \tag{36}$$

For the test kernel matrix the derivation of the formula used for centering is performed in a similar way, namely by starting out from the definition of the noncentered test kernel matrix

$$K^{tt}_{ij} = (\Phi(x^{tt}_i) \cdot \Phi(x^{tr}_j)), \tag{37}$$

and the centered test kernel matrix

$$\tilde{K}^{tt}_{ij} = (\tilde{\Phi}(x^{tt}_i)) \cdot (\tilde{\Phi}(x^{tr}_j)), \tag{38}$$

where K^{tt} and $\tilde{K}^{tt}$ are $n_{tt} \times n_{tr}$ matrices and $i = 1, \ldots, n_{tt}$ and $j = 1, \ldots, n_{tr}$. Then,

$$\begin{aligned}
\tilde{K}_{ij}^{tt} &= (\Phi(x_i^{tt}) - \frac{1}{n_{tr}} \sum_{k=1}^{n_{tr}} \Phi(x_k^{tr})) \cdot (\Phi(x_j^{tr}) - \frac{1}{n_{tr}} \sum_{l=1}^{n_{tr}} \Phi(x_l^{tr})) \\
&= \Phi(x_i^{tt}) \cdot \Phi(x_j^{tr}) - \frac{1}{n_{tr}} \Phi(x_i^{tt}) \cdot (\sum_{l=1}^{n_{tr}} \Phi(x_l^{tr})) - \frac{1}{n_{tr}} (\sum_{k=1}^{n_{tr}} \Phi(x_k^{tr})) \cdot \Phi(x_j^{tr}) \\
&+ \frac{1}{n_{tr}^2} (\sum_{k=1}^{n_{tr}} \Phi(x_k^{tr})) \cdot (\sum_{l=1}^{n_{tr}} \Phi(x_l^{tr})).
\end{aligned}$$

Here, the term

$$\begin{aligned}
\Phi(x_i^{tt}) \cdot (\sum_{l=1}^{n_{tr}} \Phi(x_l^{tr})) &= \Phi(x_i^{tt}) \cdot (\Phi(x_1^{tr}) + \cdots + \Phi(x_{n_{tr}}^{tr})) \\
&= \sum_{l=1}^{n_{tr}} \Phi(x_i^{tt}) \cdot \Phi(x_l^{tr}) \\
&= \sum_{l=1}^{n_{tr}} K_{il}^{tt} = \sum_{l=1}^{n_{tr}} K_{il}^{tt} 1_{lj}.
\end{aligned}$$

Likewise, the other terms in the equation can be rewritten in a similar way, leading to

$$\begin{aligned}
\tilde{K}_{ij}^{tt} &= K_{ij}^{tt} - \frac{1}{n_{tr}} \sum_{l=1}^{n_{tr}} K_{il}^{tt} 1_{lj} - \frac{1}{n_{tr}} \sum_{k=1}^{n_{tr}} 1_{ik} K_{kj}^{tr} + \frac{1}{{n_{tr}}^2} \sum_{k,l=1}^{n_{tr}} 1_{ik} K_{kl}^{tr} 1_{lj} \\
&= (K^{tt} - K^{tt} 1_{n_{tr}} - 1_{n_{tt}} K^{tr} + 1_{n_{tt}} K^{tr} 1_{n_{tr}})_{ij},
\end{aligned}$$

where $1_{n_{tt}}$ is an $n_{tt} \times n_{tr}$ matrix in which each element equals $1/n_{tr}$ and $1_{n_{tr}}$ is an $n_{tr} \times n_{tr}$ in which each element equals $1/n_{tr}$. The formula for centering the test kernel matrix is then

$$\tilde{K}^{tt} = K^{tt} - 1_{n_{tt}} K^{tr} - K^{tt} 1_{n_{tr}} + 1_{n_{tt}} K^{tr} 1_{n_{tr}}. \tag{39}$$

B The source code

In this section, parts of the MATLAB-source code that was used in the experiments are included.

B.1 Function: Evaluating prediction accuracy

The code presented in this section is the main program, used for running the software. This function depends on the other functions either directly or indirectly. The function loads the prespecified database and divides the data set into a training set and test set. The kernel matrices are then computed and the data points in the test set are labeled using two different methods: Kernel regression and the nearest neighbor method. Lastly, the results are displayed and the projection of the test points onto the three eigenvectors with largest corresponding eigenvalues are plotted (the class of the points are indicated by color).

```
function [KR_perc NN_perc] = prediction_accuracy(sigma,gamma,delt)
%Computes the prediction accuracy of kernel regression for a test set. The
%test set and the training set are selected randomly from the pool of data.
%
%Run this code as (example):
%[KR_perc,NN_perc] = prediction_accuracy(0.001,0.000006,-0.33);

tic %start timer

tr_percentage = 0.2; %Percentage of points that are used for training

nr_evecs = 5; %Number of extracted Eigenvectors

datatype = 2; %1 is Iris data and 2 is Sugar data
if datatype == 1
    data = load('Iris_data.dat');
else
    load Sugar_data; data = sugardata;clear sugardata;
end
%Center raw data (except label-column)
data(:,1:end-1)=data(:,1:end-1)-ones(size(data,1),1)*mean(data(:,1:end-1));
labels = data(:,end);
X = data(:,1:end-1);
n = size(X,1); %number of data points
d = size(X,2); %dimension of data

ntr = round(tr_percentage*n);
ntt = n - ntr; %The remaining points are test points

%---Create the data sets---
ind = randperm(n);
ind_tr = ind(1:ntr);
ind_tt = ind(ntr+1:end);
Xtr = X(ind_tr,1:end-1);
Xtt = X(ind_tt,1:end-1);
labels_tr = labels(ind_tr,end);
```

```
classes = unique(labels_tr);
nr_classes = length(classes); %number of classes
labels_tt = labels(ind_tt,end);
%----------------------------

%Compute the kernel matrices
[Ktr,Ktr_n,Ktt,Ktt_n] = compute_kernel_matrices(Xtr,Xtt,sigma);

KR_labels = zeros(ntt,1);
KR_ytt_list = zeros(ntt,nr_classes);
NN_labels = zeros(ntt,1);
NN_ytt_list = zeros(ntt,nr_classes);

%Perform binary classification on multiclass data points
for j = 1:nr_classes
    q = classes(j); %Class that should have +1 in binary labeling

    %Assign binary labels
    ytr = labels_tr; ytr(ytr≠q) = -1; ytr(ytr==q) = +1;
    ytt = labels_tt; ytt(ytt≠q) = -1; ytt(ytt==q) = +1;

    [KR_ytt_list(:,j),a] = kernel_regression(Ktr_n,Ktt_n,ytr,gamma,delt);
    NN_ytt_list(:,j) = nearest_neighbor(Ktr,Ktr_n,Ktt,ytr,nr_evecs);

    KR_labels(KR_ytt_list(:,j)==1) = classes(j);
    NN_labels(NN_ytt_list(:,j)==1) = classes(j);
end

%Evaluate prediction accuracy of Kernel Regression and Nearest neighbor
KR_perc = sum(KR_labels==labels_tt)/ntt;
NN_perc = sum(NN_labels==labels_tt)/ntt;

%Print results
disp(sprintf('\n\n-----Prediction Accuracy-----\n'))
disp(sprintf('Total number of points: %d', n))
disp(sprintf('       Training points: %d (%0.2g%%)', ntr, 100*ntr/n))
disp(sprintf('           Test points: %d (%0.2g%%)', ntt, 100*ntt/n))
disp(sprintf('\nNumber of classes: %d', nr_classes))
disp(sprintf('Extracted eigenvectors: %d', nr_evecs))
disp(sprintf('\nSigma = %0.4g', sigma))
disp(sprintf('Gamma = %0.4g', gamma))
disp(sprintf('delt = %0.4g', delt))

disp(sprintf('\nKernel regression'))
disp(sprintf('Prediction accuracy: %0.4g%%', 100*KR_perc));
disp(sprintf('Prediction error: %0.4g%%', 100*(1-KR_perc)));

disp(sprintf('\nNearest neighbor'))
disp(sprintf('Prediction accuracy: %0.4g%%', 100*NN_perc));
disp(sprintf('Prediction error: %0.4g%%', 100*(1-NN_perc)));

disp(sprintf('\nComputation time: %0.4g seconds', toc));

disp(sprintf('\n-----------------------------\n'))

%Project the tests points onto Eigenvectors and plot
```

```
[V,D] = eigs(Ktr_n,3,'LM'); %Extract 3 Evecs w/ largest corresp Eigenvalues
chi_tt = Ktt_n*V; %Project test points in F onto Eigenvectors V

hold on,
l=1;plot3(chi_tt(labels_tt==l,1),chi_tt(labels_tt==l,2),chi_tt(labels_tt==l,3),'.r')
l=2;plot3(chi_tt(labels_tt==l,1),chi_tt(labels_tt==l,2),chi_tt(labels_tt==l,3),'.g')
l=3;plot3(chi_tt(labels_tt==l,1),chi_tt(labels_tt==l,2),chi_tt(labels_tt==l,3),'.b')
l=4;plot3(chi_tt(labels_tt==l,1),chi_tt(labels_tt==l,2),chi_tt(labels_tt==l,3),'.k')

end
```

B.2 Function: Computing the kernel matrices

The function returns the noncentered and centered training kernel matrices and test kernel matrices. Required input is the training set, test set and a value for σ.

```
function [Ktr,Ktr_n,Ktt,Ktt_n] = compute_kernel_matrices(Xtr,Xtt,sigma)
%Computes the kernel matrices, from the training set Xtr and test set Xtt.
%Ktr = Training kernel matrix
%Ktt = Test kernel matrix
%Ktr_n = Centered training kernel matrix
%Ktt_n = Centred test kernel matrix

ntr = size(Xtr,1);
ntt = size(Xtt,1);

%Compute kernel matrices
Ktr = compute_train_kernel(Xtr,sigma);
Ktr_n = center_train_kernel(Ktr);

Ktt = compute_test_kernel(Xtr,Xtt,sigma);
Ktt_n = center_test_kernel(Ktt,Ktr);

end
```

The following function is used for computing the training kernel matrix:

```
function Ktr = compute_train_kernel(Xtr,sigma)
%Computes the ntr x ntr training kernel matrix Ktr.

    ntr = size(Xtr,1);
    Ktr = zeros(ntr,ntr);
    for i=1:ntr
        for j=1:ntr
            diff = Xtr(i,:)-Xtr(j,:);
            Ktr(i,j) = exp(-sigma*abs(diff * diff'));
        end
    end
end
```

The following function is used for computing the test kernel matrix:

```
function Ktt = compute_test_kernel(Xtr,Xtt,sigma)
%Computes the  ntt x ntr test kernel matrx Ktt.

    ntr = size(Xtr,1);
    ntt = size(Xtt,1);

    Ktt = zeros(ntt,ntr);
    for i=1:ntt,
        for j=1:ntr,
            Ktt(i,j) = exp(-sigma*norm(Xtt(i,:)-Xtr(j,:))^2);
        end
    end

end
```

B.3 Function: Centering the kernel matrices

The following function centers the training kernel matrix:

```
function Ktr_n = center_train_kernel(Ktr)
%Centers the training kernel matrix Ktr.

    ntr = size(Ktr,1);
    unit = ones(ntr, ntr)/ntr;
    Ktr_n = Ktr - unit*Ktr - Ktr*unit + unit*Ktr*unit; %Centering

end
```

The following function centers the test kernel matrix:

```
function Ktt_n = center_test_kernel(Ktt,Ktr)
%Centers the test kernel matrix. The input Ktr should not be centered.
    ntr = size(Ktr,1);
    ntt = size(Ktt,1);

    unit = ones(ntr, ntr)/ntr;
    unit_test = ones(ntt,ntr)/ntr;

    Ktt_n = Ktt - unit_test*Ktr - Ktt*unit + unit_test*Ktr*unit; %Centering

end
```

B.4 Function: Kernel regression

An implementation of kernel regression in MATLAB. The function seeks to fit a vector c as well as possible to the training kernel matrix, considering the label of each points (see [12] for details).

```
function [KR_ytt,fx] = kernel_regression(Ktr_n,Ktt_n,ytr,gamma,delt)
%Predicts the labels of the data points using kernel regression.
%Output:
```

```
%KR_ytt = Vector containing the predicted (binary) labels of the test set.
%fx = Non-binary label vector. KR_ytt = sign(fx+delt)

ntr = size(Ktr_n,1);

c = (ntr*gamma*eye(ntr)+Ktr_n)\ytr;
fx = Ktt_n*c;
KR_ytt = sign(fx + delt);

end
```

B.5 Function: The nearest neighbor method

The following is a function for performing classification by using the nearest neighbor method. The method assigns the same label to each unknown test point as that of its nearest training point.

```
function NN_ytt = nearest_neighbor(Ktr,Ktr_n,Ktt,ytr,nr_evecs)
%Classification using the Nearest neighbor method.
%Labels the data of the test kernel matrix Ktt, based on
%the data in the training kernel matrix Ktr, that have the
%labels given by ytr.
%
%Input:
%The training and test kernel matrices Ktr and Ktt, the centered
%training kernel matrices Ktr_n, the number of eigenvectors that are
%extracted from the (centered) training kernel matrix Ktr_n.
%
%Output:
%The predicted labels my_ytt of the test set.

ntr = size(Ktr,1);
ntt = size(Ktt,1);

Kaug = [Ktr;Ktt];

Kaug_n = center_test_kernel(Kaug,Ktr);

[V,D] = eigs(Ktr_n,nr_evecs,'LM');

chi = Kaug_n * V;
chi_tr = chi(1:ntr,:);
chi_tt = chi(ntr+1:end,:);

ind = nearest_reference(chi_tt,chi_tr);

NN_ytt = zeros(ntt,1);
for i=1:ntt
    NN_ytt(i) = ytr(ind(i));
end

end

function ind_list = nearest_reference(x,xref)
%Finds the nearest center xc for each point x. The output is a vector, the
```

```
%length of x, where the k:th vector element contains the index of the
%center that is nearest to the k:th point. The points are represented as
%rows in x.

    nr_points = size(x,1);
    nr_ref = size(xref,1);

    ind_list = zeros(nr_points,1);

    for i = 1:nr_points
        dx = ones(nr_ref,1)*x(i,:) - xref;

        dist = zeros(nr_ref,1);
        for j=1:size(dx,2)
            dist = dist + dx(:,j).^2;
        end

        [a,I] = min(dist);
        ind_list(i) = I;
    end

end
```

C Detailed Numerical Results

This section gives some of the detailed numerical results of the experiments that were presented in Section 3. All experiments were performed with randomly selected training sets. The sugar database is used in all the tests of this section.

C.1 Comparison of PCA and kernel PCA

In these experiments, kernel PCA was compared to standard PCA. The two methods were applied to sugar data. Table 5 shows the results.

The following parameters were used in the experiments:

- PCA20: 20% training points, $\delta = 0$.
- dPCA20: 20% training points, $\delta = -1/3$.
- KPCA20: 20% training points, $\sigma = 0.0006$, $\gamma = 0.000006$, $\delta = 0$.
- dKPCA20: 20% training points, $\sigma = 0.0006$, $\gamma = 0.000006$, $\delta = -1/3$.

C.2 Comparison of $\delta = 0$ and $\delta = -1/3$

These tests were performed to measure the impact that the parameter δ had on the prediction accuracy in kernel regression. Several other values of δ were tried as well (both positive and negative), but are not included in this report. $\delta = -1/3$ appeared to be optimal for the sugar data. The results are shown in Table 6. Also, some tests were performed in which kernel regression with noncentered kernel matrices were used. The following parameters were used in the experiments:

- KPCA20c: 20% training points. $\sigma = 0.25$, $\gamma = 0.7$, $\delta = 0$. In these tests a noncentered kernel matrix is used.
- KPCA40: 40% training points. $\sigma = 0.0006$, $\gamma = 0.000006$, $\delta = 0$.
- dKPCA40: 40% training points. $\sigma = 0.0006$, $\gamma = 0.000006$, $\delta = -1/3$.
- KPCA60: 60% training points. $\sigma = 0.0006$, $\gamma = 0.000006$, $\delta = 0$.
- dKPCA60: 60% training points. $\sigma = 0.0006$, $\gamma = 0.000006$, $\delta = -1/3$.

Nr.	PCA20	dPCA20	KPCA20	dKPCA20
1	66.20	80.00	76.00	86.80
2	68.40	80.90	76.00	82.80
3	63.70	78.40	77.40	84.20
4	67.30	79.50	77.40	85.90
5	67.50	74.50	76.50	83.10
6	68.80	73.00	78.50	83.90
7	65.30	78.00	82.40	83.30
8	69.40	80.90	76.20	82.80
9	68.10	74.10	79.60	77.10
10	67.90	78.50	78.20	79.10
11	70.60	75.40	76.50	84.40
12	66.40	79.30	71.70	86.20
13	68.80	77.60	75.80	84.20
14	64.20	78.70	75.10	84.20
15	68.60	76.50	76.20	83.70
16	67.70	76.30	73.00	80.40
17	64.40	77.10	78.50	84.80
18	65.90	74.10	72.30	84.00
19	71.40	77.30	77.80	84.40
20	68.80	79.30	73.40	81.30
21	67.70	80.40	79.10	80.90
22	66.40	71.70	75.20	83.30
23	65.70	76.50	78.40	84.00
24	65.90	74.50	77.30	82.90
25	68.60	80.70	78.20	85.10
Average:	67.40	77.30	76.70	83.30

Table 5: Prediction accuracies of PCA and KPCA applied to sugar data.

Nr.	KPCA20c	KPCA40	dKPCA40	KPCA60	dKPCA60
1	79.80	79.70	87.50	82.00	86.80
2	79.30	79.70	86.30	81.30	87.90
3	80.70	78.70	85.10	82.70	88.20
4	80.90	82.20	87.30	80.20	83.80
5	78.90	80.90	85.10	84.60	85.30
6	80.70	80.90	86.60	80.50	90.40
7	83.50	80.00	85.30	84.60	90.80
8	79.80	81.70	87.00	79.40	85.30
9	80.90	80.00	83.90	79.80	87.90
10	83.50	80.00	85.80	80.90	88.60
11	79.80	79.70	86.60	82.40	85.30
12	81.30	79.70	84.10	80.20	86.00
13	83.30	75.80	85.80	81.60	86.40
14	81.80	78.70	84.10	79.40	83.50
15	79.50	79.70	88.00	76.10	86.40
16	83.30	79.20	84.60	84.60	84.20
17	82.80	78.50	81.40	84.60	83.80
18	83.70	79.20	85.10	82.40	84.20
19	80.60	82.60	86.60	81.30	88.60
20	80.00	76.50	84.60	82.40	86.80
21	79.60	77.00	85.10	82.70	88.20
22	81.80	80.70	86.30	80.90	88.60
23	79.60	76.80	86.80	80.90	90.40
24	82.60	79.20	87.00	83.80	87.10
25	83.10	80.00	84.10	80.20	87.10
Average:	81.20	79.50	85.60	81.60	86.90

Table 6: Comparison of prediction accuracies using $\delta = 0$ and $\delta = -1/3$.

D Other attempted procedures

In this section, a summary of two additional methods that were tested in the project will be given: Random projections and a serie of experiments involving semidefinite programming applied to label-based weights.

D.1 Random projections

Random projections (RP) is a dimensionality reduction method that projects a set of high dimensional data points onto randomized base vectors, so that the resulting data points have a lower dimensionality than the original data [2, 3]. The projection does not cause any major distortion of the data: A result by Johnson and Lindenstrauss [6] guarantees that the distances between the points will be preserved through the projection to some certain accuracy. The following lemma presents the result.

Lemma D.1 (Johnson-Lindenstrauss)**.** *Given $\epsilon > 0$ and an integer n, let k be a positive integer such that $k \geq k_0 = O(\epsilon^{-2} \log n)$. For every set X of n points in R^d there exists $f : R^d \to R^k$ such that for all $u, v \in P$*

$$(1 - \epsilon)\|u - v\|^2 \leq \|f(u) - f(v)\|^2 \leq (1 + \epsilon)\|u - v\|^2. \tag{40}$$

RP can more specifically be defined in the following way: Let X be an $n \times d$ matrix representing n data points with d attributes each, so that the i:th data point is represented by the i:th row in the matrix. The goal is to reduce the dimensionality of the data in X to k dimensions ($k < d$), represented in the $n \times k$ matrix E.

Let R be a $d \times k$ matrix in which the elements are determined by the probability distribution

$$R_{ij} = \sqrt{3} \times \begin{cases} +1 & \text{with probability } 1/6 \\ 0 & \text{with probability } 2/3 \\ -1 & \text{with probability } 1/6. \end{cases} \tag{41}$$

Let E, i.e. the low-dimensional representation of the points, be determined by

$$E = \frac{1}{\sqrt{k}} XR, \tag{42}$$

where X is the $n \times d$-matrix containing the data. The matrix E thus contains the random projection of the data points in X. While E has significantly lower dimensionality than the original data set X, the Johnson-Lindenstrauss lemma gives a guarantee that the overall relative distance between the data points in E are nearly the same as in X.

In experiments, RP was tested for the Iris data and Sugar data. While the computation time of RP is significantly lower than for kernel PCA, it could not match the ability of kernel PCA to recognize features in the different data classes. In the experiments of [5], RP underperforms PCA. The experiments carried out on the data of this project seem to point to the same conclusion.

However, as RP preserves distances between points quite accurately through the dimension reduction, the method could possibly serve as a preprocessing tool for kernel PCA; By using RP to reduce the dimensionality of the data before computing the kernel matrices, the computation time could possibly be reduced for kernel PCA. Some ways of doing this have in fact been suggested in [1].

D.2 Optimizing label based-weights using SDP

The use of semidefinite programming (SDP) for kernel PCA has been suggested by [7] as a way of finding a better kernel matrix for a data set. A semidefinite program is a problem with the following form:

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & c^T u \\ \text{subject to} \quad & F^j(u) = F_0^j + u_1 F_1^j + \cdots + u_q F_q^j \preceq 0, \quad j = 1, \ldots, L \\ & Au = b, \end{aligned} \tag{43}$$

where $u \in R^q$ contains the decision variables, $c \in R^q$ is the objective vector and the symmetric matrices $F_i^j \in R^{p \times p}$ are given. An introduction to semidefinite programming can be found in [16].

In this section, we consider the SDP problem

$$\begin{aligned} \underset{w}{\text{maximize}} \quad & \sum_{i=1}^{n} \Lambda_i(\hat{K}) \\ \text{subject to} \quad & M = M_0 + \sum_{l=1}^{q} w_l M^l \\ & M \succeq 0 \\ & \hat{K}_{ij} = M_{ij} \cdot K_{ij}, \quad i, j = 1, \ldots, n \end{aligned} \tag{44}$$

where $\hat{K}$, K and M are $n \times n$ matrices and w is a weight vector of length q, i.e. the number of classes in the data. $\Lambda_i(\hat{K})$ denotes the i:th eigenvalue of $\hat{K}$. Matrix M is a linear combination of q matrices M^l, which are defined as

$$M_{ij}^l = \begin{cases} 1 & \text{if} \quad label(i) = label(j) = l \\ 0 & \text{otherwise.} \end{cases} \tag{45}$$

for $l = 1, \ldots, q$. M_0 is a $n \times n$ matrix in which each element is one.

The constraint $M \succeq 0$ indicates that M is required to be positive semidefinite. The optimization problem in (44) seeks to maximize the sum of the eigenvalues of $\hat{K}$, by using the weight vector w to modify the kernel matrix.

The problem formulation is based on the following reasoning: The eigenvalues give an indication of how much variance the corresponding eigenvectors capture. The idea is thus that by maximizing the sum of eigenvalues a kernel matrix is produced that have eigenvectors that capture a large amount of the information in the data set.

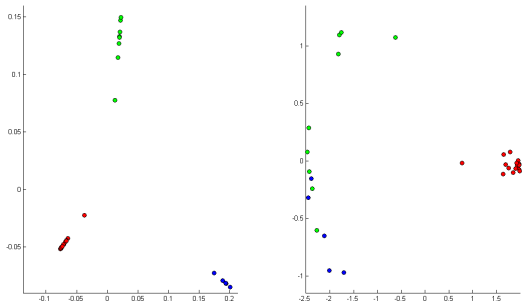

Figure 9: Separation of Iris data according to class, using weights (left) and no weights (right). The weights are based on the label information of the data points and optimized by using semidefinite programming. When the label information from the entire data set was used, the weights caused the data points to be well-separated.

It can be shown that if a square matrix K is symmetric, then

$$\text{trace}(K) = \sum_{i=1}^{n} \Lambda_i(K), \tag{46}$$

where $\Lambda_i(K)$ denotes the i:th eigenvalue of K and $\text{trace}(K) \triangleq \sum_{i=1}^{n} K_{ii}$. Thus, the optimization problem in (44) can be rewritten as

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & -t \\ \text{subject to} \quad & M = M_0 + \sum_{l=1}^{q} w_l M^l \\ & M \succeq 0 \\ & \hat{K}_{ij} = M_{ij} \cdot K_{ij}, \quad i,j = 1,\ldots,n \\ & \text{trace}(\hat{K}) \geq t \end{aligned} \tag{47}$$

The SDP problem was solved in MATLAB by using the solver SeDuMi [15]. Using SDP to determine the weights increased the separation, but only when the label-information for the whole set is available. Figure 9 shows the projection of points before (left) and after (right) the SDP-optimized label weights were applied.

It was found that the method is computationally very demanding for larger sets, probably due to the large number of variables that is contained in M. Because of this, tests with only a small number of points could be performed. Furthermore, the method did not improve the prediction accuracy in transduction problems.

It is possible to improve the problem formulation above. For example, it is not necessary to maximize the sum of all eigenvalues, because usually only a small number of eigenvectors are used in kernel PCA, and always the eigenvectors with the largest corresponding eigenvalues. Therefore, an alternative objective function would be to maximize the r first (i.e. largest) eigenvalues. This way the largest amount of variance is stored in the r first eigenvectors. Thus the objective function could possibly find weights w that capture most of the variance on a small number of eigenvectors. While this formulation is thought to be an improvement of the previous optimization problem, a feasible way of formulating this problem in SeDuMi could not be found.

References

[1] D. Achlioptas, F. Mcsherry, and B. Schölkopf. Sampling techniques for kernel methods. In *In Annual Advances in Neural Information Processing Systems 14: Proceedings of the 2001 Conference*, pages 335–342. MIT Press, 2001.

[2] Dimitris Achlioptas. Database-friendly random projections. *In Proc. ACM Symp. on the Principles of Database Systems*, pages 274–281, 2001.

[3] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: Applications to image and text data. *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)*, pages 245–250, 2001.

[4] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Trans. Info. Theory*, 13:21 – 27, 1967.

[5] Dmitriy Fradkin and David Madigan. Experiments with random projections for machine learning. In *In KDD 03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 517–522. ACM Press, 2003.

[6] William Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemporary Mathematics*, pages 189–206. American Mathematical Society, 1984.

[7] Gert R.G. Lackriet, Nello Christiani, Peter Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[8] Jin-Seon Lee and Il-Seok Oh. Binary classification trees for multi-class classification problems. In *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 770, Washington, DC, USA, 2003. IEEE Computer Society.

[9] Renqiang Min, Anthony Bonner, and Zhaolei Zhang. Modifying kernels using label information improves svm classification performance, 2007.

[10] A. Narasimhamurthy. Theoretical bounds of majority voting performance for a binary classification problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(12):1988 –1995, dec. 2005.

[11] Panos M. Pardalos and Pierre Hansen, editors. *Data Mining and Mathematical Programming*. American Mathematical Society, 2008.

[12] Tomaso Poggio and Steve Smale. The mathematics of learning: Dealing with data. *Notices of the AMS*, 50:537 – 544, 2003.

[13] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299 – 1319, 1998.

[14] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[15] Jos F. Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones, 1998.

[16] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, 1994.