Universitat Politècnica de Catalunya

Bachelor's Degree in Engineering Physics

Bachelor's Thesis

Least Squares Regression Principal Component Analysis

A supervised dimensionality reduction method for machine learning in scientific applications

---

Author:
Héctor Pascual Herrero

Supervisor:
Dr. Xin Yee
Dr. Joan Torras

June 29, 2020

Abstract

Dimension reduction is an important technique in surrogate modeling and machine learning. In this thesis, we present three existing dimension reduction methods in detail and then we propose a novel supervised dimension reduction method, 'Least Squares Regression Principal Component Analysis" (LSR-PCA), applicable to both classification and regression dimension reduction tasks. To show the efficacy of this method, we present different examples in visualization, classification and regression problems, comparing it to state-of-the-art dimension reduction methods. Furthermore, we present the kernel version of LSR-PCA for problems where the input are correlated non-linearly. The examples demonstrated that LSR-PCA can be a competitive dimension reduction method.

Acknowledgements

I would like to express my gratitude to my thesis supervisor, Professor Xin Yee. I would like to thank her for giving me this wonderful opportunity and for her guidance and support during all the passing of this semester, putting herself at my disposal during the difficult times of the COVID-19 situation. Without her, the making of this thesis would not have been possible. I would like to extend my thanks to Mr. Pere Balsells, for allowing students like me to conduct their thesis abroad, as well as to the Balsells Foundation for its help and support throughout the whole stay.

In addition, I would like to express my thanks to the second supervisor of this thesis, Professor Joan Torras, for helping me in the final stretch of the project, being as helpful as attentive.

Finally, I wish to express my most sincere appreciation to my parents and my sister, Alicia, and to my friends, for their support and encouragement during the whole stay.

Contents

Introduction

During the last decade, the access to large amounts of data has risen considerably. These sets of data, the data sets, are very useful to make estimations or predictions of the response or target values of new similar sets of data. However, analyzing and organizing data in high-dimensional spaces can lead to difficulties in processing this data. In addition, the points we have in the data set are a small and non-representative representation of the overall data. This is known as the *curse of dimensionality*.

The curse of dimensionality is a major challenge for machine learning (ML) in scientific applications. Canonical scientific examples of this challenge are found in the learning of quantitative structure-property relationships for drug design and gene-expression microarray experiments used in diagnosis and prognosis of diseases. In addition, the visualization of the data becomes extremely difficult beyond three dimensions. To alleviate the curse of dimensionality, it is necessary to reduce the number of input dimensions to a ML model.

Take for example this situation: we are an experimenter who is trying to understand some phenomenon by measuring various quantities (e.g. spectra, voltages, velocities) in our system. Unfortunately, we may not be able to figure out what is happening because the data appears to be unclear and even redundant. This is problem is a fundamental obstacle in empirical science. Examples abound from complex systems such as neuroscience, photometry, meteorology and oceanography, where the number of variables to measure can be heavy and sometimes even deceptive, because the underlying relationships between the input and response variables can often be quite simple [1].

Being an ignorant experimenter, we might not know which axes and dimensions are important to measure. We often do not know which measurements best reflect the dynamics of our system in question. Thus, we sometimes record more dimensions than we actually need! Moreover, we have to deal with that annoying, real-world problem of noise [1].

The goal of dimensionality reduction methods is to project the data onto a subspace with fewer dimensions. In this lower dimensional subspace the transformed data must be similar to the original one. Determining this fact allows an experimenter to discern which dynamics are important, which are just redundant and which are just noise. One can conclude that the most interesting dynamics occur only in the first k dimensions. This process of throwing out the less important axes can help reveal hidden, simplified dynamics in high dimensional data. This is aptly named *dimensionality reduction*.

The fewer input dimensions of our data set, the fewer parameters and the simpler structure in the machine learning model. If a model has too many input parameters, it is likely to overfit the training data set and therefore might not perform well on new data. By performing dimension on a data set, we can reduce the time and storage required to process it and we can improve the performance of the learning model. Moreover, it can provide an easy visualization of the data when it is reduced to two or three dimensions.

The techniques in dimension reduction refer to the possible processes of converting a set of data from a vast dimension into a data with lesser dimensions. Dimension reduction methods include: feature selection, linear algebra methods, projection methods and autoencoders. In this thesis, we will focus in projection methods. Similar to other types of machine learning, the efficacy of a dimension reduction method is problem dependent. Also, the methods can be classified in supervised and unsupervised learning.

Principal component analysis (PCA) [2] has been the work-horse of data-dimensionality reduction. Its popularity owes to its simplicity in theory and implementation. However, the principal components from PCA are selected completely independent of the target. Therefore PCA only explains the variance of the *input data*, and not the variance of the *target data*. This down-side lead to the development of several *supervised* dimension reduction methods [3, 4, 5]. In this thesis, we will present a new supervised dimension reduction method, which has been named as *Least Squares Regression Principal Component Analysis* (LSR-PCA) [6]. It is a method that identifies the components of the input data that contribute linearly to constructing a kernel-transformed target data. We will also show that PCA is a special case of LSR-PCA, and that LSR-PCA can be solved in closed-form with a simple generalized eigenvalue problem. Besides, we will show how LSR-PCA can be extended to solve non-linear dimensionality reduction tasks by means of a kernel formulation.

The thesis is organized as follows: Chapter 1 describes the theoretical background needed to understand the foundations of this work. A detailed overview of three state-of-the-art dimension reduction methods is given in Chapter 2. In Chapter 3, we present and describe the proposed new supervised dimension reduction method, Least Squares Regression PCA. Chapter 4 gives a detailed description of how to transform the methods to be applicable to non-linear problems and presents the kernel formulation for LSR-PCA. Next, Chapter 5, gives an overview of the code implementation that we have applied in this thesis to compare the performance of the LSR-PCA method against the other existing methods explained in Chapter 2. The results of this comparison are provided in Chapter 6, with examples in visualization, classification and regression problems. Finally, the thesis concludes in Chapter 7.

Chapter 1

Theoretical Background

The aim of this chapter is to establish the foundations of the theoretical terminology that will appear throughout the thesis. First, we will introduce the difference between supervised and unsupervised learning. Second, we will explain the concept of cross validation. Finally, we will give a description of the two machine learning models we will use in the thesis: k-Nearest Neighbors and Linear Regression.

1.1 Supervised and Unsupervised Learning

Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are two major categories: supervised learning and unsupervised learning. The main difference between the two types is that supervised learning is done using a prior knowledge of what the output values for the samples should be, while unsupervised learning is not.

1.1.1 Supervised learning

The goal of supervised learning [7] is to learn a function that, given a sample of data and desired outputs, best approximates the relationship between input and output observable in the data.

It is called supervised learning because the process of an algorithm learning from the training data set can be thought of as a teacher supervising the learning process. Since we know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the "teacher". Learning stops when the algorithm achieves an acceptable level of performance.

Supervised learning problems can be further grouped into classification and regression problems. In classification, the task is to map input data to output labels, called classes. In regression, the task is to predict a target numerical value given a set of features called predictors. In both regression and classification, the goal is to find specific relationships or structure in the input data that enable to effectively produce correct output data. Determining whether the output is correct or not is done entirely from

the training data. Therefore, despite having a ground truth that the model will assume is true, it is not to say that that data labels are always correct in real-world situations. Noisy or incorrect data labels will clearly reduce the effectiveness of the model.

Some of the most important supervised machine learning algorithms are:

- Linear and Logistic regression
- Decision Trees and Random Forests
- Support vector machines (SVMs)
- k-Nearest Neighbors

1.1.2 Unsupervised learning

In unsupervised learning [7], the training data is unlabeled, so its goal is to infer the natural structure present within a set of data points. These are called unsupervised learning because unlike supervised learning there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data.

The most common use-cases for unsupervised learning are clustering, association and dimensionality reduction. In a clustering problem one tries to discover the inherent groupings in the data. The most used algorithms for this problem are k-Means, Hierarchical Cluster Analysis (HCA) and Expectation Maximization. Another common unsupervised task is association rule learning, in which the goal is to dig into large amounts of data and discover rules and interesting relations between attributes. Some useful algorithms are Apriori and Eclat.

Finally, dimensionality reduction has been typically known for its utility in unsupervised learning. Its renown comes from the capacity of making data processing much less intensive from eliminating redundant features. The most popular dimensionality reduction techniques are Principal Component Analysis (PCA), Kernel PCA (KPCA) and Locally-Linear Embedding (LLE). However, recently, some other methods that incorporate information about the target variable have been developed. This is due to the fact that taking information from both the input and target variables can lead to a better dimension reduction performance. These methods are known as supervised dimension reduction method, and are the case of our proposed method, LSR-PCA. There exist other supervised methods, like two of the methods we will compare against LSR-PCA: Partial Least Squares Regression and Supervised PCA. But there are many more approaches that will not be considered here, like Fisher's Discriminant Analysis (FDA), the family of methods known as Metric Learning (ML), or the family of Sufficient Dimension Reduction (SDR) algorithms, among others.

1.2 Cross Validation

Cross validation is a form of model validation. In data analysis validation refers to the process of deciding whether the numerical results quantifying hypothesized relationships between variables are acceptable as descriptions of the data. Generally, these estimations are made by residual evaluations, i.e., an error estimation for the model after the training is made. In this process, a numerical estimate of the difference in predicted and original responses is done, also called the training error. The problem with residual evaluations is that they do not give an indication of how well the learner will do when it is asked to make new predictions for data it has not already seen.

One way to overcome this problem is to not use the entire data set when training a learner. Some of the data is removed before training begins. Then, after the training is done, the data that was removed can be used to test the performance of the learned model on "new" data. This is the basic idea for a whole class of model evaluation methods called cross validation.

1.2.1 Non-exhaustive methods

Non-exhaustive cross validation methods do not compute all ways of splitting the original sample. The most used ones are the Holdout method and the K-fold cross-validation.

The holdout method [8], sometimes called test sample estimation, is the simplest kind of cross validation. It partitions the data into two mutually exclusive subsets called a training set and a test set, or holdout set. The size of each of the sets is arbitrary although it is common to designate 2/3 of the data as the training set and the remaining 1/3 as the test set.

In contrast to other cross-validation methods, the holdout method involves a single run. The function approximator fits a function using the training set only. Then, the function approximator is asked to predict the output values for the data in the testing set, values which have never been seen. The errors it makes are accumulated to give the mean absolute test error, which is used to evaluate the model. This method can be effective and computationally inexpensive on very large data sets and therefore, usually preferable to the residual method. However, its evaluation can have a high variance. The evaluation may depend heavily on which data points end up in the training set and which end up in the test set, and thus the evaluation may be significantly different depending on how the division is made.

As there is never enough data to train your model, removing a part of it for validation poses a problem of underfitting. By reducing the training data, there is the risk of losing important trends in data set, which in turn increases the error induced by bias. Hence, it is required a method that provides sample data for training the model and also leaves sample data for validation. K-Fold cross validation does exactly that.

In K-Fold CV [9], the data is divided into k subsets and the holdout method is repeated k times. Each time, one of the k subsets is used as the validation set and the other $k-1$ subsets are put together to form a training set. The error estimation

is averaged over all k trials to get total effectiveness of the model. Every data point gets to be in a validation set exactly once, and gets to be in a training set $k-1$ times. This significantly reduces bias as most of the data is being used for fitting, and also significantly reduces variance as most of the data is also being used in validation set. Interchanging the training and test sets also adds to the effectiveness of this method. As a general rule and empirical evidence, $k = 5$ or 10 is generally preferred, but nothing's fixed and it can take any value [10].

1.2.2 Exhaustive methods

Leave-P-Out CV [11] is an approach that involves using p observations as the validation set and the remaining observations as the training set. For instance, if there are n data points in the original sample then, $n-p$ samples are used to train the model and p points are used as the validation set. This is repeated for all combinations in which original sample can be separated this way, and then the error is averaged for all trials, to give overall effectiveness. This method is exhaustive in the sense that it needs to train and validate the model for all possible combinations, and for moderately large p, it can become computationally infeasible.

A particular case of this method is when $p = 1$ [10]. This is known as Leave one out cross validation. This method is generally preferred over the previous one because it does not suffer from the intensive computation, as number of possible combinations is equal to number of data points in the original sample or n. Leave-one-out cross validation is K-fold cross validation taken to its logical extreme, with k equal to n, the number of data points in the set. That means that n separate times, the function approximator is trained on all the data except for one point and a prediction is made for that point. As before, the average error is computed and used to evaluate the model

1.3 K-Nearest Neighbors

As it has been previously mentioned, k-Nearest Neighbors (KNN) [12] is a method of supervised learning used for classifying objects based on closest training examples in the feature space. In pattern recognition, KNN is the fundamental and simplest classification technique when there is no prior knowledge about how the data is distributed. Basically, KNN assigns to each new data point a class represented by the majority label of its k-nearest neighbors in the training set. The simplest form of KNN is the Nearest Neighbor rule (NN), which corresponds to the case when $k = 1$.

The concept of this method is to classify each unknown sample similarly to its surrounding samples. In this way, one is predicting the unknown label by considering the classification of its nearest neighbor samples. The first step when given an unknown sample is to calculate all the distances between this new sample and all the samples in the training set. Then, the collection of distances and indices must be sorted from smallest to largest. The smallest k distances correspond to the samples in the training

set closest to the unknown sample. Therefore, the unknown sample is classified based on the labels of its nearest neighbors.

It is worth noting that the performance of the method has a high dependence on the selected value for k, since it will determine the neighborhood size, as well as the distance metric applied. The radius of the local region determined by the distance of the kth nearest neighbor to the new data point may yield different conditional class probabilities for different k values. Sometimes, a small k value might be enough for a good prediction, although it tends to be very sensitive to noisy data, leading to ambiguous or mislabeled points. By increasing k and selecting a large region around the unknown sample one can smooth the estimation. However, a large value of k will not always be the solution, as it can make estimation over smoothed when introducing outliers from other classes. We can see how important is to know how to select the most suitable neighborhood size k in each situation. This selection is a key issue that can largely affect the classification performance of KNN.

1.4 Linear Regression

As for regression problems in supervised learning, linear regression [13] [14] is one of the most attractive methods thanks to its simple representation. In statistics, regression is a method for modelling a target value based on independent predictors. This method is typically used for forecasting and finding out cause and effect relationship between variables. There are different regression techniques depending on the number of independent variables and the type of relationship between the independent and dependent variables.

The representation of linear regression is a linear equation that combines a specific set of input values to give a predicted output. In the case of simple linear regression, with a single explanatory variable, the form of the model would be:

$$\hat{y} = \theta_0 + \theta_1 \cdot x \tag{1.1}$$

where $\hat{y}$ is the predictive output and θ_0 and θ_1 are the model parameters. This linear equation assigns one scale factor θ_1 to the input value called coefficient. One additional coefficient θ_0 is also added, the bias coefficient or intercept, which gives the line an additional degree of freedom. In the general case of multiple input features, the linear regression model form is:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n = \theta^T \cdot X \tag{1.2}$$

where again $\hat{y}$ is the predicted value and θ_j are the coefficients for the x_i feature values.

In order to obtain the model parameters that best fit the training set we would need a measure of how well is it fitting it. The most common performance measure of a regression model is the Root Mean Square Error (RMSE), but in practice it is simpler to minimize the Mean Square Error (MSE) and it leads to the same result. Therefore,

to train the Linear Regression model, one needs to find the value of θ that minimizes the MSE. We have converted the search problem into a minimization problem where we would like to minimize the error between the predicted value and the actual value. The MSE is defined as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2 \tag{1.3}$$

Hence, our minimization problem is described as:

$$\text{minimize } \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2 \tag{1.4}$$

With that, we have defined the cost function, that is, the function we want to minimize, as the MSE.

The next important concept needed when applying Linear Regression is an iterative algorithm capable of finding optimal solutions to a wide range of problems. Gradient Descent is a very generic and useful method for updating the model parameters to reduce the cost function. This method attempts to measure the local gradient of the error function with regards to the parameter vector θ, and goes in the direction of the steepest slope (descending gradient) until the gradient is zero and the minimum is reached. In concrete, Gradient Descent starts by selecting a random θ value and improves it gradually, each step attempting to decrease the cost function until the algorithm converges to a minimum.

In the case of Linear Regression, the MSE cost function happens to be a convex and continuous function. This implies that there is just one global minimum with a slope that never changes abruptly, and guarantees a close approach to the global minimum.

The implementation of Gradient Descent yields on the computation of the gradient of the cost function with regards to each model parameter θ_j. This is done through the computation of partial derivatives or more directly applying the gradient vector which contains all the partial derivatives of the cost function.

$$\nabla_\theta MSE(\theta) = \begin{bmatrix} \frac{\partial}{\partial\theta_0}MSE(\theta) \\ \frac{\partial}{\partial\theta_1}MSE(\theta) \\ \vdots \\ \frac{\partial}{\partial\theta_n}MSE(\theta) \end{bmatrix} = \frac{2}{n}X^T \cdot (X \cdot \theta - y) \tag{1.5}$$

Finally, each step of this algorithm should go in the *downhill* direction, thus the opposite direction to the gradient vector obtained. The size of the step will be determined by the learning rate η.

$$\theta^{(i+1)} = \theta^{(i)} - \eta\nabla_\theta MSE(\theta) \tag{1.6}$$

It is worth mentioning the importance of the η parameter, since while a learning rate too small will take a long time to converge, a too high value of this step can make the algorithm diverge, failing to find a good solution.

Chapter 2

Related works

The following chapter will explain in detail 3 different existing dimension reduction methods, one unsupervised, Principal Component Analysis, and two supervised, Partial Least Squares Regression and Supervised PCA. The aim of this sections is to understand not only the purpose of each method but also the mathematical framework from which they are developed, so that we can understand the theoretical framework in which rises our proposed method.

2.1 Principal Component Analysis

Principal Component Analysis (PCA) [15] has been called one of the most valuable results from applied linear algebra. PCA is used abundantly in all forms of analysis, from neuroscience to computer graphics, because it is a simple, non-parametric method of extracting relevant information from confusing data sets. With minimal additional effort PCA provides a roadmap for how to reduce a complex data set to a lower dimension to reveal the, sometimes hidden, simplified structure that often underlie it.

The goal of PCA is to find another basis which is a linear combination of the original basis and best re-expresses the original data set. To do so, the first assumption that PCA makes is linearity. Given an $n \times d$ matrix X, the original data set where each column is a single sample and each, and another $n \times p$ matrix $\widetilde{X}$ related by a linear transformation $U \in \mathbb{R}^{d \times p}$ [16].

$$\widetilde{X} = XU \tag{2.1}$$

In addition, let us define the $d \times n$ matrix $\hat{X}$, as the reconstructed data set:

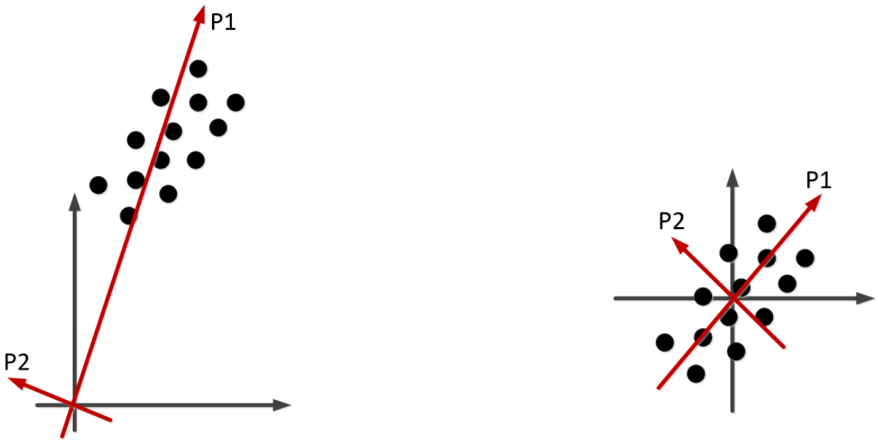
$$\widehat{X} = XUU^T \tag{2.2}$$

If the vectors of U are orthonormal, that is, the matrix U is orthogonal, then we have $U^T = U^{-1}$ and thus $U^T U = I$.

The Equation 2.1 can be interpreted as the projection onto the PCA subspace: The XU projects X onto the row space of U. It is a subspace because we have $p \leq d$ where

p and d are the dimensions of the PCA subspace and the original X, respectively. Likewise, Equation 2.2 can be interpreted as the reconstruction form the PCA, since $(XU)U^T$ projects XU, the projected data, back onto the column space of U. By assuming linearity the problem reduces to finding the appropriate change of basis. The column vectors of U will become the principal components of X. With the problem set out, the question that arises now is what is the best way to "re-express" X, that is, what is the best choice of basis U.

It is important to note that in PCA, all the data points should have zero mean, that is, be centered. The reason is shown in Figure 2.1. From now on, we assume X to be the centered data matrix.



(a) (b)

Figure 2.1: The principal components P1 and P2 for 2.1a non-centered and 2.1b centered data. As it is shown, the data should be centered for PCA. [16]

2.1.1 PCA Intuitive goal

In order to solve the laid out problem we will first build up an intuitive answer to the question. The main factors to take into account in a linear system are noise, rotation and redundancy [1].

Generally, all noise is measured relative to the measurement. If the measurement noise is very high, no information about the system can be extracted. A typical measure is the *signal-to-noise ratio (SNR)*, defined as:

$$SNR = \frac{\sigma^2_{signal}}{\sigma^2_{noise}}. \tag{2.3}$$

A high SNR indicates high precision in the data. Therefore, quantitatively we assume that directions with largest variances, and presumably highest SNR, in our measurement

vector space contain the dynamics of interest. The naive chosen basis for the experiment might not reflect the directions of largest variance. Maximizing the variance corresponds to finding the appropriate rotation of the naive basis, in such a way that the new directions reveal the dynamics of interest. The last confounding factor in the data is redundancy. This has to do with the fact that sometimes, there are variables which are highly correlated, in the sense that one can calculate one of these from another, and solely one response would express the data more concisely and reduce the number of features.

2.1.2 PCA Using Eigen-Decomposition

Recalling Equation 2.2, the projected data are $\widetilde{X} \in \mathbb{R}^{n \times p}$ and the reconstructed data are $\widehat{X} \in \mathbb{R}^{n \times d}$. The squared Frobenius norm of this reconstructed matrix $\widehat{X}$ is [16]:

$$\begin{aligned} \|\widehat{X}\|_F^2 &= \|XUU^T\|_F^2 \\ &= \mathbf{tr}((XUU^T)(XUU^T)^T) \\ &= \mathbf{tr}(XUU^TUU^TX^T) \\ &= \mathbf{tr}(XUU^TX^T) \\ &= \mathbf{tr}(U^TX^TXU) \end{aligned}$$

where $\mathbf{tr}(.)$ stands for the trace of matrix, and we have used the orthogonality of U and the cyclic property of the trace. Considering $X = [x_1, ..., x_n]^T \in \mathbb{R}^{n \times d}$, we have:

$$S := \sum_{i=1}^{n} x_i^T x_i = X^T X, \tag{2.4}$$

where $S \in \mathbb{R}^{d \times d}$ is called the *covariance matrix*. Therefore, we can rewrite the squared Frobenius norm as:

$$\|\widehat{X}\|_F^2 = tr(U^TSU) \tag{2.5}$$

Note that we can also say that U^TSU stands for the variance of the projected data onto PCA subspace. Therefore, $U^TSU = \mathbb{V}ar(XU)$ and U^TSU can be interpreted in two ways. The first one, as the squared Frobenius norm of reconstruction and the second one as the variance of the projection, as argued in the section 2.1.1. The problem posed relies on finding the projection matrix of directions U which maximize the squared Frobenius norm of reconstruction (or variance of projection) [16]:

$$\begin{aligned} &\max_U \ tr(U^TSU), \\ &\text{subject to } \ U^TU = I, \end{aligned} \tag{2.6}$$

where the orthogonality of U is ensured in the constraint. To solve the optimization problem we can use the method of Lagrange multipliers [17], and get:

$$\mathcal{L} = tr(U^TSU) - tr(\Lambda^T(U^TU - I)),$$

where $\Lambda \in \mathbb{R}^{p \times p}$ is a diagonal matrix containing the Lagrange multipliers. Taking the derivative of the Lagrangian and setting it to zero gives:

$$\frac{\partial L}{\partial U} = 2SU - 2U\Lambda = 0$$

$$\Rightarrow SU = U\Lambda \tag{2.7}$$

which is the eigenvalue problem for the matrix S. The eigenvectors of S are stored in the columns of the U matrix. The eigenvalues of S are stored in the diagonal of the Λ matrix. Moreover, the variances associated with each direction u_i quantify how *principal* each direction is. Therefore, the eigenvectors and eigenvalues must be sorted in decreasing order, since we are maximizing in the optimization problem. To sum up, the PCA directions stored in U are the leading eigenvectors of the covariance matrix of data X, and are called *principal components*.

So far, we have discussed how can be the U matrix selected and why. However, there still remains the question about how many directions shall include U, that is, what is the appropriate dimensionality of the PCA subspace. Generally, one would truncate U to have $U \in \mathbb{R}^{d \times p}$. Truncating U means that we take a subset of the leading eigenvectors rather than the whole d eigenvectors with non-zero eigenvalues. Therefore, one should only be keeping the $p \leq d$ leading eigenvectors and not those directions that correspond to smaller eigenvalues to have $U \in \mathbb{R}^{d \times p}$. Finally, the constraint of orthogonality, $U^T U = I$, will always be fulfilled as long as the columns of the matrix U are orthonormal, mindless of the value p. If the orthogonal matrix U were not to be truncated, one would still have the orthogonality constraint fulfilled [16].

Performing PCA in practice is quite simple. With all things considered, PCA could be performed by a simple algorithm:

Algorithm 1 Principal Component Analysis

Input: Training data matrix, $\mathbf{X}$, test data matrix, $\mathbf{X_t}$, and training data size, $\mathbf{n}$.
Output: Dimension reduced training and test data, $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{X}}_\mathbf{t}$.

1: $X \leftarrow X - mean(X)$
2: $S \leftarrow 1/(n-1)X^T X$
3: **Compute basis:** $U \leftarrow$ eigenvectors of S corresponding to the top p eigenvalues
4: **Project training data:** $\widetilde{X} \leftarrow XU$
5: **Project test data:** $\widetilde{X}_t \leftarrow X_t U$

2.2 Partial Least Squares Regression

Partial Least Squares Regression (PLS) is a recent supervised dimension reduction technique that generalized and combines features from PCA and multiple regression. It is especially useful when ones needs to predict a set of dependent variables from

a large set of independent variables. It originated in the social sciences but became popular first in chemometrics, its so-called domain of origin, and in sensory evaluation. However, PLS is also becoming a tool of choice in the social science as a multivariate technique for non-experimental and experimental data alike. It was first presented as an algorithm akin to the power method but was rapidly interpreted in a statistical framework [4].

The goal of PLS is to predict $Y \in \mathbb{R}^{n \times l}$ from $X \in \mathbb{R}^{n \times d}$ and to describe their common structure. If Y was a vector and X full rank, this could be accomplished using *Ordinary Multiple Regression*. When the number of dimensions, d, is larger than the number of observations, n, the X matrix will probably be singular, and so the regression approach is no longer possible. Several approaches have been developed to deal with this problem. One of them is Principal Component Regression (PCR), which is closely related to PLS. PCR performs a PCA of the X matrix and then use the principal components of X as regressors of Y. The multicolinearity problem is eliminated thanks to the orthogonality of the principal components. However, the problem of choosing an optimum set of predictors remains. One possibility would be to keep only a few of the first components. But since they are chosen to explain X rather than Y, nothing guarantees that those principal components are also relevant for Y.

In contrast, PLS creates orthogonal components by using existing correlations between explanatory variables and corresponding outputs while also keeping most of the variance of explanatory variables. PLS has proven to be useful in situations when the number of observed variables d is significantly greater than the number of observations n and high multicollinearity among the variables exists[18].

In order to find components from X that also relevant for Y, PLS decomposes both the design matrix X and response matrix Y like in PCA and then perform regression between T and U [19]. PLS searches for a set of components that performs a simultaneous decomposition of X and Y with the constraint that these components explain as much as possible of the covariance between X and Y. These components are uncorrelated latent variables which are linear combinations of the original regressors in the first step. The basic point of the procedure is that the weights used to determine these linear combinations of the original regressors are proportional to the covariance among input and output variables. A least squares regression is then performed on the subset of extracted latent variables.

From now on, we assume centered input and output variables, i.e., the columns of X and Y are zero mean. The general model of multivariate PLS is [20]:

$$\begin{aligned} X &= TP^T + F \\ Y &= UQ^T + G \end{aligned} \tag{2.8}$$

where $T \in \mathbb{R}^{n \times p}$ and $U \in \mathbb{R}^{n \times p}$ matrices of the extracted p latent vectors, also known as score vectors. $P \in \mathbb{R}^{d \times p}$ and $Q \in \mathbb{R}^{l \times p}$ represent matrices of loadings, and $F \in \mathbb{R}^{n \times d}$ and $G \in \mathbb{R}^{n \times l}$ are the matrices of residuals.

The basic form of the PLS method is based on the nonlinear iterative partial least squares (NIPALS) algorithm [18]. NIPALS is a robust procedure for solving singular

value decomposition problems and is closely related to the power method. Using this method, PLS finds weight vectors w, c such that [20]:

$$\max_{|r|=|s|=1} [cov(Xr, Ys)]^2 = [cov(Xw, Yc)]^2 = [cov(t, u)]^2$$

where $cov(t, u) = t^T u/n$ denotes the sample covariance between the score vectors t and u. The idea behind partial least squares is that we want the decomposition of X and Y to be done by taking information from each other into account. One intuitive way to achieve that is to apply the NIPALS algorithm with the aim to sequentially extract the latent vectors t, u and weight vectors w, c from X and Y matrices in decreasing order of their corresponding singular values. The NIPALS algorithm starts with an initialization of the Y-score vector u and repeats a sequence steps until convergence. After the convergence, by regressing X on t and Y on u, the loading vectors p and q can be computed. The basic algorithm of PLS is as follows [21]:

1. Set u to the first column of Y
2. $w = X^T u/\|X^T u\|$
3. Scale w to be of length one
4. $t = Xw$
5. $c = Y^T t/\|Y^T t\|$
6. Scale c to be of length one
7. $u = Yc$
8. Repeat steps 2-7 until convergence
9. X-loadings: $p = X^T t/(t^T t)$
10. Y-loadings: $q = Y^T u/(u^T u)$
11. Regression coefficient: $b = u^T t/(t^T t)$
12. Deflat X and Y: $X \leftarrow X - tp^T$, $Y \leftarrow Y - btc^T$

The PLS regression is an iterative process; i.e. after extraction of one component the algorithm starts again using the residual matrices X and Y computed in step 12. Thus we can achieve the sequence of the models up to the point when the rank of X is reached. The vectors t, u, w, c, p and q are to be stored in the corresponding matrices every iteration, and the scalar b is stored as a diagonal element of B.

The PLS regression model can be written in matrix form as:

$$\widehat{Y} = XB_{PLS} \tag{2.9}$$

where $B_{PLS} \in \mathbb{R}^{d \times l}$ is the matrix of the regression coefficients. This is the same equation as the one used in other regression problems such as Multiple Linear Regression, Ridge

Regression or Principal Components Regression. In PLS, the dependent variable is predicted using the multivariate regression formula as:

$$\widehat{Y} = TBC^T \tag{2.10}$$

Using that $T = XW$ from step 4 and defining $B = (P^TW)^{-1}$ we get the following derivation [20]:

$$\begin{aligned}\widehat{Y} = TBC^T &= XWBC^T \\ &= XW(P^TW)^{-1}C^T\end{aligned}$$

Defining $B_{PLS} = W(P^TW)^{-1}C^T$ we arrive to the original PLS regression model expression in Equation 2.9. Due to the fact that $p_i^T w_j = 0$ for ij and in general $p_i^T wj \neq 0$ for $i \leq j$ the matrix P^TW is upper triangular and thus invertible.

If all the latent variables of X are used, this regression is equivalent to principal component regression. When only a subset of the latent variables is used, the prediction of Y is optimal for this number of predictors.

2.2.1 PLS Using Eigen-Decomposition

The steps described before might seem somewhat mysterious. Having a look at the computation of step 2:

$$\begin{aligned}w &= X^Tu/\|X^Tu\| \\ &= X^TYc/\|X^TYc\| \\ &= X^TYY^Tt/\|X^TYY^Tt\| \\ &= X^TYY^TXw/\|X^TYY^TXw\| \\ &= \frac{1}{\lambda}(Y^TX)^T(Y^TX)w\end{aligned} \tag{2.11}$$

We can see that w is an eigenvector of the covariance matrix of Y^TX [19]. Another way to see this fact is recalling the optimization problem:

$$\begin{aligned}\arg\max_{|w|=1} cov(Y^TXw, Y^TXw) &= \arg\max_{|w|=1} \frac{1}{k-1}(Y^TXw)^T(Y^TXw) \\ &= \arg\max_{|w|=1} \frac{1}{k-1}w^TX^TYY^TXw\end{aligned}$$

with the constraint that the loadings fulfill $w^Tw = 1$. Using the method of Lagrange multipliers [17], we have:

$$\mathcal{L} = w^TX^TYY^TXw - \lambda(w^Tw - 1)$$

where λ is the Lagrange multiplier. Taking the derivative of Lagrangian with respect to w and λ and setting them to zero:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} &= 2X^T Y Y^T X w - 2\lambda w = 0 \\ &\Rightarrow \; X^T Y Y^T X w = \lambda w \end{aligned} \tag{2.12}$$

which is again the eigenvalue problem shown in Equation 2.11. Now we take a look at the computation of step 5.

$$\begin{aligned} c &= Y^T t / \|YTt\| \\ &= Y^T X w / \|Y^T X w\| \\ &= Y^T X X^T u / \|Y^T X X^T u\| \\ &= Y^T X X^T Y c / \|Y^T X X^T Y c\| \\ &= \frac{1}{\lambda}(X^T Y)^T (X^T Y) c \end{aligned} \tag{2.13}$$

We see that c is an eigenvector of the covariance matrix of $X^T Y$. A similar derivation as the one did with w, could be carried out for the vectors c, t and u, and we would arrive to the following eigenvalue problems [21]:

$$Y^T X X^T Y c = \lambda c \tag{2.14}$$

$$X X^T Y Y^T t = \lambda t \tag{2.15}$$

$$Y Y^T X X^T u = \lambda u \tag{2.16}$$

The power method tells us that λ is the maximum eigenvalue of the eigenvalue problem. The vectors w, u, c and t are thus the eigenvectors of the appropriate matrices corresponding to the maximum eigenvalue. The PLS algorithm can thus be viewed as follows. For each set of residual matrices X and Y, one computes the maximum eigenvalue and associated eigenvectors of the matrices $X^T Y Y^T X$, $Y^T X X^T Y$, $X X^T Y Y^T$ and $Y Y^T X X^T$, and the eigenvectors are used to compute new residual matrices [21].

Concerning the projection into a lower dimensional feature space, the P matrix of X-loadings acts as the rotation matrix. The training data matrix, X, and the test data matrix, X_t, are projected taking the leading eigenvectors of the P matrix:

$$\widetilde{X} = XP \tag{2.17}$$

$$\widetilde{X}_t = X_t P \tag{2.18}$$

2.3 Supervised Principal Component Analysis

Supervised Principal Component Analysis (SPCA) [3] is a supervised dimensionality reduction which arises from the necessity of estimating a sequence of principal components that not only harvest the variance of the input response, but also the dependence

of those on the target variable. The aim of SPCA is to find a subspace that maximizes the dependency between the inputs and outputs. As we will show, its derivation leads to a closed-form solution, similar to the PCA one. One advantage of SPCA is that it takes into account the quantitative value of the target response, a property which makes it applicable to both classification and regression problems. Furthermore, it has a kernel version which makes it applicable for estimating nonlinear transformation data. This kernel version though, is not the subject for this section and will be presented in the following sections.

2.3.1 Hilbert-Schmidt Independence Criterion

The dependence of two random variables $\mathcal{X}$ and $\mathcal{Y}$ can be easily measured using the value of the correlation between them. According to [22], two random variables are independent if and only if any bounded continuous functions of them are uncorrelated. Therefore, by measuring the correlation of $\phi(x)$ and $\phi(y)$, the mapping of $\mathcal{X}$ and $\mathcal{Y}$ to two different Reproducing Kernel Hilbert Spaces (RKHSs), we can estimate the dependence of the original variables, $\mathcal{X}$ and $\mathcal{Y}$.

The Hilbert-Schmidt Independence Criterion (HSIC) is an independence criterion in RKHSs proposed in [23]. The HSIC is a measure of the dependence between two random variables, $\mathcal{X}$ and $\mathcal{Y}$. The measure is obtained by computing the Hilbert-Schmidt norm of the cross-covariance operator associated with their RKHSs.

Let us now derive the formulation of the HSIC, which will be useful in the derivation of the SPCA derivation. Define $\mathcal{F}$ as a separable RKHS containing all continuous bounded real-valued functions of x from $\mathcal{X}$ to $\mathbb{R}$. Similarly, let $\mathcal{G}$ be a separable RKHS containing all continuous bounded real-valued functions of y from $\mathcal{Y}$ to $\mathbb{R}$. Then the cross-covariance between elements of $\mathcal{F}$ and $\mathcal{G}$ is [3]:

$$Cov(f(x), g(y)) = E_{x,y}[f(x)g(y)] - E_x[f(x)]E_y[g(y)] \tag{2.19}$$

It can be shown that there exists a unique linear operator $C_{x,y} : \mathcal{G} \to \mathcal{F}$ mapping elements of $\mathcal{G}$ to the elements of $\mathcal{F}$ which can be defined as [3]:

$$C_{x,y} := E_{x,y}[(\Phi(x) - \mu_x) \otimes (\Psi(y) - \mu_y)] \tag{2.20}$$

where $\mu_x = E[\Phi(x)]$, $\mu_y = E[\Psi(y)]$, $\otimes$ is the tensor product and Φ and Ψ are the associated feature maps of $\mathcal{F}$ and $\mathcal{G}$, respectively. A useful norm for this operator is the Hilbert-Schmidt (HS) norm, defined as:

$$\|C\|^2_{HS} := \sum_{i,j} \langle Cv_i, u_j \rangle^2_{\mathcal{F}} \tag{2.21}$$

where u_j and v_i are orthogonal bases of $\mathcal{F}$ and $\mathcal{G}$, respectively. Finally, we can define the HSIC as the square of the Hilbert-Schmidt norm of the cross-covariance operator.

Empirical HSIC

Using the explained intuition, an empirical estimation of the HSIC is introduced [3]:

$$HSIC := \frac{1}{(n-1)^2}\mathbf{tr}(K_x H K_y H) \tag{2.22}$$

where H is the centering matrix defined as $H := I - (1/n)\mathbf{1}\mathbf{1}^T$, and $\mathbf{1}$ is a vector of full ones. K_x and K_y are the kernels over x and y, respectively. In other words, $K_x = \phi(x)^T\phi(x)$ and $K_y = \phi(y)^T\phi(y)$. The term $1/(n-1)^2$ is used for normalization.

Based on this empirical result, we can deduce that the dependence between two kernels will be maximized if we increase the value of the empirical estimate, i.e., $\mathbf{tr}(K_x H K_y H)$. Therefore, the greater the HSIC, the greater the dependence they have. The HK_yH double centers the K_y in HSIC. Also, note that if one of the kernel matrices K_x or K_y is already centered, say K_y, then $HK_yH = K_y$ amd thus we may simply use the objective function $\mathbf{tr}(K_x K_y)$ which no longer includes the centering matrix H. Similarly, if $HK_xH = H$ we may rewrite $\mathbf{tr}(K_x H K_y H) = \mathbf{tr}(HK_xHK_y)$ and arrive at identical results [3].

2.3.2 SPCA Derivation

Given a $n \times d$ matrix X of features and a $n \times l$ matrix Y of outcome measurements, SPCA addresses the problem of finding the subspace XU such that the dependency between the projected data XU and the outcome Y is maximized [3]. In order to measure this dependency, SPCA uses the Hilbert-Schmidt Independence Criterion.

For the projected data points, it uses a linear kernel defined as [16]:

$$K_x = (XU)(XU)^T = XUU^TX^T. \tag{2.23}$$

For the Y matrix, it uses an arbitrary kernel K_y. The HSIC in SPCA case becomes [16]:

$$HSIC = \frac{1}{(n-1)^2}\mathbf{tr}(XUU^TX^THK_yH), \tag{2.24}$$

where $U \in \mathbb{R}^{d\times p}$ is the orthogonal transformation matrix which we are looking for and which maps the data points to a space where the features are uncorrelated. The desired dimensionality of the subspace is p and usually $p \ll d$.

In order to maximize the dependence of XU and Y, we should maximize the HSIC. Hence, the optimization problem becomes:

$$\begin{aligned} \max_{U} \ & \mathbf{tr}(XUU^TX^THK_yH) \\ & \text{subject to } \ U^TU = I, \end{aligned} \tag{2.25}$$

where the constraint ensures that the U is an orthogonal matrix and therefore, the SPCA directions are orthonormal. To solve the optimization problem we can use the

method of Lagrangian multipliers [17]:

$$\begin{aligned}\mathcal{L} &= \mathbf{tr}(XUU^TX^THK_yH) - \mathbf{tr}(\Lambda^T(U^TU - I)) \\ &= \mathbf{tr}(UU^TX^THK_yHX) - \mathbf{tr}(\Lambda^T(U^TU - I)),\end{aligned}$$

where we have used the cyclic property of trace to rearrange the first term. The $\Lambda \in \mathbb{R}^{p\times p}$ matrix is a diagonal matrix containing the Lagrange multipliers. Setting the derivative with respect to U to zero gives:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial U} &= 2X^THK_yHXU - 2U\Lambda = 0 \\ \Rightarrow X^THK_yHXU &= U\Lambda, \end{aligned} \tag{2.26}$$

which is the eigendecomposition of X^THK_yHX. The columns of U and the diagonal of Λ are the eigenvectors and eigenvalues of X^THK_yHX, respectively. The eigenvectors and eigenvalues are sorted in decreasing order because of the maximization of the optimization problem. In other words, the columns of the projection matrix U are the p leading components of SPCA.

Taking all the above into consideration, the SPCA procedure is summarized in Algorithm 2 [3].

Algorithm 2 Supervised PCA

Input: training data matrix, $\mathbf{X}$, testing data matrix, $\mathbf{X_t}$, kernel matrix over target variable, $\mathbf{K_y}$, and training data size, $\mathbf{n}$.

Output: Dimension reduced training and testing data, $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{X}}_\mathbf{t}$.

1: $H \leftarrow I - n^{-1}\mathbf{1}\mathbf{1}^T$
2: $Q \leftarrow X^THLHX$
3: **Compute basis:** $U \leftarrow$ eigenvectors of Q corresponding to the top p eigenvalues
4: **Project training data:** $\widetilde{X} \leftarrow XU$
5: **Project test data:** $\widetilde{X}_t \leftarrow X_tU$

2.3.3 SPCA Connection to PCA

Take the case of where the response variable is unknown, that is, an unsupervised problem. This means that the kernel K_y would be equivalent to the identity matrix, a kernel which only captures the similarity between a point and itself. Hence, the maximization of the dependence of these two kernels would be like computing the maximal diversity across all the original data points. This is equivalent to PCA.

Recall that SPCA is the eigendecomposition of $X^T H K_y H X U = U\Lambda$. In the unsupervised case, this matrix becomes [3]:

$$\begin{aligned} X^T H K_y H X &= X^T H I H X \\ &= (HX)^T (HX) \\ &= [(I - n^{-1}\mathbf{1}\mathbf{1}^T)X]^T [(I - n^{-1}\mathbf{1}\mathbf{1}^T)X] \\ &= (X - \mu_x)^T (X - \mu_x) \\ &= S \end{aligned}$$

where mu_x refers to the mean of data X, and S is the covariance matrix whose eigenvectors are the PCA directions.

This result shows that the eigendecomposition of the covariance matrix of the data, S, is the same as the eigendecomposition of $X^T H I H X$ and consequently maximizing the $\mathbf{tr}(XUU^T X^T H I H)$. Therefore, PCA is a particular case of a more general framework proposed by SPCA.

Chapter 3

Least Squares Regression Principal Component Analysis

In this chapter, we propose a novel supervised dimension reduction method, Least Squares Regression Principal Component Analysis (LSR-PCA) [6], which is a generalization of the classical linear regression, applicable to both classification and regression dimension reduction tasks. This method identifies the components of the input data that contribute linearly to constructing a kernel-transformed target data.

3.1 LSR-PCA Formulation

Given a *zero-centered* data matrix $X \in \mathbb{R}^{n\times d}$, where n is the number of data points and d is the number of features, and a corresponding matrix of target outcomes, $Y \in \mathbb{R}^{n\times l}$, where l is the dimension of the target outcomes, LSR-PCA consists in solving a generalized eigenvalue problem:

$$X^T K_y X w = \lambda X^T X w.$$

The matrix $K_y \in \mathbb{R}^{n\times n}$ is a kernel matrix for the target outcomes defined as $K_y = k(y, y^{'})$, where y and $y^{'}$ are the $1 \times l$ row vectors of the target matrix, Y. LSR-PCA can be shown to be equivalent to linear least squares regression if the kernel matrix $K_y = Y^T Y$. Hence, the phrase "least squares regression" was chosen for its name. In general, the kernel function $k(y, y^{'})$ is problem dependant.

For classification problems, a choice for $k(y, y^{'})$ is the Kronecker-Delta kernel, $\delta_{y,y^{'}}$:

$$\delta_{y,y^{'}} := \begin{cases} 1 & \text{if } y = y^{'} \\ 0 & \text{if } y \neq y^{'} \end{cases} \tag{3.1}$$

For regression problems, a common choice for $k(y, y^{'})$ is the squared exponential kernel:

$$k(y, y^{'}) = exp\left(-\frac{\|y_i - y_j\|^2}{2\sigma^2}\right). \tag{3.2}$$

3.2 LSR-PCA Derivation

The generalized eigenvalue problem for LSR-PCA can be derived using the following minimization problem and then apply the kernel trick for the target outcomes y:

$$\min_{w \in \mathbb{R}^d} \left\| y^T - (y^T \cdot Xw)Xw \right\|^2 \tag{3.3}$$

subject to the constraint that:

$$\|Xw\|^2 = 1 \tag{3.4}$$

The minimization problem finds a linear combination of the input data, Xw, that will maximize the projection of the transformed input data onto the target outcome matrix, y, where w is a $d \times 1$ vector that contains the coefficients of the linear combination of the input features.

The objective function in Equation 3.3 can be rewritten as:

$$\max_{w \in \mathbb{R}^d} (y^T \cdot Xw)^2 \tag{3.5}$$

$$\text{subject to } \|Xw\|^2 = w^T X^T Xw = 1. \tag{3.6}$$

Using the method of Lagrange multipliers, we have:

$$\begin{aligned} \mathcal{L} &= (y^T \cdot Xw)^2 - \lambda(w^T X^T Xw - 1), \\ &= (w^T X^T y^T yXw) - \lambda(w^T X^T Xw - 1) \end{aligned}$$

where λ is the Lagrange multiplier. Taking the derivative of Lagrangian with respect to w and setting them to zero:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w} &= 2Xy^T yXw - 2\lambda X^T Xw = 0, \\ &\Rightarrow X^T y^T yXw = \lambda X^T Xw \end{aligned} \tag{3.7}$$

The generalized eigenvalue problem in equation 3.7 only requires the target outcomes in the form of its inner products $y^T y$. We can apply the kernel trick to the target outcome by substituting for $y^T y$ any valid kernel of the form $K_y = k(y, y')$. After the application of the kernel trick, we arrived at the the final generalized eigenvalue problem for RPCA in equation 3.8.

$$\Rightarrow X^T K_y Xw = \lambda X^T Xw. \tag{3.8}$$

The eigenvectors $w \in \mathbb{R}^d$ that correspond to the largest eigenvalues are the leading principal components for LSR-PCA. The leading p eigenvectors of LSR-PCA are stored in the W matrix. The solution of the generalized eigenvalue problem requires that the covariance matrix $X^T X$ has a full rank. In the event that $X^T X$ has rank $r < d$, a singular decomposition of the covariance matrix is necessary to extract the subspace of

p singular vectors that correspond to nonzero singular values. LSR-PCA can then be applied to the reduced data matrix $\widetilde{X} \in \mathbb{R}^{n \times p}$ so that $\widetilde{X}^T \widetilde{X}$ has full rank.

Taking the p leading eigenvalues, the projection of the training data X according to Equation 3.8 is:

$$\widetilde{X} = XW \tag{3.9}$$

and for the out-of-sample/test data:

$$\widetilde{X}_t = X_t W \tag{3.10}$$

The reconstruction of the training data X and out-of-sample data X_t after projection onto the LSR-PCA subspace is:

$$X = XWW^T = \tilde{X}W^T \tag{3.11}$$

$$X_t = X_t WW^T = \tilde{X}_t W^T \tag{3.12}$$

The Least Squares Regression PCA procedure is summarized in Algorithm 3.

Algorithm 3 Least Squares Regression PCA

Input: training data matrix, $\mathbf{X}$, testing data matrix, $\mathbf{X_t}$, kernel matrix over target variable, $\mathbf{K_y}$.

Output: Dimension reduced training and testing data, $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{X}}_\mathbf{t}$.

1: $D \leftarrow X^T K_y X$
2: $B \leftarrow X^T X$
3: **Compute basis:** $W \leftarrow$ generalized eigenvectors of (D, B) corresponding to the top p eigenvalues
4: **Project training data:** $\widetilde{X} \leftarrow XW$
5: **Project test data:** $\widetilde{X}_t \leftarrow X_t W$

3.2.1 LSR-PCA Connection to PCA

The formulation of LSR-PCA is closely connected to the projection interpretation of PCA [2]. PCA can be derived from picking a $d \times 1$ unit vector w that minimizes the distance between the n training data points $\{x_i\}_{i=1}^n$ and their projection onto the direction w:

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \left\{ \sum_{i=1}^{n} \|x_i - (x_i \cdot w)w\|^2 \right\}, \tag{3.13}$$

subject to the constraint that:

$$\|w\|^2 = 1.$$

The minimization problem for PCA can be transformed into the following eigenvalue problem:

$$X^T X w = \lambda w, \tag{3.14}$$

where X is the $n \times d$ data matrix. The ith row of X is the training data point x_i.

The LSR-PCA formulation is the application of the same projection interpretation except on the target outcome vector y.

Chapter 4

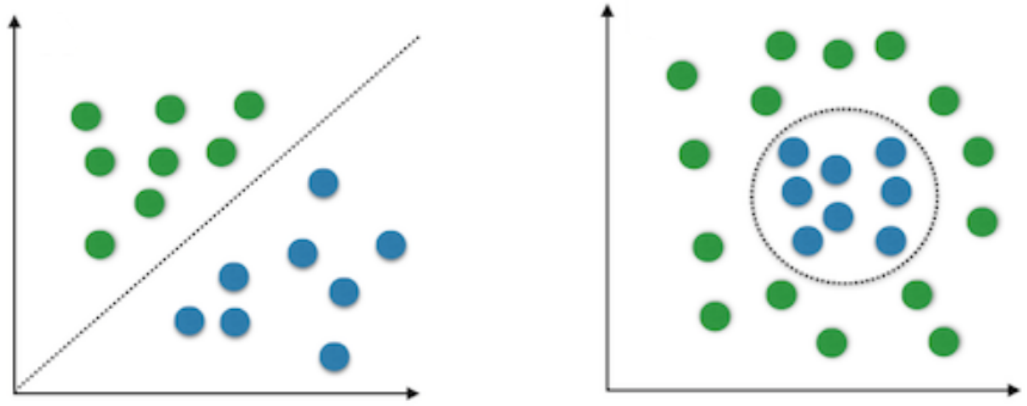
Non-linear transformation methods

In this chapter, we will derive the non-linear versions of the methods described before, PCA, SPCA and LSR-PCA, which are called 'Kernel Principal Component Analysis', 'Kernel Supervised Principal Component Analysis' and 'Kernel Least Squares Regression Principal Component Analysis', respectively. We will start with a motivation and informal explanation and then we will discuss the mathematical ideas behind each method.

All the methods described above have one thing in common: linearity. They all perform a linear projection technique of the input data matrix X. This is a good approach provided that the data might be linearly separable. However, in the case that the data points exist in a non-linear sub-manifold, the linear methods considered above might not be entirely effective. Let's see an illustrative example to understand this problem.

Take the case of a 2-dimensional data where the data points can be classified using two classes. The data points in Figure 4.1a allow a linear separation, whereas the data points in Figure 4.1b, do not. In order to handle this problem, we can either change the linear dimension reduction method to become a non-linear method or, instead, we can leave them to be linear but transform the data with the hope that it will fall on a linear subspace. The methods presented below do the latter, so they change the data. The cost to do that is that we need map the data to a feature space with higher dimensionality hoping that in this new subspace, it falls on a linear manifold. This implies that we will have to increase the dimensionality of data.

Let us know consider a 2-dimensional binary classification problem with features x_1 and x_2 shown in Figure 4.2a. This is a very similar problem to the one presented in Figure 4.1b and again, it can be easily seen that the problem is not solvable by any linear classifier. Introducing a new feature $x_1^2 + x_2^2$ and extending our space of features we get the data distribution shown in Figure 4.2b. As we can see, the new distribution now does allow us to use a linear classifier in this new feature space. The expression used to extend the data is what is known as *kernel function*, and it is the key idea behind non-linear transformation methods.



(a) (b)

Figure 4.1: 4.1a[24]:2-dimensional data classified in two classes, blue and green, linearly separable. 4.1b[24]: 2-dimensional data classified in two classes, blue and green, not linearly separable.

4.1 Kernels and Hilbert Space

In this section we will provide a mathematical description of the intuition described above. Consider the function *phi* which maps the data x to the Hilbert space (feature space) such that:

$$\Phi : x \to \mathcal{H},$$
$$x \mapsto \phi(x).$$

where $\phi(x)$ is a set of t basis functions that span the feature space, $\mathcal{H}$.

If $\mathcal{X}$ denotes the set of points $x \in \mathcal{X}$, the kernel of two vectors x_1 and x_2 is $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and is defined as [16]:

$$k(x_1, x_2) := \phi(x_1)^T \phi(x_2), \tag{4.1}$$

which is a measure of "similarity" between the two vectors. The inner product captures the similarity. Next we defined $\Phi(X)$ to be a $t \times n$ data matrix where the original data matrix X have been transformed to the new feature space.

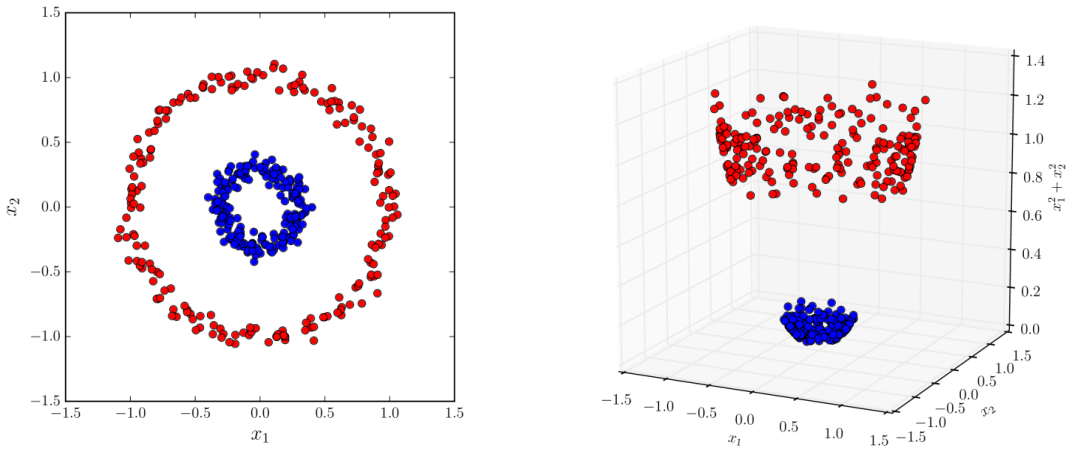
$$\Phi(X) = \{\phi(x_1), \phi(x_2), \cdots, \phi(x_n)\}. \tag{4.2}$$

One can build the kernel of two matrices $X_1 \in \mathbb{R}^{n_1 \times d}$ and $X_2 \in \mathbb{R}^{n_2 \times d}$ and have a *kernel matrix* $K \in \mathbb{R}^{n_1 \times n_2}$:

$$K(X_1, X_2) := \Phi(X_1)^T \Phi(X_2). \tag{4.3}$$

We can also build the kernel matrix of the original data matrix X over itself $K_x \in \mathbb{R}^{n \times n}$:

$$K_x := K(X, X) = \Phi(X)^T \Phi(X) \tag{4.4}$$



(a) (b)

Figure 4.2: 2-dimensional data classified in two classes, red and blue. In 4.2a[25], the classes are not linearly separable. In 4.2b[25], the data is extended to the feature space $x_1^2 + x_2^2$ and is linearly separable.

It is important to note that in kernel methods, the mapped data $\Phi(X)$ is usually not available. Only the kernel matrix $K(X, X)$, which is the inner product of the mapped data with itself, is available [16]. Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional feature space without the need of computing the explicit mapped data $\Phi(X)$. This approach of simply computing the inner product between the images of all pairs of data in the feature space is known as the *kernel trick*.

There exist different types of kernel functions. Some of the most well-known kernels are:

$$\text{Linear:} \quad k(x_1, x_2) = x_1^T x_2 + c_1, \tag{4.5}$$

$$\text{Polynomial:} \quad k(x_1, x_2) = (c_1 x_1^T x_2 + c_2)^{c_3}, \tag{4.6}$$

$$\text{Squared Exponential:} \quad k(x_1, x_2) = exp(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}), \tag{4.7}$$

where c_1, c_2, c_3 and σ are parameters that need to adjusted depending on the data. Note that the Squared Exponential kernel can also be expressed as $exp(-\gamma\|x_1 - x_2\|^2)$, where $\gamma = 1/(2\sigma^2)$ and $\gamma > 0$.

In the Squared Exponential kernel, the dimensionality of the feature space is infinite. This can be easily shown by computing the Maclaurin series expansion of this function: [16]

$$exp(-\gamma r) \sim 1 - \gamma r + \frac{\gamma^2}{2!} r^2 - \frac{\gamma^3}{3!} r^3 + \dots$$

where $r := \|x_1 - x_2\|^2$.

4.2 Kernel Principal Component Analysis

Kernel Principal Component Analysis (KPCA) is the non-linear version of the linear method Principal Component Analysis. This method uses the kernel functions explained above to efficiently compute principal components in high-dimensional feature spaces, related to input space by some non-linear map [26].

Let us denote a covariance matrix in a new feature space as:

$$\Sigma = \frac{1}{N}\sum_{n=1}^{N}\phi(x_n)\phi(x_n)^T = \frac{1}{N}\Phi^T\Phi \tag{4.8}$$

The eigendecomposition of Σ is given by:

$$\Sigma v_i = \lambda_i v_i \tag{4.9}$$

By the definition of Σ, we get:

$$[\frac{1}{N}\sum_{n=1}^{N}\phi(x_n)\phi(x_n)^T]v_i = \frac{1}{N}\sum_{n=1}^{N}(\phi(x_n)\cdot v_i)\phi(x_n)^T = \lambda_i v_i.$$

We see that v_i is a linear combination of $\phi(x_n)$ and thus can be written as:

$$v_i = \sum_{n=1}^{N} a_{i_n}\phi(x_n). \tag{4.10}$$

where $a_{i_n} = \frac{1}{\lambda_i N}(\phi(x_n)\cdot v_i)$. Substituting v_i in Equation 4.2 and writing it in matrix notation, we get:

$$K_x a_i = \lambda_i N a_i \tag{4.11}$$

where K_x is the kernel matrix of the training data and a_i are the N eigenvectors of K_x, which can be obtained by solving the eigenvalue problem for K_x. The a_i eigenvectors should be orthonormal, therefore, we get the following constrain:

$$N\lambda_i a_i^T a_i = 1.$$

Any new data point $x_t \in \mathbb{R}^d$ can now be mapped onto the high-dimensional feature space by performing the projection onto the i-th eigenvector:

$$\phi(x_t)\cdot v_i = (\phi(x_t)\cdot\sum_{n=1}^{N} a_{i_n}\phi(x_n)) = \sum_{n=1}^{N} a_{i_n}(\phi(x_t)^T\phi(x_n)) = \sum_{n=1}^{N} a_{i_n}k(x_t, x_n)$$

The above discussion is based on the assumption that the data points have zero mean, and thus, the covariance of two data points is the same as their correlation.

However, this is not always the case, and one needs to ensure the zero mean of the data in the feature space too [16]:

$$\phi_c(x_i)) := \phi(x_i) - \frac{1}{N}\sum_{n=1}^{N}\phi(x_n) \tag{4.12}$$

Since the mapping is not explicit and $\phi(x_n)$ is never available, we must center the kernel matrix too:

$$\begin{aligned}
K_c :&= \phi_c(x_i)^T\phi_c(x_j) \\
&= (\phi(x_i) - \frac{1}{N}\sum_{k_1=1}^{N}\phi(x_{k_1}))^T(\phi(x_j) - \frac{1}{N}\sum_{k_2=1}^{N}\phi(x_{k_2})) \\
&= \phi(x_i)^T\phi(x_j) - \frac{1}{N}\sum_{k_1=1}^{N}\phi(x_{k_1})^T\phi(x_j) \\
&- \frac{1}{N}\sum_{k_2=1}^{N}\phi(x_i)^T\phi(x_{k_2}) \\
&+ \frac{1}{N^2}\sum_{k_1=1}^{N}\sum_{k_2=1}^{N}\phi(x_{k_1})^T\phi(x_{k_2}).
\end{aligned}$$

In matrix form, the double centered kernel matrix of the training data points, $K_c \in \mathbb{R}^{N\times N}$ is:

$$K_c = K_x - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^T K_x - \frac{1}{N}K_x\mathbf{1}_N\mathbf{1}_N^T + \frac{1}{N^2}\mathbf{1}_N\mathbf{1}_N^T K_x\mathbf{1}_N\mathbf{1}_N^T \tag{4.13}$$

where $\mathbf{1}_N := [1,\dots,1]^T$ is a vector of N ones.

Regarding any new data point, such as a test data sample x_t, its corresponding kernel matrix for the training data and the test data is:

$$K_t = \Phi(X)^T\Phi(X_t) \tag{4.14}$$

which also needs to be centered using the mean of the training data. A similar derivation can be carried out to obtain the following double-centered kernel matrix over training and test data, $K_{t_c} \in \mathbb{R}^{N\times n_t}$ [16]:

$$K_{t_c} = K_t - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^T K_t - \frac{1}{N}K_x\mathbf{1}_N\mathbf{1}_{n_t}^T + \frac{1}{N^2}\mathbf{1}_N\mathbf{1}_N^T K_x\mathbf{1}_N\mathbf{1}_{n_t}^T \tag{4.15}$$

where n_t is the number of test samples, and $\mathbf{1}_{n_t} := [1,\dots,1]^T$ is a vector of $n-t$ ones.

Taking all of the above into consideration, we can summarize the steps of KPCA in Algorithm 4.

Algorithm 4 Kernel PCA

Input: Double-centered kernel matrix of training data, $\mathbf{K_c}$, double-centered kernel of test data, $\mathbf{K_{t_c}}$.

Output: Dimension reduced training and testing data, $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{X}}_\mathbf{t}$.

1: **Compute basis:** $V \leftarrow$ eigenvectors of K_c corresponding to the top p eigenvalues
2: **Project training data:** $\widetilde{X} \leftarrow V^T K_c$
3: **Project test data:** $\widetilde{X}_t \leftarrow V^T K_{t_c}$

The KPCA steps described above require to define some kernel function for K_x and K_t, e.g., the squared exponential kernel. Besides, the corresponding parameters of the kernel function will need to be adjusted depending on the data. It should be noted that the number of non-linear principal components in the general case is infinite. However, since we are computing the eigenvectors of a $N \times N$ matrix, at maximum we can calculate N nonlinear principal components.

4.3 Kernel Supervised Principal Component Analysis

In this section, we derive the kernel version of Supervised Principal Component Analysis (SPCA), Kernel Supervised Principal Component Analysis (KSPCA) [3], which allows SPCA to be extended to non-linear mappings of the data.

One way to do it is by direct formulation. According to representation theory [27], any solution $u \in \mathcal{H}$ must lie in the span of "all" the training vectors mapped to $\mathcal{H}$, that is, $\Phi(X) \in \mathbb{R}^{t \times n}$ [16]. Therefore, the transformation matrix U can be expressed as a linear combination of the mapped data points:

$$U = \Phi(X)\Theta, \tag{4.16}$$

where $\Theta \in \mathbb{R}^{n \times p}$ contains the θ_i unknown vector coefficients, and $U \in \mathbb{R}^{t \times p}$ contains the KSPCA directions in the Hilbert space, u_i.

In the feature space, the HSIC in Equation 2.24 becomes:

$$HSIC = \frac{1}{(n-1)^2}\mathbf{tr}(\Phi(X)^T U U^T \Phi(X) H K_y H) \tag{4.17}$$

The trace expression can be simplified as:

$$\begin{aligned}\mathbf{tr}(\Phi(X)^T U U^T \Phi(X) H K_y H) &= \mathbf{tr}(U U^T \Phi(X) H K_y H \Phi(X)^T) \\ &= \mathbf{tr}(U^T \Phi(X) H K_y H \Phi(X)^T U)\end{aligned} \tag{4.18}$$

By plugging Equation 4.16 in Equation 4.18 we get:

$$\mathbf{tr}(\Theta^T \Phi(X)^T \Phi(X) H K_y H \Phi(X)^T \Phi(X) \Theta) = \mathbf{tr}(\Theta^T K_x H K_y H K_x \Theta) \tag{4.19}$$

where K_x is the kernel function described in the previous sections.

Moreover, the constraint of orthogonality of the transformation matrix in the feature space becomes:

$$\begin{aligned} U^T U &= (\Phi(X)\Theta)^T(\Phi(X)\Theta) \\ &= \Theta^T\Phi(X)^T\Phi(X)\Theta \\ &= \Theta^T K_x \Theta \end{aligned} \tag{4.20}$$

The optimization problem in Equation 2.25 has been re-expressed in terms of inner products between data points. Therefore, the new optimization problem has the following form:

$$\begin{aligned} \max_{\Theta} \ & \mathbf{tr}(\Theta^T K_x H K_y H K_x \Theta) \\ \text{subject to } & \Theta^T K_x \Theta = I \end{aligned} \tag{4.21}$$

where the objective variable is the unknown Θ.

Using the method of Lagrange multipliers [17], we have:

$$\begin{aligned} \mathcal{L} &= \mathbf{tr}(\Theta^T K_x H K_y H K_x \Theta) - \mathbf{tr}(\Lambda^T(\Theta^T K_x \Theta - I)) \\ &= \mathbf{tr}(\Theta\Theta^T K_x H K_y H K_x) - \mathbf{tr}(\Lambda^T(\Theta^T K_x \Theta - I)) \end{aligned}$$

where Λ is a diagonal matrix containing the p Lagrange multipliers. Setting the first derivative of Lagrangian with respect to Θ to zero gives:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \Theta} &= 2K_x H K_y H K_x \Theta - 2K_x \Theta \Lambda = 0 \\ &\Rightarrow K_x H K_y H K_x \Theta = K_x \Theta \Lambda \end{aligned} \tag{4.22}$$

which is the generalized eigenvalue problem for Kernel SPCA.

Taking all of the above into consideration, we can summarize the steps of KSPCA using Algorithm 5 [3].

Algorithm 5 Kernel Supervised PCA

Input: Kernel matrix of training data, $\mathbf{K_x}$, kernel matrix of testing data, $\mathbf{K_t}$, kernel matrix over target variable, $\mathbf{K_y}$, and training data size, $\mathbf{n}$.

Output: Dimension reduced training and testing data, $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{X}}_\mathbf{t}$.

1: $H \leftarrow I - n^{-1}\mathbf{1}\mathbf{1}^T$
2: $Q \leftarrow K_x H K_y H K_x$
3: **Compute basis:** $\Theta \leftarrow$ generalized eigenvectors of (Q, K_x) corresponding to the top p eigenvalues
4: **Project training data:** $\widetilde{X} \leftarrow \Theta^T K_x$
5: **Project test data:** $\widetilde{X}_t \leftarrow \Theta^T K_t$

4.4 Kernel Least Squares Regression Principal Component Analysis

As we have seen, when the data matrix exists on a non-linear sub-manifold, the linear subspace learning becomes ineffective. This motivates the need for a kernel formulation for LSR-PCA [6].

Consider a t dimensional feature space $\mathcal{H}$ such that:

$$\begin{aligned}\Phi : x &\to \mathcal{H},\\ x &\mapsto \phi(x).\end{aligned}$$

where $\phi(x)$ is a set of t basis functions that span the feature space, $\mathcal{H}$.

Next we defined $\Phi(X)$ to be a $t \times n$ data matrix where the original data matrix X have been transformed to the new feature space.

$$\Phi(X) = \{\phi(x_1), \phi(x_2), \cdots, \phi(x_n)\}.$$

The idea behind kernel LSR-PCA is to express the coefficients w as a linear combination of the transformed data matrix:

$$w = \Phi(X)\theta,$$

where $\theta \in \mathbb{R}^n$ are the coefficients of the linear combination, and w is a $t \times 1$ vector.

Using the kernel trick on the inputs x, we can extend the feature space $\mathcal{H}$ to an infinite dimensional feature space,

$$k(x, x^{'}) = \phi(x)^T \phi(x^{'}),$$

and the resulting $n \times n$ kernel matrix:

$$K_x = \Phi(X)^T \Phi(X).$$

Given a $n \times 1$ target outcome vector y, we can rewrite the objective function in Equation 3.3 as:

$$\begin{aligned}&\min_{\omega \in \mathbb{R}^t} \left\| y - y \cdot \Phi(X)^T w)\Phi(X)^T w \right\|^2,\\ \implies &\min_{\theta \in \mathbb{R}^n} \left\| y - (y \cdot \Phi(X)^T \Phi(X)\theta)\Phi(X)^T \Phi(X)\theta \right\|^2,\\ \implies &\min_{\theta \in \mathbb{R}^n} \left\| y - (y \cdot K_x\theta) K_x\theta \right\|^2,\end{aligned}$$

with the constraint:

$$\left\| K_x\theta \right\|^2 = \theta^T K_x^T K_x \theta = 1. \tag{4.23}$$

The above optimization is equivalent to the following optimization problem:

$$\max_{\theta \in \mathbb{R}^n} (y \cdot K_x\theta)^2, \tag{4.24}$$

$$\text{subject to } \|K_x\theta\|^2 = 1. \tag{4.25}$$

Using the method of Lagrangian multipliers, we get:

$$\begin{aligned}\mathcal{L} &= (y \cdot K_x\theta)^2 - \lambda(\theta^T K_x^T K_x\theta - 1) \\ &= (\theta^T K_x^T y^T y K_x\theta) - \lambda(\theta^T K_x^T K_x\theta - 1)\end{aligned}$$

Setting the first derivative of Lagrangian with respect to θ to zero gives:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\theta} &= 2K_x^T yy^T K_x\theta - 2\lambda K_x^T K_x\theta = 0, \\ \Rightarrow K_x^T y^T y K_x\theta &= \lambda K_x^T K_x\theta \end{aligned} \tag{4.26}$$

which is the generalized eigenvalue problem for kernel LSR-PCA. We can apply the kernel trick again to the target outcome by substituting for $y^T y$ any valid kernel of the form $K_y = k(y, y')$. Once again, we arrive at the final generalized eigenvalue problem for Kernel LSR-PCA in Equation 4.27:

$$\Rightarrow K_x^T K_y K_x\theta = \lambda K_x^T K_x\theta \tag{4.27}$$

where $\theta \in \mathbb{R}^n$ are the eigenvectors of $(K_x^T K_y K_x, K_x^T K_x)$, and λ are the eigenvalues. Our Kernel LSR-PCA procedure is summarized in Algorithm 6.

Algorithm 6 Kernel LSR-PCA

Input: Kernel matrix of training data, $\mathbf{K_x}$, kernel matrix of test data, $\mathbf{K_t}$, kernel matrix over target variable, $\mathbf{K_y}$.

Output: Dimension reduced training and testing data, $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{X_t}}$.

1: $D \leftarrow K_x^T K_y K_x$
2: $B \leftarrow K_x^T K_x$
3: **Compute basis:** $\Theta \leftarrow$ generalized eigenvectors of (D, B) corresponding to the top p eigenvalues
4: **Project training data:** $\widetilde{X} \leftarrow \Theta^T K_x$
5: **Project test data:** $\widetilde{X_t} \leftarrow \Theta^T K_t$

Chapter 5

Code implementation

The aim of this chapter is to give an overview of the code used in this thesis. We will comment the purpose, the structure, and its main functions.

In order to provide a trustful comparison between all the methods, we developed a code which returned the good or bad performance of a method in terms of some error rate. The challenge here, relied on the way of checking if a method performs well or not, since as we have already explained, it has a strong dependency on the data we are treating. Therefore, we have tested several data sets from different research fields. The data sets with more conclusive results will be shown in the next chapter.

The code of this thesis is ran in Python 3.7. The files containing all the work are available on the following GitHub repository: `https://github.com/hpascualh/dim-reduction-py`. This repository consists of different Python modules with the necessary functions to test the performance of the methods. In this repository there is not a single *main* file but a collection of *main-like* files, one for each data set.

5.1 Dimension reduction routine

The routine that computes the results that allow us to properly compare all the methods is the `DimReductionClass` function. This function is inside the MainRoutineClass.py Python module and it is defined as a class. This means that this function creates an object which can have, in Python terms, attributes and methods. In a Python class, the attributes and the methods refer to the outputs and the subfunctions of the class. Also, the inputs needed to initiate the class are called the parameters.

The goal of this routine is to give a measure of the performance of each dimension reduction method in terms of the number of components, p, to which they are reducing the original data set, X and Y. Thus, once the data set has been projected onto a lower dimensional subspace, it needs to be validated using some sort of machine learning model. The way to do this is using cross-validation techniques. For the purpose of this thesis, we have used the holdout method, splitting the data in 70% training and 30% test. Both the training and test subsets are given to each of the dimension reduction

methods to obtain their corresponding projected training and test subsets. Then, the training and test subsets are given to the appropriate machine learning model along with the response variables, to predict and give a measure of the error rate obtained. The errors from the machine learning models are averaged over 50 random splits of the data. See Figure 5.1 for an illustrative representation.

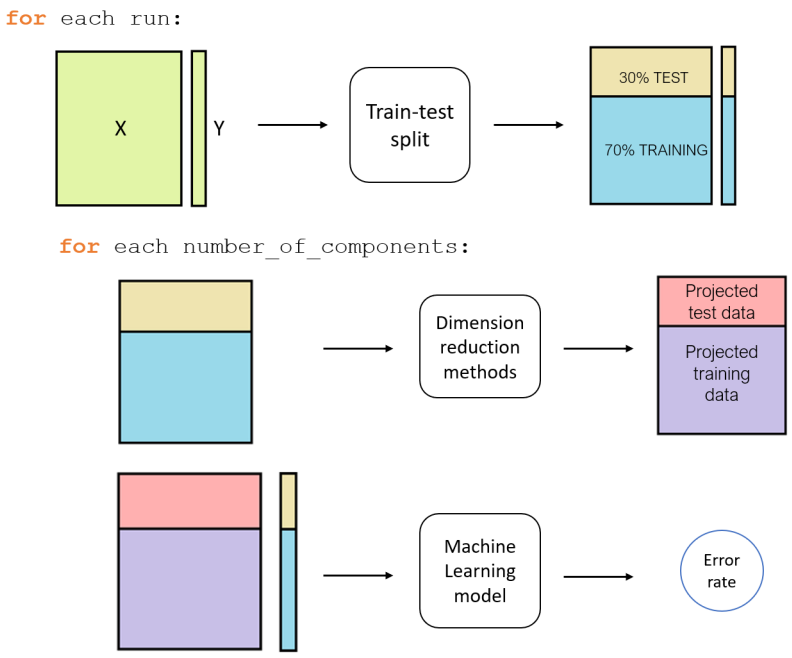


Figure 5.1: Schematic representation of the dimension reduction routine.

Some of the parameters of the `DimReductionClass` are the number of runs, the maximum number of components and whether to use a classification or regression machine learning model. Also, the attributes consist of collection of arrays, one for each method, containing the error rate in terms of the number of components to which the data set has been reduced. Finally, one of the main methods is the `_get_gamma` function, which will be explained in the next section.

5.1.1 Hyperparameter optimal search

In this section, we will explain the foundation of the hyperparameter optimal search implemented in the code. In machine learning it is very typical to deal with certain parameters that need to be adjusted before applying the machine learning model in question. In order to do that, one would need to split the data using cross-validation techniques, while going over a set of possible values for the desired parameter. Then, store the accuracy of the predicted data with each of the parameter values and finally select the one that gave the better results.

In our case, the parameter we need to adjust is the γ coefficient of the squared exponential kernel for the training data. Recall that $\gamma = 1/(2\sigma^2)$.

$$k(x, x') = exp(-\gamma \|x_i - x_j\|^2) \tag{5.1}$$

The value of the γ parameter will determine the performance of the dimension reduction methods which use that function, i.e., kernel PCA, kernel SPCA and kernel LSR-PCA. Nonetheless, its optimal value will vary for each of this methods because each method has its own eigenvalue problem and thus, they find its own matrix of projection. Therefore, we have seen that we will need to compute an optimal γ for each kernel method, γ_{KPCA}, γ_{KSPCA} and $\gamma_{KLSR-PCA}$.

In addition, the object that will give us the accuracy (or error) for each one of the values of γ is a machine learning model. Therefore, our estimation object should be a combination between a kernel dimension reduction method and a machine learning model. The Python class that can combine these two objects in one is called *Pipeline*. `Pipeline` is available at the Scikit-learn Python's library. It is a powerful tool that can assemble several steps that can be cross-validated together while setting different parameters [28].

Finally, the function that finds the optimized parameters of an estimator, by a cross-validated grid-search over a parameter grid is the Scikit-learn's class, `GridSearchCV`. The estimator will be the object built using the *pipeline*. The parameter grid given is an array containing 40 different values from 10^{-3} to 10^2 in a logarithmic scale.

5.2 Implementation of the methods

This section will give an overview of the way that all the methods explained above have been implemented in order to compare their performance.

The advantage of using Python's libraries is that some of these methods are already implemented as Python classes. This is the case of PCA, PLS or kernel PCA, which are available at Scikit-learn's decomposition and Scikit-learn's cross_decomposition packages [28].

Regarding the other methods, both SPCA and LSR-PCA were implemented using our own code, based on Algorithms 2 and 3, respectively. Finally, as for kernel SPCA and kernel LSR-PCA, they were also implemented using our own code. However, they have been written in a Scikit-learn-like interface, so that the `GridSearchCV` class recognized them as valid methods.

Chapter 6

Computational examples

In this chapter we will apply LSR-PCA to a set of visualization, classification and regression problems. We will compare LSR-PCA with Principal Component Analysis (PCA) [2], Partial Least Squares Regression (PLS) [4], and Supervised PCA (SPCA) [3] and show that LSR-PCA is a competitive dimension reduction method for the select problems. We will also apply the kernel formulation of LSR-PCA to a few problems to show its efficacy on non-linear problems.

The data sets used in this thesis are taken from the UCI Machine Learning Repository [29] and the Scikit-learn's datasets Python's package [28]. The description of each data set is shown in Table 6.1.

As we mentioned before, for data sets used for classification and regression, we randomly split the data into 70% training and 30% test data sets. The errors from the machine learning models are averaged over 50 random splits of the data.

6.1 Comparing linear methods

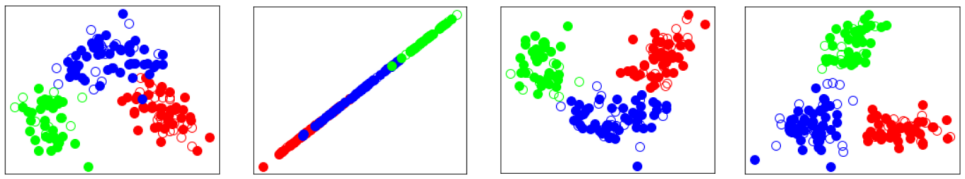
6.1.1 Visualization

For visualization, we chose two data sets from the UCI machine learning repository: the wine and sonar data sets. The descriptions of these data sets are shown in Table 6.1. For both examples, we used only the first two principal components from each dimension reduction method. The input data from the data sets are then projected onto the two principal directions and plotted in Figures 6.1 and 6.2. The training data is represented by filled circles and the test data is represented by hollow circles.

The Wine data set [29] projected on the leading two principal directions are plotted in Figure 6.1. The input data to the Wine data set consists of 13 attributes tabulated for three types of wines (no. of classes). As shown in Figure 6.1, both PCA and PLS show small regions of overlap; both SPCA and LSR-PCA show complete separation of the three classes of wine, but the gaps between the three classes are slightly wider in LSR-PCA than SPCA.

Data set	No. of Instances	No. of Features	Problem Type
Automobile	205	26	Regression
Boston House Prices	506	13	Regression
Breast Cancer	569	30	Classification
Diabetes	442	10	Regression
Iris	150	4	Classification
Mice Protein Expression	1080	82	Classification
Moons	500	2	Classification
Parkinsons	197	23	Classification
Sonar	208	60	Classification
Soybean	307	35	Classification
Wine	178	13	Classification
Yacht Hydrodynamics	308	7	Regression

Table 6.1: Description of the data sets used to analyse the classification and regression problems.



(a) PCA (b) PLS (c) SPCA (d) LSR-PCA

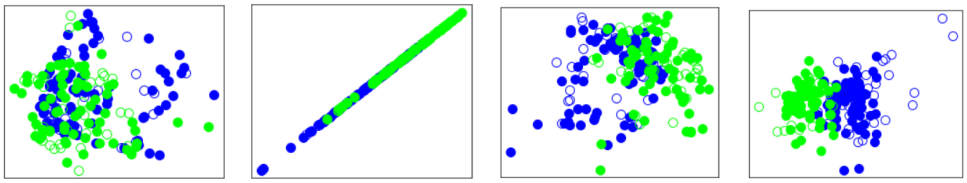
Figure 6.1: 2-dimensional projection of the leading components of the *Wine* data set. Training data is represented by filled markers. Test data is represented by hollow markers. From [6].

In the second visualization example, we use the sonar data set [29]. The data set consists of 208 measurements of sonar signals bouncing off a metal cylinder and rocks at different angles and conditions. Each measurement results in 60 real numbers ranging between 0 and 1. Figure 6.2 shows that all four methods show some overlap between the two classes. However, the overlap in the LSR-PCA is the least out of the four methods.

6.1.2 Classification

In this section, we will present the results for classification problems. We chose the following data sets: Iris, Mice Protein Expression, Soybean and Wine data sets.

The implementation of SPCA and LSR-PCA required a kernel over the target values, K_y. For the classification problems, we chose $k(y, y')$ to be the Kronecker-Delta kernel,



(a) PCA (b) PLS (c) SPCA (d) LSR-PCA

Figure 6.2: 2-dimensional projection of the leading components of the *Sonar* data set. Training data is represented by filled markers. Test data is represented by hollow markers. From [6].

$\delta_{y,y'}$:

$$\delta_{y,y'} := \begin{cases} 1 & \text{if } y = y' \\ 0 & \text{if } y \neq y' \end{cases} \tag{6.1}$$

After the input training data has been projected to the leading principal directions, we apply k-Nearest Neighbor (KNN) model to predict the class for the test data. The test error is computed using the zero-one-loss function, L_{0-1}:

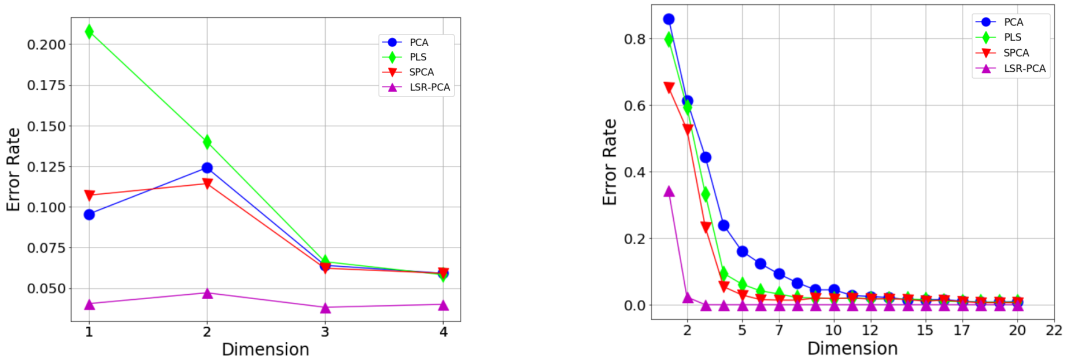
$$L_{0-1}(y, \hat{y}) = I\{\hat{y}_j \neq y_j\} \tag{6.2}$$

where y_j is the observed class label, $\hat{y}_j$ is the predicted class label corresponding to the j^{th} row of the data, and $I(x)$ is the indicator function.

The first example is the Iris data set [29]. This data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. In this example, see Figure 6.3a, LSR-PCA out performs the other three methods. Next, we have the Mice Protein Expression data set [29], which consists of the expression levels of 77 protein modifications that produced detectable signals in the nuclear fraction of cortex of mice. The aim is to identify subsets of proteins that are discriminant between the classes. LSR-PCA reaches the 0% rate in less dimensions than the other methods in this case, see Figure 6.3b. The third example is the Soybean data set [29]. The data set contains 35 properties and characteristics that classify 19 classes of soybeans. The performance of LSR-PCA in this example, see Figure 6.3c, is slightly similar to the other methods until the 14^{th} component, where starts performing slightly better. The last classification example is the Wine data set [29], already described in the previous section. In this example, see Figure 6.3d, LSR-PCA has a better performance than the other three methods.

6.1.3 Regression

Next we show the performance of LSR-PCA in regression problems. For this study, we selected the Automobile, Boston House Prices, the Diabetes, and the Yatch Hydrodynamics data sets. The properties of the data sets are described in Table 6.1.



(a) Iris

(b) Mice protein expression

(c) Soybean

(d) Wine

Figure 6.3: Classification zero-one loss error computed by the methods PCA, PLS, Supervised PCA (SPCA) and Least Squares Regression PCA (LSR-PCA). From [6].

For regression problems, we chose $k(y, y')$ to be the squared exponential kernel over Y:

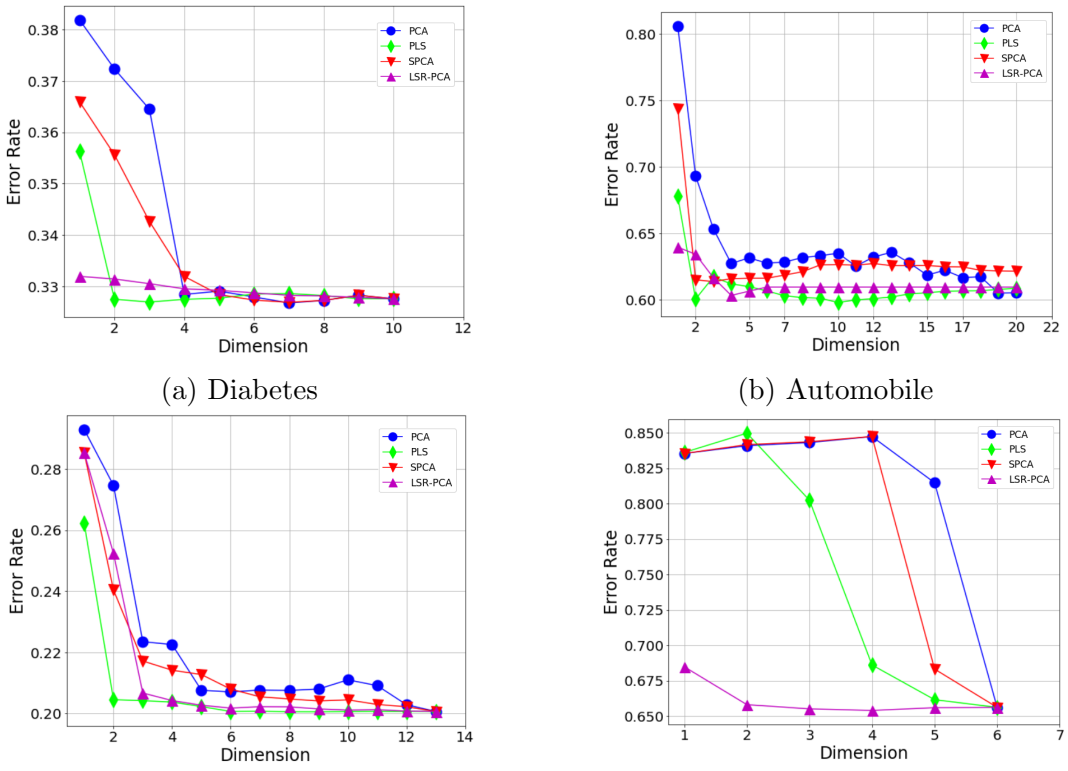
$$k(y, y') = exp\left(-\frac{\|y_i - y_j\|^2}{2}\right). \tag{6.3}$$

where y and y' are $1 \times l$ row vectors of the $n \times l$ target matrix Y. For the regression examples, after the input data is projected onto the selected principal components, linear regression was applied to fit the training data. The error of the linear regression model is measured using the mean squared error:

$$MSE(x, y) = \frac{1}{n}\sum_{j=0}^{n-1}(\hat{y}_j - y_j)^2, \tag{6.4}$$

where x_j is the j^{th} sample of data, y_j is the true response to x_j, and $\hat{y}_j$ is the predicted response for input x_j.

Looking at the results in figures 6.4 we first have a look at the Diabetes data set [28], which contains 10 different measurements obtained for each of 442 patients. In this example, see Figure 6.4a, all 4 methods perform comparably from the 4 component. Then



(a) Diabetes

(b) Automobile

(c) Boston House Prices

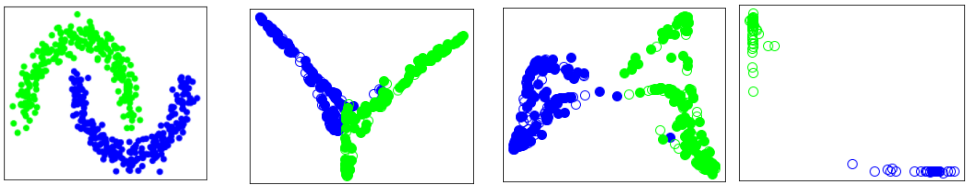
(d) Yacht Hydrodynamics

Figure 6.4: Regression mean square errors computed by the methods PCA, PLS, Supervised PCA (SPCA) and Least Squares Regression PCA (LSR-PCA). From [6].

we see the Automobile data set [29]. This data set consists in 26 different properties from different cars to predict the level of security. Figure 6.4b shows that LSR-PCA performs slightly better than PCA and and SPCA, but slightly worse than PLS. A similar performance can be seen at the Boston House Prices data set [28], see Figure 6.4c, where both PLS and LSR-PCA perform comparably and slightly better than PCA and SPCA. Each sample of the Boston House Prices data set describes a Boston suburb or town. The fourth regression example is the Yacht Hydrodynamics data set [29], which contains the prediction of residual resistance of sailing yachts at the initial design stage. This is used to evaluate the ship's performance and for estimating the required propulsive power. As we can observe in Figure 6.4d, LSR-PCA clearly out performs the other three methods.

6.2 Comparing kernel methods

Lastly, we assess the efficacy of the kernel formulation of LSR-PCA (KLSR-PCA), compared to kernel PCA [26] and kernel SPCA [3]. First, we will take a look at the 2-dimensional projection of the moons data set from the Scikit-learn machine learning



(a) Moons (b) KPCA (c) KSPCA (d) KLSR-PCA

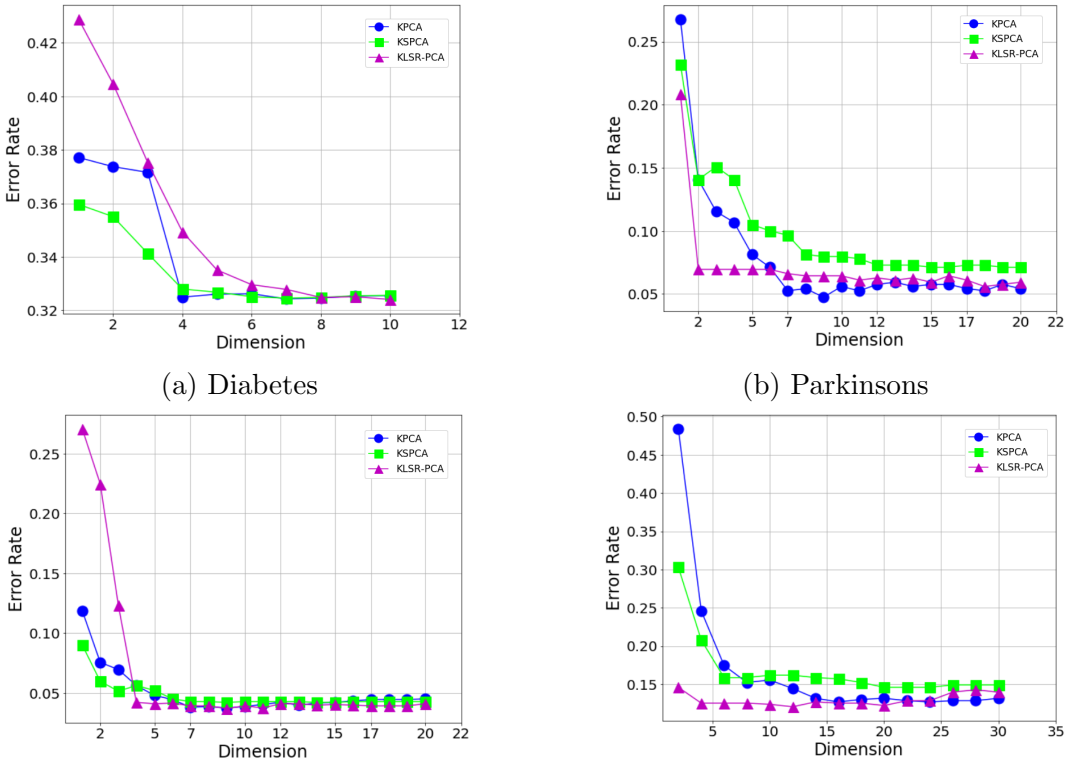
Figure 6.5: 6.5a: Original form of the *Moons* data set. 6.5b, 6.5c, 6.5d: 2-dimensional projection of the leading components of the *Moons* data set by KPCA, KSPCA and KLSR-PCA. Training data is represented with filled markers. Test data is represented with hollow markers. From [6].

library. This is a simple toy data set that makes two interleaving half circles and it is very useful to visualize the performance of classification algorithms. Then, we applied KLSR-PCA to the Diabetes, Parkinsons, Breast Cancer and Sonar data sets, which are described in Table 6.1. We used an squared exponential kernel for the input data matrix:

$$k(x, x^{'}) = exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \tag{6.5}$$

where the length-scale hyperparameter, σ was selected using a 5-fold cross validation scheme, explained in section 5.1.1. Figure 6.5 shows the 2-dimensional projection by three different kernel dimension reduction methods: KPCA, KSPCA and our method, KLSR-PCA. For this example, for the selection of the length-scale hyperparameter, σ, we used support vector machine with a linear kernel as the classification method in order to enforce the *linear* separability of the kernel transformed input data. We found that while all three kernel methods showed linear separation of the two classes, KSPCA and KLSR-PCA provided separations of the two class with a larger distance.

In the next four examples, we compared the classification and regression errors using the three kernel dimensional reductions methods. In the regression example, the Diabetes data set, see Figure 6.6a, KPCA and KSPCA performed better than KLSR-PCA for the first six dimensions, but all three kernel methods show comparable performance as the dimensions increase further. A similar trend can be seen in the next two classification examples, the Parkinsons [29] and Breast Cancer [29] data sets, see Figures 6.6b and 6.6c, respectively. The Parkinsons data set is composed of a range of biomedical voice measurements from 31 people, 23 with Parkinson's disease. The main aim of the data is to discriminate healthy people from those with the disease. As for the Breast Cancer data set, its features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. Lastly, in the sonar data set, see Figure 6.6d, KLSR-PCA performed the best out of all three for the first fifteen dimensions, and then it shows comparable performance to KPCA for the remaining subsequent dimensions.



(a) Diabetes

(b) Parkinsons

(c) Breast Cancer

(d) Sonar

Figure 6.6: Classification and Regression errors computed by the methods Kernel PCA, Kernel SPCA, Kernel Regression PCA. From [6].

Chapter 7

Conclusion

This thesis was devoted to the introduction of a new supervised dimensionality reduction method, the Least Squares Regression Principal Component Analysis (LSR-PCA) [6], a method that finds a set of principal components which are relevant for both the input and output variables by revealing how the input data contributes linearly to building a kernel-transformed target data. We showed how the method can be solved in closed-form. In addition, we developed a kernel formulation for LSR-PCA [6] for input data that are non-linearly correlated.

Furthermore, we have described in detail some of the existing state-of-the-art dimension reduction methods, Principal Component Analysis [15], Partial Least Squares Regression [4] and Supervised PCA [3], which represent different categories of prominent methods of unsupervised and supervised dimension reduction.

Our computational examples in different classes of learning problems, visualization, classification and regression, showed that LSR-PCA performed well compared to the aforementioned existing dimension reduction methods.

In 1997, David Wolpert and William Macready coined the term "no free lunch" (NFL) theorem in their paper *No Free Lunch Theorems for Optimization* [30]. In their paper, the first theorem states that "any two optimization algorithms are equivalent when their performance is averaged across all possible problems" [31]. In the spirit of the "no free lunch" theorem in machine learning, we believe that there is not a single dimension-reduction method that can out-perform all other methods in every learning problem. We have shown in our examples that LSR-PCA should be one of the candidate dimension reduction methods when it comes to picking the best dimension-reduction methods for a learning problem.

To conclude, we would like to state that as of June 29, 2020, we are in preparation of a paper which will introduce LSR-PCA in theory and will gather all the computational examples presented in this thesis.

Bibliography

[1] Shlens, J., "A tutorial on principal component analysis," 2014.

[2] F.R.S., K. P., "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[3] Barshan, E., Ghodsi, A., Azimifar, Z., and Zolghadri Jahromi, M., "Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds," *Pattern Recogn.*, vol. 44, pp. 1357–1371, July 2011.

[4] Geladi, P. and Kowalski, B. R., "Partial least-squares regression: a tutorial," *Analytica chimica acta*, vol. 185, pp. 1–17, 1986.

[5] Bair, E., Hastie, T., Paul, D., and Tibshirani, R., "Prediction by supervised principal components," *Journal of the American Statistical Association*, vol. 101, no. 473, pp. 119–137, 2006.

[6] Pascual, H. and Yee, X., "Least squares regression principal component analysis: a supervised dimensionality reduction method," 2020. Unpublished manuscript.

[7] Géron, A., *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[8] Kohavi, R. *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, pp. 1137–1145, Montreal, Canada, 1995.

[9] McLachlan, G. J., Do, K.-A., and Ambroise, C., *Analyzing microarray gene expression data*, vol. 422. John Wiley & Sons, 2005.

[10] Gupta, P., "Cross-validation in machine learning." https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f, 2017. Accessed: 2020-03-04.

[11] Celisse, A. *et al.*, "Optimal cross-validation in density estimation with the $l\{2\}$-loss," *The Annals of Statistics*, vol. 42, no. 5, pp. 1879–1910, 2014.

[12] Imandoust, S. B. and Bolandraftar, M., "Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background," *International Journal of Engineering Research and Applications*, vol. 3, no. 5, pp. 605–610, 2013.

[13] Brownlee, J., "Linear regression for machine learning." https://machinelearningmastery.com/linear-regression-for-machine-learning/, 2016. Accessed: 2020-02-14.

[14] Gandhi, R., "Introduction to machine learning algorithms: Linear regression." https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a, May 2018. Accessed: 2020-02-14.

[15] Jolliffe, I., *Principal Component Analysis*. Springer Verlag, 1986.

[16] Ghojogh, B. and Crowley, M., "Unsupervised and supervised principal component analysis: Tutorial," *arXiv preprint arXiv:1906.03148*, 2019.

[17] Boyd, S., Boyd, S. P., and Vandenberghe, L., *Convex optimization*. Cambridge university press, 2004.

[18] Wold, H., "Soft modelling by latent variables: The non-linear iterative partial least squares (nipals) approach," *Journal of Applied Probability*, vol. 12, no. S1, p. 117–142, 1975.

[19] Ng, K. S., "A simple explanation of partial least squares," *The Australian National University, Canberra*, 2013.

[20] Rosipal, R. and Clancy, D., "Kernel partial least squares for nonlinear regression and discrimination," 2002.

[21] Höskuldsson, A., "Pls regression methods," *Journal of chemometrics*, vol. 2, no. 3, pp. 211–228, 1988.

[22] Hein, M. and Bousquet, O., "Kernels, associated structures and generalizations," 2004.

[23] Gretton, A., Bousquet, O., Smola, A., and Schölkopf, B., "Measuring statistical dependence with hilbert-schmidt norms," in *International conference on algorithmic learning theory*, pp. 63–77, Springer, 2005.

[24] Raschka, S., Linear, P., Gaussian, R., and LLE, L.-L. E., "Kernel tricks and nonlinear dimensionality reduction via rbf kernel pca," *Blog, September*, 2014.

[25] Tiuplin, A., "A tutorial on kernel principal component analysis." https://atiulpin.wordpress.com/2015/04/02/a-tutorial-on-kernel-principal-component-analysis/, 2015. Accessed: 2020-05-25.

[26] Schölkopf, B., Smola, A., and Müller, K.-R., "Kernel principal component analysis," in *Artificial Neural Networks — ICANN'97* (Gerstner, W., Germond, A., Hasler, M., and Nicoud, J.-D., eds.), (Berlin, Heidelberg), pp. 583–588, Springer Berlin Heidelberg, 1997.

[27] Alperin, J. L., *Local representation theory: Modular representations as an introduction to the local representation theory of finite groups*, vol. 11. Cambridge University Press, 1993.

[28] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[29] Dua, D. and Graff, C., "UCI machine learning repository," 2017.

[30] Wolpert, D. H. and Macready, W. G., "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[31] Wolpert, D. H. and Macready, W. G., "Coevolutionary free lunches," *IEEE Transactions on evolutionary computation*, vol. 9, no. 6, pp. 721–735, 2005.