Imperial College London

Imperial College London

Department of Mathematics

Guided Dimensionality Reduction for Anomaly Detection

Author:
Tudor Trita Trita

Supervisor:
Dr. Andrew Duncan

M4R project associated with the degree of

MSci Mathematics

This is my original own work, except where stated otherwise.

June 9, 2020

Abstract

Dimensionality reduction techniques are becoming increasingly important as a step in processing large amounts of data efficiently. By reducing the dimension of data, we can gain rewards in terms of computational speed, model complexity as well as easy visualisation and interpretation of data. We can also reduce the amount of noise present in data.

In this report, we explore a technique for guided dimension reduction through maximising the dependence between our transformed input variables and our output variables. We quantify the degree of dependence through the Hilbert-Schmidt independence criterion (HSIC). With this method we aim to learn the transformation that finds directions in our input variable space which are most dependent to our output data. We then compare the effectiveness of this method with two other guided or supervised dimension reduction methods; partial least squares (PLS) and gradient-based Kernel Dimension Reduction (gKDR).

Finally, we explore using the HSIC method for anomaly detection on a dataset from social media analytics and compare performance with PLS and gKDR.

Contents

Chapter 1

Introduction

1.1 Motivation

Recent advances have meant that the ability to generate data has increased exponentially. The advent of big data, most notably seen in fields such as finance and social media [1] has meant that fast and efficient algorithms for mining and processing data are becoming increasingly important. Dimensionality reduction (DR) techniques aims to tackle this in two main ways:

First, we can reasonably assume that the speed of any data analysis algorithm, whether it is used for regression, classification, or otherwise, is dependent on the number of inputs or variables that it takes. In fact, there are many examples of algorithms where this dependence is very bad, and we may end up in a situation where the speed of the algorithm scales with dimension exponentially [2]. This is known as the curse of dimensionality, and it is often a limiting factor on the feasibility of many algorithms in real-life applications. By reducing the dimension (number) of the inputs, we can often make vast improvements in the running times of the algorithms, thus reducing the cost, energy and time requirements of any machine that runs the algorithms.

Secondly, many algorithms in statistics require a high signal to noise ratio to output meaningful results, that is, we would like to get rid of meaningless variables in our input as well as somehow 'combine' variables which are correlated. Unfortunately, a lot of multi-dimensional data generated in real life scenarios contain a high amount of noise. DR methods mitigate this by getting rid of a lot of the noise in data, thus potentially increasing the signal to noise ratio in our data, improving the efficacy of the succeeding algorithms.

So far, we have described the use cases of DR techniques as a preprocessing step as part of a larger process. However, reducing the dimension of the data can be the final objective too. These types of DR techniques are also known as data compression techniques and are used extensively for image/video compression and data transmission over long distances.

In this project, we explore DR techniques with the objective of detecting anomalies in data. This is known as anomaly detection and it is a whole field on its own. We can use DR techniques for anomaly detection to identify an underlying low-dimensional and thus smaller model capturing all of the inherent structure of the data. If this low-dimensional subspace is identified, then we can detect anomalies much more efficiently, with less data and furthermore, we are far less prone to false positives.

For this project, we demonstrate how we can use DR techniques on a real social-media

analytics data-set [3]. The data-set monitors the amount of time 10 large publicly-traded companies are mentioned by name on the Twitter social network. Suppose that we want to detect any anomalies in the number of times that a company mentioned. A reason for doing this could be if we were trying to predict the price of the company's stock, we might assume that a high volume of tweets mentioning this company might be strongly correlated with large price movements in the company's stock. Now, if we wish to detect or predict anomalies before they occur, we may want to build a model which takes into account all of the data. In this case, with today's modern computing, this could be achievable, as monitoring data from 10 sources every 5 minutes is not very computationally expensive. However, if we were to scale the monitoring up and want to monitor the volume of tweets for many more companies many times a second, this would be challenging. By using a DR technique as a preprocessing step, we can potentially reduce the amount of data we would need to monitor drastically and thus we can afford to scale our monitoring model up.

So, the main motivation behind this project is to introduce a novel DR technique which allows us to speed up processes by reducing the number of inputs any model needs to work with, resulting in gains in speed and lower costs of operation for whoever uses these models.

1.2 Overview of the Project

This project is organised as follows:

In the remaining parts of this chapter, we provide a non-mathematical introduction to DR as well as provide some technical theoretical background needed to formalise the main concepts of supervised DR.

In chapter 2, we introduce four supervised dimension reduction techniques. The first of which is partial least squares, a now standard linear method for dimension reduction [4]. We will be using PLS as a benchmark to compare the other methods against. We then present a detailed account of the use of positive definite kernels for DR with the gradient-based kernel dimensionality reduction (gKDR) method [5] and introduce a novel technique of using an independence criterion known as the Hilbert-Schmidt independence criterion (HSIC) [6] for DR through manifold learning. Finally, we introduce a DR technique best suited to speeding up numerical computations known as Active subspaces, developed extensively by [7] [8]. Active subspaces take a different approach compared to the other modules, since it is not purely data driven. With active subspaces, we explore a much stronger functional relationship between our input and output data by constructing a lower-dimensional model of whatever process is creating the data

In chapter 3, we will be testing the basic functionality of the DR methods discussed in chapter 2 as a pre-processing step by comparing the regression error for a simple nearest neighbors regressor trained on the low-dimensional data. We will be creating synthetic data to test the basic funcionality of the methods.

In chapter 4, we will be applying our DR techniques for anomaly detection on a synthetic dataset as well as on the social media dataset from Twitter.

Finally, in chapter 5, we will be summarising the results found in this project as well as giving some ideas for future work.

The appendix of this project contains relevant code used to implement the DR methods, as well as helper code for performing model selection through K-Fold cross-validation and code used for the cleaning of the Twitter dataset.

1.3 Overview on Dimension Reduction

In this section, we give a brief overview of the types of techniques available to us for performing DR. There are two broad types of DR techniques. These are supervised and unsupervised DR techniques. Roughly speaking, the difference between these two types is in the use of the labels of the data.

In unsupervised DR, we are interested in finding patterns within the input variables only during training. Perhaps the most widely used unsupervised DR technique is principal component analysis (PCA).

Supervised DR makes use of the label data during training and can therefore find information within the data not available to unsupervised methods by finding a functional relationship between the inputs and the outputs. Whilst the performance of supervised DR tends to be higher when compared to unsupervised DR, the amount of cases where supervised DR can be applied is lower. This is because we require labeled data for the supervised DR techniques to work, whereas we can apply the unsupervised DR techniques to both labeled and unlabeled data.

We can further divide our techniques into linear and non-linear DR techniques:

- In linear DR, the general approach is to find hyper-planes or directions in our input data which best summarise the data, either through variance, dependence criteria or otherwise. These methods tend to be computationally cheap and also tend to be easier to interpret.
- In non-linear DR, we search for a general map from a higher dimensional space to a lower dimensional space, also known as manifold learning. Non-linear DR has the huge advantage that we can extract information that is inaccessible to linear DR techniques at the expense of mathematical and computational complexity.

In this project, we will be concentrating on supervised DR techniques.

1.4 Theoretical Background to Supervised Dimension Reduction

We assume that we have a data matrix $X \in \mathbb{R}^{n\times p}$, where $n-$number of data points, $p-$number of dimensions/features as well as a matrix of responses $Y \in \mathbb{R}^{n\times d}$, where $d-$number of response dimensions/variables. We frame the DR in terms of the expectation of our response given the input data, in other words, finding the conditional expectation $\mathbb{E}[Y|X]$. However, in most applications the number of dimensions in our data p may be very large, making computations too expensive to perform. The type of DR techniques that are introduced next gives one way of tackling this problem.

Definition 1: (Dimension Reduction [9])

Let $R : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times q}, q \leq p$ be a function and $X \in \mathbb{R}^{n\times p}$ be a matrix, i.e. R maps X to a subset $\mathbb{R}^{n\times q} \subseteq \mathbb{R}^{n\times p}$. If $q < p$, then we say that $R(X)$ is a *dimension reduction*.

We wish to use the dimension reduction to estimate $\mathbb{E}[Y|X]$ by a conditional expectation from a lower dimensional subspace i.e. $\mathbb{E}[Y|R(X)]$. We say that the $R(X)$ is a *sufficient dimension reduction* if we can retain all the information about Y after reducing the dimensionality of our data [9]. The following definition formalises this idea.

Definition 2: (Sufficient Dimension Reduction [9])

Let $R : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times q}, q \leq p$ be as in definition 1. Then R is *sufficient* if either of the following statements holds:

1. $X|(Y, R(X)) \overset{i.i.d}{\sim} X|R(X)$ (*inverse reduction*)
2. $Y|X \overset{i.i.d}{\sim} Y|R(X)$ (*forward reduction*)
3. $X \perp\!\!\!\perp Y|R(X)$ (*joint reduction*)

Using an idea from the field of regression graphics, we make use of *sufficient summary plots*, which are lower-dimensionality graphical representations (e.g. scatter plots) of our problem keeping all of the available regression information [10]. Since we can only visualise up to 3 dimensions, these plots will be at most in three dimensions.

The following definition formalises the idea of finding the lower-dimensional subspace associated with a dimension reduction function.

Definition 3: (Dimension Reduction Subspace [11])

Let $\mathbf{x} \in \mathbb{R}^p$ be a vector, an observation from our feature space and suppose that we can express a sufficient dimension reduction $R(x) := B^T\mathbf{x}$, for some matrix $B \in \mathbb{R}^{p\times q}$, i.e. $y \perp\!\!\!\perp \mathbf{x}|B^T\mathbf{x}$. Denote $\mathcal{S}(\mathcal{B})$ to be the subspace of $\mathbb{R}^p$ spanned by the columns of B. The space $\mathcal{S}(\mathcal{B})$ is called a *dimension reduction subspace* (DRS).

We can always to obtain a DRS trivially by setting B equal to the identity matrix in p-dimensional space. It would seem reasonable to try to find a DRS which most reduces the dimensionality of our data, i.e. minimising q in the definition above. This motivates the following definition, which makes this more idea more concrete.

Definition 4: (Central Subspace [10] [11])

Let $\mathcal{S}_{y|\mathbf{x}}$ be the intersection of all DRS's. It can be shown that $\mathcal{S}_{y|\mathbf{x}}$ is a DRS under certain conditions. If we assume that $\mathcal{S}_{y|\mathbf{x}}$ is a DRS, then it is known as the *central subspace*, with associated *structural dimension* $d = \dim(\mathcal{S}_{y|\mathbf{x}})$.

The ultimate goal of any supervised DR technique is to find the dimension reduction function that spans the central subspace. If two DR techniques find a central subspace, then we should pick the one that is the least computationally expensive to use.

Chapter 2

Dimension Reduction Methods

2.1 Partial Least Squares

Partial least squares (PLS) [4] [12] is a popular method for linear supervised DR and is often used as the supervised analogue to principal components analysis (PCA). PLS is actually a regression method, known as PLS regression that uses the DR technique as part of the algorithm, much in the same way principal components regression (PCR) uses PCA to do regression. The basic idea behind PLS is to find directions between components of X and Y which are most correlated.

It is worth noting that the PLS algorithm is sensitive to the sizes of the inputs, therefore we will need to center and scale our data X, i.e. subtracting the column means and dividing through by the column variances. From now on, we will assume that X, Y are centered and scaled matrices.

In this method, we decompose our inputs into

$$X = TP^T + E_X,$$
$$Y = UQ^T + E_Y,$$

where

- $T, U \in \mathbb{R}^{n \times q}$ are the low-dimensional representations of X and Y respectively, known as the *score* matrices in the literature. Here, q is a parameter that governs the dimension of these matrices.
- $P \in \mathbb{R}^{p \times q}, Q \in \mathbb{R}^{d \times q}$ are orthogonal matrices, used to transform data back to the original form, known as the *loading* matrices in the literature.
- E_X, E_Y are the error matrices, assumed to be non-zero in general for a non-perfect fit.

To obtain the lower dimensional representation that PLS gives, we will to find the decompositions of T, U. There exist several algorithms for obtaining these components. The Python module scikit-learn [13] provides a fast implementation of one such PLS algorithm through the class `sklearn.cross_decomposition.PLSRegression`. After fitting this model on some training data, we can use the fitted model on test data to generate new predictions.

2.2 Kernel Dimension Reduction

In this section, we present the theory and implementation of a method developed by [14][15][5] for performing supervised dimension reduction through the use of positive definite kernels, known as kernel dimension reduction (KDR) [15]. Specifically, we will be introducing the faster gradient-based kernel dimension reduction (gKDR) [5] as the optimization procedure in gKDR is many times faster than that of KDR. We note, that this entire section follows the paper [5] closely, both in terms of notations and concepts, with added explanations and concepts where needed.

We begin this section by presenting some background theory on Hilbert spaces, positive definite kernels and reproducible kernel Hilbert spaces.

2.2.1 Hilbert Spaces

Before we can introduce the concept of a Hilbert space, we introduce the following definition of an inner product space, which is a vector space with an associated inner product function satisfying the basic properties of the inner product:

Definition 5: (Inner product space)

Let $\mathcal{V}$ be a vector space over a field of scalars $\mathcal{F}$. An inner product space consists of $\mathcal{V}$ and an inner product $\langle \cdot, \cdot \rangle : \mathcal{V} \times \mathcal{V} \to \mathcal{F}$ which satisfies the following properties:

- $\langle x, y \rangle = \overline{\langle y, x \rangle}, \quad \forall x, y, \in \mathcal{V}$ (Conjugate Symmetry)
- $\langle ax + by, z \rangle = a\langle x, z \rangle + b\langle y, z \rangle, \quad \forall x, y, z \in \mathcal{V}, \forall a, b \in \mathcal{F}$ (Bilinearity)
- $\langle x, x \rangle \geq 0$ and $\langle x, x \rangle = 0$ if and only if $x = 0, \forall x \in \mathcal{V}$ (Positive-Definiteness)

Note, that the inner product space can be generalised to work with functions, and we can define the norm associated with the inner product of the space $\mathcal{V}$ by

$$\|f\|_{\mathcal{V}} = \sqrt{\langle f, f \rangle_{\mathcal{V}}}.$$

We now need one more technical definition before we introduce the concept of a Hilbert space.

Definition 6: (Cauchy sequence)

A sequence $x_n{}_{n=1}^{\infty}, x_n \in \mathcal{V}$, where $\mathcal{V}$ is an inner product space is Cauchy if $\forall \epsilon > 0, \exists N \in \mathbb{N}$ such that $\forall n, m \geq N, \|x_n - x_m\|_{\mathcal{V}} < \epsilon$.

We are now ready to give the definition of a Hilbert space, which forms the foundation of much of the theory underpining KDR.

Definition 7: (Hilbert space)

A Hilbert space $\mathcal{H}$ is an inner product space such that the limits of all Cauchy sequences are contained in $\mathcal{H}$. In this project, we say that all Hilbert spaces are separable.

2.2.2 Theory on Reproducible Kernel Hilbert Spaces

This section provides the theoretical foundation required to understand operations involving kernels and reproducible kernel Hilbert spaces (RKHS). In this project, we concentrate

on real-valued positive definite kernels, but the definitions and results shown can be generalised for use with more general fields such as complex numbers or graphs.

Definition 8: (Positive definite Kernel [16])

Let $\mathcal{H}$ be a set. A positive definite (symmetric, real-valued) kernel, k on $\mathcal{H}$ is a function $k : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ such that the following properties hold:

- $k(x, y) = k(y, x)\ \forall x, y \in \mathcal{H}$ (Symmetry)
- $\sum_{i,j=1}^{n} c_i c_j k(x_i, x_j) \geq 0, \forall x_i \in \mathcal{H}, c_i \in \mathbb{R}, i = 1, \ldots, n,\ n \in \mathbb{Z}$ (Positive Def.)

From this definition, it follows that the matrix $K = (k(x_i, x_j))_{i,j=1}^{n}$, called the Gram matrix associated with the kernel k, is symmetric and positive definite.

We can also characterise the positive definite kernel via the following proposition, which says that we can express a positive definite kernel as the inner product of a feature map in a different dimensional space.

Proposition 1: [17]

Let V be an inner product space. If we let Φ be the feature map

$$\phi : \mathcal{X} \to V, \quad x \to \Phi(x), \tag{2.1}$$

then the kernel on $\mathcal{X}$ defined by $k(x, y) = \langle \Phi(x), \Phi(y) \rangle$ is positive definite.

The result above tells us that we an operation involving a kernel on some data in a dimensional space can be thought of as the inner product of a transformation of the data into a space of different dimensionality. In kernel dimension reduction, this property allows us to extract non-linear features of a lower dimensional space by transforming our data into a space with a different dimensionality where the features have (potentially) become linear.

Examples of popular (real-valued) kernels, for $x, y \in \mathbb{R}^n$:

- Polynomial kernel:

$$k(x, y) = (x^T y + c)^d, \quad (c \geq 0, d \in \mathbb{N}), \tag{2.2}$$

- Laplacian kernel:

$$k(x, y) = \exp\left(-\alpha \sum_{i=1}^{n} |x_i - y_i|\right), \quad (\alpha > 0). \tag{2.3}$$

- Gaussian radial basis function (RBF) kernel:

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right), \quad (\alpha \in \mathbb{R}). \tag{2.4}$$

- Sigmoid kernel:

$$k(x, y) = \tanh(ax^T y + b), \quad (a, b \in \mathbb{R}). \tag{2.5}$$

We are now ready to give the definition of a RKHS, together with the very important reproducing property of positive definite kernels.

Definition 9: (RKHS)

Let Ω be a set and $\mathcal{H}$ a Hilbert space. If $\mathcal{H}$ consists of functions on Ω s.t. $\forall x \in \Omega, \exists$ a function $k_x \in \mathcal{H}$ satisfying the reproducing property

$$\langle f, k_x \rangle_{\mathcal{H}} = f(x), \quad (\forall f \in \mathcal{H}), \tag{2.6}$$

then $\mathcal{H}$ is said to be a reproducible kernel Hilbert space and $k(\cdot, x) := k_x(\cdot)$ is called the reproducing kernel of $\mathcal{H}$.

It can be shown that the reproducible kernel of a RKHS is a positive definite kernel, and the following theorem proves that every positive definite kernel has an associated RKHS.

Theorem 1: (Moore-Aronszajn [16])

Let $k : \Omega \times \Omega \to \mathbb{R}$ be a positive definite kernel. Then, there exists an unique RKHS $\mathcal{H}_k$ on Ω such that:

- $k(\cdot, x) \in \mathcal{H}_k, \forall x \in \Omega$
- $span\{k(\cdot, x) | x \in \Omega\}$ is dense in $\mathcal{H}_k$
- k is the reproducing kernel on $\mathcal{H}_k$

An useful fact about positive definite kernels is that the positive definiteness is preserved under addition, product and limits, i.e., if we have positive definite kernels $k_i : \Omega \times \Omega \to \mathbb{R}$, then the following operations will produce a positive definite kernel as well: $\alpha k_1 + \beta k_2, (\alpha, \beta \geq 0)$, $k_1(x, y) \cdot k_2(x, y)$ and $\lim_{i \to \infty} k_i(x, y)$ if it exists. This means that we can easily construct different types of positive definite kernels from others [16].

2.2.3 Cross-Covariance Operators and Estimator for Conditional Expectation

The objective of the gKDR method is to obtain an estimator for the central subspace (see definition 4) given a finite sample from the distribution of our data $p(Y|X)$. The conditional independence condition seen previously $Y \perp\!\!\!\perp X | B^T X$ is equivalent to saying $p(Y|X) = \tilde{p}(Y|B^T X)$, where $p, \tilde{p}$ are conditional PDFs and B is a projection matrix.

Proceeding from a regression viewpoint, we assume that the conditional expectation (the regression function) $\mathbb{E}[Y|X = x]$ is differentiable with respect to x. Then, assuming that we can exchange integrals and partial derivatives,

$$\begin{aligned}
\frac{\partial}{\partial x}\mathbb{E}[Y|X = x] &= \frac{\partial}{\partial x} \int y p(y|x) \, \mathrm{d}y, \\
&= \int y \frac{\partial \tilde{p}(y|B^T x)}{\partial x} \, \mathrm{d}y, \\
&= B \int y \left. \frac{\partial \tilde{p}(y|z)}{\partial z} \right|_{z = B^T x} \mathrm{d}y
\end{aligned}$$

implying that this gradient is contained in the central subspace.

Following the notation in the [5] paper, we let $(\mathcal{X}, \mathcal{B}_{\mathcal{X}}, \mu_{\mathcal{X}}), (\mathcal{Y}, \mathcal{B}_{\mathcal{Y}}, \mu_{\mathcal{Y}})$ be measure spaces, and (X, Y) be a random variable on $\mathcal{X} \times \mathcal{Y}$ with joint probability distribution P_{XY}. Also,

let $k_{\mathcal{X}}, k_{\mathcal{Y}}$ be measurable positive definite kernels on $\mathcal{X}, \mathcal{Y}$ with associated RKHS $\mathcal{H}_{\mathcal{X}}, \mathcal{H}_{\mathcal{Y}}$. We know that these must exist by Theorem 1.

To proceed, we define the cross-covariance operator. In the definition below, $\mathbb{E}_{XY}$ is taken to be the joint expectation associated with the joint probability distribution P_{XY} and $\mathbb{E}_X, \mathbb{E}_Y$ are the expectations associated with the marginal probability distributions P_X, P_Y respectively.

Definition 10: (Cross-Covariance Operator [5])

Let $\Phi_{\mathcal{X}} : \mathcal{X} \to \mathcal{H}_{\mathcal{X}}$, $\Phi_{\mathcal{Y}} : \mathcal{Y} \to \mathcal{H}_{\mathcal{Y}}$ be the feature maps defined by $x \to k_{\mathcal{X}}(\cdot, x), y \to k_{\mathcal{Y}}(\cdot, y)$ respectively. Then, we define the following cross-covariance operators by

- $C_{YX} : \mathcal{X} \to \mathcal{Y}$ such that the inner product

$$\langle g, C_{YX} f \rangle_{\mathcal{H}_{\mathcal{Y}}} = \mathbb{E}_{XY}[f(X)g(Y)] = \mathbb{E}_{XY}[\langle f, \Phi_{\mathcal{X}}(X) \rangle_{\mathcal{H}_{\mathcal{X}}} \langle \Phi_{\mathcal{Y}}(Y), g \rangle_{\mathcal{H}_{\mathcal{Y}}}]. \quad (2.7)$$

- $C_{XX} : \mathcal{X} \to \mathcal{X}$ such that the inner product

$$\langle f_2, C_{XX} f_1 \rangle_{\mathcal{H}_{\mathcal{X}}} = \mathbb{E}_X[f_2(X)f_1(X)] = \mathbb{E}_X[\langle f_1, \Phi_{\mathcal{X}}(X) \rangle_{\mathcal{H}_{\mathcal{X}}} \langle \Phi_{\mathcal{X}}(X), f_2 \rangle_{\mathcal{H}_{\mathcal{X}}}]. \quad (2.8)$$

which hold $\forall f, f_1, f_2 \in \mathcal{H}_{\mathcal{X}}, g \in \mathcal{H}_{\mathcal{Y}}$.

Now, using the reproducing property of the the positive definite kernels, and taking $g = k_{\mathcal{Y}}(\cdot, y)$, $f_1 = k_{\mathcal{X}}(\cdot, x), f_2 = f$, then

$$\begin{aligned} \langle k_{\mathcal{Y}}(\cdot, y), C_{YX} f \rangle_{\mathcal{H}_{\mathcal{Y}}}(y) &= (C_{YX} f)(y), \\ &= E_{XY}[f(X)k_{\mathcal{Y}}(y, Y)] \quad \text{by definition}, \\ &= \int k_{\mathcal{Y}}(y, \tilde{y}) f(\tilde{x}) \, \mathrm{d}P_{XY}(\tilde{x}, \tilde{y}), \quad \text{by definition of expectation}, \end{aligned}$$

and similarly

$$\begin{aligned} \langle k_{\mathcal{X}}(\cdot, X), C_{XX} f \rangle_{\mathcal{H}_{\mathcal{X}}}(x) &= (C_{XX} f)(x), \\ &= E_X[f(X)k_{\mathcal{X}}(x, X)] \quad \text{by definition}, \\ &= \int k_{\mathcal{X}}(x, \tilde{x}) f(\tilde{x}) \tilde{P}_X(\tilde{x}), \quad \text{by definition of expectation}. \end{aligned}$$

Now, if we wish to estimate these cross-covariance quantities given an IID sample

$$\{(X_i, Y_i)\}_{i=1}^n,$$

where each of the $(X_i, Y_i) \overset{i.i.d}{\sim} P_{XY}$, we can use the empirical estimates for the covariance operators

$$\widehat{C}_{YX}^{(n)} f = \frac{1}{n} \sum_{i=1}^n k_{\mathcal{Y}}(\cdot, Y_i) \langle k_{\mathcal{X}}(\cdot, X_i), f \rangle_{\mathcal{H}_{\mathcal{X}}}, \quad (2.9)$$

$$= \frac{1}{n} \sum_{i=1}^n f(X_i) k_{\mathcal{Y}}(\cdot, Y_i), \quad \text{by reproducing property}. \quad (2.10)$$

and similarly

$$\widehat{C}_{XX}^{(n)} f = \frac{1}{n} \sum_{i=1}^n f(X_i) k_{\mathcal{Y}}(\cdot, X_i). \quad (2.11)$$

Now, the following result, given in [14], provides the connection between the cross-covariance operators and the conditional expectation, which will be used to construct the gKDR estimator for the central subspace.

Theorem 2

Let $g \in \mathcal{H}_\mathcal{Y}$, then if $\mathbb{E}[g(Y)|X = \cdot] \in \mathcal{H}_\mathcal{Y}$, then

$$C_{XX}\mathbb{E}[g(Y)|X = \cdot] = C_{XY}g \tag{2.12}$$

Furthermore, if C_{XX} is injective, then the inverse C_{XX}^{-1} exists, and we can express the above as

$$\mathbb{E}[g(Y)|X = \cdot] = C_{XX}^{-1}C_{XY}g \tag{2.13}$$

The most natural estimator of 2.13 would be $(\widehat{C}_{XX}^{(n)})^{-1}\widehat{C}_{XY}^{(n)}g$, where $\widehat{C}_{XX}^{(n)}, \widehat{C}_{XY}^{(n)}$ were found earlier. If the matrix $\widehat{C}_{XX}^{(n)}$ is nearly singular, then finding the inverse will be very ill-conditioned. To overcome this, Thikonov-type regularization [14] is used, which adds a constant to the diagonals of the matrix $\widehat{C}_{XX}^{(n)}$, thus making it more invertible. We can then obtain a regularized empirical estimator

$$\mathbb{E}[g(Y)|X = \cdot] \approx (\widehat{C}_{XX}^{(n)} + \varepsilon_n I)^{-1}\widehat{C}_{XY}^{(n)}g, \tag{2.14}$$

where ε_n is a regularization coefficient. Note that ϵ_n is usually very small.

Since we are interested in estimating the gradient of the regression function, we make use of the following result:

Proposition 2: [17]

Let $k(x, y)$ be a positive definite kernel, with corresponding RKHS $\mathcal{H}_\mathcal{X}$ be continuously differentiable with respect to x and y. If for any $f \in \mathcal{H}_\mathcal{X}$, f continuously differentiable and $\partial k(\cdot, x)/\partial x \in \mathcal{H}_\mathcal{X}$, then

$$\frac{\partial f(x)}{\partial x} = \left\langle f, \frac{\partial}{\partial x}k(\cdot, x)\right\rangle_{\mathcal{H}_\mathcal{X}} \tag{2.15}$$

This is saying that we can estimate the derivative of the function f through an inner product with a kernel function.

2.2.4 Gradient-based Kernel Dimension Reduction (gKDR)

This section closely follows section 2.3.1 in the paper [5], with added explanations.

Sticking to the notation used in the paper, we let (X, Y) be a random vector on $\mathbb{R}^m \times \mathcal{Y}$, where $\mathcal{Y}$ is a measurable space with respective measure $\mu_\mathcal{Y}$. Let $k_\mathcal{X} : \mathbb{R}^m \to \mathcal{H}_\mathcal{X}, k_\mathcal{Y} : \mathcal{Y} \to \mathcal{H}_\mathcal{Y}$ be positive definite kernels with RKHS's $\mathcal{H}_\mathcal{X}, \mathcal{H}_\mathcal{Y}$.

Assume that $p(Y|X) = \tilde{p}(Y|B^TX)$ for a matrix $B \in \mathbb{R}^{m\times d}$ such that $B^TB = I_d$. We can then express the conditional expectation

$$\mathbb{E}[g(Y)|X] = \int g(y)\tilde{p}(y|B^TX)\,\mathrm{d}\mu_\mathcal{Y} := \varphi_g(B^TX), \tag{2.16}$$

for any $g \in \mathcal{H}_\mathcal{Y}$.

We also need to make the following assumptions to ensure that our results are theoretically valid [5]:

1. The RKHS's $\mathcal{H}_\mathcal{X}, \mathcal{H}_\mathcal{Y}$ are separable.
2. The kernels $k_\mathcal{X}, k_\mathcal{Y}$ are measurable and the expectations $\mathbb{E}[k_\mathcal{X}(X,X)], \mathbb{E}[k_\mathcal{Y}(Y,Y)]$ are finite.
3. $k_\mathcal{X}(\tilde{x}, x)$ is continuously differentiable and
$$\frac{\partial k_\mathcal{X}(\cdot, x)}{\partial x^i}$$
is in the range of the operator C_{XX}.
4. $\mathbb{E}[k_\mathcal{Y}(y, Y)|X = \cdot] \in \mathcal{H}_\mathcal{X}, \forall y \in \mathcal{Y}$, which implies that $\mathbb{E}[g(Y)|X = \cdot] \in \mathcal{H}_\mathcal{X}, \forall g \in \mathcal{H}_\mathcal{Y}$.
5. $\varphi_g(z)$ as defined in 2.16 is differentiable w.r.t. x, and
$$g \to \frac{\partial \varphi_g(z)}{\partial z^a}$$
is continuous $\forall z \in \mathbb{R}^d, a = 1, \ldots, d$. This implies that $\exists \Phi_a(z) \in \mathcal{H}_\mathcal{X}$ such that for $a = 1, \ldots, d$,
$$\langle g, \Phi_a(z) \rangle_{\mathcal{H}_\mathcal{Y}} = \frac{\partial \varphi_g(z)}{\partial z^a}$$
and we let $\Phi_a(z) := \nabla_a \varphi(z)$.

We now want to obtain an expression for the derivative of the conditional expectation in terms of the matrix B. Firstly, using equation (2.16), we work out that for any $g \in \mathcal{H}_\mathcal{Y}$,

$$\frac{\partial}{\partial x^i}\mathbb{E}[g(Y)|X = x] = \frac{\partial \varphi_g(B^T x)}{\partial x^i} = \sum_{a=1}^{d} B_{ia} \langle g, \nabla_a \varphi(B^T x) \rangle_{\mathcal{H}_\mathcal{Y}}. \tag{2.17}$$

But we also know from Theorem 2 and Proposition 2 that for any $g \in \mathcal{H}_\mathcal{Y}$

$$\frac{\partial}{\partial x^i} E[g(Y)|X = x] = \left\langle C_{XY} g, C_{XX}^{-1} \frac{\partial k_\mathcal{X}(\cdot, x)}{\partial x^i} \right\rangle = \left\langle g, C_{YX} C_{XX}^{-1} \frac{\partial k_\mathcal{X}(\cdot, x)}{\partial x^i} \right\rangle. \tag{2.18}$$

Hence, combining these, we obtain that

$$C_{YX} C_{XX}^{-1} \frac{\partial k_\mathcal{X}(\cdot, x)}{\partial x^i} = \sum_{a=1}^{d} B_{ia} \nabla_a \varphi \left(B^T x \right) \tag{2.19}$$

and therefore, we can obtain an expression for the matrix B by

$$\left\langle C_{YX} C_{XX}^{-1} \frac{\partial k_\mathcal{X}(\cdot, x)}{\partial x^i}, C_{YX} C_{XX}^{-1} \frac{\partial k_\mathcal{X}(\cdot, x)}{\partial x^j} \right\rangle_{\mathcal{H}_\mathcal{Y}} = \sum_{a,b=1}^{d} B_{ia} B_{jb} \left\langle \nabla_a \varphi \left(B^T x \right), \nabla_b \varphi \left(B^T x \right) \right\rangle_{\mathcal{H}_\mathcal{Y}}. \tag{2.20}$$

where B_{ij} denotes the ij-th element of the matrix B $i, j = 1, \ldots, m$ [5]. Finding $B \in \mathbb{R}^{m \times d}$ then involves finding the eigenvectors corresponding to the d largest eigenvalues of the matrix defined by

$$M_{ij}(x) = \left\langle C_{YX} C_{XX}^{-1} \frac{\partial k_\mathcal{X}(\cdot, x)}{\partial x^i}, C_{YX} C_{XX}^{-1} \frac{\partial k_\mathcal{X}(\cdot, x)}{\partial x^j} \right\rangle_{\mathcal{H}_\mathcal{Y}}. \tag{2.21}$$

The matrix B is then constructed by taking these eigenvectors as its columns in order, and these are contained within the effective dimension reduction space.

Using the regularised estimator found in equation (2.14), we can construct a regularised estimator for the matrix M, given some data $\{X_i, Y_i\}_{i=1}^n$ and a small real number ϵ_n,

$$\widehat{M}_n(x) = \left\langle \frac{\partial k_{\mathcal{X}}(\cdot, x)}{\partial x}, \left(\widehat{C}_{XX}^{(n)} + \varepsilon_n I\right)^{-1} \widehat{C}_{XY}^{(n)} \widehat{C}_{YX}^{(n)} \left(\widehat{C}_{XX}^{(n)} + \varepsilon_n I\right)^{-1} \frac{\partial k_{\mathcal{X}}(\cdot, x)}{\partial x} \right\rangle. \tag{2.22}$$

We can rewrite this in terms of Gram matrices G_X G_Y, whose elements are given by $(G_X)_{ij} = k_{\mathcal{X}}(X_i, X_j)$ and $(G_Y)_{ij} = k_{\mathcal{Y}}(Y_i, Y_j)$, as well as defining the gradient of the kernels as

$$\nabla \mathbf{k}_X(x) = \left(\frac{\partial k_{\mathcal{X}}(X_1, x)}{\partial x}, \cdots, \frac{\partial k_{\mathcal{X}}(X_n, x)}{\partial x} \right)^T \in \mathbb{R}^{n \times m}. \tag{2.23}$$

Using these notations, we can rewrite $\widehat{M}_n(x)$ as

$$\widehat{M}_n(x) = \nabla \mathbf{k}_X(x)^T \left(G_X + n\varepsilon_n I\right)^{-1} G_Y \left(G_X + n\varepsilon_n I\right)^{-1} \nabla \mathbf{k}_X(x) \tag{2.24}$$

Since we can find eigenvectors of $M(x)$ that are in the effective dimension reduction subspace independently for each x, we use the average of $\widehat{M}_n(X_i)$ over all of the data points X_i, i.e.

$$\tilde{M}_n := \frac{1}{n} \sum_{i=1}^{n} \widehat{M}_n\left(X_i\right), \tag{2.25}$$

and construct the estimate $\widehat{B}$ using the eigenvectors associated with the d largest eigenvalues of this new matrix $\tilde{M}_n$ [5]. The matrix $\widehat{B}$ is known as the gKDR estimator.

2.2.5 gKDR with Common Kernels

It remains to define the various kernel derivatives for the kernel functions shown earlier. In the case for the polynomial kernel, each element of the gram matrix is given by equation (2.2) and the j-th row of $\nabla \mathbf{k}_X(X_i)$ is given by

$$dX_i^T (X_i^T X_j + c)^d \tag{2.26}$$

In the case for the Gaussian RBF kernel, each element of the gram matrix is given by equation (2.4) and the j-th row of $\nabla \mathbf{k}_X(X_i)$ is given by [5]

$$\left(1/\sigma^2\right)\left(X_i - X_j\right) \exp\left(-\left\|X_i - X_j\right\|^2 / \left(2\sigma^2\right)\right) \tag{2.27}$$

Finally, the last kernel we consider is the sigmoid kernel. Each element of the gram matrix is given by equation (2.5) and the j-th row of $\nabla \mathbf{k}_X(X_i)$ is given by

$$aX_i^T(1 - \tanh(aX_i^T x + b)^2) \tag{2.28}$$

It is worth noting that although the gKDR is a general method for dimensionality reduction that works on a variety of datasets, the choice of kernel and hyperparameters as well as regularization coefficient ε_n is very much dependent on the dataset at hand. Therefore, when working with gKDR, we will be using cross-validation techniques to optimise the hyperparameters.

2.3 Dimension Reduction through HSIC

In this section, we will be presenting a novel method for *guided* DR developed in this project. A brief overview of the method is as follows: given a data matrix X and a matrix of responses Y, we wish to estimate the central subspace (see definition 4) by finding a matrix B with $BB^T = I$ such that $B^T X$ is as statistically dependent to Y as possible. We call this method guided DR since we are not only interested in the correlation between input and response, but we want the stronger condition of statistical dependence between input and response.

There are many criteria that we could use to measure independence, however, in this project we concentrate on using the Hilbert-Schmidt independence criterion (HSIC) [6]. The HSIC is a function of two random random vectors with output that can be interpreted as a measure of statistical (in)dependence between these two vectors. More concretely, if the HSIC value is zero, then the two inputs are statistically independent.

During literature review on HSIC for dimensionality reduction, a couple of papers were found where HSIC is used for dimension reduction that are very relevant:

- The paper [18] uses HSIC for *unsupervised* dimension reduction. The method presented here can be summarised as trying to find a lower-dimensional representation X' of the data X that is as statistically dependent to X as possible. This is achieved through maximising the $HSIC(X, X')$.
- The paper [19] presents the same concept as the method presented here, but the optimisation process is different; an eigendecomposition method is used.

This section is structured as follows. We first present some theoretical background to $HSIC$ to show that it is indeed an criterion of independence. Then we present a (biased) estimator of $HSIC$ using a finite sample of data. Finally, we will present the (novel) method used in this project for using $HSIC$ for dimensionality reduction.

2.3.1 Derivation of HSIC as an Independence Criterion

Let (X, Y) be a random vector on $\mathcal{X} \times \mathcal{Y}$. Let $k_{\mathcal{X}} : \mathcal{X} \to \mathcal{H}_{\mathcal{X}}$ be a positive definite kernel with associated RKHS $\mathcal{H}_{\mathcal{X}}$ and feature map ϕ. Likewise, let $k_{\mathcal{Y}} : \mathcal{Y} \to \mathcal{H}_{\mathcal{Y}}$ be another positive definite kernel, with associated RKHS $\mathcal{H}_{\mathcal{X}}$ and feature map φ i.e. $k_{\mathcal{X}}(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}_{\mathcal{X}}}$ and similarly for $k_{\mathcal{Y}}$.

The following definitions seen in [6] and [18] will help us in the derivation of the HSIC.

Definition 11: Hilbert-Schmidt Norm

Let $C : \mathcal{H}_{\mathcal{Y}} \to \mathcal{H}_{\mathcal{X}}$ be a linear operator. We can define the Hilbert-Schmidt (HS) norm of C by

$$\|C\|_{HS}^2 := \sum_{i,j} \langle C v_i, u_j \rangle_{\mathcal{H}_{\mathcal{X}}}^2 \tag{2.29}$$

where u_i and v_j are orthonormal bases of $\mathcal{H}_{\mathcal{X}}$ and $\mathcal{H}_{\mathcal{Y}}$ respectively.

If the HS norm of a linear operator C exists, then we call C a Hilbert-Schmidt operator. We can define the inner product of two HS operators $C_1, C_2 : \mathcal{H}_{\mathcal{Y}} \to \mathcal{H}_{\mathcal{X}}$ by

$$\langle C_1, C_2 \rangle_{HS} := \sum_{i,j} \langle C_1 v_i, u_j \rangle_{\mathcal{H}_{\mathcal{X}}} \langle C_2 v_i, u_j \rangle_{\mathcal{H}_{\mathcal{X}}} \tag{2.30}$$

Next we give the definition of the tensor product in the context of HS operators used to provide an alternative definition to the cross-covariance operator seen before.

Definition 12: Tensor Product

Let $f \in \mathcal{H}_{\mathcal{X}}$ and $g \in \mathcal{H}_{\mathcal{Y}}$. The tensor product is defined as the operator $f \otimes g : \mathcal{H}_{\mathcal{Y}} \to \mathcal{H}_{\mathcal{X}}$ such that $\forall h \in \mathcal{H}_{\mathcal{Y}}$,

$$(f \otimes g)h := f\langle g, h\rangle_{\mathcal{H}_{\mathcal{Y}}}. \tag{2.31}$$

We can show that the tensor product is a HS operator with HS norm given by

$$\|f \otimes g\|_{HS}^2 = \|f\|_{\mathcal{H}_{\mathcal{X}}}^2 \|g\|_{\mathcal{H}_{\mathcal{Y}}}^2 \tag{2.32}$$

Let P_x, P_y be the (marginal) probability measures associated with $\mathcal{X}$ and $\mathcal{Y}$ together with corresponding expectations $\mathbb{E}_X, \mathbb{E}_Y$. The mean elements μ_x, μ_y with respect to the measures P_x, P_y are defined by [18]

$$\langle \mu_x, f\rangle_{\mathcal{H}_{\mathcal{X}}} := \mathbb{E}_x\left[\langle \phi(x), f\rangle_{\mathcal{H}_{\mathcal{X}}}\right] = \mathbb{E}_x[f(x)], \text{ by reproducing property.} \tag{2.33}$$

$$\langle \mu_y, g\rangle_{\mathcal{H}_{\mathcal{Y}}} := \mathbb{E}_y\left[\langle \varphi(y), g\rangle_{\mathcal{H}_{\mathcal{Y}}}\right] = \mathbb{E}_y[g(y)], \text{ by reproducing property.} \tag{2.34}$$

and by taking the expectation twice over x, x' from the distribution P_x, we can obtain

$$\|\mu_x\|_{\mathcal{H}_{\mathcal{X}}}^2 = \mathbb{E}_{x,x'}\left[\left\langle \phi(x), \phi\left(x'\right)\right\rangle_{\mathcal{H}_{\mathcal{X}}}\right] = \mathbb{E}_{x,x'}\left[k\left(x, x'\right)\right] \tag{2.35}$$

We have already seen the cross-covariance operator (definition 10), and we can rewrite it in terms of the tensor product as [18]

$$C_{xy} := \mathbb{E}_{x,y}\left[(\phi(x) - \mu_x) \otimes (\varphi(y) - \mu_y)\right]. \tag{2.36}$$

The HSIC is then defined as the HS norm of the cross-covariance operator, i.e.

$$HSIC(X, Y) := \|C_{xy}\|_{HS}^2 = \mathbb{E}_{x,y}\left[\|(\phi(x) - \mu_x) \otimes (\varphi(y) - \mu_y)\|_{HS}^2\right]. \tag{2.37}$$

This can be written in terms of the kernel functions $k_{\mathcal{X}}, k_{\mathcal{Y}}$ as follows. First, note that we can write the cross-covariance as [6]

$$\mathbb{E}_{x,y}\left[(\phi(x) - \mu_x) \otimes (\varphi(y) - \mu_y)\right] = \mathbb{E}_{x,y}[\phi(x) \otimes \varphi(y)] - \mu_x \otimes \mu_y := \tilde{C}_{xy} - \mu_x \otimes \mu_y, \tag{2.38}$$

where we let $\tilde{C}_{xy} = \mathbb{E}_{x,y}[\phi(x) \otimes \varphi(y)]$. We can therefore write the HSIC as a function of expectations of kernels [6]

$$\begin{aligned}
HSIC(X, Y) &= \mathbb{E}_{x,y}\left[\|(\phi(x) - \mu_x) \otimes (\varphi(y) - \mu_y)\|_{HS}^2\right], && (2.39)\\
&= \left\|\tilde{C}_{xy} - \mu_x \otimes \mu_y\right\|_{HS}^2, && (2.40)\\
&= \langle \tilde{C}_{xy}, \tilde{C}_{xy}\rangle_{HS} + \langle \mu_x \otimes \mu_y, \mu_x \otimes \mu_y\rangle_{HS} - 2\langle \tilde{C}_{xy}, \mu_x \otimes \mu_y\rangle_{HS}, && (2.41)\\
&= \mathbb{E}_{x,y}\mathbb{E}_{x',y'}[k_{\mathcal{X}}(x, x')k_{\mathcal{Y}}(y, y')] + \mathbb{E}_{x,x'}[k_{\mathcal{X}}(x, x')]\mathbb{E}_{y,y'}[k_{\mathcal{Y}}(y, y')] && (2.42)\\
&- 2\mathbb{E}_{x,y}[\mathbb{E}_{x'}[k_{\mathcal{X}}(x, x')]\mathbb{E}_{y'}[k_{\mathcal{Y}}(y, y')]] && (2.43)
\end{aligned}$$

This is known as the population HSIC. The following is an informal explanation of why HSIC measures the independence between two vectors which may help with intuition and understanding. We first note that HSIC measures correlation between two vectors indirectly. We say indirectly, because it actually measures the covariance between the kernel transformations of the vectors in the RKHS spaces.

Now, if the underlying distribution of the data was Gaussian, we know from standard statistics that if two vectors are Gaussian, they are uncorrelated if and only if they are statistically independent. Hence, if we used the identity kernel in HSIC with two uncorrelated vectors, we should see that the value of HSIC is equal to zero, implying that both of these vectors are independent.

However, when a non-trivial kernel transformation is used together with HSIC, we can interpret this as saying that the kernel transformations of the two input vectors are independent in the RKHS space. This property means that the HSIC can extract dependence features in the RKHS of the two input vectors [18].

2.3.2 Estimating HSIC

In practice, we must find a finite-sample estimator to the population HSIC described in the previous section. To do this, we make the following approximations to the values $\tilde{C}_{XY}, \mu_x$ and μ_y [18]. Given a sample of size N of data $\{X_i, Y_i\}_{i=1}^N$,

$$\tilde{C}_{XY} \approx \frac{1}{N}\sum_{n=1}^{N} \phi(X_i) \otimes \varphi(Y_i), \tag{2.44}$$

$$\mu_x \approx \frac{1}{N}\sum_{n=1}^{N} \phi(X_i), \tag{2.45}$$

$$\mu_y \approx \frac{1}{N}\sum_{n=1}^{N} \varphi(Y_i). \tag{2.46}$$

$$\tag{2.47}$$

Using these approximations, we can obtain estimates to each of the individual components in the HSIC population calculation [6][18],

$$\langle \tilde{C}_{xy}, \tilde{C}_{xy} \rangle_{HS} = \mathbb{E}_{x,y}\mathbb{E}_{x',y'}[k_{\mathcal{X}}(x,x')k_{\mathcal{Y}}(y,y')] \approx \frac{1}{N^2} tr(G_X G_Y), \tag{2.48}$$

$$\langle \mu_x \otimes \mu_y, \mu_x \otimes \mu_y \rangle_{HS} = \mathbb{E}_{x,x'}[k_{\mathcal{X}}(x,x')]\mathbb{E}_{y,y'}[k_{\mathcal{Y}}(y,y')] \approx \frac{1}{N^3}\mathbf{1}^T G_X G_Y \mathbf{1}, \tag{2.49}$$

$$\langle \tilde{C}_{xy}, \mu_x \otimes \mu_y \rangle_{HS} = \mathbb{E}_{x,y}[\mathbb{E}_{x'}[k_{\mathcal{X}}(x,x')]\mathbb{E}_{y'}[k_{\mathcal{Y}}(y,y')]] \approx \frac{1}{N^4}\mathbf{1}^T G_X \mathbf{1}\mathbf{1}^T G_X \mathbf{1}, \tag{2.50}$$

where G_X, G_Y are gram matrices associated with the kernels $k_{\mathcal{X}}, k_{\mathcal{Y}}$ and $\mathbf{1}$ is a vector of ones and $tr(\cdot)$ denotes the trace of a matrix. Substituting these values into the equation for the HSIC, we obtain the (biased) empirical estimator

$$\widehat{HSIC}(X,Y) = \frac{1}{(N-1)^2} tr(G_X H G_Y H), \quad H := I - \frac{1}{N}\mathbf{1}^T\mathbf{1}. \tag{2.51}$$

It is worth noting the fact that the computation of the HSIC depends on the choice of kernel. This offers flexibility and similarly to gKDR, the choice of kernel is left to the point of application by model selection algorithms such as cross-validation.

2.3.3 Dimensionality Reduction with HSIC

The method for dimensionality reduction proposed here is as follows. We search for a matrix B whose columns span the central subspace that maximises $HSIC(B^T X, Y)$, in other words, we try to make the projected low-dimensional data as conditionally dependent to Y as possible. Here is a recipe on how to achieve this:

1. Given data (X, Y), choose kernels k_X, k_Y and an initial guess for the matrix B.
2. Find the matrix B that minimises $-tr(G_X(B)HG_YH)$, where

$$(G_X(B))_{ij} = k_{\mathcal{X}}((B^T X_i, B^T X_j)$$

   by gradient descent, subject to the constraint that $B^T B = I$.
3. Once the optimisation is complete, we take the matrix B^* which optimises step 2 as our estimate for the EDR matrix. The dimension reduction is then given by $(B^*)^T X$ i.e. the projected data onto the central subspace.

The constraint $B^T B = I$ can be viewed as B lying on the manifold of orthogonal matrices. It can be shown that the gradient of the HSIC w.r.t to B $\nabla_B HSIC$ also lies in the manifold of orthogonal matrices.

The reason why we are not using the HSIC as our cost function is as follows. First, we can see that the preceding constant in HSIC does not affect the optimisation procedure, so we get rid of it. Second, since we are performing gradient descent, we want to minimise the negative of HSIC, which is equivalent maximising HSIC, and this is why we take the negative trace as our objective function here.

The technique used in this method is known as manifold learning. In the implementation, we make use of the Python module `pymanopt` [20], a library written especially for optimisation of functions on manifolds.

2.4 Active Subspaces

In this section, we present a method [7] [8] for linear supervised dimension reduction that takes a different approach to the general dimension reduction methods seen so far. For this section, we formulate the problem in the following way: given a general function $f : \mathcal{X} \to \mathbb{R}, \mathcal{X} \in \mathbb{R}^n$, depending on an n-dimensional vector of parameters, say x, can we find a lower-dimensional affine subspace in the domain of f in which f changes the most? In other words, can we find directions in the space of parameters $\mathcal{X}' \subseteq \mathcal{X}$ in which f changes the most?

In discovering the active subspace, we require that we are able to generate random samples from f and additionally from the gradient ∇f. Here, we essentially treat f as a black box, since only care about being able to evaluate the function and not its contents in general. This is a departure from the previous approaches seen in PLS and gKDR for example, where we only needed some data generated from f (say) but we didn't explicitly need to generate new data at the point of 'training'. This requirement limits the type of problems active spaces may be applied to. It has been shown [7] that active spaces are very powerful in many engineering applications, where we may have numerical simulations that depend on many parameters, which may be very expensive to perform. If we treat the numerical simulation process as our f, we can potentially find an active subspace, potentially reducing the number of parameters needed for the simulation, which in turn should reduce the time that it takes to evaluate.

This section will be structured as follows. We first introduce a mathematical explanation for the active subspace of a function f (if it exists). We then give practical method of estimating the active subspace through evaluations of f only. Lastly, we will be showing how the active subspace can be used for DR.

2.4.1 Discovery of the Active Subspace

We note here that this section closely follows section 2 in [7], with added explanations where it felt appropriate.

Let $f : \mathcal{X} \subseteq \mathbb{R}^n \to \mathbb{R},\ f = f(x),\ x \in \mathcal{X}$ be a continously differentiable function. Furthermore, let $p_X : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ be the probability density function (PDF) associated with the input parameters. Since p_X is a pdf, we must have that $p_X(x) > 0, x \in \mathcal{X}$ and $p_X(x) = 0, x \notin \mathcal{X}$. Also let $\mathbb{E}_X$ be the expectation in the usual way. We also assume that f is square-integrable with respect to p_X.

Since we assumed that f is continuously differentiable, the gradient

$$\nabla_x f(x) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)^T \tag{2.52}$$

is assumed to exist.

We now define $\mathcal{C} \in \mathbb{R}^{n \times n}$, a positive semidefinite matrix, to be the expectation (in matrix form):

$$\mathcal{C} = \mathbb{E}[(\nabla_x f)(\nabla_x f)^T]. \tag{2.53}$$

Since $\mathcal{C}$ is positive semidefinite, we can decompose it into its real eigendecomposition

$$\mathcal{C} = \mathbf{V} \Lambda \mathbf{V}^T, \tag{2.54}$$

where $\Lambda \in \mathbb{R}^{n \times n}$ is the matrix of eigenvalues sorted in descending order, i.e. $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n),\ \lambda_1 \geq \ldots \geq \lambda_n \geq 0$

To see the how we can obtain the active subspace from the eigenvalues and eigenvectors, we need the following two results, the proof of which can be found in [7] and [8]:

Lemma 1

The mean-squared directional derivative of f with respect to the eigenvector v_i is equal to the corresponding eigenvalue, i.e.

$$\mathbb{E}[((\nabla_x f)^T v_i)^2] = \lambda_i \tag{2.55}$$

What this lemma is saying is that we can quantify the variation of the function f in the direction of the eigenvector v_i with the eigenvalue λ_i. Since we are interested in active directions of high variation, our intuition might say that we should pick the eigenvectors with the higher eigenvalues as our active variables. This turns out to be the case.

Suppose that we have a sequence of eigenvalues $\lambda_1 \geq \ldots \geq \lambda_{n'} >> \lambda_{n'+1} \geq \ldots \lambda_n \geq 0$, for some $n' < n$, where the first n' eigenvalues are much larger than the rest. Then, in a similar argument as the one used in principal components, we can say that the function f has greater variation along the components with larger eigenvalues. If we then decompose Λ and $\mathbf{V}$ into two block-matrix parts corresponding to the larger and smaller sets of eigenvalues,

$$\Lambda = \text{diag}(\Lambda_1, \Lambda_2), \quad \mathbf{V} = [\mathbf{V_1 V_2}], \tag{2.56}$$

where $\Lambda_1 = \text{diag}(\lambda_1, \ldots, \lambda_{n'})$, $\Lambda_2 = \text{diag}(\lambda_{n'+1}, \ldots, \lambda_n)$ are diagonal matrices of eigenvalues and $\mathbf{V}_1 \in \mathbb{R}^{n \times n'}, \mathbf{V}_2 \in \mathbb{R}^{n \times (n-n')}$ are matrices containing eigenvectors corresponding to the eigenvalues of the diagonal matrices Λ_1, Λ_2.

Since we can think of the action of the eigenvectors as rotations, we define the rotated vectors $y \in \mathbb{R}^{n \times n'}$ and $z \in \mathbb{R}^{n-n'}$ by

$$y = \mathbf{V}_1^T x, \quad z = \mathbf{V}_2^T x. \tag{2.57}$$

It is worth noting that $x = \mathbf{V}\mathbf{V}^T x = \mathbf{V_1}\mathbf{V_1}^T x + \mathbf{V_2}\mathbf{V_2}^T x = \mathbf{V}_1 y + \mathbf{V}_2 z \implies f(x) = f(\mathbf{V}_1 y + \mathbf{V}_2 z)$, which looks like a decomposition of x into the variables y and z.

For clarity, we define the gradients:

$$\nabla_y f(x) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_{n'}} \right)^T, \quad \nabla_z f(x) = \left(\frac{\partial f}{\partial x_{n'+1}}, \ldots, \frac{\partial f}{\partial x_n} \right)^T$$

The second result that we need is the following [7] [8]:

Lemma 2

The mean-squared gradients of f with respect to the vectors y and z satisfy

$$\mathbb{E}[(\nabla_y f)^T (\nabla_y f)] = \sum_{i=1}^{n'} \lambda_i, \quad \mathbb{E}[(\nabla_z f)^T (\nabla_z f)] = \sum_{i=n'+1}^{n} \lambda_i. \tag{2.58}$$

This results tells us that our function "f varies more on average along the directions defined by the columns of V_1 than along the directions defined by the columns of V_2" [7]. We call the matrix V_1 to be the basis of the *active subspace* of f, and $y = \mathbf{V}_1^T x$ to be the *active variables*. Informally, we can say that the amount of total variability in the active subspace

is equal to the sum of the eigenvalues of the basis of that subspace, namely the eigenvectors corresponding to the eigenvalues in the sum.

We can say something about the other variables z as well, which we name *inactive variables* here. In the case that the eigenvalues $\lambda_{n'+1} = \ldots = \lambda_n = 0$, then $\nabla_z f(x) = 0, \forall x \in \mathcal{X}$ and we call a function f with this property z-invariant.

The dimension reduction happens when $n' < n$ and ideally $n' << n$ for good performance. In the ideal case, n' is very small (2 or 3) and the function is z-invariant.

2.4.2 Estimation of the Active Subspace

We have described a theoretical way of finding the active subspace, but in practice, computing the matrix $\mathcal{C}$ directly may not be possible, as it involves integrating in many dimensions. Instead, given N samples from our density P_X over $\mathcal{X}$, we can approximate $\mathcal{C}$ approximate using the estimator [7]

$$\mathcal{C} \approx \frac{1}{N} \sum_{i=1}^{N} (\nabla_x f(x_i))(\nabla_x f(x_i))^T \tag{2.59}$$

Another challenge that may occur is in the case that we cannot compute the gradients of the function directly, i.e. we do not have a functional form for $\nabla_x f(x)$ available to us. In this case, we can use a finite difference approach to approximate each of the partial derivatives.

In summary, we can express the method for calculating the active subspace as follows:

Algorithm [8]

1. Draw N independent samples $\{X_i\}$ from the sample density P_X.
2. Compute the gradient $\nabla_x f(X_i)$, possibly using finite differences approximations.
3. Compute the estimator matrix $\widehat{\mathcal{C}}$ of $\mathcal{C}$

$$\widehat{\mathcal{C}} = \frac{1}{N} \sum_{i=1}^{N} (\nabla_x f(x_i))(\nabla_x f(x_i))^T \tag{2.60}$$

4. Calculate the eigen-decomposition of the matrix $\widehat{\mathcal{C}} = \widehat{V}\widehat{\Lambda}\widehat{V}^T$.
5. Obtain a basis for the active subspace by inspecting the eigenvalues contained within the matrix $\widehat{\Lambda}$ to see if an active subspace is present.

Remarks:

- The number of samples drawn in step 1 needs to be chosen. In [8], a formula was obtained for this in the form of $N = \alpha k \log(n)$, where α is an oversampling factor, between 2 and 10, k is defined as $k = m + 1$, where m denotes the largest target dimensionality torelable of the target low-dimensional space and n is the original dimensionality of the problem function f.
- If finite differences are not feasable either, we can construct a (linear) model of $f(x)$ and then compute the gradients in that way. For more information, see Algorithm 1.2 in [8].
- In the ideal case, the value of n' is small (i.e. $n' < 5$) and all eigenvalues $\lambda_{n'+1}, \ldots, \lambda_n$ are zero. However, this will in general not be the case. In the worst case scenario, we could have that all eigenvalues are roughly similar in magnitude, which would imply

that no active subspace is present. In other words, every parameter in our model contributes to the variation of f roughly equally.

- In other cases, we might find there is a jump in eigenvalue magnitude, and we should pick n' equal to the number eigenvalues before that jump occurs.

2.4.3 Making use of the Active Subspace

Once the active subspace has been estimated, we still have not addressed one problem, which is that we cannot use our original f with the lower-dimensional active variables directly. Hence, we must construct a new function $g : \mathbb{R}^{n \times n'} \to \mathbb{R}$ such that

$$f(x) \approx g(V_1^T x) = g(y). \tag{2.61}$$

The construction of the function g is addressed in chapter 4 of [8] as well as section 3 of [7].

The simplest approximation would be to simply fix $z = 0$ and take $g(y) = f(\mathbf{V_1}x)$. It turns out that if f is z-invariant, this approach works and we can say that we have achieved DR using Active Subspaces.

If the function f is not exactly z-invariant, i.e. z is not exactly equal to the zero vector, it turns out that this approximation is not optimal. [8] outlines a technique which involves constructing a regression surface which approximates f well. We give an outline of the method here. The idea is that given a set of points $\{x_i, f(x_i)\}_{i=1}^M$, we use them to construct a regression surface which interpolates these points with a lower dimensional input. A regression surface is nothing more than an interpolating function that is able to approximate the outputs of $f(x)$ but takes in a lower-dimensional input.

Regression surface for the active subspace [8]

1. Sample M points from the distribution $x_i \sim P_x$ and compute $q_i := f(x_i)$.
2. For each x_i, let $y_i = \mathbf{V_1}^T x_i$.
3. Fit a regression surface $g(y)$ using the points $\{y_i, q_i\}$.
4. The regression surface $g(y) = g(\mathbf{V_1}x)$ should approximate the function $f(x)$.

We can use this regression surface $g(y)$ as our dimension reduction function to approximate function calls to f at a lower computational cost.

A recent non-linear extension of active subspaces, called active manifolds [21] provides a non-linear analogue to active subspaces for finding a manifold along which the function f has greatest variation.

Chapter 3

Dimension Reduction in Regression

In this chapter, we will be testing the dimensionality reduction performance of the methods discussed in chapter 2 from a regression viewpoint. In section 3.1, we test these on synthetically generated data.

To quantify the performance of the DR methods, we will be using a similar process to the approach used in [5] for comparing DR methods. That is, if we let $(X_i, Y_i)_{i=1}^N, X_i \in \mathbb{R}^p, Y_i \in \mathbb{R}$ denote an independent, identically distributed (IID) sample of data drawn from some distribution P_{XY}, we perform cross-validation using a k-nearest-neighbors (kNN) regressor and pick the model that produces the smallest mean-squared error (MSE) value.

3.1 Tests on synthesised data

The data used in this section has been generated specifically for this project, and gives a good insight into the basic functionality of the DR methods. In statistics, it is always a good idea to critically test any algorithm using synthetic data because (1) the data is already clean and no pre-processing is required, (2) all the properties of the data are known, such as the exact distribution of the data.

We introduce some notation which will be used when defining the models for the data generation. Let $X_i^{(j)}$ denote the j-th component of the vector X_i and $X_i^{(j,\dots,k)}$ denote the j-th to k-th components of the vector X_i, where we assume j, k are positive integers and $j < k$. We also denote $U[a, b]$ for the uniform distribution on the interval $[a, b]$ and $\mathcal{N}(\mu, \sigma^2)$ the normal distribution with mean μ and variance σ^2. Also, denote $U[a, b]^m$ an m-dimensional vector where each of the components are uniformly distributed on the interval $[a, b]$.

For this project, we will be looking at the following four types of data:

1. We define each of the samples of this model (X_i, Y_i) by

$$X_i \sim U[-1, 1]^{20}, \quad W_i \sim \mathcal{N}(0, 0.1), \quad Y_i = \frac{1}{4}(X_i^{(1)} + 3X_i^{(2)} - X_i^{(3)}). \tag{3.1}$$

   This is a linear model with a 20-dimensional input with additive Gaussian noise, but only 3 of these are used, namely $X_i^{(1)}, X_i^{(2)}, X_i^{(3)}$. With this data, we are effectively testing a DR's ability to select relevant variables in the input data.

2. We define each of the samples of this model (X_i, Y_i) by

$$(A_i, B_i) \sim U[-1, 1]^2, \quad W_i \sim \mathcal{N}(0, 0.1),$$

$$X_i^{(1,\ldots,10)} = A_i + \mathcal{N}(0, 0.01), \quad X_i^{(11,\ldots,20)} = B_i + \mathcal{N}(0, 0.01),$$

and finally,

$$Y_i = \sum_{j=1}^{20} X_i^{(j)}. \tag{3.2}$$

This is a linear model with a 20-dimensional input, but there is a high degree of collinearity in the data, i.e. the components $X_i^{(1,\ldots,10)}$ are highly correlated and so are $X_i^{(11,\ldots,20)}$. This means that the effective dimensionality of our data is actually two, and we should expect a DR method to pick up on this.

3. We define each of the samples of this model (X_i, Y_i) by

$$X_i \sim U[-1, 1]^{20}, \quad W_i \sim \mathcal{N}(0, 0.1), \quad Z_i = X_i^{(1)} + 0.5X_i^{(2)},$$

$$Y_i = Z_i \cos(Z_i) + W_i. \tag{3.3}$$

This is a non-linear model with additive Gaussian noise that effectively depends only on the first two components of X. Note that this is a very similar model to model (A) used in section 3.1 in [5].

4. We define each of the samples of this model (X_i, Y_i) by

$$X_i \sim U[-1, 1]^{50}, \quad W_i \sim Beta(2, 4), \quad Z_i = ((X_i^{(1)})^2 + X_i^{(5)})(((X_i^{(2)})^2 - X_i^{(4)})),$$

$$Y_i = W_i Z_i^2. \tag{3.4}$$

This is a non-linear model with multiplicative noise drawn from a Beta distribution, hence the noise will be skewed. The effective dimensionality of the model is 4, since we are only using the first, second, fourth and fifth components of our input X. Here, we are testing DR models with high-dimension data, as well as the performance in a non-linear setting with non-Gaussian noise.

N.B: the Python implementation code to generate the data for each of the models can be found in the appendix:

	PLS	gKDR	HSIC
Model 1	0.008	0.006	0.333
Model 2	0.458	0.027	63.495
Model 3	0.102	0.087	0.333
Model 4	0.043	0.075	0.093

Table 3.1: MSE values for out-of-sample (test) data on cross-validated models. MSE was calculated on the predicitions of the kNN model after dimension reduction using 5-nearest neighbours.

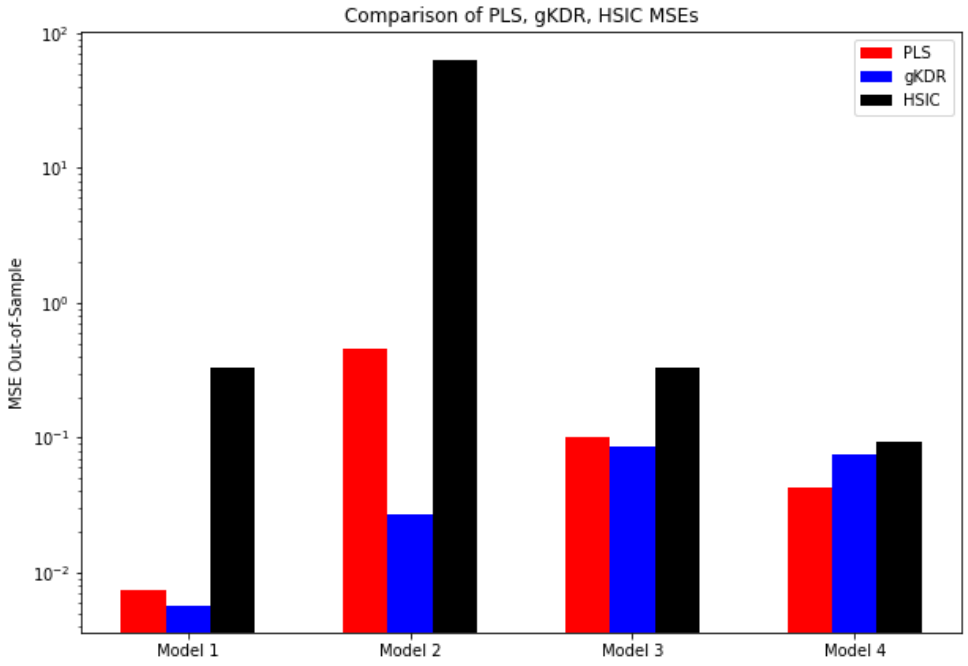

Figure 3.1: Bar chart showing the MSE (in log scale) comparisons for all of the PLS, gKDR and HSIC methods. Lower MSE means better performance here. We can see that the gKDR model performs the best by having the lowest MSE in Models 1, 2, 3. HSIC surprisingly performs consistently the worst. We will see however, that we get good results using HSIC in the next chapter.

As we can see from figure 3.1, except the very large MSE in the HSIC DR technique on the model 2 data, the overall MSEs are reasonably low, suggesting that all three of the DR techniques are performant.

To select the hyper-parameters of the DR methods, we used the following procedure:

First, we split the data into training and test sets by first randomly shuffling the data and then taking the first 75 percent of the data as the training set and the remaining 25 percent as the test set. We do 5-fold cross-validation on the training set by splitting it further into sub-training and validation sets 5 times. By computing the MSE on the validation sets and averaging, we chose the hyper-parameters that gave the lowest in-sample MSE.

We can then use this cross-validated model for testing on the test dataset and report the out-of-sample MSE which are the values reported in table 3.1 as well as figure 3.1.

Chapter 4

Anomaly Detection

In this chapter, we will be looking at using DR techniques for the purpose of anomaly detection, also known as outlier detection.

First, we must define what we mean by an anomaly in our data. The following definition is used "An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism"[22]. The interpretation of an anomaly is very different and it depends on the context.

The process of finding anomalies in a dataset is known as anomaly detection and there are many methods for achieving this goal [23]. To illustrate how DR techniques can be used for anomaly detection, we give the following example:

Suppose that we have a machine running as part of an industrial process. We wish to monitor the machine's operation by installing many sensors on it, each of which capturing information about certain attributes of the machine. Now, suppose we have built a statistical model for detecting anomalies which treats the information coming out of each sensor as one of its input variables, measured at short, regular time intervals. The goal here is to detect any deviations from the normal operation of the machine.

We can reasonably assume that this model is complex and therefore computationally expensive when using all of the input variables. To make the detection model faster, DR techniques can be used as a preprocessing step to reduce the number of inputs (thus dimensionality) of the data as far as possible whilst keeping as much information as possible.

For this chapter, we will be using a k-nearest neighbours approach for detecting anomalies in the low-dimensional space [23]. Specifically, in the low-dimensional space, for each point, we will be calculating the average distance to the nearest x neighbours. We will then look at the distribution of the average distances and take points as being anomalous if their average distance to the nearest x neighbours is greater than a certain threshold. The advantage of using this approach is that it is cheap to use and it is visually easy to interpret, especially if the low-dimensional space is of two or three dimensions, as in most cases, we will see most points clustered together and we will see anomalous points outside of these clusters.

We will be using the PLS method for DR as it is fast to compute for large datasets and we will aim to project our high-dimensional data to two to three dimensions wherever possible to be able to visualise the anomalous points easily. We first begin by using synthetic data, where we label the anomalous points ourselves and then proceed to look at some real data examples.

4.1 Synthetic Data

The goal here is to illustrate how we perform anomaly detection via dimension reduction for a very simple case. We will be using points from two different distributions, labeled normal and anomalous accordingly. Each of these points will then be passed through a function f.

For the normal points, we will be taking 20-dimensional vectors $X_{1,i} \in \mathbb{R}^{20}$, with normally distributed components, i.e. $X_{1,i}^{(j)} \sim \mathcal{N}(0,1)$ [24]. For the anomalous points, we will we taking 20-dimensional vectors $X_{2,i} \in \mathbb{R}^{20}$, but this time, the first two components are distributed as $X_{2,i}^{(1)}, X_{2,i}^{(2)} \sim \mathcal{N}(\mu, 0.1)$ while the other components are standard normals $\mathcal{N}(0,1)$. We will then compute the norm of all of the vectors, in other words, we let the response variables $Y_i := f(X_{\cdot,i}) = \|X_{\cdot,i}\| = \sqrt{(X_{\cdot,i}^{(1)})^2 + \ldots + (X_{\cdot,i}^{(20)})^2}$.

The experiment is therefore as follows. For each value of μ, perform 100 iterations of the following steps:

- Generate data as detailed above, specifically, 1000 normal data points of type $X_{1,i}$ and 40 anomalous datapoints of type $X_{2,i}$. Compute $f(X)$ on every data point.
- Split the data randomly into 80% training and 20% test sets, making sure to include 800 normal datapoints and 32 anomalous datapoints in the training set and 200 normal data points and 8 anomalous datapoints in the test set.
- Train the PLS algorithm for DR on the training set and transform the test set to a low-dimensional representation.
- Compute the average distance to the nearest 10 neighbours for every point in the low-dimensional representation of the test set using a kNN algorithm.
- Compute the mean and the standard deviation of the average distances.
- Label any points that are outside 3 standard deviations as anomalous and compute the accuracy, defined here by the fraction of correctly classified points as anomalous.

The experiments done with PLS using 2 and 3 components, and plot the plots of the accuracy attained. We should expect the accuracy to increase with the value of μ, since increasing μ makes the anomalous points more detectable. For small μ, the $X_{2,i}$ will be within the cluser of the $X_{2,i}$, so we expect to not detect anomalies when μ is small.

The results of the experiment is shown in figure 4.1. According this figure, the PLS with 3 components yields a higher accuracy, hence for this data, it will be beneficial to choose PLS with 3 components as the DR method.

Figure 4.2 illustrates the low-dimensional data after projection to the low-dimensional space, taken when the value of $\mu = 10$, making the anomalous point stand out. As we can see, the PLS algorithm has picked up on this. If we look at the components separately, we can see that the first component contains the information on whether the points are anomalous or not, since if we were to project the two clusters onto the axis, we would still see two clearly defined clusters on the component 1-axis.

Of course, in real data sets, things will not be as nice. In most practical cases, we don't know which points are actually outliers, so computing accuracy plots is not usually possible without making some assumptions about the data first.

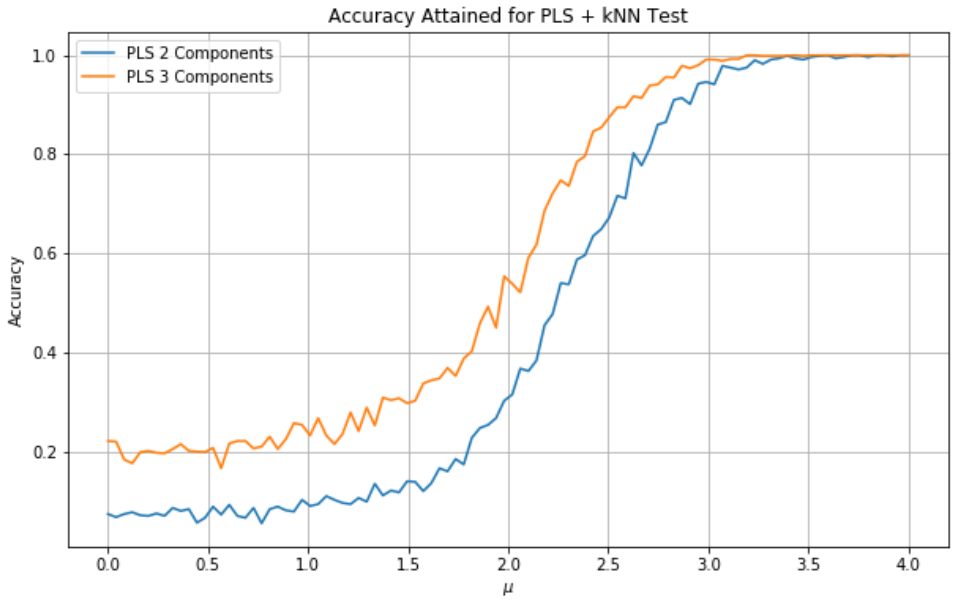

Figure 4.1: Accuracy curves attained after testing with partial least squares using two and three components. As we can see, the orange line corresponding to three components has a higher accuracy in detecting anomalies.

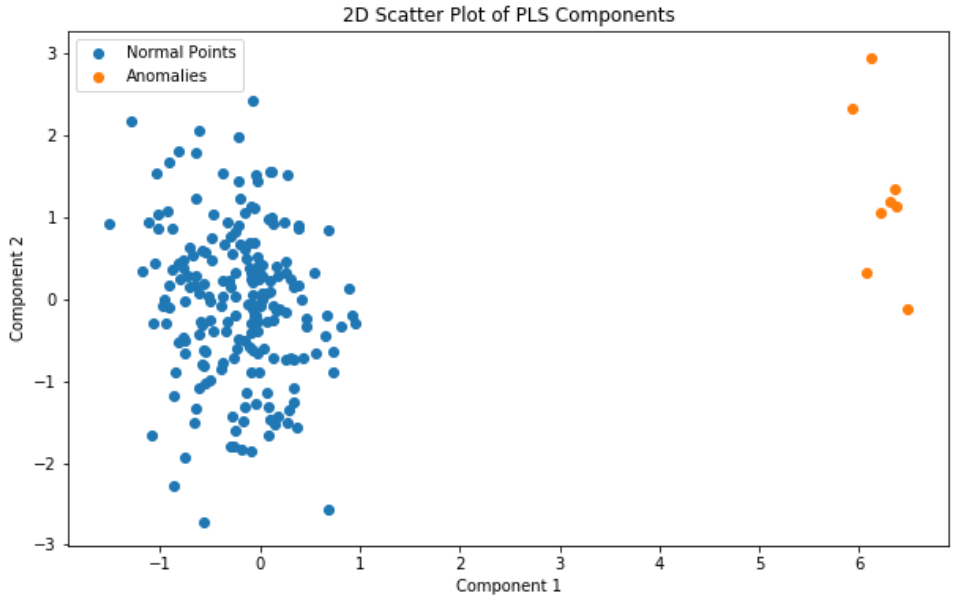

Figure 4.2: Scatter plot of the PLS low-dimensional representation of our X data when using a high value of $\mu = 10$. The 8 points on the RHS are anomalous, and these have been labeled as such by the algorithms.

4.2 Twitter Data Set

In this section, we use a dataset composed of social analytics data taken from the GitHub repository NAB associated with the paper [3]. Specifically, the data is composed of number of mentions for specific publicly traded companies, measured every 5 minutes over a span of approximately two months (26th February 2015 to 23rd April 2015).

The raw dataset is composed of 15902 observations on 10 variables corresponding to data of 10 publicly traded different companies, and each are indexed by the tickers (industry):

- AAPL: Apple (Technology)
- AMZN: Amazon (Technology)
- CRM: Salesforce (Technology)
- CVS: CVS Health (Healthcare)
- FB: Facebook (Technology/Social Networks)
- GOOG: Google (Technology)
- IBM: International Business Machines Corporation - IBM (Technology)
- KO: Coca-Cola (Beverages)
- PFE: Pfizer (Pharmaceuticals)
- UPS: United Parcel Service - UPS (Logistics)

As we can see, there are many companies in the technology sector, so we should expect some correlation between the volume of tweets mentioning these companies.

We will define $Volume(ticker, t)$ to be the number of mentions at time t for the company name associated with ticker, and $Volume(t-k)$ to be the number of mentions at time $t-k(5mins)$. Since we are dealing with time series data, we will augment our 10-dimensional data into a 100-dimensional data by monitoring the values $Volume(ticker, t-k)$ for $k = 1, \ldots, 10$ for every ticker. For this project, we will be taking $Volume(AAPL, t)$ as our response variable for anomaly detection. In other words, we are attempting to monitor for anomalies in the current volume of Apple tweets given data on all the volumes of every company for the past 10 time-steps (55 minutes).

By using supervised dimension reduction, we will be attempting to find a functional relationship between our 100 input variables and the response variables and identify a low-dimensional subspace in three dimensions. This will then enable us to do anomaly detection using the kNN approach discussed in section 4.1 as well as enable easy visualisation of the low-dimensional subspace.

Remarks:

- Special care was taken when cleaning the data. We had to get rid of rows with missing data and make sure that all of the observation times were aligned in our dataset.
- When constructing our input variables, care was taken to make sure we didn't introduce artifial missing values in the process of staggering the time series.
- After data cleaning, the full model is $X \in \mathbb{R}^{nxd}, Y \in \mathbb{R}^n$, where $n = 15821, d = 100$, i.e. 100-dimensional on 15821 observations.

Experiment:

We will be performing the same experiment for the PLS, gKDR and HSIC methods, inspect the low-dimensional space as well as the anomalies that have been detected by each of the methods.

It is worth noting that whilst PLS is a very fast method, gKDR and HSIC are very slow in comparison, since gKDR requires the inversion of the gram matrix, in which case it will scale as $\mathcal{O}(n^3)$, where n is the number of observations and HSIC requires optimisation over the Stiefel manifold which is also scales worse than linearly with n. Memory considerations are also problematic. During experiments, we tried running the gKDR and HSIC methods on the full dataset but the computer ran out of memory very fast, 16GB of RAM. This is because the computations involving Gram matrices require vasts amount of memory to complete.

To solve the problem of memory space, we have trained both gKDR and HSIC on a much smaller subset of our data. In this case, we chose the first 300 points only to keep the computational time reasonable. That is, we attempt to discover the central subspace of the data using the first 300 points only and never considering the rest. This is a small portion of the dataset $300/15821 \approx 2\%$ of the dataset, leading us to assume that their performance may be compromised by this. However, we will see that even with this little data used for training, both gKDR and HSIC will give us very comparable results to PLS trained on the whole dataset. We can then say that gKDR and HSIC are very good at learning the central subspace even from small amounts of data.

Results:

The figures seen on the next three pages show the resulting low-dimensional 3D space for the three methods as well as the dates that have been labeled as anomalous too. It is very encouraging to see that all of the three methods have found a similar central subspaces to the data; one big cluster of normal points close to the origin and various anomalous points away from the cluster codified by 1 or 2 of the component directions found. These yield very similar labeling of anomalies as seen in figures 4.4, 4.5, 4.6. In fact, after counting the number of anomalies, we find that the PLS method found 99 anomalies, the gKDR method found 109 anomalies and the HSIC method found 101 anomalies. As the results are very similar for all of the three methods, and we are using the PLS as a benchmark algorithm, we can say with some confidence that the HSIC method does dimension reduction well, even with much less data than PLS.

Since we have the anomalous dates available to us, the curious reader might find interesting to know that the date with the highest volume of Apple tweets was the 27th of February 2015, and after some searching, we can find that this was the date another company sued Apple over patent rights, causing a flurry of tweets mentioning Apple on that day.

The upshot of this experiment is that we have verified the dimensionality reduction capabilities of maximising HSIC through manifold learning. Furthermore, we have shown that we need only use small amounts of data to find the central subspace. This is extremely encouraging, as using this method as part of a larger framework can save time and running costs.

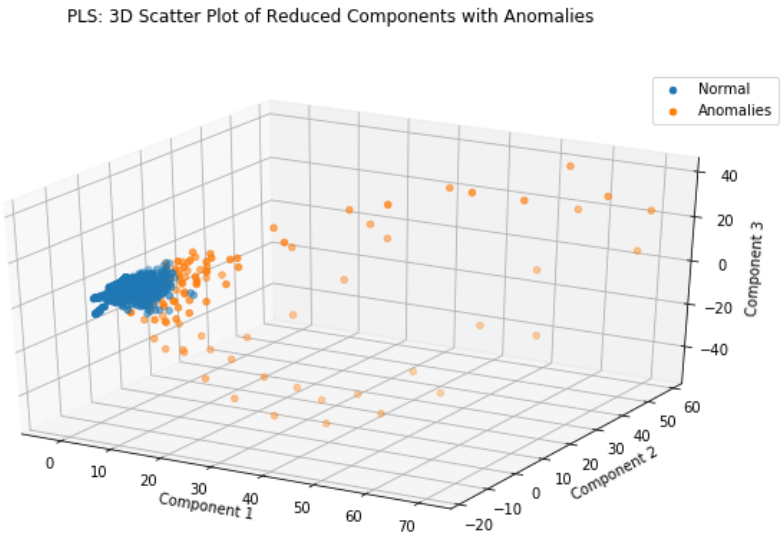

Figure 4.3: Low-dimensional space after PLS dimension reduction. As we can see here, there is a cluster close to the origin of the axes, which the kNN algorithm has labeled as normal points. We can also see many anomalous points departing away from the cluster in two branches, one being negative in the third component and the other one being positive.

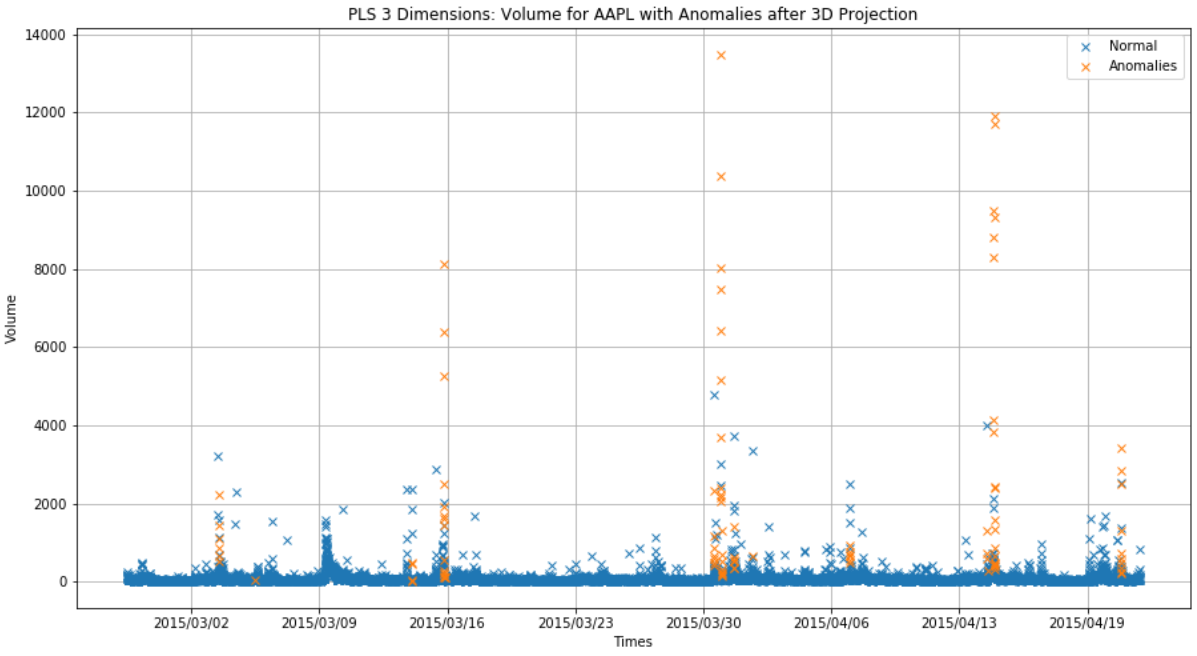

Figure 4.4: Volume against time for Apple mentions with the anomalous points found in figure 4.3 shown using PLS. The algorithm has labeled days with unusually high volume as anomalous.

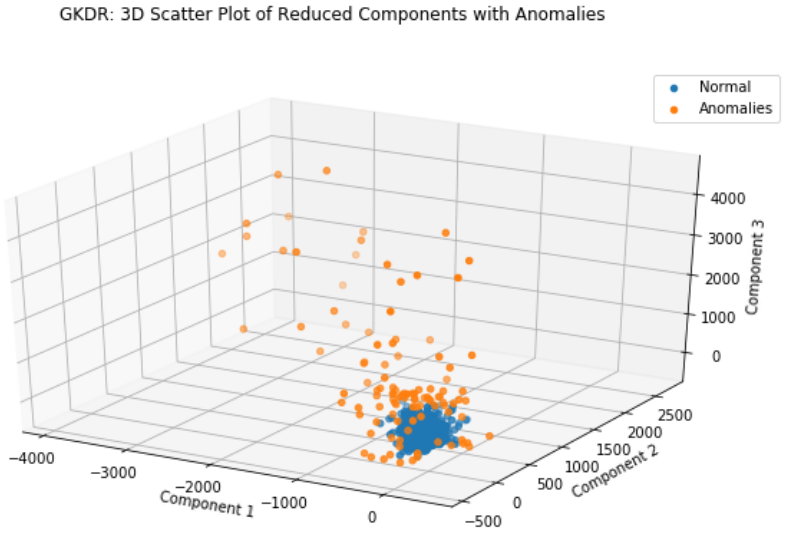

Figure 4.5: Low-dimensional space after gKDR dimension reduction. As we can see here, there is a cluster close to the origin of the axes, which the kNN algorithm has labeled as normal points and anomalous points are the ones further away from the cluster.

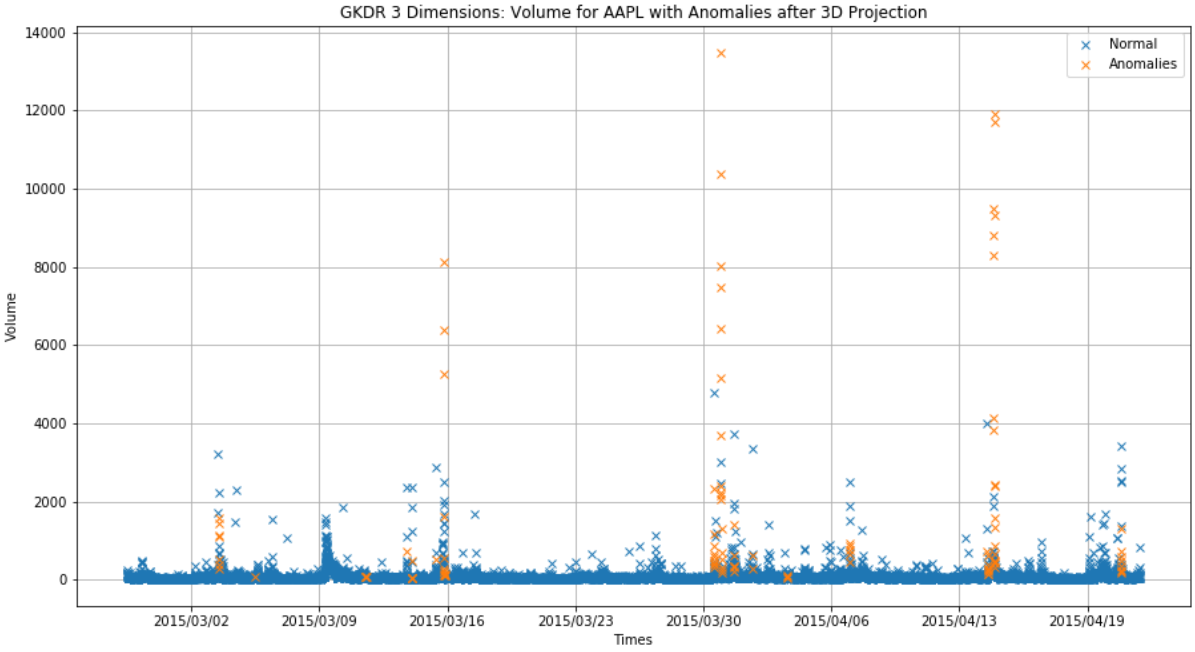

Figure 4.6: Volume against time for Apple mentions with the anomalous points found in figure 4.5 shown using gKDR. Again, the algorithm has labeled days with unusually high volume as anomalous.

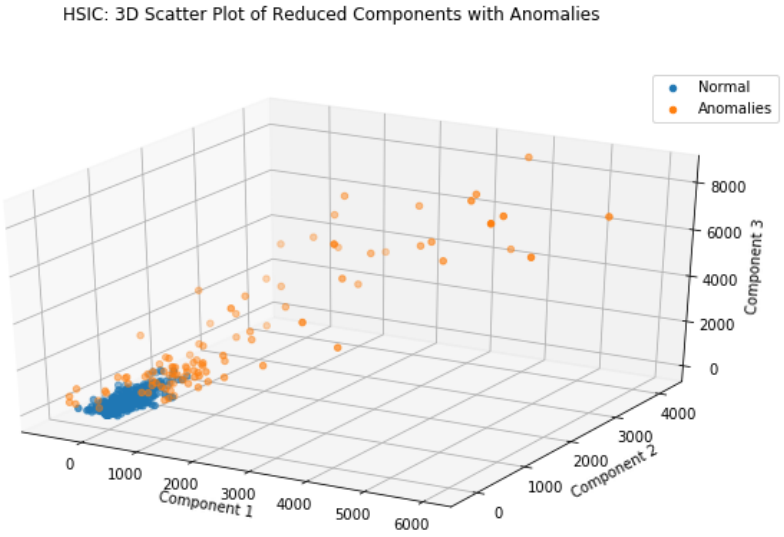

Figure 4.7: Low-dimensional space after HSIC dimension reduction. As we can see here, there is a cluster close to the origin of the axes, which the kNN algorithm has labeled as normal points and anomalous points are the ones further away from the cluster.

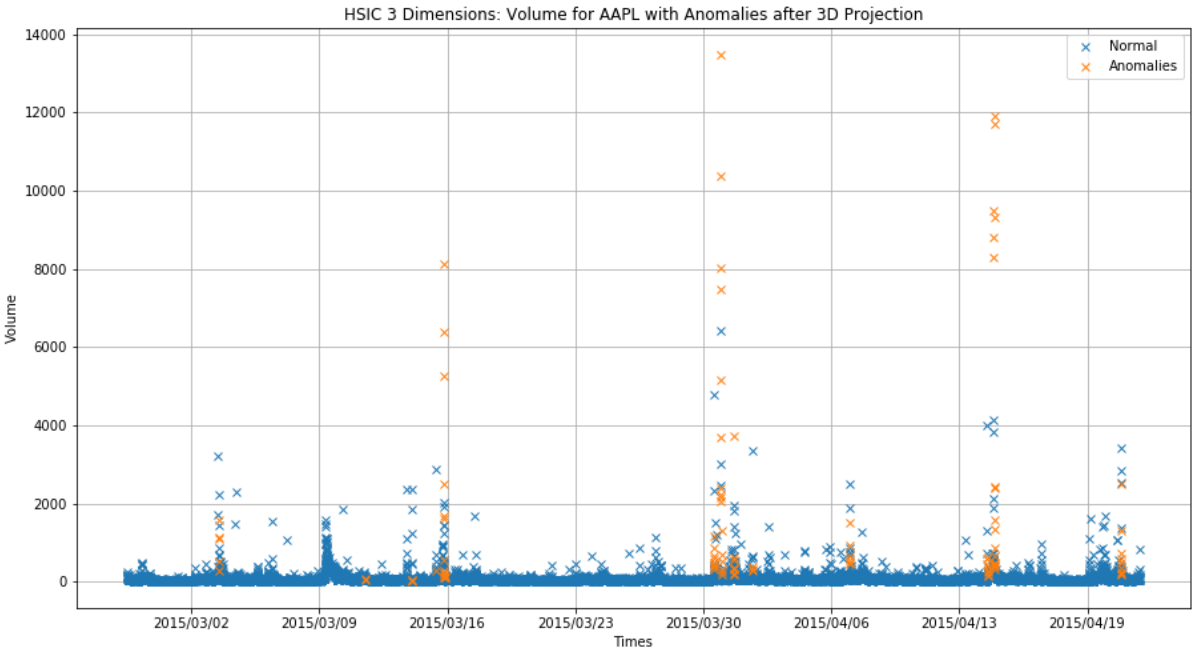

Figure 4.8: Volume against time for Apple mentions with the anomalous points found in figure 4.7 shown using gKDR. Just like with PLS and gKDR algorithm has labeled days with unusually high volume as anomalous.

Chapter 5

Summary of Results and Future Work

We briefly conclude this project by talking about the work accomplished as well as ideas for future research on this topic.

The central objective of this project has been to demonstrate the dimension reduction capabilities of the HSIC method described in chapter 2. Through the experiments carried out in chapters 3 and 4, we can conclude that the HSIC yields promising results for both regression and anomaly detection applications on synthetic and real data.

Unfortunately, we weren't able to find a suitable novel example of a functional dataset to test active subspaces on. As mentioned before, not every system contains an active subspace and the task of finding such systems which do contain active subspaces is an active research subject.

Ideas for further research:

- Exploring performance of HSIC with other kernel functions.
- Implementation of a different gradient descent method that is less prone to local minima for the optimisation of the objective function in the HSIC method.
- Further investigation of HSIC on collinear data similar to model 2 of the synthetic data in chapter 3.
- Comparison of the HSIC method with other independence criteria for DR.
- Performance of the HSIC when used together with a classification algorithm.

Appendices

Appendix A

Computer Code

A.1 Implementation of gKDR Method

In this section, we present a Python implementation of the gKDR method for the polynomial, gaussian radial basis function and sigmoid kernels. The MATLAB code provided by [5] was used as a basis for the gKDR function below as well as computing the gaussian RBF kernel functions.

The following code listing contains a general gKDR function that accepts different types of kernels. The function consists of three main parts. Lines 18 to 38 call other functions to compute the gram matrices G_X and G_Y using different kernels as well as the quantity $\nabla \mathbf{k}_X(\cdot)$ for the input data. Lines 40 to 49 handle the computation of the inverses of the regularised gram matrix $(G_X + n\varepsilon_n I)^{-1}$ and the construction of the estimator $\tilde{M}_n$. Finally, lines 51 to 56 compute the eigenvalues and corresponding eigenvectors of $\tilde{M}_n$ and sets the output matrix B.

```
1  def gKDR(X, Y, d, eps, kernel_type, params):
2      """ Computes gKDR estimates of effective dimension reduction subpsace.
3
4      Parameters
5      ----------
6      X : np.ndarray | Explanatory variables
7      Y : np.ndarray |  Response variables
8      d : int | Dimension of effective subspaces
9      eps : float | Regularization coefficient
10     kernel_type : str | Type of kernel to use
11     params : tuple | Tuple containing hyperparameters
12
13     Returns
14     -------
15     B:  orthonormal column vectors (M x d)
16     t:  value of the objective function
17     """
18     N, M = X.shape  # Dimensions of our data
19     I = np.eye(N)
20
21     # Computing Gram Matrices and kernel derivatives:
22     if kernel_type == "Polynomial":
23         Kx = gram_matrix_polynomial(X, params)
24         Ky = gram_matrix_polynomial(Y, params)
25
26         gradK = polynomial_kernel_deriv(X, params)
27
28     elif kernel_type == "Gaussian_RBF":
```

```
        Kx = gram_matrix_gaussian_RBF(X, params)
        Ky = gram_matrix_gaussian_RBF(Y, params)

        gradK = gaussian_RBF_kernel_deriv(X, params)

    elif kernel_type == "Sigmoid":
        Kx = gram_matrix_sigmoid(X, params)
        Ky = gram_matrix_sigmoid(Y, params)

        gradK = sigmoid_kernel_deriv(X, params)

    # Computing estimator of M_n:
    a = np.linalg.lstsq(Kx + N*EPS*I, Ky, rcond=None)[0]
    b = Kx + N*EPS*I
    F = np.linalg.lstsq(b.T, a.T, rcond=None)[0]
    Hm = H.reshape(N, N*M, order='F').copy()
    HH = (Hm.T @ Hm).reshape(N, M, N, M, order='F').copy()
    HHm = np.transpose(HH, (0, 2, 1, 3)).reshape(N*N, M, M, order='F').copy
    ()
    Freshaped = F.reshape(N*N, 1, order='F').copy()
    Fm = np.tile(Freshaped[:, None], [1, M, M])
    R = np.sum(HHm*Fm, axis=0).reshape(M, M, order='F').copy()

    # Computing eigenvalues and eigenvectors of estimator of M_n:
    L, V = np.linalg.eig(R)
    idx = L.argsort()[::-1][0:K]
    L = L[idx]
    B = V[:, idx]  # Setting final projection matrix B
    t = np.sum(L)
    return B, t

```

Listing A.1: gKDR Function

```
def gram_matrix_polynomial(Z, params):
    """Computes gram matrix using gaussian polynomial kernel function.

    Parameters
    ----------
    Z : np.ndarray | Input data
    params : tuple | Contains hyperparameters

    Returns
    -------
    Kx : np.ndarray | Gram matrix
    """
    N, M = X.shape  # Dimensions of our data
    c, d = params
    return (X.T @ X + c) ** d

def polynomial_kernel_deriv(X, params):
    """Computes derivative of kernel function for polynomial kernel.
    """
    N, M = X.shape
    c, d = params
    return d * X.T @ ((X.T @ X + c) ** (d-1))

```

Listing A.2: Polynomial kernel helper functions

```
def gram_matrix_gaussian_RBF(Z, params):
    """Computes gram matrix using Gaussian RBF kernel function.

    Parameters
```

```
    ----------
    Z : np.ndarray | Input data
    params : tuple | Contains hyperparameters

    Returns
    -------
    Kx : np.ndarray | Gram matrix
    """
    N, M = X.shape  # Dimensions of our data
    sgx, sgy = params
    sx2 = 2 * (SGX**2)

    ab = Z @ Z.T
    aa = np.diag(ab) if type(ab) == np.ndarray else ab
    D = np.tile(aa, [N, 1])
    xx = np.maximum(D + D.T - 2*ab, np.zeros((N, N)))
    Kx = np.exp(-xx/sx2)
    return

def gaussian_RBF_kernel_deriv(X, params):
    """Computes derivative of kernel function for Gaussian RBF kernel.
    """
    N, M = X.shape  # Dimensions of our data
    sgx, sgy = params

    rep_mat = np.tile(X, [N, 1])
    Dx = rep_mat.reshape(N, N, M, order='F').copy()
    Xij = Dx - np.transpose(Dx, (1,0,2))
    Xij = Xij/(sgx**2)
    H = Xij*np.tile(Kx, [M, 1, 1]).T
    return

```

Listing A.3: Gaussian RBF kernel helper functions

```
def gram_matrix_sigmoid(Z, params):
    """Computes gram matrix using gaussian sigmoid kernel function.

    Parameters
    ----------
    Z : np.ndarray | Input data
    params : tuple | Contains hyperparameters

    Returns
    -------
    Kx : np.ndarray | Gram matrix
    """
    N, M = X.shape  # Dimensions of our data
    a, b = params
    return np.tanh(a*X.T @ X + b)

def sigmoid_kernel_deriv():
    """Computes derivative of kernel function for sigmoid kernel.
    """
    N, M = X.shape  # Dimensions of our data
    a, b = params
    return a * X.T @ (1 - np.tanh(a*X.T @ X + b)**2)

```

Listing A.4: Sigmoid kernel helper functions

A.2 Implementation of HSIC Method

The code below, is the implementation for DR using HSIC.

- The first function `median_distance` computes the median distance of the rows of a vector, and is used to calculate the optimal value of σ for the Gaussian RBF.
- The second function `gaussian_rbf_gram` computes the Gaussian RBF Gram matrix.
- The third function `HSIC_gRBF` calculates the trace component of HSIC as described in section 2.3.
- The last function `hsic_cost_function` is used to compute the objective function as detailed in section 2.3.

```
# HSIC CODE:
import autograd.numpy
from pymanopt.manifolds import Stiefel
from pymanopt import Problem
from pymanopt.solvers import SteepestDescent

from scipy.spatial.distance import pdist

def median_distance(X):
    return np.mean(pdist(X))


def gaussian_rbf_gram(MAT1, MAT2, sigma_val):
    """ Computes Gaussian RBF Gram matrix kernel using autograd.numpy:

            k(X, Y) = - exp[||X - Y||^2 / (2sigma^2)]

    REFERENCE: https://github.com/amber0309/HSIC

    Parameters
    ----------
    MAT1 : np.ndarray | Matrix X in the formula.
    MAT2 : np.ndarray | Matrix Y in the formula.
    sigma_val : float | Value for regularising coeff.
    Returns
    -------
    np.ndarray | Gram matrix
        Description of returned object.
    """
    G = autograd.numpy.sum(MAT1*MAT1, 1).reshape(MAT1.shape[0],1)
    H = autograd.numpy.sum(MAT2*MAT2, 1).reshape(MAT2.shape[0],1)

    Q = autograd.numpy.tile(G, (1, MAT2.shape[0]))
    R = autograd.numpy.tile(H.T, (MAT1.shape[0], 1))

    H = Q + R - 2* autograd.numpy.dot(MAT1, MAT2.T)
    return autograd.numpy.exp(-H/2/(sigma_val**2))


def HSIC_gRBF(XMAT, YMAT, sgx, sgy):
    """ Computes the Hilbert-Schmidt Independence Criterion between
    matrices
        X and Y without normalising constant, i.e. just the trace trace(Gx
    H Gy H)

    Parameters
    ----------
    XMAT : np.ndarray | Matrix X in the formula.
    YMAT : np.ndarray | Matrix Y in the formula.
```

```
    sgx : float | sigma_x parameter in gaussian RBF kernel
    sgy : float | sigma_y parameter in gaussian RBF kernel

    Returns
    -------
    float | value of HSIC calculated
    """
    N, p = XMAT.shape
    HMAT = autograd.numpy.eye(N) - autograd.numpy.ones((N, N))

    Gx = gaussian_rbf_gram(XMAT, XMAT, sgx)

    if len(YMAT.shape) == 2:
        Gy = gaussian_rbf_gram(YMAT, YMAT, sgy)
    elif len(YMAT.shape) == 1:  # Just a vector, convert to column matrix
        Gy = gaussian_rbf_gram(YMAT[:, None], YMAT[:, None], sgy)
    else:
        raise Exception("Error with Y Dimensions")
    return autograd.numpy.trace(Gx @ HMAT @ Gy @ HMAT)


def hsic_cost_function(P):
    """ Helper function for Pymanopt optimisation."""
    global XHSIC, YHSIC, sgx, sgy
    return - HSIC_gRBF(XHSIC @ P, YHSIC, sgx, sgy)

```

Listing A.5: HSIC Implementation Gaussian RBF

To use this code, we will need to use a version of the following code listing:

```
# HSIC Code Usage
# Declare global variables:
global XHSIC, YHSIC, sgx, sgy

# Set current data e.g. X, Y
XHSIC = autograd.numpy.array(X.copy())
YHSIC = autograd.numpy.array(Y.copy())

# Compute sgx, sgy:
sgx = median_distance(XHSIC)
sgy = median_distance(YHSIC[:, None])  # If Y is a matrix, remove [:, None]

# Using pymanopt:
manifold = Stiefel(X_dims, target_dims)  # Manifold B from R^X_dims to R^
    target_dims
problem = Problem(manifold=manifold, cost=hsic_cost_function)
solver = SteepestDescent()
Popt = solver.solve(problem)

# To transform any data, use:
Xtransformed = X @ Popt

```

Listing A.6: HSIC Usage using pymanopt

The code above is a skeleton code for performing dimensionality reduction using HSIC as described in chapter 3. Cross-validation on the parameters sgx, sgy and the target dimensionality should always be performed whenever possible.

A.3 Generation of data (section 3.1)

```
def synthetic_data_generator(N, model_number):
    """Generates data as explained in section 3.1 of the project.

    Parameters
    ----------
    N : int | Number of samples to generate
    model_number : int | Model to follow

    Returns
    -------
    X : np.array | Array containing input variables
    Y : np.array | Array containing response variables
    """

    if model_number == 1:
        X = np.random.uniform(low=-1.0, high=1.0, size=(N, 20))
        W = np.random.normal(loc=0.0, scale=np.sqrt(0.1), size=N)
        Y = 0.25 * (X[:, 0] + 3*X[:, 1] - X[:, 2])

    elif model_number == 2:
        A = np.random.uniform(low=-1.0, high=1.0, size=N)
        B = np.random.uniform(low=-1.0, high=1.0, size=N)
        W = np.random.normal(loc=0.0, scale=np.sqrt(0.1), size=N)

        X1 = A[:, None] + np.random.normal(
            loc=0.0, scale=np.sqrt(0.01), size=(N, 10))
        X2 = B[:, None] + np.random.normal(
            loc=0.0, scale=np.sqrt(0.01), size=(N, 10))
        X = np.concatenate([X1, X2], axis=1)
        Y = np.sum(X, axis=0)

    elif model_number == 3:
        X = np.random.uniform(low=-1.0, high=1.0, size=(N, 20))
        W = np.random.normal(loc=0.0, scale=np.sqrt(0.1), size=N)
        Z = X[:, 0] + 0.5*X[:, 1]
        Y = Z * np.cos(Z) + W

    elif model_number == 4:
        X = np.random.uniform(low=-1.0, high=1.0, size=(N, 50))
        W = np.random.beta(2, 4, size=N)
        Z = (X[:, 0]**2 + X[:, 4])*(X[:, 1]**2 - X[:, 3])
        Y = W * Z**2
    return X, Y

```

Listing A.7: Function which generates synthetic data according to the models described in 3.1

A.4 Helper functions used for Cross-Validation in Regression:

```
def PLS_cross_validation(X, Y, dim_space):
""" Performs cross-validation to find optimal dimension on
    data (X, Y) through the values of the array dim_space.

Parameters
----------
X : np.array | Array containing input variables
Y : np.array | Array containing response variables
dim_space : np.array | Array of integers containing dims.

Returns
```

```
--------
optimal_dim : int | Value for the optimal dimension found.
MSE_min_insample : float | Value for in-sample MSE
MSE_out_sample : float | Value for out-of-sample MSE
"""
# First normalise the data:
Xscaled = sklearn.preprocessing.StandardScaler().fit_transform(X)

# Splitting data into training and test sets.
# This is to compute in-sample and out-of-sample errors.
Xtrain, Xtest, Ytrain, Ytest = sklearn.model_selection.train_test_split(
    Xscaled, Y, test_size=0.25, shuffle=True)

# Cross-Validation:
kfold_maker = sklearn.model_selection.KFold(n_splits=5, shuffle=True)

# Initialising MSE array for choosing optimal dimension:
mse_array = np.zeros(dim_space.shape)

for idx, d in enumerate(dim_space):
    kfold_split = kfold_maker.split(Xtrain, Ytrain)
    for k, (train_id, val_id) in enumerate(kfold_split):
        Xtrain_insample = Xtrain[train_id]
        Ytrain_insample = Ytrain[train_id]

        Xval_insample = Xtrain[val_id]
        Yval_insample = Ytrain[val_id]

        # Fit Partial Least Squares:
        PLS = sklearn.cross_decomposition.PLSRegression(n_components=d)
        PLS.fit(Xtrain_insample, Ytrain_insample)
        Xtrain_insample_pls = PLS.transform(Xtrain_insample)

        # Train simple kNN model:
        kNN_5 = sklearn.neighbors.KNeighborsRegressor(n_neighbors=5)
        kNN_5.fit(Xtrain_insample_pls, Ytrain_insample)

        # Transform validation data using Partial Least Squares:
        Xval_insample_pls = PLS.transform(Xval_insample)

        # Prediction using kNN_5:
        Ypreds = kNN_5.predict(Xval_insample_pls)

        mean_square_error = sklearn.metrics.mean_squared_error(
    Yval_insample, Ypreds)

        mse_array[idx] += mean_square_error

# Finding minimum error:
mse_array /= 5
optimal_dim = np.unravel_index(np.argmin(mse_array, axis=None), mse_array.
    shape)[0] + 1
MSE_min_insample = mse_array[optimal_dim - 1]

# Computing out-of-sample errors:
PLS = sklearn.cross_decomposition.PLSRegression(n_components=optimal_dim)
PLS.fit(Xtrain, Ytrain)
Xtrain_PLS = PLS.transform(Xtrain)

# Train kNN on full training data:
kNN_5 = sklearn.neighbors.KNeighborsRegressor(n_neighbors=5)
kNN_5.fit(Xtrain_PLS, Ytrain)

```

```
# Transform test data using PLS:
Xtest_PLS = PLS.transform(Xtest)

# Prediction using kNN_5:
Ypreds = kNN_5.predict(Xtest_PLS)

MSE_out_sample = sklearn.metrics.mean_squared_error(Ytest, Ypreds)
return optimal_dim, MSE_min_insample, MSE_out_sample

```

Listing A.8: Cross-validation for Partial Least Squares.

```
def gKDR_cross_validation(X, Y, param_space, VERBOSE=True):
    """ Performs K-Fold cross validation on data using gKDR and kNN approach.
    Parameters
    ----------
    X : np.array | Array containing input variables
    Y : np.array | Array containing response variables
    param_space : dict | Dictionary containing parameter space to perform Grid-Search
    target_dimension : int | Final dimensionality to use

    Returns
    -------
    optimal_dimension : int | Value for the optimal dimension found.
    SGX_optim : float | Value for the optimal gamma for the X-kernel
    SGY_optim : float | Value for the optimal gamma for the Y-kernel
    MSE_min_insample : float | Value for in-sample MSE
    MSE_out_sample : float | Value for out-of-sample MSE
    """
    # First normalise the data:
    Xscaled = sklearn.preprocessing.StandardScaler().fit_transform(X)

    # Splitting data into training and test sets.
    # This is to compute in-sample and out-of-sample errors.
    Xtrain, Xtest, Ytrain, Ytest = sklearn.model_selection.train_test_split(
        Xscaled, Y, test_size=0.25, shuffle=True)

    # Cross-Validation:
    kfold_maker = sklearn.model_selection.KFold(n_splits=5, shuffle=True)

    dims_array = param_space['dims']
    SGX_array = param_space["SGX"]
    SGY_array = param_space["SGY"]

    mse_array = np.zeros((dims_array.size, SGX_array.size, SGY_array.size), dtype=float)

    for a, dim in enumerate(dims_array):
        for b, SGX in enumerate(SGX_array):
            for c, SGY in enumerate(SGY_array):
                if VERBOSE:
                    print(f"dim={dim}, SGX={SGX}, SGY={SGY}")

                kfold_split = kfold_maker.split(Xtrain, Ytrain)
                for k, (train_id, val_id) in enumerate(kfold_split):
                    Xtrain_insample = Xtrain[train_id]
                    Ytrain_insample = Ytrain[train_id]

                    Xval_insample = Xtrain[val_id]
                    Yval_insample = Ytrain[val_id]

```

```
                    # Take EPS = 10^(-dim) as suggested in gKDR paper.
                    EPS = 10**(-float(dim))

                    params = (SGX, SGY)
                    B = gKDR.gKDR(Xtrain_insample, Ytrain_insample,
                                        dim, EPS, "Gaussian_RBF", (params))
[0].real
                    Xtrain_insample_gKDR = Xtrain_insample @ B  # Perform
dimensionality reduction

                    # Train simple kNN model:
                    kNN_5 = sklearn.neighbors.KNeighborsRegressor(
n_neighbors=5)
                    kNN_5.fit(Xtrain_insample_gKDR, Ytrain_insample)

                    # Transform validation data using gKDR:
                    Xval_insample_gKDR = Xval_insample @ B

                    # Prediction using kNN_5:
                    Ypreds = kNN_5.predict(Xval_insample_gKDR)

                    mean_square_error = sklearn.metrics.mean_squared_error(
Yval_insample, Ypreds)

                    mse_array[a, b, c] += mean_square_error
    # Averaging since we took errors 5 times due to 5-fold cross validation
    mse_array /= 5
    index_optim = np.unravel_index(np.argmin(mse_array, axis=None),
mse_array.shape)
    MSE_min_insample = mse_array[index_optim]
    optimal_dim = dims_array[index_optim[0]]
    SGX_optim = SGX_array[index_optim[1]]
    SGY_optim = SGY_array[index_optim[2]]


    # Test optimal model:
    EPS = 10**(-float(optimal_dim))
    B = gKDR.gKDR(Xtrain, Ytrain, optimal_dim, EPS, "Gaussian_RBF", (
SGX_optim, SGY_optim))[0].real
    Xtrain_gKDR = Xtrain @ B
    Xtest_gKDR = Xtest @ B

    kNN_5 = sklearn.neighbors.KNeighborsRegressor(n_neighbors=5)
    kNN_5.fit(Xtrain_gKDR, Ytrain)
    Ypreds = kNN_5.predict(Xtest_gKDR)
    MSE_out_sample = sklearn.metrics.mean_squared_error(Ytest, Ypreds)
    return optimal_dim, SGX_optim, SGY_optim, MSE_min_insample,
MSE_out_sample

    # gKDR with Gaussian RBF Kernel Dictionary of parameters:
    param_space = {
        'dims' : np.array([2, 3, 4], dtype=int),  # Dimension settings to
explore
        'SGX' : np.logspace(-4, 2, num=16),  # sigma for X gram matrix
        'SGY' : np.logspace(-4, 2, num=16),  # sigma for Y gram matrix
    }

```

Listing A.9: Cross-validation for gKDR RBF method.

Bibliography

[1] Avanidhar Subrahmanyam. Big data in finance: Evidence and challenges. *Borsa Istanbul Review*, 19(4):283–287, 2019.

[2] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. volume 3512, pages 758–770, 06 2005.

[3] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134 – 147, 2017. Online Real-Time Learning Strategies for Data Streams.

[4] Vincenzo Esposito Vinzi, Wynne W. Chin, Jrg Henseler, and Huiwen Wang. *Handbook of Partial Least Squares: Concepts, Methods and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[5] Kenji Fukumizu and Chenlei Leng. Gradient-based kernel dimension reduction for regression. *Journal of the American Statistical Association*, 109(505):359–370, 2014.

[6] Arthur Gretton, Kenji Fukumizu, Choon Hui Teo, Le Song, Bernhard Schölkopf, and Alexander J. Smola. A kernel statistical test of independence. *Advances in Neural Information Processing Systems 20 - Proceedings of the 2007 Conference*, pages 1–8, 2009.

[7] Paul G. Constantine, Eric Dow, and Qiqi Wang. Active subspace methods in theory and practice: Applications to kriging surfaces. *SIAM Journal on Scientific Computing*, 36(4):A1500–A1524, Jan 2014.

[8] Paul G. Constantine. *Active Subspaces*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2015.

[9] Kofi P. Adragni and R. Dennis Cook. Sufficient dimension reduction and prediction in regression. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1906):4385–4405, 2009.

[10] R.D. Cook. *Regression Graphics: Ideas for Studying Regressions Through Graphics*. Wiley Series in Probability and Statistics. Wiley, 1998.

[11] Francesca Chiaromonte and R. Cook. Sufficient dimension reduction and graphics in regression. *Annals of the Institute of Statistical Mathematics*, 54:768–795, 01 2002.

[12] Joseph Hair, G. Tomas M. Hult, Christian Ringle, and Marko Sarstedt. *A Primer on Partial Least Squares Structural Equation Modeling*. 01 2014.

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

[14] Kenji Fukumizu, Francis R. Bach, and Michael I. Jordan. Dimensionality reduction for supervised learning with reproducing kernel Hilbert spaces. *Journal of Machine Learning Research*, 5:73–99, 2004.

[15] Kenji Fukumizu, Francis R. Bach, and Michael I. Jordan. Kernel dimension reduction in regression. *Annals of Statistics*, 37(4):1871–1905, 2009.

[16] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

[17] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[18] Zhengming Ma, Zengrong Zhan, Xiaoyuan Ouyang, and Xue Su. Nonlinear dimensionality reduction based on HSIC maximization. *IEEE Access*, 6:55537–55555, 2018.

[19] Yin Zhang and Zhi Hua Zhou. Multi-Label dimensionality reduction via dependence maximization. *Proceedings of the National Conference on Artificial Intelligence*, 3:1503–1505, 2008.

[20] James Townsend, Niklas Koep, and Sebastian Weichwald. Pymanopt: A python toolbox for optimization on manifolds using automatic differentiation. *Journal of Machine Learning Research*, 17(137):1–5, 2016.

[21] Robert A. Bridges, Anthony D. Gruber, Christopher R. Felder, Miki E. Verma, and Chelsey Hoff. Active manifolds: A non-linear analogue to Active Subspaces. *36th International Conference on Machine Learning, ICML 2019*, 2019-June:1204–1212, 2019.

[22] D.M. Hawkins. *Identification of Outliers*. Monographs on applied probability and statistics. Chapman and Hall, 1980.

[23] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(2):427–438, 2000.

[24] Sevvandi Kandanaarachchi and Rob J Hyndman. Dimension reduction for outlier detection using DOBIN. (August), 2019.

[25] Chieh Wu, Jared Miller, Yale Chang, Mario Sznaier, and Jennifer Dy. Solving Interpretable Kernel Dimension Reduction. (NeurIPS):1–11, 2019.

[26] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009.

[27] Guoqing Chao, Yuan Luo, and Weiping Ding. Recent Advances in Supervised Dimension Reduction: A Survey. *Machine Learning and Knowledge Extraction*, 1(1):341–358, 2019.

[28] Jonathan H. Manton and Pierre-Olivier Amblard. A primer on reproducing kernel hilbert spaces. *Foundations and Trends® in Signal Processing*, 8(1-2):1–126, 2015.

[29] Kenji Fukumizu, Francis R. Bach, and Michael I. Jordan. Dimensionality reduction for supervised learning with reproducing kernel hilbert spaces. *J. Mach. Learn. Res.*, 5:73–99, December 2004.

[30] Donglin Niu, Jennifer G. Dy, and Michael I. Jordan. Dimensionality reduction for spectral clustering. *Journal of Machine Learning Research*, 15:552–560, 2011.

[31] Kun Zhang, Jonas Peters, Dominik Janzing, and Bernhard Schoelkopf. Kernel-based conditional independence test and application in causal discovery, 2012.

[32] Jianbo Chen, Mitchell Stern, Martin J. Wainwright, and Michael I. Jordan. Kernel feature selection via conditional covariance minimization. *Advances in Neural Information Processing Systems*, 2017-Decem(Nips):6947–6956, 2017.

[33] Charu C. Aggarwal. *Outlier Analysis*. Springer Publishing Company, Incorporated, 2nd edition, 2016.