The Pennsylvania State University
The Graduate School
Elberly College of Science

A STUDY OF APPROXIMATIONS AND DATA REDUCTION TECHNIQUES FOR KERNEL REGULARIZED LEAST SQUARES

A Thesis in
Statistics
by
Benjamin M. Straub

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2018

The thesis of Benjamin M. Straub was reviewed and approved* by the following:

Bharath K Sriperumbudur
Assistant Professor of Statistics
Thesis Adviser

Donald Richards
Professor of Statistics

Naomi Altman
Professor of Statistics
Chair of Graduate Studies

*Signatures are on file in the Graduate School.

Abstract

Researchers using machine-learning algorithms are encountering problems of data storage and computation time with the advent of Big Data in almost all aspects of life and work. Popular machine-learning techniques that perform computationally well on small datasets often suffer from computational efficiency problems on larger datasets. A second concern is the need for applied researchers to understand and interpret underlying phenomena, which can prove challenging with machine-learning algorithms. The kernel regularized least squares algorithm (KRLS) is a machine-learning technique that has great promise for allowing this understanding, but still suffers from computational issues. In this thesis, I study several data reductions techniques such as clustering, random sampling and random sketching within the KRLS framework in order to decrease computational time while seeking to only sacrifice a little in prediction performance.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1
Background

In this Chapter, we discusses the history of learning algorithms over the last 60 years. We shall describe how algorithms have been invented to overcome certain difficulties and others dusted off due to poor performance of current algorithms when applied to new situations. After the history, we briefly discuss the ordinary least square algorithm (OLS), its advantages and disadvantages. The disadvantages of OLS leads to a discussion of remedies by Tikhonov regularization. However, regularization techniques, such as ridge and lasso, are linear modeling techniques, which when faced with nonlinear data, have poor prediction performance. I then focus on how kernel methods can be introduced into the variations of the ridge regularization algorithm to overcome the challenge of nonlinear data. Surprisingly, a trade-off ensues, as introduction of the kernel methods can drastically increases computational challenges, but the algorithms can now be applied to nonlinear data.

1.1 A Very Brief History of Learning Algorithms

The search for patterns is as old as science itself. The search for effective learning algorithms to aid us in the search for these patterns is of a more modern time. Learning algorithms often have a mysterious oracle-like properties to them both in scientific and non-scientific communities. However, these algorithms all operate under the same basic principles. Given a set of data, we would like to construct an algorithm that can detect patterns in the data. If a pattern has been detected by the algorithm and has the ability to make predictions on new data, then the algorithm has achieved a form of learning. Learning systems can be divided into two broad categories of supervised learning and unsupervised learning. Supervised learning systems have the responses for the data already given, e.g., regression is a supervised learning method. Unsupervised learning systems has data that do not possess corresponding responses, algorithms such as clustering are an example of unsupervised learning systems.

Now that a learning algorithm is available to discover patterns in the data, the next step is to determine whether it is an *effective* algorithm. Here, we adopt the definition of an effective algorithm from Cristianni and Shawe-Taylor (p 12-14) with some input from Vapnik (p 275-276) : computational efficiency, robustness and capacity, and statistically stability and generalization to determine if an algorithm is effective[7],[21].

Framework for an effective algorithm

1. **Computational efficiency** - Algorithms are sought whose resource requirements scale polynomially with the size of the input.

2. **Robustness and Capacity** - Algorithms can handle noisy data and identify approximate patterns even for complex and highly nonlinear data.

3. **Statistical stability and Generalization** - Algorithms , when rerun on a new sample from the same source will identify a similar pattern, i.e. algorithm is not sensitive to a particular dataset.

To better understand the need for the framework of an effective algorithm, a short history of learning algorithms is needed.

The first algorithm for pattern recognition was by R.A. Fisher in 1936. Fisher proposed a method to distinguish between two or more groups, the linear discriminant algorithm (LDA)[8]. Several assumptions are needed for linear discriminant analysis to work optimally. First, the data classes need to be normally distributed and second, the classes have equal covariances. However, even with violations of these assumptions, the linear discriminant algorithm can still provide insights into the data. For the purposes of this history and thesis, we are particularly interested in the linear part of the linear discriminant algorithm and will not discuss the other assumptions of LDA. In Figure 1, Fisher was interested in constructing a linear decision boundary to help classify between groups of iris. The algorithm does a good job of separating the 3 groups by its sepal width and sepal length. However, the method is not perfect as it has trouble separating the virginica and versicolor species, which is indicated by the vertical linear boundary having both species on

either side. This brings us to the issue of using linear modeling tools on nonlinear data. While the linear methods will still work on the nonlinear data it can present problems for the analysis and prediction. Future researchers either tried to augment or invent new methods for pattern recognition when facing nonlinear data. A recurring theme in this thesis will be comparisons of linear versus nonlinear techniques in pattern recognition.

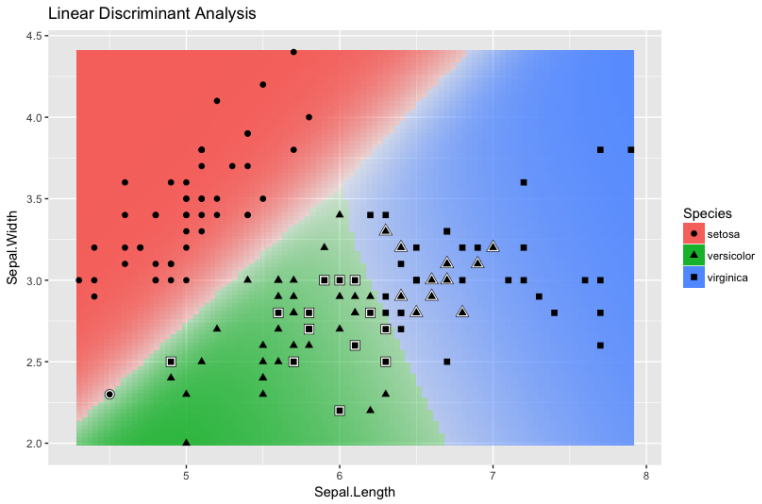

Figure 1.1: Fisher's Iris Data and use of linear discriminant analysis to separate iris based on sepal width and length

In the late 1950s, there came the advent of more effective algorithms that could quickly detect a linear relation in the data. Rosenblatt constructed his perceptron algorithm in 1957, which is the most famous algorithm of this period. The perceptron consists of connected neurons, where each neuron implements a separating hyperplane, so the perceptron as a whole implements a piecewise linear separating surface. Here again, the keyword is a linear[16]. The Perceptron algorithm was to herald a new era, where the computer would be able to walk, talk, see, write, reproduce itself and be conscious of its existence [15]. However, the Perceptron had difficulty recognizing other types of patterns, particularly nonlinear, which showed that the algorithm lacked robustness. In the late 1960s, the famous book Perceptrons by Minsky and Papert, showed this issue in great detail using

the XOR problem. XOR, meaning ”exclusive or” problem is the classical neural networks problem and is essentially a simple classification problem. See Figure 2 for a brief explanation. The big take-away on the XOR problem is the simple classification problem was proving too much for the Perceptron algorithm. Neural networks were now seen as a dead-end for machine learning, and the 1970s has sometimes been called the *AI Winter*, as government and corporations cut funding in this area.

Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1

Figure 1.2: The XOR, or “exclusive or”, problem is a classic problem in neural network research. It is the problem of using a neural network to predict the outputs of XOR logic gates given two binary inputs. An XOR function should return a true value if the two inputs are not equal and a false value if they are equal. All possible inputs and predicted outputs are shown in above Figure.

The 1980s saw new techniques in neural networks to better incorporate nonlinear data, but their statistical behavior were not well understood. Neural networks using back-propagation, for example, was one of the nonlinear machine learning techniques that could now deal with the XOR problem. Unfortunately, neural networks were still suffering from multiple local minima in that the formulation is a nonlinear optimization problem as well as the issue of computational time and their heuristic nature.

In the 1990s, we see the the emergence of kernel-based learning methods, which enabled scientists to apply linear tools to nonlinear data. This was a powerful marriage of ideas as researchers prefer tools that are well-understood and can also be used in a wide array applications of data. Perhaps the most famous kernel-based machine-learning algorithm is the support vector machine, proposed by Vapnik and Cortes in 1995[21]. In Figure 3, again using Fisher’s Iris dataset, and a very naive support vector machines algorithm, the algorithm is able to separate the three iris

types much better than Fisher's linear discriminant analysis algorithm. How are kernel methods able to perform such a feat? I will explore this topic thoroughly, but the underlying principle of any kernel-based machine-learning algorithm is that the input vectors are mapped into a higher dimensional space through some mapping chosen beforehand. Now, in the higher dimensional space, one can now use the more robust and well-understood linear techniques. This approach was based on a very powerful idea that opened the floodgates of machine-learning in the late 1990s.

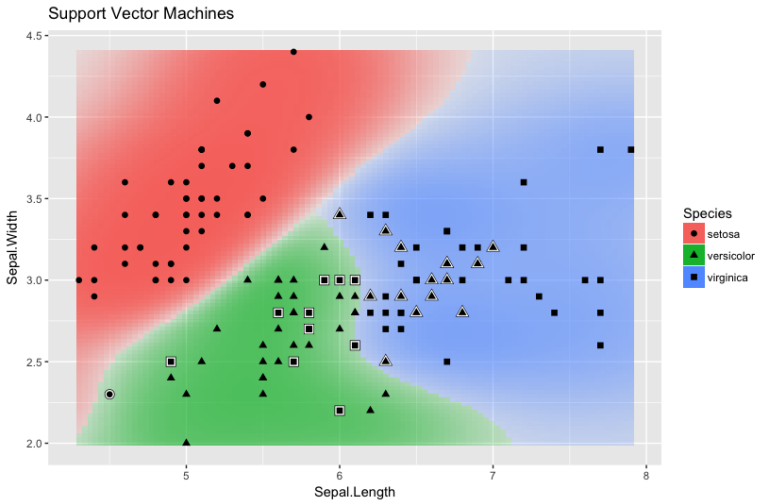

Figure 1.3: Fisher's Iris data set and Vapnik's Support Vector Machines

To summarize this brief, albeit dense short history, researchers prefer linear learning techniques as they are well-understood and in general produce effective algorithms. However, as data becomes more abundant, researchers are encountering more and more nonlinear data. Nonlinear learning techniques such as neural networks try to address this situation, but often require more computation time and often involve nonlinear optimization problems with multiple local minima. Kernel methods are an attractive option for researchers as they merge the robustness and statistical stability of linear techniques with the need of designing algorithms to resolve nonlinear data problems. This thesis will explore extensively these kernel methods needed to discover patterns in nonlinear data and how to increase their

computational efficiency.

1.2 Regression and Its Limitations

The classical linear model forms the foundation of many complex and powerful modeling techniques in statistics. A statistician is interested in understanding the relationship between some response variable y and some predictor variables $x_1, ..., x_p$, i.e. we want to model the relationship between the y and the x's with some function $f(x_1, ..., x_p)$. However, the relationship is not deterministic and we need to allow for some random noise, defined as ε, in our model. The model is now written as,

$$y = f(x_1, ..., x_p) + \varepsilon \tag{1.1}$$

The function f has several assumptions on it that will drive the future discussion and analysis in this thesis. The first assumption for the function, f, is that it is modeled as a linear combination of predictors with parameters $\beta_0, ..., \beta_p$ that are unknown and will need to be estimated from the data.

$$f(x_1, ..., x_p) = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p \tag{1.2}$$

This linear combination can be vectorized as $x_i = (1, x_{i1}, ..., x_{ip})^T$, $i = 1, ..., n$ and $\beta = (\beta_0, ..., \beta_p)^T$ giving us the form

$$f(x) = x^T \beta \tag{1.3}$$

This assumption that f can be constructed as a linear function will be of primary interest for this thesis. The second assumption, as stated above, is that the model is has the random noise/error as additive. Now our model is written as follows,

$$y = x^T \beta + \varepsilon \tag{1.4}$$

To estimate the unknown parameters β, data is collected, (x_i, y_i), where $i = 1, ..., n$

and for every data point an equation is constructed

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_k x_{ip} + \varepsilon_i = x_i^T \beta + \varepsilon_i \tag{1.5}$$

Again, vectorizing the list of equations it can be written as

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} , \boldsymbol{\epsilon} = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix} and \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \dots & \vdots \\ 1 & x_{1n} & \cdots & x_{np} \end{bmatrix} = \begin{bmatrix} 1 & x_1^T \\ \vdots & \\ 1 & x_n^T \end{bmatrix} = \boldsymbol{X}$$

The model is written in its matrix equation form, where the $\boldsymbol{y}$ matrix is the response, the X matrix is the explanatory data, β are the coefficients that are to be estimated and the epsilon matrix is the noise in our data. The X matrix is classically known as our design matrix. The matrix format will be simplified into the following form and this form will be used throughout this thesis,

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \dots & \vdots \\ 1 & x_{1n} & \cdots & x_{np} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

$$y = X\beta + \varepsilon \tag{1.6}$$

The design matrix, X, has the assumption that it is of full rank, which implies that the columns of X (the variables) are linearly independent. For this thesis, of interest, is what happens when this assumption that the design matrix is not of full rank. For example, what if certain variables are linear combinations of other variables in our design matrix, which is a common issue in large datasets. How will this effect the estimates of the coefficients for the linear model? Is it possible to remedy this situation. Also, the assumption that our $\varepsilon \sim N(0, \sigma^2 I)$ must be made and is independent, but further investigation of distribution of the noise in the data is outside the scope of this thesis.

To obtain our estimators, $\hat{\beta}$, the process of minimizing the sum of squared residuals will be used. First, define e as the residuals, that is the difference between

the observed and estimated. Here the ordinary least squares method is trying to find the best line that fits the data by minimizing the residuals.

$$e = y - X\beta \tag{1.7}$$

Now, the residual sum of squares, in vector notation, is just $e^T e$. Using calculus, the solution for our estimator is $\hat{\beta} = (X^T X)^{-1} X^T y$ with the assumptions that $X^T X$ is invertible. If $X^T X$ is not invertible, then we can introduce regularization techniques to allow us to invert $X^T X$.

The beauty of the OLS estimators is that only a few assumptions are needed to derive the estimators. First, the model is linear in the parameters. Second, there is random sampling of the observations. Third, there is no multicollinearity. Fourth, no autocorrelation in the errors. If there is multicollinearity present, then the matrix, $X^T X$, will not be of full rank. The issues with multicollinearity is that either unique estimators can not be derived or the matrix $X^T X$ will not be invertible. Another issue is that OLS when facing an outlier will not be robust, in that the OLS algorithm will attempt to incorporate this point into the coefficients and in the end inflate the coefficients. However, as stated above, other methods are available to remedy this situation, but OLS will form as a benchmark to compare other methods.

1.3 Bias-Variance Trade-off

Before discussing techniques to remedy the many issues facing the classical linear model/learning algorithm, I will quickly discuss the idea of the bias-variance trade-off. The idea of the trade-off for other issues will be a recurring theme within this thesis. In most modeling problems, the researcher wants to learn some functional relationship from the data and then with future data use the learned function to make some predictions. The prediction error can be broken down into three parts for any learning algorithm: Bias Error, Variance Error and Irreducible Error. Equation (1.8) provides the classical mean-square prediction error formula.

$$E\left[\left(y_0 - \hat{f}(\mathbf{x}_0)\right)^2\right] = \text{bias}\left(\hat{f}(\mathbf{x}_0)\right)^2 + \text{var}\left(\hat{f}(\mathbf{x}_0)\right) + \sigma^2 \tag{1.8}$$

The irreducible error, σ^2 is inherently **not** reducible. Perhaps due to some latent variable or data collection process, this error is immutable. However, bias and variance error can be reduced, but only by trading one for the other since we are trying to minimize both at the same time. The bias error is introduced into the model based on the assumptions made on the model. These assumptions allow for the model to have an easier time of learning from the data. For example, the linear learning algorithm tends to have high bias and is not flexible with its assumptions, but the trade-off is that it is fast to learn from the data and interpretable. If high bias is encountered within the linear model, then this would indicate that other assumptions, maybe even more complicated ones, are needed for this problem.

Ideally, the model that has been learned from the data will not change too much when another similar data set is used. Variance error is the measurement of how much the estimates will change when another training set is used. High variance indicates that the model is not picking out the underlying structure of the data. Methods that are flexible to the data tend to have high variance.

In Figure 1.4, is the bias variance trade-off with the yellow line being the sum total of the bias and variance. Decreasing the bias leads to an increase in the variance and decreasing the variance leads to an increase in the bias. Finding an optimal trade-off is important to the predictive performance of an algorithm.

1.4 Ridge Regression

The failure of invertibility for the matrix $X^T X$ in the classical linear model is encountered frequently in real-word applications. In the real-word, perhaps the data gathering process created variables that are linear combination of other already collected variables or the number of variables is greater than the number of samples. Under these conditions estimates can either be unstable for the linear model or estimates will not even be found. In order to alleviate this problem, the method of

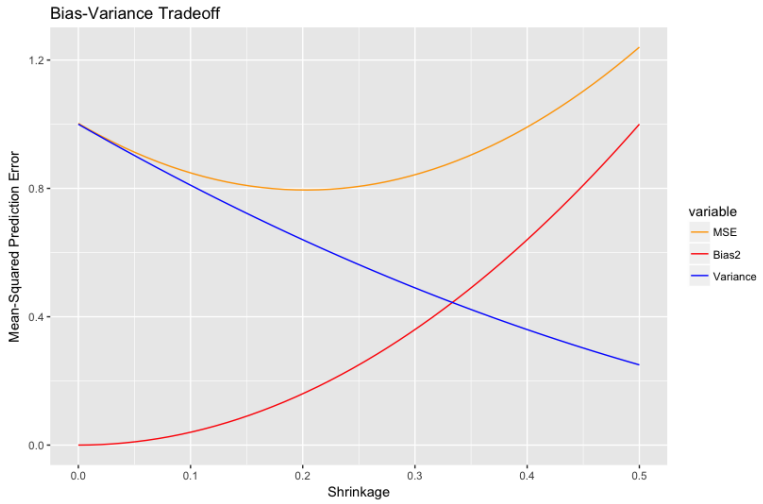

Figure 1.4: Bias-variance Tradeoff

regularization was introduced to allow the $X^T X$ to be properly inverted[20]. There are many variations of regularization techniques available, but this report will focus on the ridge regularization technique. Ridge regression is a classical statistical technique that allows one to invert the $X^T X$ matrix as well as find stable estimates with the trade-off between bias and variance trade-off, discussed in section 1.2.2, in the design of linear regression models.

The setup is the same as OLS estimation technique, that is the y is a vector of responses, X is a design matrix of data, β is our coefficients to estimate and ε is the random noise found in collection of the data.

$$y = X\beta + \varepsilon \tag{1.9}$$

Again, we want to minimize the residual sum of squares as done in the OLS estimation, however, a penalty term is introduced into this equation. By introducing this penalty term, the idea of the variance-bias trade off is introduced into the model, i.e., regularize the coefficients to control the variance, but introduces some bias, then we arrive at the idea of the ridge constraint. Recall, that the residuals are defined as $e = y - X\beta$

$$\min_{\beta} \sum_{i=1}^{n} (y_i - X\beta)^2 s.t. \sum_{i=1}^{p} \beta_i^2 \leq c, c \geq 0 \tag{1.10}$$

Now, rewriting in common L_2 penalty notation and using theory for optimization under constraints, the equation will be written as

$$\min_{\beta} (y - X\beta)^T (y - X\beta) + \lambda ||\beta||^2 \tag{1.11}$$

Utilizing the same technique as we did in the OLS derivation, we first take the first derivative of our new penalized/regularized sum of squares.

$$\begin{aligned}
e^T e &= (y - X\beta)^T (y - X\beta) + \lambda ||\beta||_2^2 \\
&= \frac{\partial e^T e}{\partial \beta} = 0 \\
&= -2X^T (y - X\beta) + 2\lambda\beta = 0 \\
&= -X^T y + X^T X\beta + \lambda\beta = 0 \\
X^T X\beta + \lambda\beta &= X^T y \\
\beta &= (X^T X + \lambda)^{-1} X^T y \\
\beta &= (X^T X + \lambda I_p)^{-1} X^T y
\end{aligned} \tag{1.12}$$

The I_p is an identity matrix that is the dimension of $X^T X$. Now, the estimator for the ridge problem is (1.12). The choice of λ is critical to this technique. Methods, such as cross validation, will be discussed and implemented later to help in finding an optimal λ.

Ridge is a powerful technique when OLS fails. However, it exists in the inflexible domain of linearity. When ridge or any other linear algorithm encounters nonlinear data, then it will have poor predictive performance. Just like with the failure of OLS to deal with multicollinearity, a new technique, that will be explored extensively in this thesis, is introduced to deal with nonlinear data, which is the kernel method.

1.5 Kernels and Nonlinear Problems

Traditional data analysis techniques, such as linear regression, assume the unknown functional form is linear, which makes them unable to extract more complex, nonlinear patterns that may lie in the data. To circumvent this issue of nonlinear data a range of techniques have been introduced, and some were briefly discussed in the history section. The kernel methods are one of those techniques.

A motivating example will be presented to begin the brief discussion of kernel methods for this Chapter. In Chapter 2, I will discuss kernel methods in great detail. Observe in Figure 1.5, that the left plot has two types of data, circles and squares, which are distributed on a plane. The squares are clustered around the origin while the circles are distributed around the outside of the squares. The data are not linearly separable, but can be identified as two separate groups by using one's eye. If we allow some function, ϕ, that can map the data into some higher dimension, then the data can be separated by a hyperplane, which is a linear separating object. The higher dimensional space is called the *feature space* and it can be of infinite dimensions. Now, the underlying idea of the function ϕ is that it can map the data into a space where the patterns can be separated by a linear object[7]. For Figure 1.5, observe the data in higher dimensions and now can be separated by a hyperplane, making it linearly separable. The function, ϕ, will be known as the feature map and as discussed later in detail will have this ability of mapping data to higher dimensions. However, mapping into higher dimensional feature spaces can be fraught with peril, especially if one is mapping into infinite dimensional feature spaces. Nonetheless, if one is using the solid mathematical framework of the Reproducing Kernel Hilbert Space, then the kernel functions can be situated to only return the pairwise inner products of the inputs from the feature space. This important fact, explored in Chapter 2, allows the kernel method algorithms to never venture into the feature space.

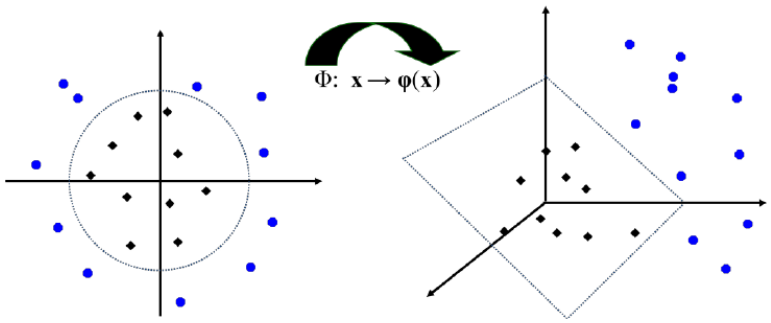

Figure 1.5: The mapping ϕ transforms the data from the 2-dimensional space into a higher dimensional space, feature space, where a linear model (a hyperplane) can now separate the data.

To quickly summarize the kernel method approach I have provided a sketch of the kernel method algorithm. While my focus is on the kernel regularized least squares, this sketch generalizes well to other methods that use kernels.

Algorithm 1 Kernel Method

1: **Inputs:**

Data, Choice of Kernel, Choice of Kernel Parameters

2: Inputs are embedded into the feature space via a specified kernel function, ϕ

3: Linear relations are sought among the inputs in the feature space

4: Algorithms are implemented in such a way that the coordinates of the points in the feature space only need the pairwise inner products.

5: The pairwise inner products are computed directly from the original data using a kernel function

1.6 Computational Challenges for Kernel Methods

Time and space complexity are of primary interest for investigating the effectiveness of any algorithm. Here time complexity is the measure of time taken to run the algorithm. The common way to estimate time complexity is by counting the number of elementary operations performed by the algorithm. Revisiting the framework of an effective algorithm, most researchers like to have this as a guiding

principle:

1. **Computational efficiency** - Algorithms are sought whose resource requirements scale polynomial with the size of the input.

The kernels methods that I will be exploring in this thesis will have a time complexity of $O(n^3)$, where n is the sample size of the data. Unfortunately, as n increases in size, the computational issues one faces becomes prohibitive, which makes using kernel methods difficult on very large datasets. Space complexity involves the number of storage locations the algorithm needs, i.e. the biggest operation within that algorithm that will be needed at any point in the algorithm. In this thesis space complexity for the kernel methods will typically be of $O(n^2)$ As will be explored more in depth in Chapter 2, kernel methods have desirable statistical properties, but can suffer from computational challenges as the sample increases in size. Several methods will be explored to try and reduce these computational challenges.

1.7 Goals of this Thesis

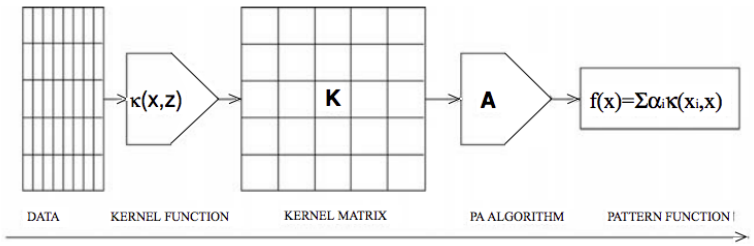

Figure 1.6: Stages of the Kernel Method from Cristianni and Shawe's *Kernel Methods for Pattern Analysis*

The end goal of this thesis is to compare methods to reduce computation time for kernel regularized least squares. Observe in Figure 1.6, the stages of the kernel method in graphical form. The first set of methods that I explore look at the first stage, the Data. I seek ways to either reduce the training data that is fed into the

kernel function via k-means clustering or random sampling. The second set of methods that I explore look at the third stage of the kernel method. I seek ways to reduce the dimensions of the kernel matrix, by random projection. As stated before, large datasets prove challenging for kernel methods as computing inner products for each pairwise observations can quickly balloon in costs. Sacrificing some prediction performance in order to reduce computational time could prove beneficial in certain situations and therefore warrants exploration.

Chapter 2
Kernel Methods

Discovering and learning from patterns in data is the primary goal of any learning technique. As discussed in Chapter 1, linear techniques attempt to discover the underlying linear functional relationship between the input and output of the data. However, the linear assumption is a rigid assumption, especially when facing non-linear data. In this chapter, I discuss the mathematical concepts and tools needed to understand kernel methods and how one moves from least squares regression to kernel regularized least squares to better incorporate nonlinear data.

2.1 Definitions and Notations

In order to move away from the rigidity of linearity, we must build up a set of mathematically sound tools to work within the kernel method framework. These tools are largely pulled from the works of *Learning with Kernels* by Scholokopf and Smola as well as *Kernel Methods for Pattern Analysis* by Shawe-Taylor and Cristianini.

Definition 2.1.1. Kernel Function - A function that takes as its input the data from the original space and returns the inner product of the vectors in the feature space is called a kernel function ϕ, i.e. given input $(\boldsymbol{x}, \boldsymbol{x'})$ contained in some space V and a function ϕ, which is a mapping from V to R^N, $\phi : V \mapsto R^N$ then $K(x,x') =< \phi(x), \phi(x') >$

Example 2.1.1. A simple example of a kernel function that takes inputs and projects into a feature space is one that maps inputs from the Cartesian domain into the polar domain.

$$\phi : R^2 \mapsto R^2$$

$$\boldsymbol{x} \mapsto \phi(\boldsymbol{x})$$

$$(\boldsymbol{x_1}, \boldsymbol{x_2}) \mapsto (r, \theta)$$

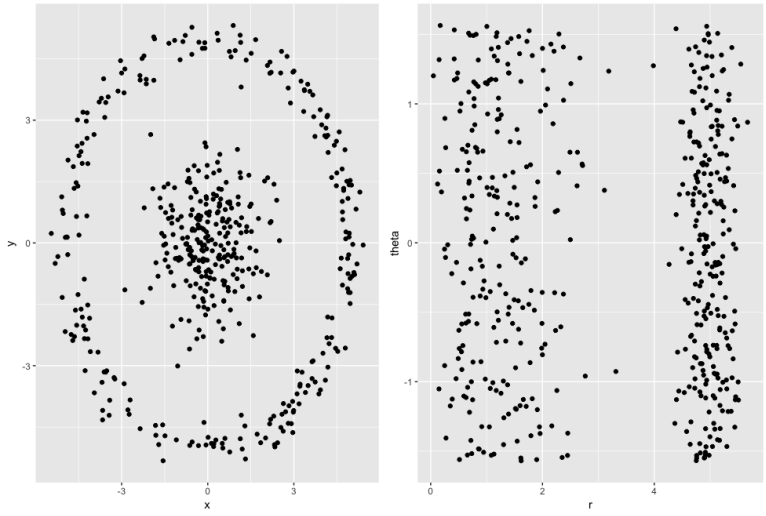

Figure 2.1: Cartesian Domain to the Polar Domain

$$r^2 = x_1^2 + x_2^2, \theta = arctan(\frac{x_2}{x_1}) \tag{2.1}$$

In Figure 2.1, using example 2.1.1, I have simulated data that has two distinct elliptical patterns. Observe that there is some elliptical boundary separating the data, but linear techniques would have an impossible time separating the data. However, projecting into the polar domain, using equation 2.1, the data has now been transformed such that linear techniques can easily separate out the two groups. Here the polar domain can be considered as the feature space.

Example 2.1.2. Using the same data and the same ideas in Figure 2.1, if we mapped the data via a nonlinear map, ϕ, where

$$\phi(x) = (z_1, z_2, z_3) = ([x]_1^2, [x]_2^2, \sqrt{(2)}[x]_1[x]_2)$$

then the data could be linearly separated by a hyperplane. Interestingly, this is due to the fact that ellipses can be written as linear equations of the entries of (z_1, z_2, z_3). Now, we just need some methods to construct and evaluate these hyperplanes without having to compute ϕ.

The end goal is that we want a kernel function that computes the dot product

in the feature space, but allows us to avoid doing any calculations in the feature space, since the feature space could be of infinite dimensions. To do this, we will need a few more tools to ensure that everything is mathematically sound.

Definition 2.1.2. Gram or Kernel Matrix
Given a set of vectors, $X = (\boldsymbol{x_1}, ...\boldsymbol{x_n})$, the Gram matrix is defined as an $n \times n$ matrix K whose entries are $K_{ij} = \langle \phi(x_i), \phi(x_j) \rangle = K(x_i, x_j)$, where K is the kernel function. Here, the Kernel Matrix measures the pairwise similarities between the inputs of the data.

Running the kernel on the set of n vectors gives us an inner product for each two samples.

$$K(x_1, \ldots, x_n) = \begin{vmatrix} \langle \phi(x_1), \phi(x_1) \rangle & \langle \phi(x_1), \phi(x_2) \rangle & \ldots & \langle \phi(x_1), \phi(x_n) \rangle \\ \langle \phi(x_2), \phi(x_1) \rangle & \langle \phi(x_2), \phi(x_2) \rangle & \ldots & \langle \phi(x_2), \phi(x_n) \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \phi(x_n), \phi(x_1) \rangle & \langle \phi(x_n), \phi(x_2) \rangle & \ldots & \langle \phi(x_n), \phi(x_n) \rangle \end{vmatrix} \tag{2.2}$$

The Gram matrix is the workhouse of the learning algorithms that use kernel methods. The algorithms receive information on the training set from the Gram matrix. Therefore, it acts as an information bottleneck, as all the information available to a kernel algorithm, be it about the distribution, the model or the noise, must be extracted from this matrix[7].

I have mentioned a few cautions when working with kernels methods. First, is avoidance of computing the feature map. To ensure that an algorithm does not have to venture into the feature space the kernel function must allow for a representation in terms of an inner product under a feature map. To ensure this representation, two key definitions are needed, positive definite kernels and reproducing kernel Hilbert space (RKHS). Second, is that the kernel matrix is fully dense, so that solving using direct methods takes $O(n^3)$ time. Finding methods to address this dense issue will also be explored in Chapter 3.

Definition 2.1.3. Positive semi-definite matrix - A symmetric $n \times n$ matrix, K, is positive semi-definite if and only if $x^T Kx \geq 0$ for all $x \in R^n$

Definition 2.1.4. Positive semi-definite kernel - $K : R^d \times R^d \mapsto R$ is positive semidefinite if

- $\forall (x,x') \in R^n \times R^n, K(x,x') = K(x,x')$.
- $\forall n \in N, \forall \xi_1 ... \xi_n \in R, \forall x_1 ... x_n \in R^d, \sum_{i,j=1}^{n} x_i x_j k(x_i,x_j) \geq 0$

Note: Use of kernels for regression in this context should not be confused with nonparametric methods that involve using a kernel to construct a weighted local estimate.

Prop 2.1 Gram and kernel matrices are positive semi-definite.

Proof. Consider the general case of a kernel matrix, let $K_{ij} = K(x_i,x_j) = \langle \phi(x_i), \phi(x_j) \rangle$, for i,j=1...n

For any vector α, we have

$$\alpha^T K \alpha = \sum_{i,j=1}^{n} \alpha_i \alpha_j K_{ij} = \sum_{i,j=1}^{n} \alpha_i \alpha_j \langle (\phi(x_i), \phi(x_j) \rangle$$

$$= \langle \sum_{i}^{n} \alpha_i \phi(x_i), \sum_{j}^{n} \alpha_j \phi(x_j) \rangle$$

$$= || \sum_{i}^{n} \alpha_i \phi(x_i) ||^2 \geq 0$$

□

Definition 2.1.5. Reproducing Kernel Hilbert Space

Let $\mathscr{H}$ be a Hilbert space of real-valued functions defined on a non-empty set X. A function $k : X \times X \mapsto R$ is called a reproducing kernel of $\mathscr{H}$, and $\mathscr{H}$ is a Reproducing Kernel Hilbert Space, if k satisfies

- $\forall x \in X, \quad k(\cdot,x) \in \mathscr{H}$,
- $\forall x \in X, \forall f \in H, \langle f, k(\cdot,x) \rangle_{\mathscr{H}} = f(x)$ (the reproducing property).

Then for any $x,y \in X$, $k(x,y) = \langle k(\cdot,x), k(\cdot,y) \rangle_{\mathscr{H}}$

For a feature space to be constructed associated with ϕ, the image of ϕ must be turned into a vector space and endowed with an inner product[17]. Now, the vector space can be defined by taking linear combinations of kernels and some coefficients. Any positive definite kernel has a feature space where it can be thought of an inner product, such that

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \tag{2.3}$$

RKHS give kernel methods a solid mathematical framework. This framework allows the kernel methods to compute inner products in a high, even infinite, dimensional feature space as a kernel function in the input space. Basically, allowing us to solve implicitly the linear algorithm in a feature space, where the data's images are more likely to be separated by a hyperplane.

The Kernel Trick

The kernel trick is the backbone of the kernel methods and has been around in the literature since the early 1900s, but was seen as a mathematical curiosity without any practical usefulness. However, by the mid 1990s it was proven that any algorithm that only depends on dot products can be kernelized, which opened up the floodgates of machine learning techniques such as support vector machines. The gist of the trick is that any inner product based algorithm can be transformed to an alternative algorithm by replacing the inner products with a nonlinear kernel, i.e., Equation (2.3) states that this alternative kernel-based algorithm is equivalent to applying the original inner product based algorithm in the feature space[17]. Now, this simple idea allows us to take the solution and implicitly solve a convex optimization problem in the feature space. To sum up, given an algorithm which is formulated in terms of a positive definite kernel k, one can construct an alternative algorithm by replacing k by another positive definite kernel $\tilde{k}$ [17].

One last theorem is needed to cement the foundation needed for kernel methods, which is the Representer Theorem. The significance of the theorem is that it allows the simplification of the optimization problem in an infinite-dimensional

space,$\mathscr{H}$, and reduces the search to an n-dimensional vector of coefficients. This representer theorem allows for the machine learning algorithm to go from the theoretical level to the practical level, i.e., we do not want to solve an optimization problem in an infinite-dimensional space[18].

Theorem 2.1.1 (**Representer Theorem**). Let $\mathscr{X}$ be a nonempty set and k a positive-definite real-valued kernel on $\mathscr{X} \times \mathscr{X}$ with corresponding reproducing kernel Hilbert space H_k. Given a training sample $(x_1, y_1), \ldots, (x_m, y_m) \in \mathscr{X} \times \mathbb{R}$, a strictly monotonically increasing real-valued function $\Omega\colon [0,\infty) \to \mathbb{R}$, and an arbitrary empirical risk function $c\colon (\mathscr{X} \times \mathbb{R}^2)^n \to \mathbb{R} \cup \{\infty\}$, then each minimizer $f \in \mathscr{H}$ of the regularized risk
$c((x_1, y_1, f(x_1)), ..., (x_n, y_n, f(x_n))) + \Omega(\|f\|_{\mathscr{H}})$
admits a representation of the form:
$f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x)$, where $\alpha_i \in \mathbb{R}$ for all $1 \leq i \leq n$.

Please see Scholkopf and Smola for a sketch of the proof on page 90-91 of their seminal work *Learning with Kernels*.

2.2 Common Kernels

This section focuses on kernels commonly seen in the literature. The kernel is at its core a similarity measure, where it serves as a function to return an inner product from the projected data in the feature space. In addition there are kernels that can be constructed for specific purposes given some prior knowledge of the data, which is a open topic of exploration.

Definition 2.2.1. Linear Kernel

$$K(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{x}^T \boldsymbol{x'}, \boldsymbol{x}, \boldsymbol{x'} \in R^d \tag{2.4}$$

Useful for forming a baseline for performance of other kernels.

Definition 2.2.2. Polynomial Kernel. For degree d polynomials,

$$K(\boldsymbol{x}, \boldsymbol{x'}) = (\boldsymbol{x}^T \boldsymbol{x'} + c)^d, \boldsymbol{x}, \boldsymbol{x'} \in R^d, c > 0 \tag{2.5}$$

x and x' are the inputs and c is a free parameter that trades off influence between higher order and lower order terms.

Definition 2.2.3. Hyperbolic Tangent Kernel. For degree d polynomials,

$$K(\boldsymbol{x}, \boldsymbol{x'}) = tanh(\lambda \langle (x, x' \rangle), \boldsymbol{x}, \boldsymbol{x'} \in R^d, \lambda > 0 \tag{2.6}$$

x and x' are the inputs and λ is a scaler parameter, this kernel is mainly used as a proxy for neural networks

Definition 2.2.4. Sequence kernels. A kernel that takes two strings from some alphabet and computes the similarity. Squence kernels are outside the purvey of this thesis, but warrants a mention as it shows the power of kernel methods, i.e., that even in situations where a vectorial representation is not available you can still use kernel methods.

Definition 2.2.5. Gaussian Kernel

$$K(\boldsymbol{x}, \boldsymbol{x'}) = exp(\frac{-||\boldsymbol{x} - \boldsymbol{x'}||^2}{2\sigma^2}), \boldsymbol{x}, \boldsymbol{x'} \in R^d, \sigma > 0 \tag{2.7}$$

The Gaussian kernel will be extensively used in this thesis and therefore necessitates a thorough introduction. First, in the numerator of the expression, $||\boldsymbol{x} - \boldsymbol{x'}||$, is the Euclidean distance between the explanatory vectors $\boldsymbol{x}$ and $\boldsymbol{x'}$, which provides the understanding of how the kernel function is a measure of similarity between two inputs. As the two inputs become closer and closer together, then the kernel function will return values closer and closer to one, while two inputs that are getting farther and farther apart will produce values close to zero. A key insight into the Gaussian kernel is that it does not model the data as a linear function, but rather leverages information about the similarity between the input data[9]. Next, σ^2 is called the bandwidth of the Gaussian kernel and is a measurement decision that incorporates how distant points need to be in the standardized covariate space

before they are considered dissimilar. The choice of bandwidth must be made in any learning procedure and how to choose the bandwidth is an open topic, but a popular method is using cross-validation.

The last two sections has reviewed the mathematical theory needed to understand kernels. The simple examples in the beginning provided some intuition in how important moving data into feature spaces can provide us with more powerful tools to find patterns. I then established how kernels can be used to compute dot products, how kernel methods need RKHS and the representer theorem and a brief discussion of common kernels. Next, I take this theory of kernels and apply it to the regularized least squares, moving this well-understood method into the domain of kernels.

2.3 Kernel Regularized Least Squares

The appeal of the Kernel Regularized Least Squares is its intepretability and flexibility, especially to applied researchers[9]. Machine learning techniques tend to have difficult interpretations to understand a phenomenon of interest and most strictly just do prediction. This thesis does not go too deep into the interpretation of the results from the KRLS method. Please see [9] [14] for more in-depth discussion. However, suffice it to say that KRLS allows interpretations in ways analogous to generalized linear models (GLM), but adds in more complex interpretation to examine nonlinearities and interactions. GLMs have strict assumptions on its functional form. KRLS permits more flexibility by taking a radically different approach to its assumptions in two ways. First, given a dataset where the functional form is unknown, KRLS is flexible enough to take in the data and discover this form by taking a similarity approach to constructing a prediction function, whereas GLMs assume that the model can be built by weighted sums of independent variables[9]. A drawback to any flexible method is that it can lead to over-fitting of the model and will necessitate additional assumptions to help address this over-fitting issue. Second, KRLS allows the researchers to use linear modeling tools on nonlinear data via the kernel method. Essentially, the algorithm is just solving a system of linear equations in a higher dimensional space. If the function preferred is smoother and

less complicated, then the algorithm can find solutions in a more constrained space, hence the need for regularization.

To introduce kernel regularized least squares (KLRS), I find it wise to revisit the Ridge problem presented in Chapter 1. The same setup for the data will be used and the given data will be assumed to exhibit issues with multicollinearity or the set of variables is more than the set of data. Now, the ultimate goal is to discover some linear functional relationship between the inputs and outputs under the ordinary least squares paradigm, but the multicollinearity prevents this from happening, However, introducing a penalty, λ, into the classical OLS problem allows one to get more stable estimates.

$$\min_{\beta}(y-X\beta)^T(y-X\beta)+\lambda||\beta||^2 \tag{2.8}$$

The L_2 norm acts as the regularization constraint that helps the model measure the complexity of the function and λ is a scalar parameter that governs the trade-off between model fit and complexity, while λ encapsulates the idea of trade-off between bias and variance, i.e., introduce some bias into our problem in order to reduce the variance. Finally, the estimate for the ridge β, derived in Chapter 1, is

$$\beta = (X^TX+\lambda I)^{-1}X^Ty \tag{2.9}$$

with I being an $n \times n$ identity matrix. If λ is allowed to go to zero, then we arrive at the classic OLS estimates for β, i.e. $\beta = (X^TX)^{-1}X^Ty$. Now, a key assumption that OLS and ridge make is that the data has some linear relationship that can be discovered, but data does not always conform to our expectations. Therefore, when facing nonlinear data the ridge and OLS estimates will perform poorly, but using the kernel methods that have been introduced the data can be projected into a feature space and the linear tools of Ridge can be utilized, hence kernel regularized regression. This is a powerful idea that the well-understood ridge tools can still be utilized on nonlinear data in a transformed space.

Before deriving the estimator involving kernels, I want to explore an another solution to the ridge problem in Equation (2.8), which can be represented in an

alternative fashion by using the Representer Theorem[12]. The theorem states that the solution for β for minimization problems, e.g., Equation (2.8) can be re-written as a linear expansion of the training inputs. For my purposes, lets go back a step for the solution to β for Equation (2.9)

$$(X^T X + \lambda I)\beta = X^T y$$

$$\beta = \lambda^{-1}(X^T y - X^T X\beta)$$

$$\beta = \lambda^{-1} X^T (y - X\beta)$$

Let

$$\alpha = \lambda^{-1}(y - X\beta)$$

then

$$\beta = X^T \alpha = \sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i, \tag{2.10}$$

which shows that the β can be written as a linear combination of the data. Using this information, the optimal α's can be found through the next derivation

$$\alpha = \lambda^{-1}(y - X\beta)$$

$$\lambda\alpha = (y - X\beta) = (y - XX^T\alpha)$$

$$(XX^T + \lambda I)\alpha = y$$

$$\alpha = (XX^T + \lambda I)^{-1} y \tag{2.11}$$

There are two solutions now to the regularization problem that was presented in (2.8), which are (2.11) and (2.10). This brings about a brief mention of primal and dual solutions. The first tells us how to find the solution by computing the weight vector and the second tells us how to find the solution by finding a linear combination of the training examples. The crucial take away from the primal and dual solution is that in equation (2.9), the XX^T can be written as just the inner

product of $\boldsymbol{x_i}, \boldsymbol{x_j}$, therefore $XX^T = K = G$, which is the Gram matrix, Equation (2.2), that was previously discussed. Now, a new input in a predictive function can be derived by using the inner products of the training points and the new input. In general, the primal solution takes less computational time, but will not be able to deal with nonlinear data. Luckily, the dual solution can deal with this added complexity of nonlinear data since the kernel trick can be applied, since we are are only dealing with inner products of the training inputs.

Now, given some problem where the data is nonlinear, we know that the linear techniques will not be sufficient in modeling. However, if the data can be projected into a feature space, then there is hope. The only assumptions for this method are that the functions and kernels lie in a reproducing kernel Hilbert space. Using some unspecified kernel ϕ, the data from the design matrix, X, will be sent into an unspecified feature space.

$$X = \begin{bmatrix} \dots & x_1 & \dots \\ \dots & x_2 & \dots \\ \dots & \vdots & \dots \\ \dots & x_n & \dots \end{bmatrix} \quad \mapsto \phi(X) = \begin{bmatrix} \dots & \phi(x_1) & \dots \\ \dots & \phi(x_2) & \dots \\ \dots & \vdots & \dots \\ \dots & \phi(x_n) & \dots \end{bmatrix}$$

Now, the regularized regression problems can be re-written as the following:

$$\min_{\beta} ||y - \phi(X)\beta)||_2^2 + \lambda ||\beta||^2 \tag{2.12}$$

where $||\beta||^2$ is the norm of β in that space, λ is the scaling parameter for trade-off and $\phi(X)$ is a mapping that moves the data to a higher-dimensional space:

$$\min_{\beta} (y - \phi(X)\beta)^2 + \lambda \beta^T \beta$$

$$\frac{\partial}{\partial \beta} \{ (y - \phi(X)\beta)^2 + \lambda \beta^T \beta \}$$

$$-2\phi(X)^T (y - \phi(X)\beta) + 2\lambda\beta = 0$$

$$-2y\phi(X)^T + 2\phi(X)\beta\phi(X) + 2\lambda\beta = 0$$

$$y\phi(X)^T = \phi(X)\beta\phi(X) + \lambda\beta$$

$$y\phi(X)^T - \phi(X)\beta\phi(X) = \lambda\beta$$

$$\phi(X)(y - \phi(X)^T\beta) = \lambda\beta$$

$$\lambda^{-1}\phi(X)^T(y - \phi(X)\beta) = \beta$$

$$\alpha = \lambda^{-1}(y - \phi(X)^T\beta)$$

$$\beta = \phi(X)^T\alpha$$

$$\beta = \sum_{i=1}^{n} \alpha_i\phi(x_i) \tag{2.13}$$

So β is re-written as a linear combination of kernels and coefficients. Using the same trick as in 2.9, the dual can be computed such that

$$\alpha = \lambda^{-1}(y - \phi(X)\beta)$$

$$\lambda\alpha = (y - \phi(X)\beta) = (y - \phi(X)\phi(X)^T\alpha)$$

$$(\phi(X)\phi(X)^T + \lambda I)\alpha = y$$

$$\alpha = (\phi(X)\phi(X)^T + \lambda I)^{-1}y \tag{2.14}$$

Since $(\phi(X)\phi(X)^T$ is just the Gram/Kernel matrix, it will be relabeled as $\phi(X)\phi(X)^T = K$

$$\alpha = (K + \lambda I)^{-1}y \tag{2.15}$$

Now, given some new inputs, we can use the predictive function and Equation (2.14) to get

$$f(x) = \phi(X)^T\beta = \langle\sum_{i=1}^{n} \alpha_i\phi(x_i), \phi(x)\rangle = \alpha^T K = y^T(K + \lambda)^{-1}k \tag{2.16}$$

where K is the kernel matrix with the original inputs and k is the kernelized new inputs. Note, that the use of kernels to compute inner products prevents us from

ever needing to explicitly perform the expansion implied by ϕ. This is the oft cited kernel trick that prevails in the machine learning community as well as presented above.

Algorithm 2 KRLS Sketch

1: **Inputs:**
   Data, Lambda, Sigma, kernel function
2: Data is fed into a kernel function creating an $n \times n$ kernel matrix.
3: Using Equation (2.16), compute predictions of new inputs

2.4 Summary

Patterns that are linear can be detected efficiently by well-understood techniques such as least squares regression. Least squares, when facing multicollinearity, can be remedied by using Tikhonov regularization. However, when least squares is facing nonlinear data, then it will perform poorly. If the data is mapped into a feature space via some nonlinear function, then one can utilize the linear tools in the feature space for discovering patterns in the nonlinear data. The caveat is that one does not want to do any computations inside the feature space as it could be infinite dimensional. Kernels make it feasible to do this calculation by providing a mathematically sound foundation to attack nonlinear problems via Reproducing Kernel Hilbert Spaces and the Representer Theorem..

Chapter 3
Data Reduction

As society continues to move through the era of Big Data, the issue of storage and computational time becomes an ever-pressing issue. Researchers now require methods that quickly reduce the size of the data, but still maintain its meaningful effects. Here a trade-off must be made, in order to gain insights and understanding into the data's patterns in a timely manner a reduced set must be explored. Data reduction is a set of techniques that can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data[10].

Data Reduction is broken down into three broad categories:

1. Dimensionality Reduction
2. Compression
3. Numerical Reduction

A typical dataset has the columns as the variables (p) of the data while the rows are the recorded observations n of the data, hence an $n \times p$ matrix of data. The variables can sometimes be reduced by dimensional reduction techniques, such as principal component analysis (PCA). Dimension reduction techniques have been widely studied and are not the focus of this thesis. Data compression involves a transformation or encoding of the data in order to have a compressed representation of the original data. A very interesting area, but again not the focus of this thesis. The data reduction technique that is the focus of this thesis is the numerical reduction technique, which has both parametric and nonparametric methods available to it. I focus on the nonparametric methods of clustering, sampling and sketching.

The clustering and sampling methods will focus on the DATA part of Figure 3.1, while the random projection will focus on the KERNEL MATRIX portion of the diagram.

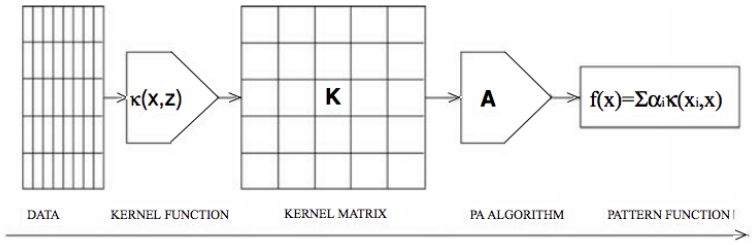

Figure 3.1: Stages of Kernel Method Revisited

3.1 Sampling

Sampling, while seeming trivial, can be used as a data reduction technique, as it allows a large data set to be represented in a much smaller fashion. Several options exist for sampling techniques such as sampling with replacement, sampling without replacement, cluster sampling and stratified sampling. I will utilize sampling with replacement as a benchmark to compare other data reduction techniques.

3.2 Clustering

Clustering is a set of well-explored and understood unsupervised learning techniques. Clustering partitions observations into groups that are similar to one another and dissimilar to observations in other clusters. The typical measurement is a distance function that checks the similarity between each observation. Finally, the clusters form a representation of the actual data that a researcher is interested in investigating. For this thesis, of interest is reducing the actual data in a way that preserves the meaningful relationship between the observations, but gives us the ability to speed up the computation for kernel regularized least squares.

3.2.1 K-means

K-means clustering is done by partitioning a data set into K distinct, non-overlapping clusters. To perform this technique, we first must specify the number of K clusters that we want to use. It is useful to have domain knowledge to help in specifying

the number of clusters beforehand. The algorithm then assigns each observation to exactly one of the clusters while trying to minimize the within cluster sum of squares. K-means has issues with converging to local minima and has issues with how the random start is initialized. However, k-means is a simple enough algorithm that multiple runs with different random starts has been shown heuristically to avoid the trap of local minima.

Algorithm 3 K-means Sketch

1: **Inputs:**
   Data, number of K clusters, number of random starts
2: **Initialize:**
   Choose a random start
3: Assign data to a partition with the closest center means, for $i = 1...N$
4: Recalculate the mean as the center of clusters and repeat until there is no change

3.2.2 Hierarchical Clustering

Hierarchical clustering builds hierarchies of clusters, which is done in two ways:

Agglomerative (bottom-up) Clustering

1. Start with each example in its own singleton cluster
2. At each time-step, greedily merge 2 most similar clusters
3. Stop when there is a single cluster of all examples, else go to 2

Divisive (top-down) Clustering

1. Start with all examples in the same cluster
2. At each time-step, remove the "outsiders" from the least cohesive cluster
3. Stop when each example is in its own singleton cluster, else go to 2

I will use the agglomerate clustering technique and not the divisive clustering and so will provide a little more information on agglomerative. Three linkage techniques are used with the agglomerative clustering technique, which are a measure of similarity or dissimilarity between clusters.

- **Single linkage:** Looks at all the pairwise distances between the items in the two clusters and takes the distance between the clusters as the minimum distance.
- **Complete linkage** Takes the maximum distance between clusters
- **Average linkage** Takes the average distance between clusters

Algorithm 4 Agglomerative Clustering Sketch

1: **Inputs:**
   Data, number of K clusters, linkage
2: **Initialize:**
   Create N clusters and assign one data point to each
3: Calculate distance measure, linkage, merge clusters which linkage is minimized.
4: Repeat until linkage is minimized

Domain knowledge is helpful in deciding which clustering technique to use. In Figure 3.2 on the left side, I provide an example of where k-means breaks down in its ability to identify the two distinct clusters. However, using hierarchical cluster with single linkage, the algorithm can now separate the two clusters quite well.

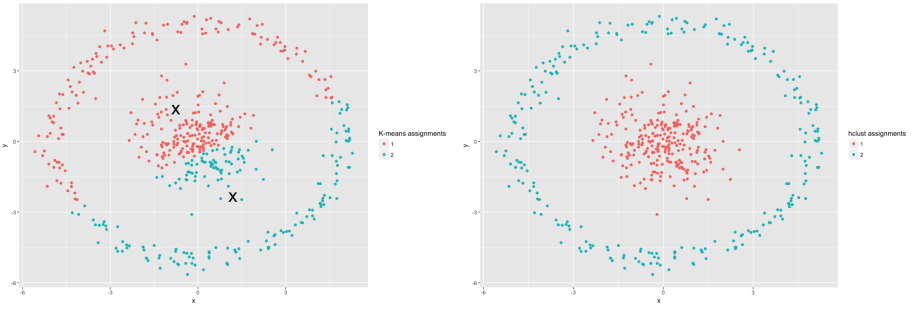

Figure 3.2: K-means separates the two groups of data incorrectly. The X's represent the means. Meanwhile, hierarchical clustering identifies the two groups quite well with single linkage

3.2.3 KRLS with Clustering

The kernel regularized least squares algorithm can be altered slightly to accommodate clustering. Using K to signify the Kernel, the kernel algorithm now takes in the Z clusters that have been identified. The Z's are fed into the kernel and again the same procedure is used that finds the optimal estimator.

$$\sum_{i=1}^{n}[y_i - [K(x_i,z_1),...,K(x_i,z_m)(\alpha_1,...,\alpha_m)^T]^2 + \lambda\alpha^T K(z,z)\alpha \quad \alpha \in R^m$$

$$\min_{\alpha}\sum_{i=1}^{n}(y_i - K^T(x_i,z)\alpha)^2 + \lambda\alpha^T K(z,z)\alpha$$

$$\frac{\partial}{\partial\alpha}\sum_{i=1}^{n}(y_i - K^T(x_i,z)\alpha)^2 + \lambda\alpha^T K(z,z)\alpha$$

$$-2K^T(x,z)(y - K(x,z)\alpha) + 2\lambda K(z,z)\alpha = 0$$

$$-K^T(x,z)y + K^T(x,z)K(x,z)\alpha + \lambda K(z,z)\alpha = 0$$

$$K^T(x,z)y = \alpha(K^T(x,z)K(x,z) + \lambda K(z,z))$$

$$\alpha = ((K^T(x,z)K(x,z) + \lambda K(z,z))^{-1}K^T(x,z)y \tag{3.1}$$

$$f(x) = \sum_{i=1}^{m}\alpha_i K(\cdot,Z_i) \tag{3.2}$$

Equation (3.1) is then used to make predictions for new inputs in Equation (3.2), similar to the procedure in Equation (2.16).

3.3 Random Projection

The kernel matrix within the KRLS algorithm is of time complexity $\mathcal{O}(n^3)$, where n is the sample size. As discussed in previous sections, the kernel matrix is a dense matrix and it becomes computationally expensive to compute as the sample size increases. If clustering algorithms are not seen as a viable method to reduce the size of the data, then one can turn to random projections as a way to reduce computational time in the KRLS algorithm.

Random projections(RP), is a dimensional reduction technique that has some similarities to Principal Components Analysis(PCA). PCA is a statistically optimal way of reduction in dimensions as it aims to project the data onto a lower dimensional orthogonal subspace that captures as much of the variation as possible[3]. However, PCA on the kernel matrix is computationally expensive for large samples sizes. In RP for KRLS, the n-dimensional kernel matrix, K, is projected onto a lower m-dimensional matrix, where $m \ll n$, using an $m \times p$ random matrix R, whose columns have unit lengths:

$$S^{RP}_{m\times n} = R_{m\times n} K_{n\times n} \tag{3.3}$$

The random projection's mathematical tractability is given by the Johnson-Lindenstrauss Lemma, which states that if points are in a high dimensional space of dimension n, then they can be projected into a lower-dimensional space of dimension m, while roughly preserving the distance between points [13].

Johnson-Lindenstrauss Lemma For any $0 < \varepsilon < 1$ and any integer n let k be a positive integer such that

$$k \geq \frac{24}{3\varepsilon^2 - 2\varepsilon^3} log\ n \tag{3.4}$$

Then for any set A of n points in R^d there exists a map $f : R^d \mapsto R^k$ such that for all $x_i, x_j \in A$

$$(1-\varepsilon)||x_i - x_j||^2 \leq ||f(x_i) - f(x_j)||^2 \leq (1+\varepsilon)||x_i - x_j||^2 \tag{3.5}$$

where Equation (3.5) states that any n point subset of Euclidean space can be embedded in a space with smaller dimensions without losing the information on the distances between any pairs of points by more than a factor of $1 \pm \varepsilon$.

I will choose between four different types of random projections : Gaussian, Achlioptas, Probability and Li.

Definition 3.3.1. (Gaussian Projection) The new $m \times n$ matrix, S, is formed by a matrix $R = \frac{1}{\sqrt{m}}G$, where each entry of G is sampled from an independent and identically distributed normal random variable and m is the number of dimensions you want the data reduced to.

Definition 3.3.2. (Achlioptas Projection) The new $m \times n$ matrix, S, is formed by a matrix R with each element of R begin chosen as follows:

$$r_{ij} = \sqrt{3} \begin{cases} +1 & \text{with probability } \frac{1}{6} \\ 0 & \text{with probability } \frac{2}{3} \\ -1 & \text{with probability } \frac{1}{6} \end{cases} \tag{3.6}$$

Definition 3.3.3. (Probability Projection) The new $m \times n$ matrix, S, is formed by a matrix R with each element of R begin chosen as follows:

$$r_{ij} = \begin{cases} +1 & \text{with probability } \frac{1}{2} \\ -1 & \text{with probability } \frac{1}{2} \end{cases} \tag{3.7}$$

Definition 3.3.4. (Li or Very Sparse Projection) The new $m \times n$ matrix, S, is formed by a matrix R with each element of R begin chosen as follows:

With $s \geq 3$

$$r_{ij} = \sqrt{s} \begin{cases} +1 & \text{with probability } \frac{1}{2}s \\ 0 & \text{with probability } 1-s \\ -1 & \text{with probability } \frac{1}{2}s \end{cases} \tag{3.8}$$

3.3.1 KRLS with Random Projection

The kernel regularized least squares algorithm can be altered slightly to accommodate sketching/random projection. Let $\alpha = S^T \omega$, use K to signify the Kernel, and S

being the random projection chosen beforehand, we again use the same procedure to find the optimal estimator.

$$\min_{\omega}(y - KS^T\omega)^T(y - KS^T\omega) + \lambda(S^T\omega)^T KS^T\omega$$

$$\frac{\partial}{\partial\omega}(y - KS^T\omega)^T(y - KS^T\omega) + \lambda(S^T\omega)^T KS^T\omega = 0$$

$$-2SK^T y + 2SK^T KS^T\omega + 2\lambda SKS^T\omega = 0$$

$$SK^T y = SK^T KS^T\omega + \lambda SKS^T\omega$$

$$SK^T y = (SK^T KS^T + \lambda SKS^T\omega$$

$$(SK^T KS^T + \lambda SKS^T)^{-1} SK^T y = \omega \tag{3.9}$$

$$f(x) = \sum_{i=1}^{n} \alpha_i K(\cdot, x_i) \tag{3.10}$$

where Equation (3.9) can be used to make predictions on new inputs in Equation (3.10), similar to the procedure in Equation (2.16).

Chapter 4
Simulation Study

The next chapter employs simulations of kernel regularized least squares with the data reduction techniques and studies the time and prediction performance of each procedure. All simulations are done using the Institute for Cyber Science's computing clusters available at the Pennsylvania State University.

4.1 Motivating Examples

For the first motivating example, I have run 15 simple simulations of kernel regularized least squares with a Gaussian kernel. The data has been generated from the following model.

$$y = sin(x) + cos(x) + \varepsilon, where\ \varepsilon \sim N(0,1) \tag{4.1}$$

I divided the data into train and test sets with a ratio of seventy to thirty percent. I clocked the amount of time it took for the algorithm to create a model from the train data and then use it to make predictions from the test data. In Figure 4.1 on the left side, I observe that as the data approaches 8,000 samples the algorithm appears to show an exponential nature in time complexity. I believe the culprit lies in the kernel matrix, as taking the inverse of X^TX, where X is 8000×8000 is very computationally intensive. For the second motivating example, I have simulated a dataset of size $n = 252$ using the same model from Equation (4.1). I again divide the data into train and test datasets with a ratio of seventy to thirty. I use the the KRLS algorithm with a Gaussian kernel and two different lambda and sigma pairs of $(0.0325, 0.0325)$ and $(1,1)$ to train a model and then try and recreate the model in Equation (4.1). In Figure 4.2, the first pair of lambda and sigma values is able to predict exactly where the data should lie while the second pair performs quite poorly in its ability to predict. It should be noted that the λ and σ values will play a great role in the search for a valid prediction model.

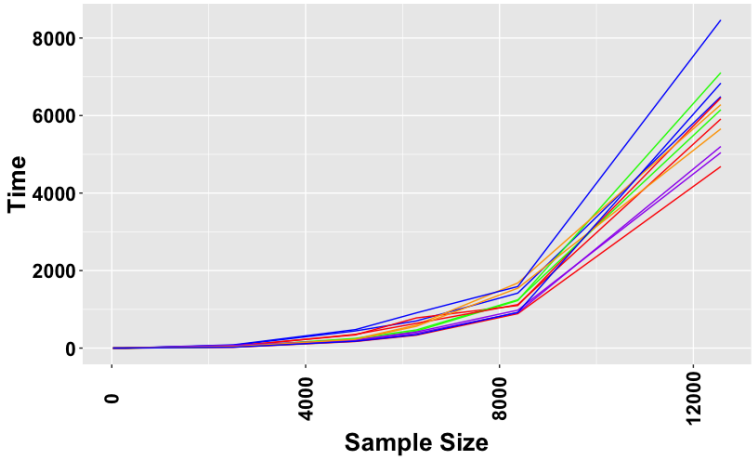

Figure 4.1: The explosion in computation time as sample size increases for KRLS algorithm. Twelve simulations with same λ and σ pairs. Once the sample size increases above 8000, the algorithm time increases in an almost exponential fashion.

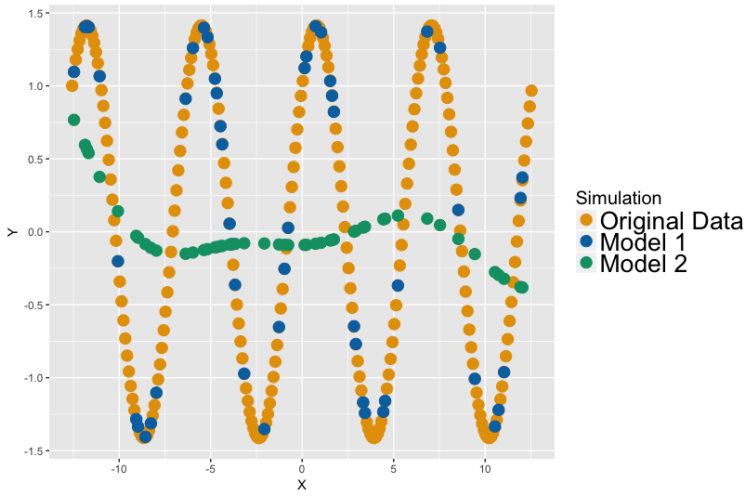

Figure 4.2: The KRLS algorithm depends greatly on the choice of λ and σ. The original data is plotted in yellow and two KRLS models are fitted using different λ and σ pairs. The blue model is able to make a perfect prediction while the green model is nowhere close to the original data.

4.2 Baseline Simulation

In order to compare the different data reduction techniques I will first establish a baseline to use for comparison. The kernel for kernel regularized least squares will be a gaussian kernel. The pairs of λ and σ go from 2^{-5} to 2^5, giving 121 pairs that can be used in the search for lowest MSPE. The data is generated from Equation (4.1) in one dimension and then scaled up to six dimensions.

Algorithm 5 Baseline for KRLS Sketch

1: **Inputs:**
   Train data, test data, 121 pairs of λ and σ
2: Divide data into 70 percent train and 30 percent test sets
3: Use five-fold cross validation on train set to find lowest MSPE for λ and σ pair
4: Folds are fitted using KRLS algorithm 2.
5: Use λ and σ pair that produces lowest MSPE through five fold on entire train set
6: Use fitted model from train data on test data
7: Record time and MSPE for test set

Table 4.1: Baseline Simulation

Dimension	**Lambda, Sigma**	**Time**	**MSPE**
1	0.03125, 0.03125	27.75	0.8×10^{-7}
2	0.03125,0.0625	38.08	1.8×10^{-6}
3	0.03125, 0.03125	39.08	1.8×10^{-6}
4	0.03125, 0.03125	42.11	2.1×10^{-6}
5	0.03125, 0.03125	43.01	3.2×10^{-5}
6	0.03125, 0.03125	48.25	2.9×10^{-3}

In Table 4.1, I observe that the Baseline Simulation shows that as the dimensions of the data increases the time the algorithm takes to build a model and do prediction also increases, which makes intuitive sense. At the same time, the MSPE aslo increases as the dimensions of the data increase.

4.3 Random Sampling

The first data reduction technique takes the data and randomly samples without replacement. I have provided a sketch of how this is done, but it very similar to the baseline algorithm with random sampling done on the training sets of data. My observations of Table 4.2 are that time is drastically reduced with random sampling without having to sacrifice too much in MSPE. However, as dimensions increases, MSPE becomes problematic. I believe that this technique will do well on data of low dimensions, but will perform poorly in higher dimensions. I also report the lambda and sigma pair. As the dimension increases the lambda and sigma pairs start to go over all over the place.

Algorithm 6 Random Sampling with KRLS Sketch

1: **Inputs:**
   Data, 121 pairs of λ and σ
2: Randomly divide data into 70 percent train and 30 percent test sets
3: Use five-fold cross validation on train set to find lowest MSPE for λ and σ pair
4: Folds are fitted using KRLS algorithm 2 with random sampling of 5 percent to 75 percent of the train data.
5: Use λ and σ pair that produces lowest MSPE through five fold on entire train data with corresponding random sampling size.
6: Use fitted model from train data on test data
7: Record time and MSPE for test data.

Table 4.2: Simulations using Random Sampling

Dimensions	Sample	Lambda, Sigma	Time	MSPE	Dimensions	Sample Size	Lambda, Sigma	Time	MSPE
1	5%	0.03125, 0.0625	0.025	3.5×10^{-6}	4	5%	0.03125,0.03125	0.2	2.2×10^{-3}
	10%	0.03125,0.03125	0.138	1.7×10^{-5}		10%	0.5, 0.03125	2.9	1.9×10^{-2}
	25%	0.03125,0.03125	1.07	$3.9 \times 10-6$		25%	0.0625, 0.03125	10.8	0.21
	30%	0.03125,0.03125	3.28	2.3×10^{-6}		30%	1, 0.5	47.5	0.25
	50%	0.03125,0.03125	7.66	4.03×10^{-6}		50%	0.03125,0.03125	78.11	0.8
	75%	0.03125,0.0625	22.17	3.56×10^{-6}		75%	0.03125,0.03125	105.3	0.8
2	5%	0.03125, 0.0625	0.12	1.1×10^{-4}	5	5%	0.03125, 1	10.2	0.87
	10%	0.03125, 0.125	0.51	2.5×10^{-3}		10%	2, 0.03125	9.2	0.21
	25%	1, 0.03125	1.88	1.8×10^{-4}		25%	0.03125, 0.5	57.2	0.62
	30%	0.03125,0.5	5.12	2.3×10^{-3}		30%	0.03125, 0.03125	102.2	0.58
	50%	0.03125, 0.0625	8.14	1.9×10^{-3}		50%	2, 16	201.4	0.74
	75%	1, 0.03125	8.49	1.6×10^{-2}		75%	16, 0.03125	301.8	0.73
3	5%	0.03125, 0.03125	1.5	2.2×10^{-3}	6	5%	1, 0.03125	28.2	0.64
	10%	1, 0.03125	1.8	2.9×10^{-4}		10%	0.03125, 1	35.7	0.5
	25%	2, 0.03125	15.4	3.2×10^{-4}		25%	0.5, 0.03125	104.5	0.74
	30%	0.03125, 0.5	25.4	1.2×10^{-5}		30%	0.03125, 2	187.1	0.87
	50%	0.03125, 0.0625	201.4	7.5×10^{-6}		50%	0.03125, 1	253.6	0.88
	75%	0.5, 0.5	351.1	2.3×10^{-6}		75%	16, 32	321.2	1.51

4.4 Clustering

I use the k-means clustering technique for my simulated data. Similar to the previous algorithm sketch, but with random sampling switched to k-means clustering. My observations are that k-means performs much better in dimensions of 1 and 2, but produces similar results as the algorithm goes into higher dimensions.

Algorithm 7 K-means with KRLS Sketch

1: **Inputs:**
   Data, 121 pairs of λ and σ
2: Randomly divide data into 70 percent train and 30 percent test sets
3: Use Five-fold cross validation on train set to find lowest MSPE for λ and σ pair
4: Folds are fitted using KRLS, algorithm 2, with k-means algorithm, algorithm 3, with clusters set from 5 percent to 75 percent of the train data.
5: Use λ and σ pair that produces lowest MSPE through five fold on entire Train data with corresponding cluster size.
6: Use fitted model from train data on test data
7: Record time and MSPE for test data.

Table 4.3: Simulations using K-means Clustering

Dimensions	Cluster Size	Lambda, Sigma	Time	MSPE	Dimensions	Cluster Size	Lambda, Sigma	Time	MSPE
1	5%	0.03125, 0.0625	0.077	1.5×10^{-4}	4	5%	0.03125,0.5	0.2	3.1×10^{-3}
	10%	0.03125,0.03125	0.199	2.3×10^{-5}		10%	0.03125, 0.03125	1.5	1.7×10^{-3}
	25%	0.03125,0.03125	1.22	$4.2 \times 10-6$		25%	0.0625, 0.03125	10.47	0.0333
	30%	0.03125,0.03125	3.39	5.5×10^{-6}		30%	1, 0.03125	38.5	0.014
	50%	0.03125,0.03125	7.566	3.6×10^{-6}		50%	0.03125,0.03125	68.11	0.78
	75%	0.03125,0.0625	26.43	2.7×10^{-6}		75%	0.03125,0.03125	98.2	0.89
2	5%	0.03125, 0.0625	0.062	1.1×10^{-4}	5	5%	0.03125, 0.03125	9.8	0.77
	10%	0.03125, 0.125	0.17	5.5×10^{-5}		10%	0.03125, 0.03125	8.5	0.021
	25%	0.03125, 0.03125	1.28	9.8×10^{-6}		25%	0.03125, 0.5	37.4	0.11
	30%	0.03125,0.03125	4.82	2.3×10^{-6}		30%	0.03125, 0.03125	62.98	0.36
	50%	0.03125, 0.03125	9.31	1.9×10^{-6}		50%	2, 0.03125	103.5	0.41
	75%	0.03125, 0.03125	4.785	1.6×10^{-6}		75%	0.5, 0.5	118.2	0.58
3	5%	0.03125, 0.03125	0.08	0.2×10^{-3}	6	5%	0.03125, 0.03125	22.5	0.55
	10%	0.03125, 0.03125	0.25	1.9×10^{-4}		10%	0.03125, 0.03125	41.3	0.04
	25%	0.03125, 0.03125	2.5	1.2×10^{-5}		25%	0.03125, 0.03125	98.2	0.87
	30%	0.03125, 0.5	7.99	1.2×10^{-5}		30%	0.03125, 0.03125	105.3	0.85
	50%	0.03125, 0.03125	18.86	1.5×10^{-6}		50%	0.03125, 0.03125	187.2	0.98
	75%	0.03125, 0.03125	66.12	2.4×10^{-6}		75%	0.03125, 0.03125	275.2	0.99

4.5 Random Projection

Random projection uses a different technique than random sampling or k-means. The first two techniques involved finding representations of the data and feeding

it into the KRLS algorithm. For random projection, the data is fed into the kernel function to create the kernel matrix. The kernel matrix is then sampled, based on one of the four projection schemes outline in Section 3.3. Unfortunately, random projection performs quite poorly even in one dimension. Perhaps this is due to how the simulated data was constructed and warrants more exploration. I will use it on real-data to see if the results continue.

Algorithm 8 Random Projection with KRLS Sketch

1: **Inputs:**
   Data, 121 pairs of λ and σ
2: Randomly divide data into 70 percent train and 30 percent test sets
3: Use five-fold cross validation on train set to find lowest MSPE for λ and σ pair
4: Folds are fitted using KRLS, but now the kernel matrix is randomly sampled after it is created based on sampling procedure outlined in chapter 3.
5: Use λ and σ pair that produces lowest MSPE through five fold on entire train data with corresponding cluster size.
6: Use fitted model from train data, with modified kernel matrix, on test data
7: Record time and MSPE for test data.

Table 4.4: Simulations using Random Projection

Dimensions	Projection	Lambda, Sigma	Time	MSPE	Dimensions	Projection	Lambda, Sigma	Time	MSPE
1	Achlioptas	16, 0.0625	12.8	0.12	4	Achlioptas	2, 0.125	20.2	0.18
	Probability	0.03125,0.03125	17.2	0.53		Probability	1, 0.03125	28.2	0.12
	Li	0.03125,0.5	28.31	0.61		Li	0.0625, 16	29.2	0.52
	Gaussian	0.03125,0.03125	36.62	0.72		Gaussian	1, 2	37.4	0.78
2	Achlioptas	2, 0.25	13.45	0.32	5	Achlioptas	0.03125, 0.03125	22.1	0.24
	Probability	1, 0.0625	16.64	0.14		Probability	2, 0.125	27.8	0.27
	Li	0.03125, 32	12.46	0.17		Li	0.03125, 0.5	27.5	0.68
	Gaussian	0.03125, 0.0625	17.72	0.41		Gaussian	1, 8	37.8	0.9
3	Achlioptas	0.03125, 0.0625	22.8	0.25	6	Achlioptas	32,32	40.1	0.91
	Probability	0.03125, 0.03125	27.3	0.67		Probability	16,2	52.7	0.97
	Li	2, 0.125	22.8	0.41		Li	1, 0.03125	47.8	0.98
	Gaussian	0.03125, 0.125	23.68	0.48		Gaussian	2, 16	51.4	1.57

4.6 Summary

Table 4.5 gives the simulation results for each best performing method for each dimension with a comparison to the original baseline. It appears that k-means is the best procedure to use within the KRLS framework with random sampling a not too distant second. Random projection preforms very poorly, which the way it works led me to believe that it would perform well, but this could be due to how the simulated data was constructed and warrants more simulations. Of particular interest, is that the k-means procedure only needs ten percent of the data to get a fairly decent MSPE with a substantial decrease in time for the algorithm. Moving the amount of clusters from ten percent to somewhere below twenty-five percent could get us more reduced MSPE without increasing the time too much. The λ and σ values are stable in the lower dimensions, but become erratic in higher dimensions. In Chapter 5, I will continue to do a grid search of lambda values from 2^{-5} to 2^{5}, but will use a different technique for sigma values. The sigma will now be chosen to ensure that the kernel matrix carries useful information about the data. The method is suggested by Scholkopf and Smola, which is $\sigma^2 = dim(X)$ and X is the data.

Table 4.5: Best performing method under Simulations

Dimensions	Best Method	Lambda, Sigma	Proportion of Data	Time	MSPE	Comparison To Baseline Time, MSPE
1	Random Sampling	0.03125,0.0625	5%	0.025	3.5×10^{-6}	27.75,0.8×10^{-7}
2	K-means	0.03125,0.125	10%	0.17	5.5×10^{-5}	38.08, 1.8×10^{-6}
3	K-means	0.03125, 0.03125	50%	18.86	1.5×10^{-6}	39.08, 1.8×10^{-6}
4	K-means	0.03125, 0.03125	10%	1.5	1.7×10^{-3}	42.11, 2.1×10^{-6}
5	K-means	0.03125, 0.03125	10%	8.5	0.021	43.01, 3.2×10^{-5}
6	K-means	0.03125, 0.03125	10%	41.3	0.04	48.25, 2.9×10^{-3}

Chapter 5
Application

For this chapter, I apply the three techniques to reduce computation time in the KRLS framework to real-world datasets. The five datasets I have chosen are available from the University of California Irvine Machine Learning Repository. Due to the limitations of the algorithm I will only work with fully numerical datasets, which greatly reduce the datasets that I can explore. I provide a brief overview for each dataset and then provide results. All data analysis is done using the Institute for Cyber Science's computing clusters available at Pennsylvania State University.

5.1 Overview

As stated above, I look at 5 datasets: Concrete, Airfoil, Wine Quality, Naval Plant and Appliance Energy. Each dataset has only numerical variables and responses. The datasets go in increasing order of sample size with Concrete being the smallest of about 1000 observations with the Appliance Energy set going up to about 20,000 observations.

5.1.1 Concrete

The concrete dataset has 1030 observations of 8 variables with the response being the concrete compressive strength. This is a small dataset, but the author has found that concrete compressive strength to be a highly nonlinear function of age and its ingredient, which I feel makes it a prime candidate for the nonlinear techniques presented in this thesis [22].

5.1.2 Airfoil

The Airfoil dataset has 1503 observations of 5 variables with the response being the scaled sound pressure in decibels. The data set has different measurements on the National Advisory Committee for Aeronautics (NACA) 0012 airfoils at various wind tunnel speeds and angles of attack. 0012 refers to the formula needed to

construct this particular airfoil. The span of the airfoil and the observer position were the same in all of the experiments [4].

5.1.3 Wine Quality

Next is the classic Wine Quality datasets from Portugal, which is really two datasets that are related to red and white vinho verde wine samples. The dataset has 4898 observations of 11 variables that are physiochemical tests with the response being the quality of wine [6]. I expect the computation time to increase substantially for this data. Referring back to Figure 4.1, the naive KRLS algorithm computation time started to increase exponentially after 5000 samples.

5.1.4 Naval Plant

The Naval Plant dataset has 11934 observations of 14 variables with two responses being the state of decay of the compressor and the turbine. I focus only one of the two response, which will be the turbine. The data was generated from a sophisticated simulator of a Gas Turbines (GT), mounted on a Frigate characterized by a combined diesel electric and gas propulsion plant type. The experiments have been carried out by means of a numerical simulator of a naval vessel (Frigate) characterized by a Gas Turbine (GT) propulsion plant [2].

5.1.5 Appliance Energy

The final dataset is the Appliances energy prediction set. It has 19735 observations with 28 variables with the response being the energy use of a household appliance. The description is as follows: The data set is at 10 min for about 4.5 months. The house temperature and humidity conditions were monitored with a ZigBee wireless sensor network. Each wireless node transmitted the temperature and humidity conditions around 3.3 min. Then, the wireless data was averaged for 10 minute periods. The energy data was logged every 10 minutes with m-bus energy meters. Weather from the nearest airport weather station (Chievres Airport, Belgium) was downloaded from a public data set from Reliable Prognosis (rp5.ru), and merged together with the experimental data sets using the date and time column [5].

5.2 Results

As in the simulation study, I compare only the Time and Mean Squared Prediction Error (MSPE) for the Test set of the different methods. To have a firmer footing, I have found the Time and MSPE for Ridge Regression for each dataset. Airfoil was problematic in that its data is highly nonlinear and gave back an unwieldy MSPE for the ridge regression. All 5 datasets benefit from using the KRLS method in order to reduce MSPE, but it does significantly increase the time moving from Ridge to KRLS.

5.2.1 Summary of Results:

In Table 5.1, I have run ridge and kernel regularized least squares, a.k.a Baseline, to use as a benchmark. Observe that the random sampling method of 5 percent to 75 percent of the data reduces computational time, but appears to have a great cost in increasing the MSPE. Only in the naval plant dataset did the method perform better than the *k*-means method. In Table 5.2, observe that *k*-means has the best results of the three methods. Not only is the method able to significantly reduce the computation time of the KRLS algorithm, but its MSPE is also able to hover around the Baseline MSPE. However, from my observations the amount of clusters you specify, being 10 to 30 percent, seems to help reduce the MSPE and Time while the rest are of a mixed bag. This came out in the simulations, i.e., only 10 to 30 percent of the data is needed for the KRLS algorithm to work well. Finally, in Table 5.2 are the four projection methods that reduce the size of the kernel matrix. The methods helps to reduce computational time, but have incredibly poor performance in terms of MSPE. I observe that the projection methods perform worst than the ridge technique for all datasets.

Table 5.1: Results of Random Sampling with Ridge and Baseline as Comparison

Dataset *n,p*	**Ridge** **Time** **MSPE**	**Baseline** **Time** **MSPE**	**Random Sampling** **Time: 5%,10%,25%,30%,50%,75%** **MSPE: 5%,10%,25%,30%,50%,75%**
Concrete	0.008	7.40	Time:, 4.43, 4.22, 5.87, 6.81, 7.01, 7.31
1030,9	3075	3054	MSPE: 3912, 4815, 3874, 3741, 3774, 3571
Airfoil	0.82	17.21	Time: 14.88, 0.11, 0.68, 0.95, 4.47, 14.88
1503,6	90258.71	15585.92	MSPE:15801.11,16731.22,15596.64,17568.20,18588.75,17551.55
Wine Quality	0.037	92.7	Time: 9.4, 4.5, 64.2, 109.3, 203.6, 334.8
4898,12	31.89	24.22	MSPE: 41.4 , 55.7, 38.3, 54.2, 60.2, 47.4
Naval Plant	0.211	10.27	Time: 18.2, 28.9, 37.1, 38.4, 41.6, 42.8
11934, 16	30.12	2.71	MSPE: 21.2 , 25.7, 18.3, 24.7, 36.1, 47.4
Appl.	10.87	87.44	Time: 35.3, 41.1, 52.2, 58.1, 54.7, 64.9
19735, 29	5482	5001	MSPE: 5874, 6241, 5914, 5741, 6417, 5991

Table 5.2: Results of K-means and Projection with Ridge and Baseline as Comparison

Dataset *n,p*	**Ridge** **Time** **MSPE**	**Baseline** **Time** **MSPE**	**K-means Clustering** **Time: 5%,10%,25%,30%,50%,75%** **MSPE: 5%,10%,25%,30%,50%,75%**	**Projection** **Ach,Li,Prob,Gaussian**
Concrete	0.008	7.40	Time: 0.04, 0.06, 0.30, 0.46, 1.95, 5.50	Time: 0.51, 0.2, 0.63, 1.91
1030,9	3075	3054	MSPE: 2932, 2755, 2676, 2677, 2631, 2614	MSPE: 3147, 3257, 3417, 3549
Airfoil	0.82	17.21	Time: 14.62, 0.1, 0.82, 1.03, 4.36, 14.95	Time: 0.61, 0.21, 0.78, 2.43
1503,6	90258.71	15585.92	MSPE:15607.73,15790.62,15526.64,15563.20,15585.99,15592.55	MSPE:16401.77,18490.62,17426.64,17763.20
Wine Quality	0.037	92.7	Time: 0.5, 0.60, 1.01, 1.41, 2.79, 7.39	Time: 1.87, 1.64, 1.47, 5.43
4898,12	31.89	24.22	MSPE: 32.34, 32.17, 32.09, 31.88, 32.04, 32.23	MSPE: 41.35, 52.16, 51.1, 81.1
Naval Plant	0.211	10.27	Time: 9.15, 4.30, 6.42, 6.37, 10.87, 25.89	Time: 8.25, 8.20, 7.12, 15.17
11934, 16	30.12	2.71	MSPE: 2.89, 3.12, 3.57, 3.87, 10.22, 10.47	MSPE: 6.1, 6.11, 5.34, 13.86
Appl.	10.87	87.44	Time: 30.22, 28.47, 28.45, 27.11, 50.1, 68.4	Time: 41.23, 42.36, 48.46 98.75
19735, 29	5482	5001	MSPE: 5103, 5087, 5041, 5121, 5122, 5101	MSPE: 6301, 7281, 71421, 9523

Chapter 6
Conclusions and Future Work

Kernel Regularized Least Squares offers researchers both flexibility and interpretability. KRLS does this by starting from the idea of similarity of observations rather than fitting a linear combination of the regressors. However, due to the space and time complexity of working with an $n \times n$ kernel matrix, where n is the size of the sample, the practical limits of working within the KRLS framework will hit most users at observations of tens of thousands. Therefore, finding techniques that can overcome this information bottleneck in the kernel matrix is of utmost importance.

In this thesis, I have compared three different data reduction methods: Random Sampling, K-means Clustering and Random Projection. The first two methods involved processing the data before feeding it into the KRLS algorithm. The third method, random projection, first fed the data into the algorithm, but sought to reduce the size of the kernel matrix. I compared the three methods by looking at the computational time and the MSPE. I found that the random projection performed the worst of the three methods, while k-means clustering performed the best.

Future work would need to look at other types of simulations other than my Equation (4.1) and building up to much more complicated data structures. Perhaps these simulations would see an improvement on the random projection technique or at least it would not fair so poorly compared to random sampling and k-means. I also believe that other types of clustering need to be investigated such as hierarchical clustering to again deal with other types of data structures, e.g. Figure 3.2. Also, the KRLS method has difficulty with categorical data and I only explored datasets that were strictly numerical. Future work would need to look into adding this feature into the algorithm in order to give researchers more options when deciding between complicated machine-learning algorithms.

Bibliography

[1] D. Achlioptas. *Database-friendly random projections: Johnson-Lindenstrauss with binary coins.* Journal of Computer and System Sciences 66 671–687, 2003.

[2] M. Altosole, G. Benvenuto, M. Figari, U. Campora. *Real-time simulation of a cogag naval ship propulsion system*, Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment 223 (1) 47-62, 2009.

[3] E. Bingham and H Mannila. . *Random Projection in dimensionality reduction: Applications to image and text data*. KDD 2001 Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, pages 245-250. 2001.

[4] T.F. Brooks, D.S. Pope, and A.M. Marcolini. *Airfoil self-noise and prediction.* Technical report, NASA RP-1218, 1989.

[5] L. Candanedo, V. Feldheim, D. Deramaix. *Data driven prediction models of energy use of appliances in a low-energy house*, Energy and Buildings, Volume 140, Pages 81-97, 2017.

[6] P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. *Modeling wine preferences by data mining from physicochemical properties.* In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

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

[8] R. A Fisher. *The use of multiple measurements in taxonomic problems* Annals of Eugenics, 7(2):180-188, 1936.

[9] J. Hainmueller and C. Hazlett. *Kernel Regularized Least Squares: Reducing Misspecification Bias with a Flexible and Interpretable Machine Learning Approach*. Oxford University Press, 2013.

[10] J. Han and M. Kamber. *Data Mining: Concept and Techniques*. Academic Press, 2001.

[11] T. Hastie, P. Li, and K. Church. Very sparse random projections. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '06). ACM, New York, NY, USA, 287-296, 2006.

[12] G. Kimeldorf and G. Wahba. *Some Results on Tchebycheffian Spline Functions*. Journal of Mathematical Analysis and Applications Vol 33 Jan 1971. pp. 82-95, 1971.

[13] J. Lindenstrauss and W. Johnson. *Extensions of Lipschitz mappings into a Hilbert space*. Conference in Modern Analysis and Probability (New Haven, Conn., 1982). Contemporary Mathematics. 26. Providence, RI: American Mathematical Society. pp. 189–206, 1984.

[14] V. Mathews and G.Sicuranza. *Polynomial signal processing*. Addison-Wesley, New York: Wiley, 2000.

[15] M. Olazaran. *"A Sociological Study of the Official History of the Perceptrons Controversy"*. Social Studies of Science. 26 (3): 611–659. doi:10.1177/030631296026003005. JSTOR 285702, 1996.

[16] F. Rosenblatt. *Principles of Neurodynamics*, Spartan Books, New York. 1962.

[17] B. Scholkopf and a. Smola. *Learning with Kernels*, Massachusetts Institute of Technology, 2002.

[18] B. Scholkopf, B. Ralf and A. Smola. *A Generalized Representer Theorem*. Computational Learning Theory, 14th Annual Conference on Computational Learning Theory, COLT 2001, 416-426, 2001.

[19] B. Sriperumbudur and G. Lanckriet. *Nearest Neighbor Prototyping for Sparse and Scalable Support Vector Machines*, Technical Report, 2007.

[20] A. Tikhonov, A. Goncharsky, V. Stepanov, A. Yagola. *Numerical Methods for the Solution of Ill-Posed Problems*. Springer Science and Business Media, 1995.

[21] V. Vapnik and C. Cortest. *Support-Vector Networks* Machine Learning, 20, 273-297. 1995.

[22] I-Cheng Yeh, *"Modeling of strength of high performance concrete using artificial neural networks,"* Cement and Concrete Research, Vol. 28, No. 12, pp. 1797-1808 (1998).