Investigation of Extending Feature Selection Algorithms to Explicit Feature Selection in Kernel Space

Qiaozhi Li

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Computer Science & Systems

University of Washington

2018

Committee:

Jie Sheng

Raghavi Sakpal

Donald Chinn

Chunming Gao

Program Authorized to Offer Degree:
Computer Science & Systems

University of Washington

Abstract

Investigation of Extending Feature Selection Algorithms to Explicit Feature Selection in Kernel Space

Qiaozhi Li

Chair of the Supervisory Committee:
Jie Sheng
Institue of Technology

Feature selection methods play important roles in the area of machine learning. Being a part of prepossessing, the technology of feature selection can select useful information from raw data. A good feature selection method can significantly improve performance of a prediction model. However, most feature selection methods only work well with linear data. Although nonlinear data can be transformed into linear data by being projected into a high dimensional space, the computation cost of calculating in high dimensional space is quite high.

Kernel Trick is a method in model training. It reduces greatly the computational cost of calculating the inner product of high dimension data, and thus is usually used to solve nonlinear problems. However, data in the high dimension space generated by Kernel Trick, the so called Kernel Space, are usually implicit. Therefore, most feature selection methods cannot use Kernel Trick to process nonlinear data. Cao et al. provided a new method that can explicitly select features in Kernel Space with limited additional cost by extending a famous margin-based feature selection method *Relief*. Inspired by their results, we propose a method that transforms the original space to a so called **Explicit Kernel Space** (EKS). Our method successfully extends and broadens the idea by Cao et al. With EKS, most traditional feature selection algorithms can use Kernel Trick to deal with real world data.

Based on several tests with different types of data and algorithms, the usefulness of EKS is verified; some of its properties are also presented and discussed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
INTRODUCTION

1.1 Motivation

In the areas of machine learning and pattern recognition, a feature is an individual measurable property or characteristic of a phenomenon being observed[1]. A sample set with features of a phenomenon is usually called *data*. Features are very important because most machine learning models are in fact a "combination" of features. The quality of features used to train a model can directly influence the model's accuracy. Meanwhile, the number of features can directly determine the computational time of a model's training. Nowadays, feature selection has become a necessary step to build a high performance machine learning model.

Feature selection is the process of finding good features. Most feature selection algorithms can be classified into two categories. One group assumes that good features can be generated from original features. These algorithms view the data as in a vector space and uses linear algebra methods to process data. In most cases, these algorithms will generate new features from raw data and thus being named feature extraction algorithms. PCA[2], SVD[3] and LDA[4] are all in this group. The other group originates from statistics and information theory. This group regards good features as a subset of original features. This type of algorithms focus on the original features. They always use some algorithms to measure the importance of each feature and select the feature with the most significance. In this thesis we focus on algorithms belonging to this group. We classify these algorithms into three types:

1. Score and null hypothesis based algorithms. These include Chi score [5], Mutual information [6], t-test and f-test [7]. These algorithms usually measure the importance of each feature separately and give a score for each feature based on the importance.

2. Model based algorithms. These include decision tree [8], linear regression [9], LARS [10] and LASSO [11]. These algorithms usually use coefficients of a regression model trained by the original data to be the score of each feature.

3. Distance based algorithms. These include Relief [12] and corresponding margin-based algorithms, Laplacian score [13] and some feature selection algorithms based on graph theory. These algorithms usually use distance between data points to measure the importance of each feature.

There is an assumption behind all the traditional feature selection algorithms listed above, i.e., the perfect model is linear for regression problem, or the data can be linear-divided for classification problem. However, many real-world problems are nonlinear and traditional algorithms are not capable of handling nonlinearities due to the linearity assumption. To solve this problem, a technique called Kernel Trick is introduced into machine learning algorithms. Kernel Trick will project data into an implicit high dimension space called "Kernel Space" to transform nonlinear problems into linear with little additional computing cost.

With Kernel Trick, machine learning models can work with nonlinear problems, but the feature selection algorithms are still difficult to work with Kernel Trick because the Kernel Space is implicit and there are either no explicit features or with infinite features to select. To select features in Kernel Space, some relevant feature selection algorithms were proposed. Kernel Principle Component Analysis (KPCA) [14], Kernel Fisher Discriminant Analysis (KFD) [15] and Generalized Discriminant Analysis (GDA) [16] are examples in this pool. However, these algorithms have their limitations. KPCA cannot make use of data label. KFD and GDA can only generate at most N-1 number of features when N is the number of categories.

Kernel Relief [17] was proposed as a solution to explicitly select features in the Kernel Space. It uses Kernel Gram-Schmidt Process (KGSP) to construct a basis set first in the Kernel Space and then extends *Relief* to select features explicitly in the Kernel Space. In-

spired by *Kernel Relief*, we aim to investigate a method that can extend feature selection algorithms to let them select features directly in the Kernel Space.

1.2 Preliminaries

In this section, we will briefly introduce certain background of our research.

1.2.1 Feature Selection and Feature Extraction

Feature, sometimes called variable or attribute, is an important concept in the area of machine learning. For a given set of data, the attributes of a data form their feature set. For example, "student" is a type of data , then each student might have four attributes: "name", "age", "gender" and "birthday". These four attributes form the feature set of "student". If we call one of these attributes, say, "name" as a "label", and try to use remaining features, i.e., "age" together with "gender" and "birthday" to predict the label, we then have a prediction problem. If we use a computer to solve this problem, then it is a machine learning problem.

Generally there are two types of prediction problems: classification problem and regression problem. The label of classification problem is a set of categories, for example, "yes" or "no". The label of regression problem is a continuous variable. To solve a prediction problem, we usually use available features to build a function or an equation. The function refers to the relationship between features and labels. We usually call the function prediction model or machine learning model. The process of building a prediction model is also called model training.

The goal of training a model is to find a set of coefficient to maximize a target function. Such target functions are measurement of predict results. There are lots of different measurement. The simplest measurements are error rate[1] for classification problem and mean

[1]Error rate = Num of Errors / Num of Samples

squared error[2] for regression problem.

If we view data as a mathematical description of real world objects, then a feature of data is a math description of an aspect of that object. No matter which aspect of the object we want to predict, the remaining aspects, or, the features, are the only thing we can use to train a prediction model. That explains why the quality of features will decide the performance of a model training.

Note that most of time, the features of data collected from real world are not perfect for training a prediction model, due to the following reasons:

1. Number of features: too many features cause an unacceptable training time cost.

2. Redundant features: some features have close meaning for a prediction target, for example, in most cases, "birthday" and "age" are highly related, i.e., they are redundant; and we may only need one of them to train the model.

3. Irrelevant features: some features are irrelevant to the prediction target. For example, if we want to predict "age", the feature "gender" is likely to be irrelevant. And if we want to train a model to predict "age", the feature "name" is likely to be useless.

To solve these issues, a variety of feature selection and extraction algorithms have been proposed. Note that the term "feature extraction" is often used in area of image processing, which need to generate numeric features from non-numeric data like images or texts. However, some of these feature extraction algorithms can also work as feature selection algorithms to select features because they will give a score to each extracted feature to measure the importance of them. The most significant difference between feature selection and feature extraction is that feature selection produces a subset of the original features, whereas feature extraction generates a new set of features from the original features.

---

[2]The MSE is defined as $MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i')^2$, the smaller the better. Y is the prediction result and Y' is the correct value

Basically, a process of feature selection contains two parts. One is to find a subset from features. The other is a measurement of these subsets. The simplest feature selection is to test every possible subset of features and find the subset that has best error rate or best mean squared error. Based on the combination of different methods to find a subset and different methods to score the result, there are mainly three types of feature selection methods:

- *Filter Methods* Filter methods measure the importance of each feature and find a subset of features before any training. Most filter methods work with a measurement of correlation between a feature and the label; and exclude features which have low correlation score, including some famous algorithms, such as Mutual Information, Pearson Correlation. Filter methods often have good computational performance and can find a general feature subset for different prediction models. However, because filter methods only use the correlation between feature and label, they often cannot find the best feature subset. Also, it is hard for filter methods to solve the redundancy problem, because features with high scores may be correlated to each other.

- *Wrapper Methods* Wrapper methods measure the performance of feature subsets after training and select the feature subset which has the best performance after training. The stepwise feature selection [18] is the most famous wrapper method. Both forward stepwise method and backward stepwise method find best subset of features based on performance of model trained by the subset. Wrapper methods usually find the best feature subset for a particular learning model. However, wrapper methods usually have a high computational cost and may cause overfitting[3].

- *Embedded Methods* Filter methods and Wrapper methods can be easily combined because their measures happen in different phases. The embedded methods try to combine advantages of filter methods and wrapper methods. LASSO [11] is one of this

[3]Overfitting is "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably"

type. When using LASSO to select features for linear regression, the process can be seen as two steps. First, use least-squares approximation (error rate) to give a score for each feature. Second, use L1-norm to suppress the features and penalize the small scores to 0.

Like traditional machine learning models, these traditional feature selection methods are also based on linear assumptions. Thus they work poorly on nonlinear problems. Furthermore, the famous nonlinear solution Kernel Trick for learning models does not work with feature selection algorithms. Before further discussion, we now introduce the Kernel Trick.

1.2.2 High dimension data projection and Kernel Trick

We want to clarify first that the Kernel Trick is not a solution for nonlinear problems but a solution for "high computational cost of inner product of high dimensional data points". A nonlinear problem is transformed into a linear problem by "projecting data into a high dimensional space". A simple example can show how this transformation works.

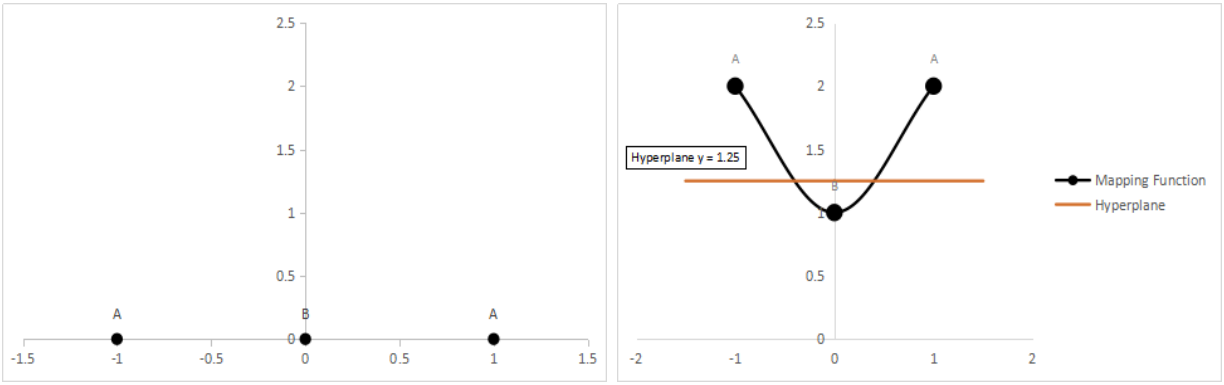


Figure 1.1: Example of high dimension projection

Assume there are three one-dimension data points (-1), (0), (1), with corresponding labels "A", "B", and "A". Obviously, we cannot find a one-dimension line $x = k$ to divide these data points into two parts perfectly. Because no matter what k is, the line cannot perfectly divide three data points into two parts, where one part only has one label inside.

However, when we project these three points into a two-dimension space with mapping function $\phi(x) \Rightarrow (x, x^2+1)$, then the three points become (-1,2), (0,1), and (1,2), correspondingly. We can find a hyperplane to classify data points easily such as the one in Figure 1.1. Note that y = 1.25 is one such potential hyperplane. Projecting data into high dimensional space allows us to find a linear solution for a nonlinear problem in the original space. In fact, if the dimension of space where data is projected into is high enough, any nonlinear problem can be solved by linear algorithms. A large number of dimensions cause high computational cost, though, especially for some similarity computation[4]. Kernel Trick was introduced to machine learning to solve the high computational cost problem.

Basically, Kernel Trick is a method that uses Kernel Function $K(x_1, x_2)$ to calculate the inner product of two high dimensional data points $\phi(x_1)$ and $\phi(x_2)$, as Equation 1.1 indicates.

$$K(x_1, x_2) =< \phi(x_1), \phi(x_2) > . \tag{1.1}$$

Here ϕ is a mapping function that projects data point x_i in the original space to a high dimensional data point $\phi(x_i)$. We call the space generated by $\phi(x_i)$ the Kernel Space.

Theoretically each Kernel Function involves a relevant mapping function $\phi(x)$. However, in the real training of learning models using Kernel Function, there is in fact no need to know what the explicit form of $\phi(x)$ is. Thus we often use the term *implicit* meaning "unknown".

For any function $K(x_1, x_2)$, if it satisfies *Mercer's condition*[5], it can be a Kernel Function. Although Equation 1.1 tells us that there is a corresponding $\phi(x)$ for that Kernel Function, however, the inner product in the data space generated by $\phi(x)$ can still be calculated even if $\phi(x)$ is unknown, since we can judge $K(x_1, x_2)$ by checking with the Mercer's condition instead of knowing about $\phi(x)$.

In short, a Kernel Function $K(x_1, x_2)$ which satisfies the Mercer's condition gives us a way to calculate the inner product between two high dimensional data points $\phi(x_1)$ and

[4]The similarity is usually measured by some type of distance, such as Euclidean distance.

[5]Mercer's condition: if a function $K(x_1, x_2)$ is positive-semidefinite, then $K(x_1, x_2)$ is a Kernel Function. See *https://en.wikipedia.org/wiki/Mercer*

$\phi(x_2)$. Neither the explicit form of $\phi(x)$, nor the dimension number (even when it is infinite) matters, in computing $K(x_1, x_2)$. That is how Kernel Trick got its name. Due to this reason, when we use Kernel Trick to train a model in a high-dimension space, i.e., the Kernel Space, we in fact have no knowledge about the space (or say, the Kernel Space is implicit). The computation cost of inner product is purely decided by the Kernel Function $K(x_1, x_2)$.

Implicit $\phi(x)$ also means that features in Kernel Space are implicit, so we can not perform feature algorithms directly in Kernel Space unless there is an explicit high dimension space with same property of Kernel Space.

1.3 Contribution

The main contribution of this research is that we provide a method to transform an original space into a so-called Explicit Kernel Space (EKS). Notice that EKS is an approximation of Kernel Space. With this proposed method, most feature selection methods can now select features from EKS, which can be seen as selecting features from Kernel Space.

We want to emphasize that our work is not a simple extension of that in Cao et al.[17] which dealt with specific feature selection methods. Our work focuses on and anwsers the following questions: does there exists an explicit Kernel Space so that most feature selection methods can work on that? If yes, what properties this space have? how can we derive this space? We are very proud that we have found answers to these questions, and will present them in this thesis. Specifically our research contributions are:

- We give a definition and specify the requirements of EKS.
- We provide a theoretical proof that EKS is a valid approximation of Kernel Space.
- We provide a theoretical proof on an important property of EKS: the inner product of data points in EKS can be calculated by Kernel Function.
- We present a general process to first transform an original space to EKS then select features with EKS.

- We apply the transformation to more than ten different feature selection algorithms via EKS. Experimental results show the effectiveness of such transformation.

Chapter 2

RELATED WORK

Traditional feature selection algorithms assume each feature influences the label independently. These algorithms are useful in many situations. As we discussed earlier, most of them are filter methods. There are still lots of situations when features are not independent. Two types of solutions for nonindependent features have been proposed. One uses feature extraction to generate more effective features from original features. The other lets algorithms themselves deal with the problem. Margin-based feature selection methods were designed as the latter solution.

Most margin-based algorithms including the work by Cao et al. in [17] were derived from the basic *Relief* proposed by Kira and Rendell in [12]. *Relief* was designed for two class classification problems. It tries to weight features to find a maximal margin between two types of data.

Let $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{|\mathcal{D}|}$ be the training data set where I is the data dimensionality, $|\mathcal{D}|$ the size of data set, $\mathbf{x}^{(n)} \in \mathcal{R}^I$ a data point which can be seen as a vector when calculating, and $y^{(n)}$ the corresponding label of data point $\mathbf{x}^{(n)}$. For any given data point $\mathbf{x}$, the margin of $\mathbf{x}$ is defined as the distance between near-hit NH($\mathbf{x}$) (nearest point with same label) and near-miss NM($\mathbf{x}$) (nearest point with different label). Because the distance is often calculated by all weighted features, weights generated by margin-based algorithms can be regarded as global information. Thus margin-based algorithms can work well with dependent features [19].

The optimization target of *Relief* to be maximized is the sum of distances between NM($\mathbf{x}$) and NH($\mathbf{x}$) for all data points in the set $\mathcal{D}$, and the corresponding math description is as follows:

$$\begin{aligned}
&\max_{\mathbf{w}} \sum_{n=1}^{|\mathcal{D}|} (D_w(\mathbf{x}^{(n)}, NM(\mathbf{x}^{(n)})) - D_w(\mathbf{x}^{(n)}, NH(\mathbf{x}^{(n)})) \\
&\text{s. t. } \|\mathbf{w}\|_2^2 = 1, w_i \geq 0 \\
&D_w(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{I} w_i |\mathbf{x}_i - \mathbf{x}'_i|
\end{aligned} \tag{2.1}$$

With the superscript n and the subscript i in Expression (2.1), $\mathbf{x}^{(n)}$ means the n^{th} data point, and w_i the i^{th} element of weighting vector $\mathbf{w}$, i.e., the weight of the i^{th} feature of the relevant data point. $\|\mathbf{w}\|_2$ is the L2 norm of vector $\mathbf{w}$ which consists of w_i. $\mathbf{x}_i$ is the value in i^{th} feature of data point $\mathbf{x}$. Notice that here $\mathbf{x}$ and $\mathbf{x}'$ mean different data points. $|\mathcal{D}|$ means the size of data set.

The solution to the problem (2.1) can be solved and is given by:

$$\begin{aligned}
&w_i = (m_i)^+ / \|(\mathbf{m})^+\|_2, \\
&\mathbf{m} = \sum_{n=1}^{|\mathcal{D}|} (\mathbf{x}^{(n)} - NM(\mathbf{x}^{(n)})) - (\mathbf{x}^{(n)} - NH(\mathbf{x}^{(n)})), \\
&(m_i)^+ = \max(m_i, 0),
\end{aligned} \tag{2.2}$$

where m_i is the i^{th} value of margin vector $\mathbf{m}$. We recommend further reading of [20] and [21] for detail of *Relief*.

Lots of existing solutions to feature selection have been proved useful in handling linear problems. Although some nonlinear models have been proposed to solve nonlinear problems, most of them are hard in training except those being able to be transformed into forms like exponential or logarithmic models. Thus, a better solution to solve nonlinear problems is first to transform them into relevant linear problems and then to solve the problem with linear methods that have been proved effective.

Making use of Kernel Trick to feature selection problems is difficult, however, there are still efforts in developing kernel feature selection algorithms. There are mainly two types of

kernel feature selection algorithms. One is the extraction type, the other type are modified from traditional algorithms. KPCA, FKD and GDA all belong to the former type. However, KPCA may generate features irrelevant to labels; FKD and GDA could only generate limited number of features. Some feature selection algorithms make use of Kernel Trick [22], but they could only select features in the original space.

Our method was inspired by a special Relief algorithm - *Kernel Relief* proposed in [17], which was designed for explicitly selecting features from Kernel Space to solve nonlinear problems, and thus different from most other nonlinear feature selection methods. Briefly, the key idea of *Kernel Relief* is: replacing $D_w(\mathbf{x}, \mathbf{x}')$ in the *Relief* optimization problem shown in Expression (2.1) using $K_w(\mathbf{x}, \mathbf{x}')$ in Kernel Space, and solve the same optimization problem. Here $K_w(\mathbf{x}, \mathbf{x}')$ is defined as follows:

$$K_w(\mathbf{x}, \mathbf{x}') = \sum_i w_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}'), \tag{2.3}$$

with $\phi_i(\mathbf{x})$ representing the i^{th} feature of high dimension data point $\phi(\mathbf{x})$ in the Kernel Space. More details on this will be continued in the next Chapter.

After Cao et al., there have been continued research about kernel feature selection. However, few of them has ever focused on explicit feature selection in Kernel Space. Some tried to develop a kernel feature selection of a given learning algorithm[23]; some tried to select features in original data space[24][25]. They also tried to find a better Kernel Function and optimization target in the Kernel Space[26]. Different from these approaches, we aim to investigate the explicit features in Kernel Space and find a framework to allow traditional feature selection algorithms explicit select features from Kernel Space without high computational cost.

Our method borrows the idea from [17] that transforms *Relief* to *Kernel Relief*, and proposes a new data space: **Explicit Kernel Space** (EKS) with a proof of its important properties. These properties allow EKS to be a general solution for explicit kernel feature selection. With EKS, most linear feature selection methods can explicitly select features in Kernel Space without complicated modifications.

Chapter 3
EXPLICIT KERNEL SPACE

In this chapter, we will first define the term EKS standing for Explicit Kernel Space; and then present theoretical analysis of its effectiveness in machine learning practice.

3.1 Kernel Relief

Calculating distance in Kernel Space is straightforward. With definition of Kernel Function, we can calculate distance in Kernel Space to find NM and NH in Kernel Space using the following equation (3.1) as that in [27]:

$$\begin{aligned} D_K(\mathbf{x},\mathbf{x}') &=< \phi(\mathbf{x}) - \phi(\mathbf{x}'), \phi(\mathbf{x}) - \phi(\mathbf{x}') > \\ &=< \phi(\mathbf{x}), \phi(\mathbf{x}) > + < \phi(\mathbf{x}'), \phi(\mathbf{x}') > -2 < \phi(\mathbf{x}), \phi(\mathbf{x}') > \\ &= K(\mathbf{x},\mathbf{x}) + K(\mathbf{x}',\mathbf{x}') - 2K(\mathbf{x},\mathbf{x}'), \end{aligned} \tag{3.1}$$

here ϕ is an implicit mapping function.

The optimization problem of *Relief* was given in (2.1). If we substitute $D_w(\mathbf{x},\mathbf{x}')$ with the weighted kernel distance $K_w(\mathbf{x},\mathbf{x}')$, then we have:

$$\begin{aligned} &\max_{\mathbf{w}} \sum_{n=1}^{|\mathcal{D}|} (K_w(\mathbf{x}^{(n)}, NM(\mathbf{x}^{(n)})) - K_w(\mathbf{x}^{(n)}, NH(\mathbf{x}^{(n)}))) \\ &\text{s. t. } \|\mathbf{w}\|_2^2 = 1, w_i \geq 0 \\ &K_w(\mathbf{x},\mathbf{x}') = \sum_i w_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}') \end{aligned} \tag{3.2}$$

Math problem described in (3.2) has showed the form of *Kernel Relief* [17]. Note that since $\phi_i(\mathbf{x})$ is implicit, we cannot get any weight from kernel weight function K_w. To solve this problem, an orthogonal basis set $\{\eta^{(i)}\}$ is induced. The basis set $\{\eta^{(i)}\}$ can be generated

by Kernel Gram-Schmidt Process. It is straightforward to extend Gram-Schmidt Process to kernel Gram-Schmidt Process, because the core step of Gram-Schmidt Process is inner product, which can be simply substituted by Kernel Function.

Algorithm 1 Kernel Gram-Schmidt Process
Input: data $x^{(i)}$
Output: an orthogonal basis set
FOR i = 1 **TO** N **DO**
  $v^{(i)} = \phi(x^{(i)})$
  FOR j=1 **TO** i-1 **DO**
    $v^{(i)} = v^{(i)} - \langle \phi(x^{(i)}), v^{(j)} \rangle v^{(j)}$
  END FOR
  Normalize $v^{(i)}$
END FOR
RETURN $\{v^{(i)}\}$

Figure 3.1: Algorithm of Kernel Gram-Schmidt Process

The input of the Algorithm shown in Figure 3.1 is the training data set $\mathcal{D}$, and its return - $\{v^{(i)}\}$, is the orthogonal basis set $\{\eta^{(i)}\}$ as we expected. It forms an approximation of the Kernel Space [17]. With it, we can calculate an approximation of $\phi(\mathbf{x})$ with $< \phi(\mathbf{x}), \eta^{(i)} >$. From [17], $\eta^{(i)}$ can be expressed as a linear combination of all data points in high dimensional spaces, that is: $\eta^{(i)} = \sum_j a_{ij}\phi(\mathbf{x}^{(j)})$ where the mixed weights a_{ij} can be calculated as follows:

$$a^{(i)} = \frac{\mathbf{e}^{(i)} - \sum_{k=1}^{i-1} a^{(k)} \sum_j a_{kj} K(\mathbf{x}^{(k)}, \mathbf{x}^{(j)})}{||(\mathbf{e}^{(i)} - \sum_{k=1}^{i-1} a^{(k)} \sum_j a_{kj} K(\mathbf{x}^{(k)}, \mathbf{x}^{(j)})) \cdot \mu||}, \tag{3.3}$$

where $\mathbf{e}^{(i)}$ is the vector whose i^{th} element is one and all remaining elements are zero. $\mu = (\phi(\mathbf{x}^{(1)}), \phi(\mathbf{x}^{(2)}), \phi(\mathbf{x}^{(3)}), ..., \phi(\mathbf{x}^{(|\mathcal{D}|)}))^T$, and $a^{(i)} = (a_{i1}, a_{i2}, ..., a_{i|\mathcal{D}|})$.

After getting all of a_{ij}, $K_w(\mathbf{x}, \mathbf{x}')$ can be calculated by the following equation:

$$\begin{aligned} K_w(\mathbf{x}, \mathbf{x}') &= \sum_{i}^{|\mathcal{D}|} w_i(\phi(\mathbf{x}) \cdot \eta^{(i)})(\phi(\mathbf{x}') \cdot \eta^{(i)}) \\ &= \sum_{i}^{|\mathcal{D}|} w_i(\sum_{j} a_{ij} K(\mathbf{x}, \mathbf{x}^{(j)}))(\sum_{j} a_{ij} K(\mathbf{x}', \mathbf{x}^{(j)})) \end{aligned} \tag{3.4}$$

3.2 Explicit Kernel Space

Our work to find the general solution of explicit kernel feature selection begins with the basis set $\{\eta^{(i)}\}$. In the paper by Cao et al., some properties of $K_w(\mathbf{x}, \mathbf{x})$ in $\{\eta^{(i)}\}$ have been proposed. We analyzed the properties of *Kernel Relief* and extended *Kernel Relief* to a general solution for most feature selection algorithms. The key of our solution is the **Explicit Kernel Space** (EKS).

Definition 1 *Let $\mathcal{K}$ be the Kernel Space Spanned by $\phi(\mathbf{x}^{(i)})$, an Explicit Kernel Space is a subspace of $\mathcal{K}$ spanned by a given orthogonal basis set $\{\eta^{(i)}\}$ which satisfies:*

1. *any $\eta^{(i)}$ is a linear combination of $\phi(\mathbf{x}^{(i)})$.*
2. *any $\phi(\mathbf{x}^{(i)})$ can be explained as a linear combination of $\{\eta^{(i)}\}$.*

EKS has an important property: the inner product of data points in EKS can be calculated by corresponding Kernel Function. As pointed out in [17], Kernel Gram-Schmidt Process is not the only method that can generate an Explicit Kernel Space; other methods such as KPCA can also be applied. It is beyond the scope of this research though.

Theorem 1 *Let $\mathcal{K}$ be the Kernel Space spanned by $\phi(\mathbf{x}^{(i)})$, $\mathcal{S}$ be the Explicit Kernel Space, $\mathbf{s}$ be the data point $\mathbf{x}$ projected into Explicit Kernel Space, $K(\mathbf{x}, \mathbf{x}')$ be the Kernel Function. If $\phi(\mathbf{x})$ and $\phi(\mathbf{x}') \in \mathcal{K}$, then $< \phi(\mathbf{x}), \phi(\mathbf{x}') > = \ K(\mathbf{x}, \mathbf{x}') \ = < \mathbf{s}, \mathbf{s}' >$.*

Proof 1 *By Equation (1.1), $K(\mathbf{x}, \mathbf{x}') =< \phi(\mathbf{x}), \phi(\mathbf{x}') >= \sum_{i} \phi_i(\mathbf{x})\phi_i(\mathbf{x}')$. Denote the coefficient in a linear combination representation as $\eta_i(x)$ for the relevant basis vector $\eta^{(i)}$. Then*

there is $K(\mathbf{x},\mathbf{x}') = \sum \eta_i(x)\eta_i(x')$, $i \in \{1,2,3...|\mathcal{S}|,|\mathcal{S}|+1...\} = <\mathbf{s},\mathbf{s}'> + \sum \eta_i(x)\eta_i(x')$, $i \in \{|\mathcal{S}|,|\mathcal{S}|+1...\}$. *Notice that* $\eta^{(i)}$, $i \in \{|\mathcal{S}|,|\mathcal{S}|+1...\}$ *is in the orthogonal complement subspace of* $\mathcal{S}$. *Then for any* $i \in \{|\mathcal{S}|,|\mathcal{S}|+1...\}$, *there is* $\eta_i(x) = <\eta_i,\phi(x)> = 0$. *Hence* $<\phi(\mathbf{x}),\phi(\mathbf{x}')> = K(\mathbf{x},\mathbf{x}') = <\mathbf{s},\mathbf{s}'>$.

As a solution to allow feature selection algorithms select features from Kernel Space explicitly, EKS works like a feature extraction algorithm. It consists of features extracted from implicit Kernel Space. Different from results of usual feature extraction algorithms, EKS tends not to be a ranked feature (even it can be ranked like KPCA). The key point in proposing EKS is to provide a set of explicit high-dimensional features which can calculate inner product with Kernel Function.

As a summary, and one of the main contributions of this research work, we propose the following procedure for the purpose of explicit feature selection in Kernel Space:

Step 1 Generate an EKS $\mathcal{S}$ with a feature extraction algorithm from raw data sets, say, $\mathcal{D}$.

Step 2 Project $\mathcal{D}$ onto the $\mathcal{S}$.

Step 3 To minimize the calculation cost of inner product, we can use Kernel Function to substitute inner product in a feature selection algorithm. Without this step, theoretically it will not influence the features selected, but influence the time cost of feature selection.

Step 4 Perform the feature selection in EKS.

This prototype has its own limitation. To demonstrate this, let us first study certain properties of EKS. No matter what algorithms are used to generate the basis set $\{\eta^{(i)}\}$, this process can be seen as an implicit orthogonal basis transformation. The Kernel Space $\mathcal{K}$ can be regarded as a vector space with infinite orthogonal basis, with an assumption that the rank of data set $\mathcal{D}$ equals to N. Extracting features from Kernel Space is in fact rotating (doing

the orthogonal transformation to) the vector space $\mathcal{K}$, until the moment when projecting the data set $\mathcal{D}$ into the rotated space resulted in that only N dimensions have non-zero value and all remaining dimensions are zero. EKS consists of these non-zero dimensions.

EKS is based on basis transformation, so does its limitation. Theoretically, it will provide good support for feature selection algorithms which are mostly based on inner product or L2-norm, since basis transformation doesn't affect the distance and inner product between data points. When using EKS to transform feature selection algorithms based on L1-norm, however, it may cause some problems since it "shrinks" the remaining dimensions to 0. Interestingly our test for some algorithms making use of L1-norm (such as *LASSO*) showed that such influence is not a major problem.

Chapter 4

EXPERIMENT AND ANALYSIS

4.1 Test Suite

In the previous Chapter, we proposed EKS as a solution to kernel feature selection. However, designing testing experiments is not easy. On the one hand, there are too many factors that can influence the performance of feature selection. On the other hand, we have already made some assumptions but we still dont know the exact properties of the Explicit Kernel Space.

After rounds of investigation and consideration, we designed the test suit as shown in Table 4.1 which we believe can best validate and show the performance of EKS. Furthermore, during our testing efforts, we kept noticing some new interesting properties of EKS and we will report them in the following sections.

Classification				
Data	Size	Features	Number of Features in EKS	Classes
wine	178	13	178	3
Breast cancer	200	30	200	2
Breast cancer	300	30	300	2
Regression				
Data	Size	Features	Number of Features in EKS	
Diabetes	200	10	200	
eunite2001	200	16	200	
eunite2001	300	16	300	

Table 4.1: Test suite

As a kernel-based method, EKS is very sensitive to data set. According to our observation, if a data set is not suitable for EKS, the performance will always be bad. Table 4.1 shows the test data suitable to EKS and used in our experiment. "Wine", "Breast cancer" and "Diabetes" are Sklearn[1] data sets, whereas "eunite2001" is LIBSVM[2] data.

Our tests focused on the performance of transformed feature selection algorithms, thus we only use Kernel Gram-Schmidt Process to generate the EKS. Because the idea was inspired by *Kernel Relief*, we transformed different types of *Relief* (as shown in Table 4.2) with EKS.

Classification		
Relief	ReliefF	RReliefF
F-score	Mutual Information	Laplacian score
Regression		
RReliefF	LASSO	F-score
Family-wise error rate	False Positive Rate test	False Discovery Rate
Mutual Information		

Table 4.2: Algorithms

Transformation of all the algorithms in Table 4.2 has followed the 4 steps described in Section 3.2. We used SVM as a baseline model for classification problem, and linear regression as a baseline model for regression problem. Regarding the performance measurement, we used accuracy[3] for classification problem and MSE for regression problem. All the test results were evaluated across 5 folds cross-validation process.

Our work has used three different Python packages. Relief, ReliefF and RReliefF utilize

[1]http://scikit-learn.org/stable/datasets/index.html

[2]https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/

[3]Accuracy = Number of Correct Classified / Number of Samples, the larger the better

the Sklearn Relief[4] package, Laplacian score utilizes the scikit-feature[5] package, and all other feature selection algorithms, machine learning algorithms and performance measurement methods utilize the scikit-learn[6] package. Parameters used in different algorithms are given in Table 4.3.

LASSO (LASSOLARS-cv)	max alpha = 1000	threshold = 1e-5
Family-wise error rate	threshold = 0.00005	
False Positive Rate test	threshold = 0.00005	
False Discovery Rate	threshold = 0.00005	

Table 4.3: Parameter used in some algorithms

We note that not all algorithms in Table 4.2 have test results with data set given in Table 4.1. *RReliefF* is one such example, and this special case will be discussed later.

4.2 Test Results and Discussion

In this section, we will show detailed results of our tests, while analyzing some interesting properties of EKS. The program environment information is summarized in Table 4.4.

CPU	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
System	Windows 10 Family x64
Platform	Python 3.6 with Anaconda 5.2

Table 4.4: Program environment

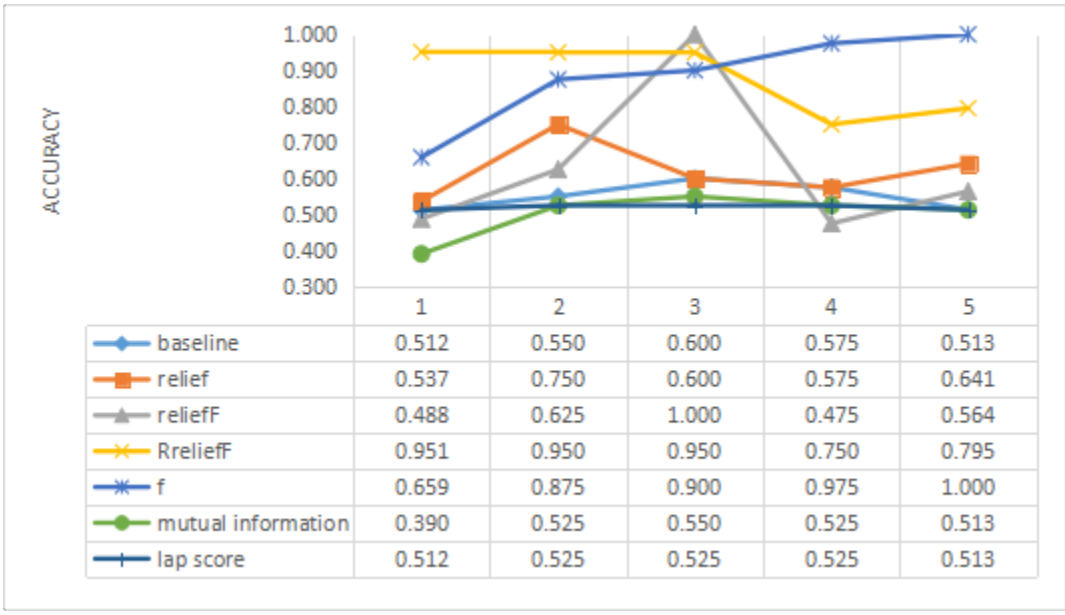
[4]https://pypi.org/project/sklearn-relief/

[5]https://pypi.org/project/skfeature-chappers/

[6]http://scikit-learn.org

4.2.1 Test Results

Figure 4.1 - 4.3 showed test results of the classification model. Obviously most of algorithms we transformed have a higher accuracy than baseline, in particular, the F-score algorithm. Transformed F-score can improve the performance of machine learning model up to 50%-60% from the baseline. These test results have validated the effectiveness of our EKS method. As we expected, not all feature selection algorithms are suitable to EKS such as Mutual Information and Laplacian score; their performance with EKS were not good and will be explained later.

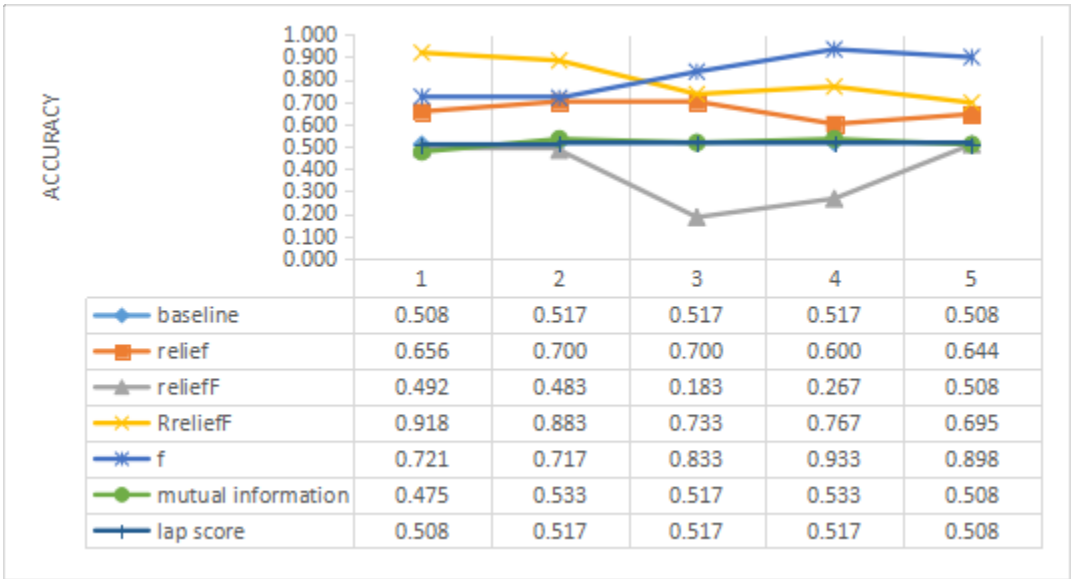


	1	2	3	4	5
baseline	0.512	0.550	0.600	0.575	0.513
relief	0.537	0.750	0.600	0.575	0.641
reliefF	0.488	0.625	1.000	0.475	0.564
RreliefF	0.951	0.950	0.950	0.750	0.795
f	0.659	0.875	0.900	0.975	1.000
mutual information	0.390	0.525	0.550	0.525	0.513
lap score	0.512	0.525	0.525	0.525	0.513

Figure 4.1: Test Results: Breast Cancer, size 200, N_features: 120

For classification problem, we set the number of features selected from EKS to about 2/3 of all N explicit features. The number of selected features is thus larger than that of features in the original data space. This may cause some additional computing cost, but the accuracy has been improved significantly.

As a summary, interesting observations from classification problem testing are given below:



	1	2	3	4	5
baseline	0.508	0.517	0.517	0.517	0.508
relief	0.656	0.700	0.700	0.600	0.644
reliefF	0.492	0.483	0.183	0.267	0.508
RreliefF	0.918	0.883	0.733	0.767	0.695
f	0.721	0.717	0.833	0.933	0.898
mutual information	0.475	0.533	0.517	0.533	0.508
lap score	0.508	0.517	0.517	0.517	0.508

Figure 4.2: Test Results: Breast Cancer, size 300, N_features: 200

- The F-score, one of the basic feature selection algorithms, shows very high performance in EKS.

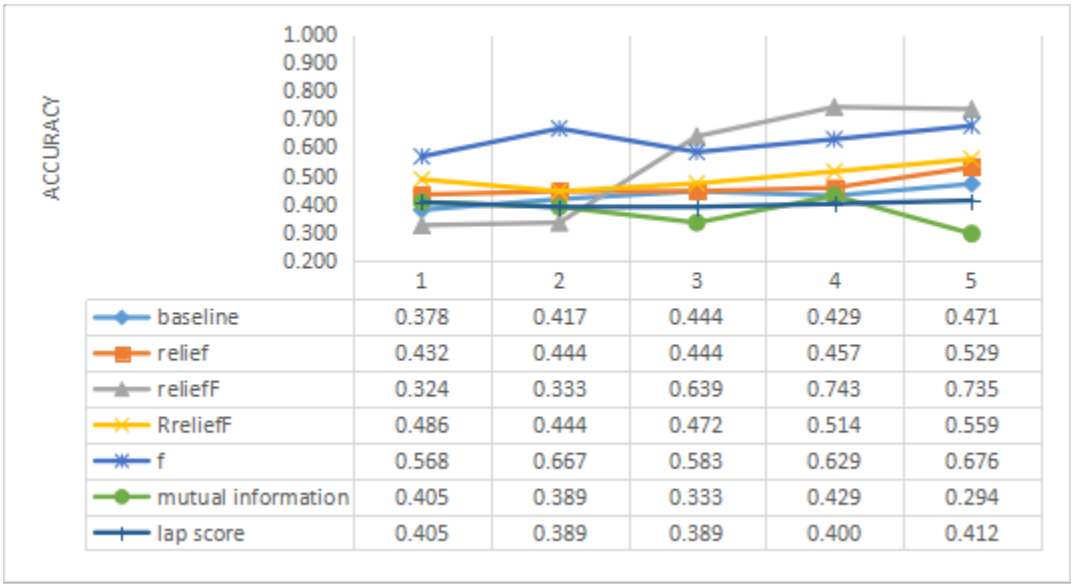
- The RReliefF, as an algorithm designed for regression but not classification, has better performance than Relief and ReliefF, which were designed for classification.

- Different from baseline, which trains models used in the original space, EKS is more sensitive to the size of data sets.

Further discussion on these observations will be presented later.

Figure 4.4 - 4.6 are test results of regression problems. Different from classification problems, we have *LASSO* algorithms, which can decide how many features should be selected, in a smart way. Thus we used different strategies to decide numbers of features to be selected. For *F-score*, *Mutual Information* and *RReliefF*, we use the same number of features selected by *LASSO*. For FDR, FPR and FWE, the threshold required by algorithms are "0.00005".

The conclusion drawn from regression tests is similar to those of classification problem.



	1	2	3	4	5
baseline	0.378	0.417	0.444	0.429	0.471
relief	0.432	0.444	0.444	0.457	0.529
reliefF	0.324	0.333	0.639	0.743	0.735
RreliefF	0.486	0.444	0.472	0.514	0.559
f	0.568	0.667	0.583	0.629	0.676
mutual information	0.405	0.389	0.333	0.429	0.294
lap score	0.405	0.389	0.389	0.400	0.412

Figure 4.3: Test Results: Wine, size 178, N_features: 125

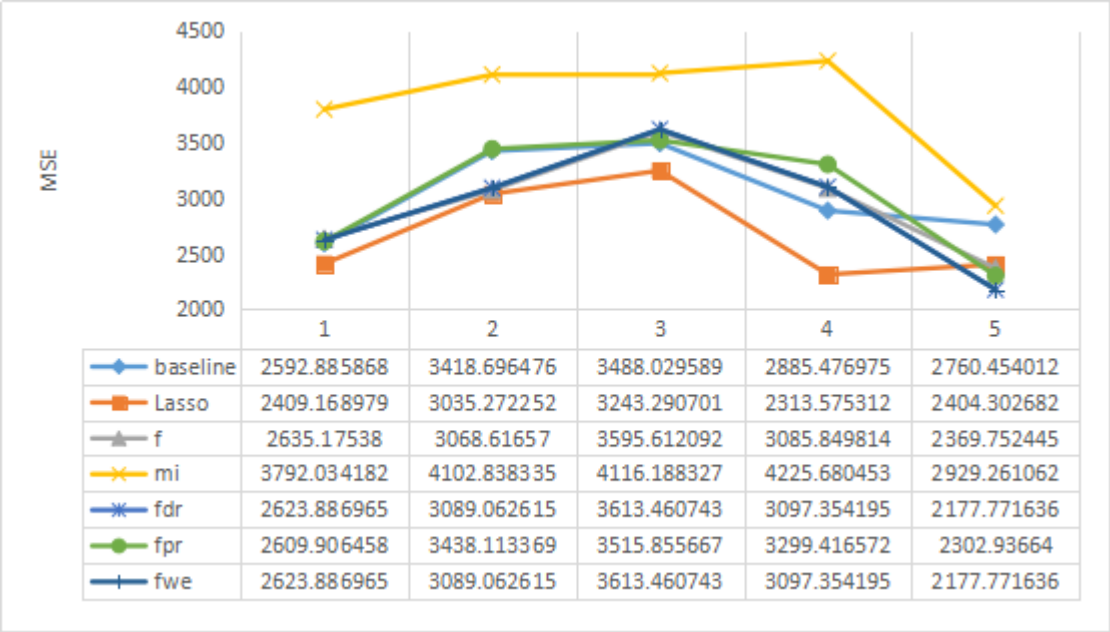
Not all feature selection algorithms are suitable to EKS; not all data sets are suitable for testing either. In fact, the case for regression problem is more complicated.

In last Section 4.1, we have claimed that *RReliefF* doesn't have all test results for the data set given in Table 4.1. It still could work with Diabetes data and eunite2001 data when size is 300. However, some of 5 folds cross-validation tests in these two data sets generated very poor models.

Below is a summary of some interesting observations from regression tests. They are similar to those of the classification test results:

- Basic feature selection algorithms showed good performance in EKS. The *LASSO* always has the best performance, which we believe is due to its usage of Randomized method and iteration of 6 times to find the best features. That might also be a potential property of EKS for regression which is still under investigation.

- EKS is more sensitive to the size of data set in regression problem than in classification problem. For regression problem, simply changing the size of data set may cause



	1	2	3	4	5
baseline	2592.885868	3418.696476	3488.029589	2885.476975	2760.454012
Lasso	2409.168979	3035.272252	3243.290701	2313.575312	2404.302682
f	2635.17538	3068.61657	3595.612092	3085.849814	2369.752445
mi	3792.034182	4102.838335	4116.188327	4225.680453	2929.261062
fdr	2623.886965	3089.062615	3613.460743	3097.354195	2177.771636
fpr	2609.906458	3438.113369	3515.855667	3299.416572	2302.93664
fwe	2623.886965	3089.062615	3613.460743	3097.354195	2177.771636

Figure 4.4: Test Results: Diabetes, size 200

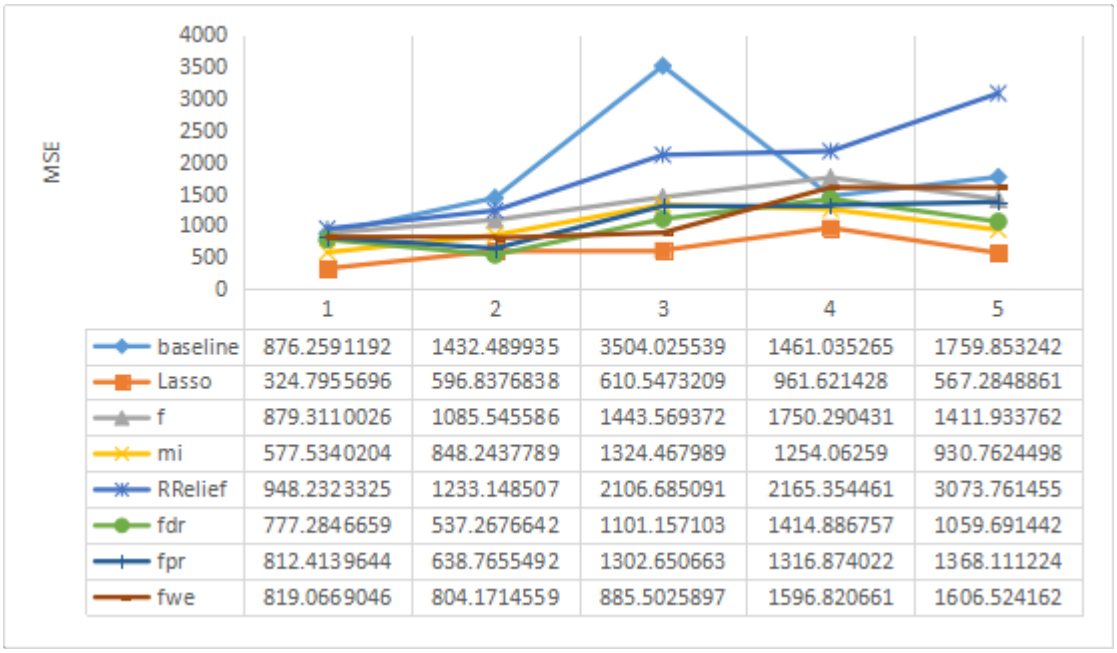
significant impact on some test results (e.g., *RReliefF*).

4.2.2 Discussion

The test results validated the effectiveness of EKS. Some transformed algorithms improved the performance of model greatly, such as *F-score* for classification test suite, and *LASSO*, *F-score* for regression test suite. In the following, we will have some discussions on the potential properties and problems we found from tests.

One of the properties is "sensitiveness". The performance of algorithms in EKS is more sensitive to data size and number of selected features, especially for regression. We believe one of the reasons is that when projecting data into high-dimension space, the data points become sparse, which causes proportion of useful features decreases. For EKS, the number of features is decided by the size of data set. Thus the size of data set will also influence how sparse the high-dimension space is.

The other reason is more important. We know that features in EKS are a linear combi-

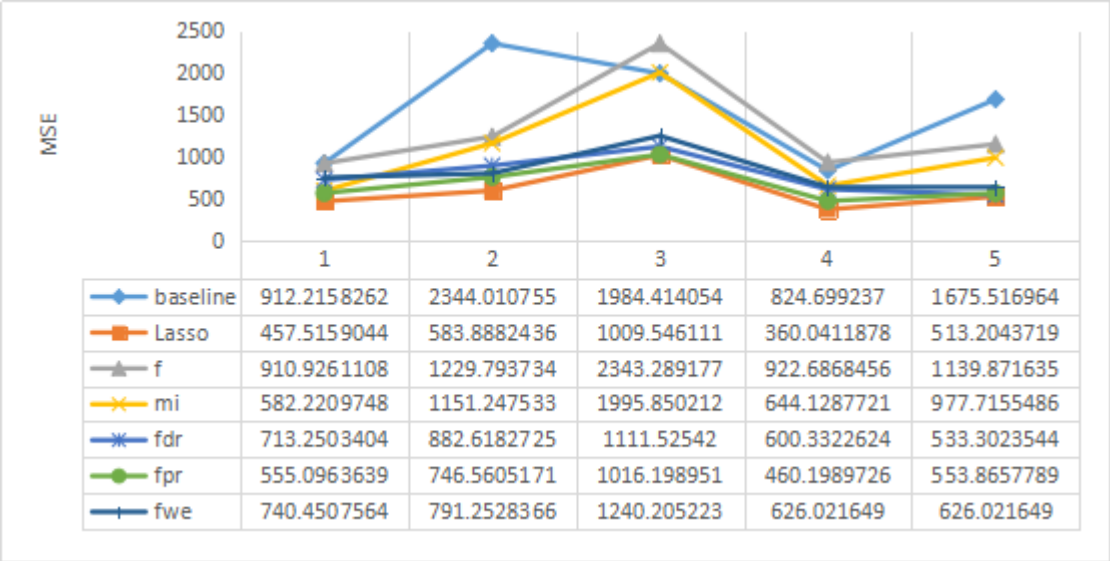


	1	2	3	4	5
baseline	876.2591192	1432.489935	3504.025539	1461.035265	1759.853242
Lasso	324.7955696	596.8376838	610.5473209	961.621428	567.2848861
f	879.3110026	1085.545586	1443.569372	1750.290431	1411.933762
mi	577.5340204	848.2437789	1324.467989	1254.06259	930.7624498
RRelief	948.2323325	1233.148507	2106.685091	2165.354461	3073.761455
fdr	777.2846659	537.2676642	1101.157103	1414.886757	1059.691442
fpr	812.4139644	638.7655492	1302.650663	1316.874022	1368.111224
fwe	819.0669046	804.1714559	885.5025897	1596.820661	1606.524162

Figure 4.5: Test Results: eunite2001, size 200

nation of high-dimension data projection $\phi(\mathbf{x})$. Two problems are caused due to this fact. First, if there exists some terrible noisy data points, features of EKS will be influenced. There will be some terrible features remaining in EKS and becoming hard to find. Once these bad features are selected by feature selection algorithms, the machine learning model will learn a bad model. Thats why sometimes increasing the size of data set cause better results and sometimes the opposite. Second, the algorithm generating orthogonal basis set has its own weakness. We all know that Gram-Schmidt process has orthogonality loss problem. Due to the increasing loss of orthogonality, the feature generated by the Gram-Schmidt process will be useless.

We already have a clue to solve bad features problem in EKS. For lots of data sets, N features (here N the size of data set) are more than enough to create a good model. We can use randomized methods to solve the problem like randomized LASSO we used. In fact, at the very beginning, we have tried original LASSO. We use randomized LASSO to substitute LASSO because sometimes LASSO have same problems as RReliefF. Randomized process



	1	2	3	4	5
baseline	912.2158262	2344.010755	1984.414054	824.699237	1675.516964
Lasso	457.5159044	583.8882436	1009.546111	360.0411878	513.2043719
f	910.9261108	1229.793734	2343.289177	922.6868456	1139.871635
mi	582.2209748	1151.247533	1995.850212	644.1287721	977.7155486
fdr	713.2503404	882.6182725	1111.52542	600.3322624	533.3023544
fpr	555.0963639	746.5605171	1016.198951	460.1989726	553.8657789
fwe	740.4507564	791.2528366	1240.205223	626.021649	626.021649

Figure 4.6: Test Results: eunite2001, size 300

will successfully remove bad features from selected features. We can perform some noisy reduction for the data set or recapture orthogonality loss of Gram-Schmidt process.

The other potential properties we found is "the simpler the algorithm, the better the performance", or, "the simpler, the better". We still can not give a theoretical proof of this property. But the simplest algorithms like *F-score* works very well in both classification test and regression test. We believe the reason is that EKS successfully extracts inner features of the data set being used. This view may also explain why *RReliefF* can also work on classification problem. With good EKS, the classification and regression may be the same thing.

Chapter 5
CONCLUSION AND FUTURE WORK

In this paper, we proposed a method to do explicit feature selection in Kernel Space. The core is to transform the original data space into a so-called **Explicit Kernel Space** (EKS). Using this method, most feature selection algorithms are allowed to calculate the inner product of data points fast in an N-dimension EKS (here N the size of data set), just like using the well-know Kernel Trick. A general process is presented to show how feature selection algorithms can be transformed to select features in Kernel Space. Our experiment includes more than 10 transformed algorithms based on the process. Experimental results are summarized to validate the effectiveness of our method using EKS. Some interesting properties of EKS indicated by tests are also discussed.

Our proposed method using EKS still needs further investigation, though. On the one hand, the tests we have done so far can only prove the usefulness of EKS. We can explain the "sensitiveness" property and "the simpler, the better" property based on specific assumptions, but there are still unknown properties to be explored. On the other hand, our current largest data set has only 300 data points. With increasing data size, the dimension of EKS will also increase, followed by 2 potential problems. First, the time cost of generating EKS will be very large. Second, the proportion of useless features in EKS may increase. To tackle the problem of increasing dimension is the main target of our future work. One effort we are going to try is to find a way selecting "important data points" from data set and use these data points to generate an optimized EKS. Hopefully, the method can also help with the problem of "bad features" and "high dimension". Meanwhile, we will analyze and test more data sets and algorithms. Our ultimate goal is to generate a guideline which can be used to tell whether or not a data set or an algorithm is suitable for our method based on EKS.

BIBLIOGRAPHY

[1] Christopher M Bishop. Pattern recognition and machine learning (information science and statistics) springer-verlag new york. *Inc. Secaucus, NJ, USA*, 2006.

[2] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[3] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[5] William Meredith. Measurement invariance, factor analysis and factorial invariance. *Psychometrika*, 58(4):525–543, 1993.

[6] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238, 2005.

[7] Erich L Lehmann and Joseph P Romano. *Testing statistical hypotheses*. Springer Science & Business Media, 2006.

[8] Philip H Swain and Hans Hauska. The decision tree classifier: Design and potential. *IEEE Transactions on Geoscience Electronics*, 15(3):142–147, 1977.

[9] John Neter, William Wasserman, and Michael H Kutner. Applied linear regression models. 1989.

[10] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[11] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[12] Kenji Kira and Larry A Rendell. The feature selection problem: Traditional methods and a new algorithm. In *Aaai*, volume 2, pages 129–134, 1992.

[13] Xiaofei He, Deng Cai, and Partha Niyogi. Laplacian score for feature selection. In *Advances in neural information processing systems*, pages 507–514, 2006.

[14] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International Conference on Artificial Neural Networks*, pages 583–588. Springer, 1997.

[15] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In *Neural networks for signal processing IX, 1999. Proceedings of the 1999 IEEE signal processing society workshop.*, pages 41–48. Ieee, 1999.

[16] Gaston Baudat and Fatiha Anouar. Generalized discriminant analysis using a kernel approach. *Neural computation*, 12(10):2385–2404, 2000.

[17] Bin Cao, Dou Shen, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Feature selection in a kernel space. In *Proceedings of the 24th international conference on Machine learning*, pages 121–128. ACM, 2007.

[18] Shigeo Abe. Feature selection and extraction. In *Support Vector Machines for Pattern Classification*, pages 331–341. Springer, 2010.

[19] Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. Margin based feature selection-theory and algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 43. ACM, 2004.

[20] Igor Kononenko. Estimating attributes: analysis and extensions of relief. In *European conference on machine learning*, pages 171–182. Springer, 1994.

[21] Yijun Sun and Jian Li. Iterative relief for feature weighting. In *Proceedings of the 23rd international conference on Machine learning*, pages 913–920. ACM, 2006.

[22] Paul S Bradley and Olvi L Mangasarian. Feature selection via concave minimization and support vector machines. In *ICML*, volume 98, pages 82–90, 1998.

[23] Bor-Chen Kuo, Hsin-Hua Ho, Cheng-Hsuan Li, Chih-Cheng Hung, and Jin-Shiuh Taur. A kernel-based feature selection method for svm with rbf kernel for hyperspectral image classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(1):317–326, 2014.

[24] Kemal Polat and Salih Güneş. A new feature selection method on classification of medical datasets: Kernel f-score feature selection. *Expert Systems with Applications*, 36(7):10367–10373, 2009.

[25] Claudio Persello and Lorenzo Bruzzone. Kernel-based domain-invariant feature selection in hyperspectral images for transfer learning. *IEEE transactions on geoscience and remote sensing*, 54(5):2615–2626, 2016.

[26] Ehsan Adeli, Guorong Wu, Behrouz Saghafi, Le An, Feng Shi, and Dinggang Shen. Kernel-based joint feature selection and max-margin classification for early diagnosis of parkinsons disease. *Scientific reports*, 7:41069, 2017.

[27] Bernhard Schölkopf. The kernel trick for distances. In *Advances in neural information processing systems*, pages 301–307, 2001.