University of Jean Monnet

Advanced Machine Learning

Practical Session 1

Kernel Methods

Author:
Allwyn Joseph
Karthik Bhaskar

Supervisor:
Dr. Amaury Habrard

November 4, 2018

Contents

1 Introduction **2**

2 Datasets **2**

3 Kernel-PCA **3**
3.1 The Algorithm 3
3.2 Mathematical Representation 3
3.3 Testing and Interpretation on Benchmark Dataset 4
3.4 Testing and Interpretation on Numerai Dataset 6

4 Kernel K-means **7**
4.1 The Algorithm 7
4.2 Mathematical Representation 8
4.3 Testing and Interpretation on Benchmark Dataset 9
4.4 Testing and Interpretation on Numerai Dataset 10

5 Kernel Logistic Regression **11**
5.1 The Algorithm 11
5.2 Mathematical Representation 12
5.3 Testing and Interpretation on Benchmark Dataset 13

6 One class SVM **14**
6.1 The Algorithm 14
6.2 Mathematical Representation 15
6.3 Testing and Interpretation on Benchmark Dataset 16

7 Conclusions **18**

List of Figures

1 K-PCA on benchmark dataset 5
2 K-PCA on Numerai dataset 6
3 K-Kmeans on benchmark dataset 10
4 Non-linear Data 14
5 Non-linear Data 14
6 One Class SVM on benchmark dataset 17

1 Introduction

The principle aim of the practical session is to reproduce kernelised versions of four commonly used machine learning algorithms namely, PCA, K-means, Logistic Regression and One class SVM, and record their behaviour on an existing benchmark and miscellaneous datasets.

This report documents our approach towards effectuating a successful practical session. It begins with presenting the vanilla version of each of the algorithms, followed by their kernalised versions. The report then dwells deeper into the workings and implementation of the kernalised version of the algorithms followed by its testing on one of the benchmark datasets. Finally, kernalised version of the algorithms are tested on an actual dataset from Numerai - an AI-run, crowd-sourced hedge fund - the findings and observations are recorded.

2 Datasets

To validate the workings of the kernelised implementations of the aforementioned algorithms benchmark datasets such as moons, clusters, blobs etc are used.
As for the second part of testing, dataset from the AI-run, crowd-sourced hedge fund platfrom Numerai is used. The dataset is a simple CSV file containing rows of features and their binary target values. Following are the characteristics of the dataset:

- The data set has been cleaned and regularized
- The feature names of the dataset have been obfuscated so as to avoid human bias.
- The target values are binary indicating as to whether the hedge fund should invest or not - Numerai's trades are determined by an AI, which is fueled by a network of thousands of anonymous data scientists.
- As for the attributes there are 24, 21 of which are representative of obfuscated features and the remaining three are `id`, `era` and `data_type`.

For analysis, only instances with `era` value equal to 'era3' - 403 samples - are considered due to computational constraints.

3 Kernel-PCA

3.1 The Algorithm

Principle Component Analysis (PCA) is one of the commonly used algorithms for dimensionality reduction. The algorithm does so by producing principle components (which are linear combinations of features of the given dataset), extracting as much as variance as possible from the dataset. Applying PCA as a prepossessing step often rids the data off correlated features while retaining the variation present in the dataset, up to the maximum extent.

But often is the case that the underlying structure of the dataset is non-linear. On such datasets, PCA tends to under perform. Hence a kernelised version of the same algorithm is used to counter non-linearity by first projecting the data in a higher dimensional space and then producing principle components from the projected data.

3.2 Mathematical Representation

3.2.1 Formalization and Solution of PCA

Let $\boldsymbol{S} = \{x_1, .., x_n\}$ be a set of vectors ($x \in\in R^d$). Assume that the data are centered:

$$\frac{1}{n}\sum_{i=1}^{n} x_i = 0$$

The orthogonal projection onto a direction $\boldsymbol{w} \in \mathrm{R}^d$ is the function
$h_w : R^d \to R$

$$h_w(x) = x^T \frac{w}{\|w\|}$$

Then empirical variance captured by h_w is:

$$\hat{var}(h_w) := \frac{1}{n}\sum_{i=1}^{n} h_w(x_i)^2 = \frac{1}{n}\frac{w^T X^T X w}{w^T w}$$

The solutions of :

$$w_i := \underset{w \perp \{w_1,..,w_{i-1}\}}{\operatorname{argmax}} \; w^T X^T X w \quad \text{s.t.} \quad \|w\| = 1$$

3.2.2 Formalization and Solution of K-PCA

Assuming the data are centered:

$$\frac{1}{n}\sum_{i=1}^{n}\varphi(x_i) = 0$$

The orthogonal projection onto direction $f \in H$ is a function
$h_f : X \to R$

$$h_f(x) = \langle \varphi(x), \frac{f}{\|f\|_H} \rangle_H$$

The empirical variance captured by h_f is:

$$\hat{var}(h_f) := \frac{1}{n}\sum_{i=1}^{n}\frac{\langle \varphi(x_i), f\rangle_H^2}{\|f\|_H^2}$$

The i^{th} principal direction $f_i \quad \forall (i = 1, ..., d)$ is defined by:

$$f_i := \underset{f \perp \{f_1,..,f_{i-1}\}}{\operatorname{argmax}} \hat{var}(h_f) \quad \text{s.t.} \quad \|f\|_H = 1$$

From the above equations it becomes evident that, in order to run a K-PCA on a given dataset, the following steps ought to be followed [1]:

- Calculate the kernel matrix using rbf, polynomial or sigmoid kernels
- Centering the symmetric kernel matrix.
- Calculate eigen values and eigen vectors of the centered matrix
- Calculate the new representation of input data using the eigen vectors

Note : The code for K-PCA can be found on Github

3.3 Testing and Interpretation on Benchmark Dataset

The benchmark dataset used for this task was the `Make Moons` dataset available on python. The aim of using this dataset was to see if a non-linear dataset could be clustured using kernel methods.

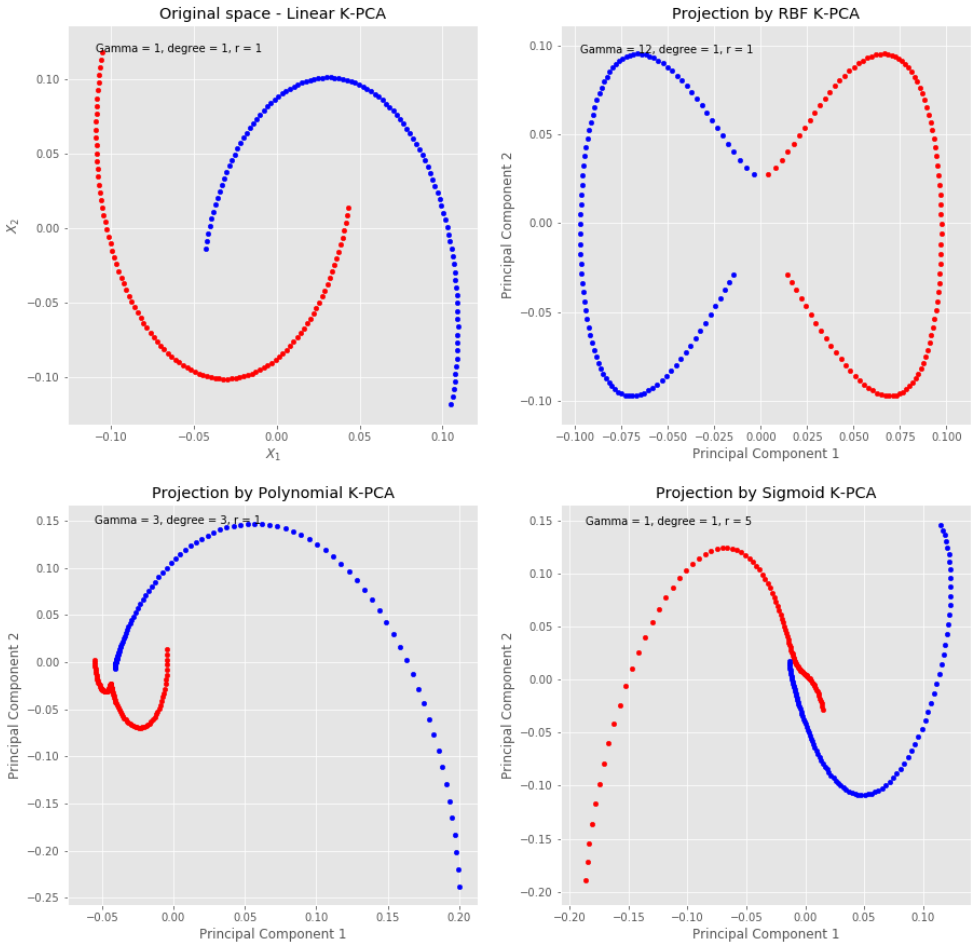

Figure 1: K-PCA on benchmark dataset

The non-linear dataset was iterated over several times using the KPCA algorithm in order to obtain the right parameter tuning for maxmimizing separation between the two classes - red and blue.

Due to the nature of the dataset, it becomes evident from the figure 1 that the RBF kernel performs the best in terms of achieving the best separation between the two classes. The polynomial and sigmoid in comparison is barely able to induce any linear separation of sorts.

3.4 Testing and Interpretation on Numerai Dataset

Having tested KPCA on benchmark dataset, here we go on to run tests on the Numerai Dataset.

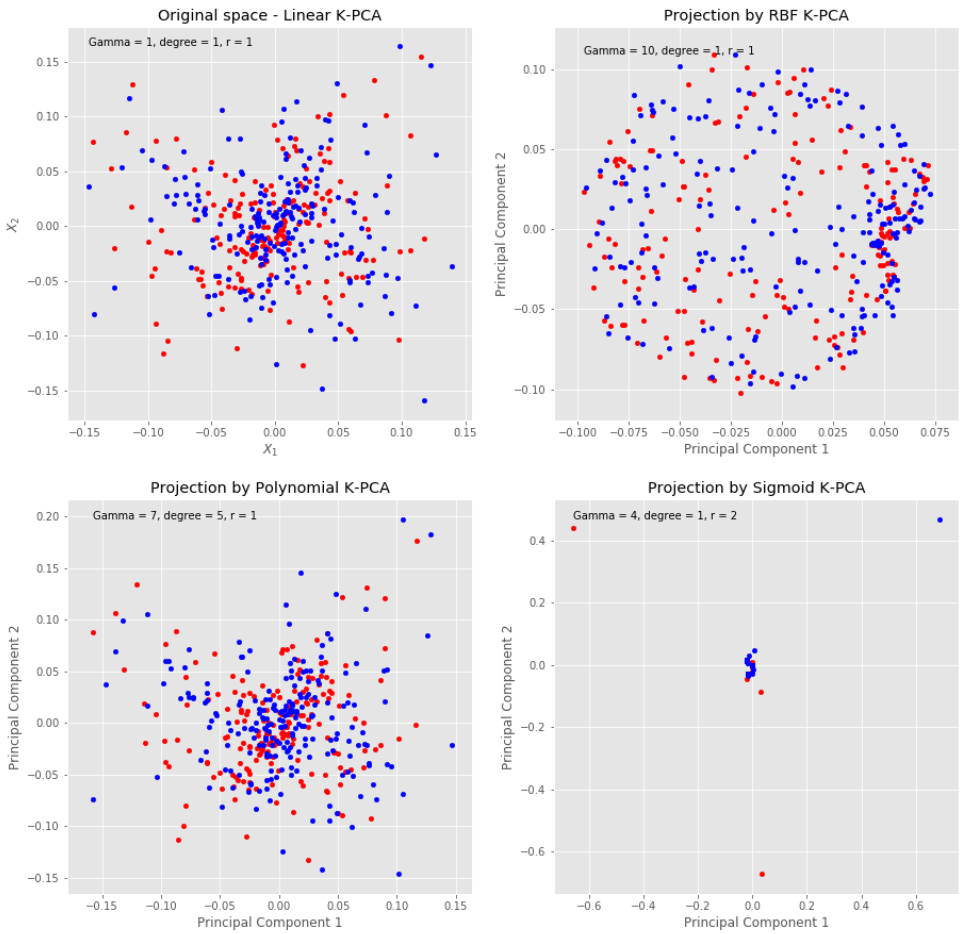

Figure 2: K-PCA on Numerai dataset

After several iteration of hyperparmeter tuning of the KPCA algorithm on the dataset, Figure 2 above is achieved. We can observe that PCA using the sigmoid kernel seems to have best separated the data, while the RBF and polynomial kernels couldn't introduce much separation.

To confirm our visual interpretation, the newly generated dataset is tested out with two machine learning algorithms: logistic regression and Support Vector Machines. The algorithms are run on the kernelised principle components using a search grid for their hyperparmeter tuning. Below are the results of the conducted tests:

Machine Learning Algorithm	**Kernels**				**Precision**	**Recall**
		Gamma	*Degree*	*Bias*		
Logistic Regression	linear	1	1	1	0.54	0.54
	RBF	10	-	-	0.59	0.58
	Polynomial	7	8	1	0.58	0.56
	Sigmoid	4	-	2	0.75	0.53
Support Vector Machines	linear	1	1	1	0.61	0.55
	RBF	0.5	-	-	0.57	0.55
	Polynomial	5.5	12	3	0.57	0.57
	Sigmoid	0.001	-	1	0.61	0.55

Table 1: Comparison of ML algorithms on Kernalised Principle Components

Table **??** confirms our interpretation, concluding that PCA in combination with the sigmoid kernel brings about the best performance on the dataset in terms of precision. This being said, vanilla PCA or linear K-PCA in combination with the machine learning algorithms performs well too. Thus corroborating the fact that performance of K-PCA is very data dependent and in combination with machine learning algorithms, often yields different results with datasets of varying nature.

4 Kernel K-means

4.1 The Algorithm

The K-means algorithm often sees its application within a unsupervised setting (when we are unaware of the labels). The algorithm works by initially selecting 'k' points with a defined space and then creating clusters based on its distances from the data points. The 'k' points keep getting redefined until stability is attained.

But again, the algorithm struggles to perform well within a non linear setting. Hence a kernelised version is of the algorithm is used to remedy the same.

4.2 Mathematical Representation

4.2.1 Formalization and Solution of K-means

Given data points $x_1, ..., x_n in R^p$, it consists of minimizing the the following cost/objective function:

$$\underset{s\in\{1,...k\}}{\mu_j \in R^p} \underset{for i=1,..,n}{\min_{for j=1,..k}} \quad \sum_{i=1}^{n} \|x_i - \mu_s\|_2^2$$

The vanilla version of K-means alternates between two steps until stability is attained.

- **Cluster Assignment**: Given fixed $\mu_1 ... \mu_k$ assign each x_i to its closest centroid:

$$\forall_i, \quad \boldsymbol{s_i} \in \underset{s\in\{1,...k\}}{\operatorname{argmin}} \|x_i - \mu_s\|_2^2$$

- **Centroid Update**: Given the previous assignments $s_1, ... s_n$, update the centroids.

$$\forall_j, \quad \mu_j = \underset{\mu\in R^p}{\operatorname{argmin}} \sum_{i:s_i=j} \|x_i - \mu_s\|_2^2$$

4.2.2 Formalization and Solution of Kernalised K-means

Modifying the above vanilla K-means objective function to operate in the RKHS space we get:

$$\underset{s\in\{1,...k\}}{\mu_j \in H} \underset{for i=1,..,n}{\min_{for j=1,..k}} \quad \sum_{i=1}^{n} \|\varphi(x_i) - \mu_s\|_H^2$$

Again, alternating between two steps:

- **Centroid Update**: Given the previous assignments $s_1, ... s_n$, update the centroids.

$$\forall_j, \quad \mu_j = \underset{\mu\in H}{\operatorname{argmin}} \sum_{i:s_i=j} \|\varphi(x_i) - \mu_s\|_H^2$$

- **Cluster Assignment**: Given fixed $\mu_1 ... \mu_k$ assign each x_i to its closest centroid.

$$\boldsymbol{s_i} \in \underset{s \in \{1,...k\}}{\operatorname{argmin}} \|\varphi(x_i) - \mu_s\|_H^2$$

The above equations can be combined and reformulated into:

$$\boldsymbol{s_i} \in \underset{s \in \{1,...k\}}{\operatorname{argmin}} (K(x_i, x_i) - \frac{2}{|C_s|} \sum_{j \in C_s} K(x_i, x_j) + \frac{1}{|C_s|^2} \sum_{j,l \in C_s} K(x_j, x_l)) \quad (1)$$

The crux of the Kernalised K-means algorithm lies within solving the above equation [3].

Putting together the Kernalised K-means algorithm entails the following steps:

- Calculate the kernel matrix using rbf, polynomial or sigmoid kernels
- Randomly assign points to 'k' clusters.
- Calculate the distance/similarity of each to the assigned cluster by following equation 1.
- Reassign point to clusters with which it has the least distance (highest similarity).
- Repeat until stability.

Note : The code for Kernalised K-means can be found on Github

4.3 Testing and Interpretation on Benchmark Dataset

The benchmark dataset used for this task was the `Make Circles` dataset available on python. The aim of using this dataset was to see if a non-linear dataset could be clustured using kernel methods.

The non-linear dataset was iterated over several times using the Kernalized K-means algorithm in order to obtain the right parameter tuning for maxmimizing separation between the two classes - red and blue. Figure 4 below displays the result of the algorithm on the dataset in combination with different kernels.

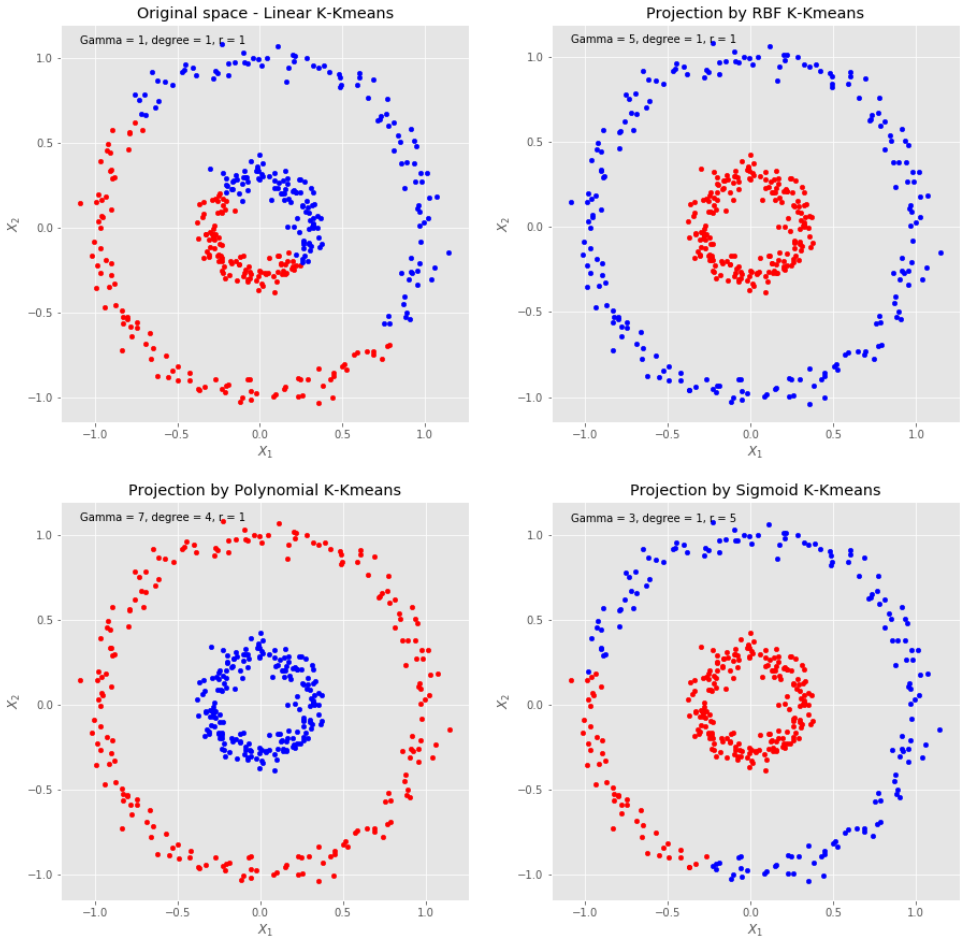

Figure 3: K-Kmeans on benchmark dataset

With the right tuning, we observe that the `RBF` and `Polynomial` kernel versions of the algorithm achieves complete separation of the datset, while the sigmoid kernel's performance isn't far off. Additionally, one is able to observe the linear kernel's futile attempt at separating the dataset, reinforcing the need for kernel methods during certain case.

4.4 Testing and Interpretation on Numerai Dataset

The algorithm is further test run on the Numerai dataset. The results of the same are tabulated below.

Machine Learning Algorithm	Kernels				Precision	Recall
		Gamma	*Degree*	*Bias*		
Kernalized K-Means	linear	1	1	1	0.54	0.53
	RBF	0.8	-	-	0.54	0.53
	Polynomial	0.33	2	1	0.54	0.53
	Sigmoid	0.9	-	2	0.55	0.54

Table 2: Kernalized K-means algorithms on Numerai Dataset

After having run several iterations of the kernalised alogrithm on the dataset table 2 above summarizes our learnings. We observe that the kernalised versions of the K-means algorithms doesn't fare any better than the vanilla version. This being said, K-means with the sigmoid kernel fares slighty better than the rest. This also goes to show that the Kernalised K-means algorithm is a bad fit for the Numerai dataset and an algorithm's performance is highly dependent on the nature of the dataset.

5 Kernel Logistic Regression

5.1 The Algorithm

Logistic regression is a technique in machine learning. It is an algorithm to perform binary classification, where the classification to be performed is on two class values.

In this algorithm we make a prediction of true or false (0 or 1) based on the logistic function which is also called sigmoid function. It is S shaped curve to make predictions to which class the data belongs.

The Logistic function is based on the following equation.

$$g(y) = \frac{1}{1 + exp(-y)} \tag{1}$$

where exp is the euler's number. Here the coefficients associated with the input is learned along with the bias using optimization algorithms such as stocastic gradient descent to predict the output.

5.2 Mathematical Representation

5.2.1 Formalization and Solution of vanilla logistic regression

Moving into the deeper mathematics of the LR algorithm; first off the probability of a point x can be calculated using the sigmoid function.

$$P(Y|X = x) = sigmoid_{\beta_1,\beta_0}(x^T\beta + \beta_o) \quad (2)$$

The derivative of the sigmoid funciton is :

$$\frac{\partial sigmoid}{\partial \beta} = sigmoid(1 - sigmoid) \quad (3)$$

Using equations 1, 2 and 3, we are able to calculate the derivate of the logit function.

$$\frac{\partial L(\beta, Y)}{\partial \beta_j} = \frac{\partial L}{\partial \beta_j} \sum_{x_i \in X} y_i ln(p_i) + (1 - y_i) ln(1 - p_i)$$

$$\frac{\partial L(\beta, Y)}{\partial \beta_j} = \sum_{x_i \in X} y_i \frac{\partial ln(p_i)}{\partial \beta_j} + (1 - y_i) \frac{\partial ln(1 - p_i)}{\partial \beta_j} \quad (4)$$

Using the following equation $\frac{\partial p_i}{\partial \beta_j} = x_{i,j} p_i (1 - p_i)$ in equation 4, we get:

$$(4) = \sum_{x_i \in X} y_i x_{i,j} (1 - p_i) x_{i,j} (1 - y_i) p_i$$

$$(4) = \sum_{x_i \in X} x_{i,j} (y_i - p_i) \quad (5)$$

$$(4) = X_j^T (Y - \hat{Y})$$

Where X_j is the jth column of the dataset and is the predicted probability.

Looking to kernelize the LR algorithm, from equation 5 we know the the optimal solution to maximise the likelihood. By using its value in the probaility formula we obtain:

$$P(Y|X = x) = (1 + exp(- \sum_{x_i \in X} \sum_{x_j \in X} x_i^T x_j (y_i - p_i)))^{-1}$$

$$P(Y|X = x) = (1 + exp(- \sum_{x_i \in X} \sum_{x_j \in X} K(x_i, x_j)(y_i - p_i)))^{-1} \quad (6)$$

The equation 6 above defines the objective function for the kernalized version of LR.

5.2.2 Formalization and Solution of kernel logistic regression

As the vanilla logistic regression is not suited for non-linear data. We Kernalised the logistic regression to make good predictions on the non-linear data.

Constructing the Kernalised Logistic Regression algorithm [2] entailed the following steps:

1. At first kernel matrix was created
2. Using RBF, Polynomial and Sigmoid kernels. The target value was predicted using stocastic gradient descent and the logistic function from the equations 1 and 2
3. The threshold of prediction was set to 0.5, if the predicted value was greater or equal to 0.5 then the predicted class is 1 else 0
4. The accuracy is calculated based on the threshold

Note : The code for Kernalised Logistic Regression can be found on Github

5.3 Testing and Interpretation on Benchmark Dataset

The Benchmark dataset used was moon dataset to obtain the non-linear data using python. the generated plot is shown bellow. As the data is non-linear the vanilla version of logistic regression cannot make better predictions. During the test the RBF and Polynomial kernels gave a good accuracy compared to sigmoid kernel. The sigma for the RBF was chossen 1 by trail and error. The polynomial kernel was tested with the gamma=0.3 and degree=2 which gave a good prediction accuracy and the sigmoid kernel was tested with gamma=3 and r=5, but we could not obtain the accuracy compared to other two kernels.

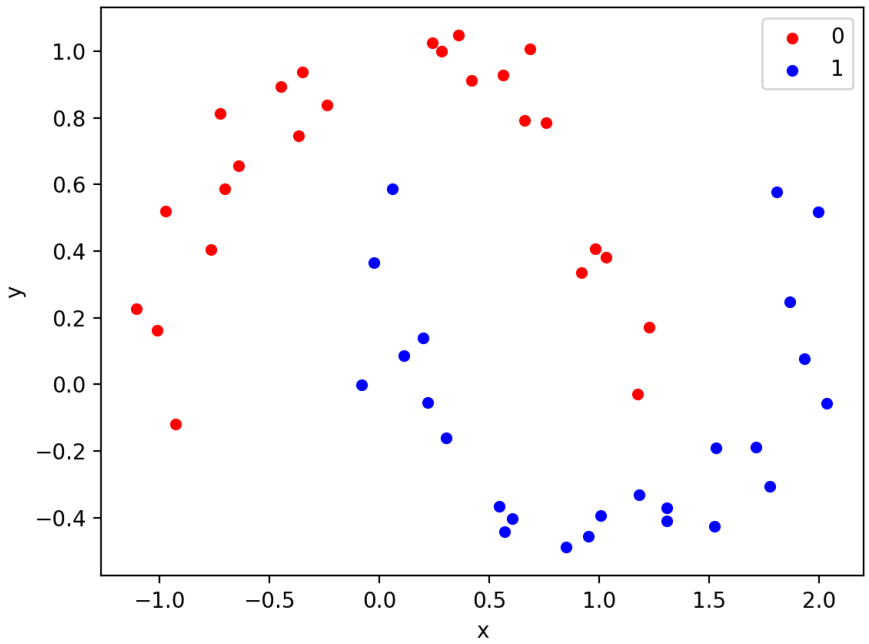

Figure 4: Non-linear Data

```
Final Test Accuracy of 3 Kernels with Linear LR
Linear LR 0.4
RBF:  0.9
Polynomial:  1.0
Sigmoid:  0.6
```

Figure 5: Non-linear Data

6 One class SVM

6.1 The Algorithm

The one class SVM is a subset of Support Vector Data Description and often used to detect outliers within a given dataset. An example of its application can be seen in removal of blemishes/anomalies from images. In SVDD, on

would usually try and find the minimum circumscribing hyper-ball in a high dimensional space.

The algorithm also suffers from finding outliers within a non-linear setting, hence they are often used along with kernels to remedy this drawback.

6.2 Mathematical Representation

6.2.1 Formalization and Solution of vanilla SVDD

The error function that we ought to minimize is:

$$F(R,a) = R^2 \quad \text{with constraints :} \|x_i - a\|^2 \leq R^2, \quad \forall_i \quad (1)$$

With the introduction of slack variables the unconstrained primal form of the objective function can be rewritten as:

$$L(R,a,\alpha_i,\gamma_i,\xi_i) = R^2 + C\sum_i \xi_i - \sum_i \alpha_i\{R^2 + \xi_i - (\|x_i\|^2 - 2a.x_i + \|a\|^2)\} - \sum_i \gamma_i\xi_i (2)$$

The primal form is then differentiated with respect to the constraints and assigned to zero to find their corresponding values.

$$\frac{\partial L}{\partial R} = 0 : \quad \sum_i \alpha_i = 1 \qquad (3)$$

$$\frac{\partial L}{\partial a} = 0 : a = \frac{\sum_i \alpha_i x_i}{\sum_i \alpha_i} = \sum_i \alpha_i x_i \qquad (4)$$

$$\frac{\partial L}{\partial \xi_i} = 0 : \quad C - \alpha_i - \gamma_i = 0 \qquad (5)$$

6.2.2 Formalization and Solution of kernalised SVDD

[4] Re-substituting equations 3,4 and 5 in 2 we get the dual formulation of the same. Maximizing the dual form is tantamount to minimizing the

$$L = \sum_i \alpha_i K(x_i,x_i) - \sum_{i,j} \alpha_i\alpha_j K(x_i,x_j) \quad (6)$$

The support vectors of the enclosing ball is given by:

$$\|x_i - a\|^2 = R^2 \rightarrow \quad 0 < \alpha_i < C, \gamma_i = 0 \quad (7)$$

And the radius of the enclosing ball is given by:

$$R^2 = K(x_k, x_k) - 2\sum_i \alpha_i K(x_j, x_i) + \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \quad (8)$$

Points lying outside of this radius will be considered outliers. Constructing the Kernalised SVDD algorithm entailed the following steps:

- Calculating the kernel matrix using rbf and polynomial kernels.
- Defining the constraints, objective function and maximizing the dual from.
- Calculating the support vector from the optimized alpha values using equation/constraints 7.
- Calculating the radius using equation 8.

Note : The code for Kernalised one class SVM can be found on Github

6.3 Testing and Interpretation on Benchmark Dataset

The benchmark dataset used for this task was the `Make Blobs` dataset available on python. Since the One class SVM algorithm is used on datasets with a single class, the datset was initialized with only a single class/cluster.

Figure 6 below are the plots realized after having run the one class SVM algorithm on the `Make Blobs` dataset. Each row corresponds to plots generated while using a specific kernel and the columns correspond to the C (penalty or regularization parameter) values used while maximising the objective function 6. The red points in the plots are the centers calculated using equation 4.

The first row of the plot clearly illustrates the effect of the varying values of C on support vector selection and with it radius of the maximum enclosing ball - Linear SVDD. Referring to equation 2, a value of C = 0.1 equates to a low penalty on the instances that lie outside the hypersphere. Thus the

optimizing the primal solution would minimize the size of the hypersphere while leaving out a few instance as the penalty on them is meager.

On the contrary, higher values of C (C = 0.8) imposes high penalties on instances outside of the hypersphere. This in turn leads to minimizing the size of the hypersphere while making sure none or close to none of the instances lie beyond it. Thereby leading to larger spheres as can be seen in figure 5. One can also notice that the support vectors lie exactly on the enclosing sphere dictating the boundary of the classifier.

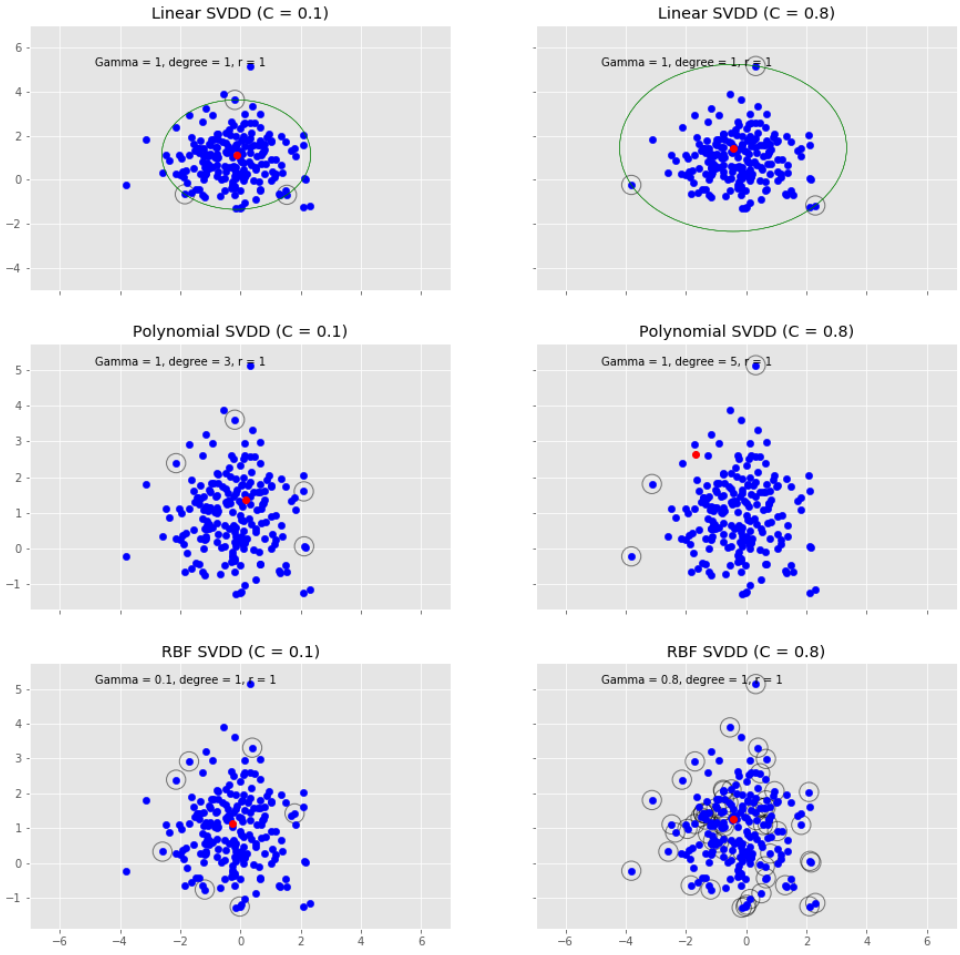

Figure 6: One Class SVM on benchmark dataset

As with the kernalized version of one class SVM (polynomial and rbf),

the enclosing ball is generated in a higher dimensional space, along with it centers and support vectors. This in turn leads to a distorted view of the hypersphere in 2-D projection (not plotted). This being said, the theory behind doesn't change, i.e. greater the value of C, higher the penalty on outliers and higher the volume of the hypersphere.

7 Conclusions

With the first lab session we revisited the concept of kernels (used with SVMs) and got to understand how the idea of using a kernel to project data into a higher dimensional wasn't exclusive to SVMs, rather it could be used in combination with PCA, K-means and even Logistic Regression. The effectiveness of these kernalized versions were further understood with experiments on the benchmark and test datasets. All in all, the lab session was extremely insightful and interesting, as we got to discover and play around with a concept that we initially imagined to be limited in its application. And we hope to use this new found knowledge to further our research, studies and contributions within the Machine Learning space.

References

[1] Kernel principal component analysis. [Online; accessed October 30, 2018].

[2] Test run - kernel logistic regression using cs. [Online; accessed October 31, 2018].

[3] Piyush Rai. Kernalised k-means, 2016. [Online; accessed October 30, 2018].

[4] David M.J. Tax and Robert P.W. Duin. Support vector data description. *Machine Learning*, 54(1):45–66, Jan 2004.