Imperial College London

Machine Learning Coursework 4

Youssef Rizk (00940962)

Questions to be marked: Problem 1, Problem 2 part A, Problem 3, and Problem 4

Examined by
Dr. András György

March 25, 2017

Contents

1 Problem 1

1.1 Part A

In this problem, we are asked to find the minimizer of the weighted empirical error with Tykhonov Regularization, $J(w)$, where

$$J(w) = \sum_{j=1}^{p}\sum_{i=1}^{n}\gamma_i((w^\top x_i)_j - y_j)^2 + \sum_{j=1}^{p}\sum_{k=1}^{d} w_{k,j}^2 \tag{1}$$

To begin this question, it is instructive to first note the dimensions of the various elements, specifically

$$\begin{aligned} x_i &\in \mathbb{R}^d \\ y_i &\in \mathbb{R}^p \\ w &\in \mathbb{R}^{d\times p} \end{aligned}$$

We can aggregate the x's and y's together into new matrices $X \in \mathbb{R}^{n\times d}$ and $Y \in \mathbb{R}^{n\times p}$. We can now rewrite the equation in (1) in terms of these matrices. This becomes

$$J(w) = (Xw - Y)^\top \Gamma (Xw - Y) + ||w||^2 \tag{2}$$

In order to minimize the above expression, we take the gradient and set it to 0 and subsequently solve for w as shown.

$$\begin{aligned} \nabla J(w) = 0 &= 2X^\top \Gamma(Xw - Y) + 2w \\ &= X^\top\Gamma(Xw - Y) + w \\ X^\top \Gamma Y &= X^\top \Gamma X w + w \\ &= (X^\top \Gamma X + I)w \\ &\boxed{\therefore w = (X^\top\Gamma X + I)^{-1}X^\top \Gamma Y} \end{aligned}$$

Thus, the boxed value of w above minimize the weighted empirical error with Tykhonov Regularization.

1.2 Part B

In this part of the question, we are asked to show that the resulting predictor for any data point, which is of the form $w^\top\phi(x)$ can be expressed using only K without computing $\phi(x)$ for any point. Defining some terms, $\phi(x)$ is a transformation from $\mathbb{R}^d \to \mathcal{Z}$, and K is the corresponding kernel, $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$.

Now, we define $Z = \phi(X)$ as the transformation of X where each row of the matrix is transformed from $\mathbb{R}^d \to \mathcal{Z}$. Taking $\phi(x) = z$, where x as a single point, we can then rewrite our predictor as

$$\begin{aligned} w^\top z &= ((Z^\top \Gamma Z + I)^{-1}X^\top\Gamma Y)^\top z \\ &= ((\gamma Z^\top Z + I)^{-1}\gamma Z^\top Y)^\top z && \text{Since } \gamma_i = \gamma \text{ for all } i \\ &= (\gamma Y^\top Z(I + \gamma Z^\top Z)^{-1})z && \text{Applying transposition} \\ &= (\gamma Y^\top (I + \gamma Z Z^\top)^{-1} Z)z && \text{Using identity } (I + AB)^{-1}A = A(I + BA)^{-1} \end{aligned}$$

Now, from this formulation of the predictor, we can note that $Z \in \mathbb{R}^{n \times q}$, where q is the dimension of the $\mathcal{Z}$-space. Examining the above expression term-by-term, we notice that $ZZ^\top$ represents the inner product of all the points with each other in the $\mathcal{Z}$-space. Furthermore, we notice that the term Zz corresponds to the inner product of all the training points with the point z. We realize, then, that we can replace both of these expressions with the appropriate kernel K which takes in input the appropriate original matrices, i.e. without the transformation applied. Thus, we can rewrite the above expression as

$$w^\top \phi(x) = \gamma Y^\top (I + \gamma K(X, X^\top))^{-1} K(X, x)$$

which shows that the predictor can be expressed only in terms of the kernel, without the need for actually computing the transformation ϕ for any feature vector.

2 Problem 2

2.1 Part A

We consider a binary classification problem with features in $\mathbb{R}^d$, where the classes are linearly separable with a positive margin. We want to show that the maximum margin classifier does not change when we remove a point from the training set that is not a support vector or add a new point which does not violate the current maximum margin. To set up the problem, we define $\mathcal{D}$ as the original training set, $\mathcal{D}^-$ as the training set obtained after removing a non-support vector data point from the set, and $\mathcal{D}^+$ as the training set obtained after the addition of a data point that does not violate the current maximum margin. When I refer to the addition of a point, I will implicitly assume that it does not violate the current maximum margin, from now on. I also define g as the maximum margin classifier of $\mathcal{D}$, g^- for $\mathcal{D}^-$, and g^+ for $\mathcal{D}^+$. I will specifically show that the maximum margin classifier is the same in these three sets in three steps.

1. The first step of this proof is to show that g is a separator in $\mathcal{D}^-$ and $\mathcal{D}^+$. This can be seen in the below figure.

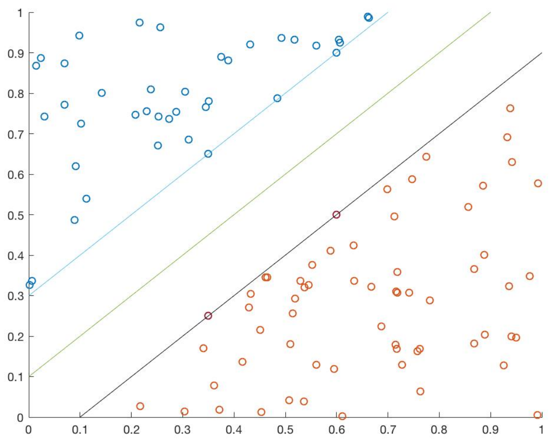


Figure 1: Example of linearly separable classes separated by a positive margin

The figure shows a hypothetical situation to aid in visualization. The green line in the middle represents g, and the distance between the green line and the two other lines is the positive margin separating the blue and red classes. Now, the data points shown here correspond to $\mathcal{D}$. Noticeably, $\mathcal{D}^-$ is created by removing one of the points from within the red class, or one from the blue class (specifically not a point on boundary lines since that would be a support vector). Similarly, $\mathcal{D}^+$ is achieved by adding a point to either class. It is trivially obvious that adding or removing a point to the corresponding class does not change the fact that g still separates the two classes.

2. As the second step, we show that the maximum margin classifier is unique. We do this by showing that the maximum margin classifier involves minimizing a convex function, namely $||w||^2$. We being by showing that $||w||$ is a convex function.

$$\begin{aligned} f(\alpha u + (1-\alpha)w) &\leq \alpha f(u) + (1-\alpha) f(w) && \text{Definition of convexity} \\ ||\alpha u + (1-\alpha)w|| &\leq ||\alpha u|| + ||(1-\alpha)w|| && \text{Substituting for } f \text{ and applying triangle inequality} \\ &= \alpha||u|| + (1-\alpha)||w|| \end{aligned}$$

Thus, $||w||$ is convex, as it satisfies the convexity inequality. Now, we notice that the function $f(t) = t^2$ is convex since $f''(t) = 2 > 0$. Thus,

$$\begin{aligned} ||\alpha u + (1-\alpha)w||^2 &= f(||\alpha u + (1-\alpha)w||) \\ &\leq f(\alpha||u|| + (1-\alpha)||w||) && \text{By convexity of } ||w|| \text{ and since } f \text{ is increasing} \\ &\leq \alpha f(||u||) + (1-\alpha) f(||w||) && \text{By convexity of } f(t) = t^2 \\ &= \alpha||u||^2 + (1-\alpha)||w||^2 \end{aligned}$$

Thus, we prove that $||w||^2$ is convex, which means that the solution to the maximum margin classifier is unique.

3. Now, we prove that g is the maximum margin classifier in $\mathcal{D}^-$ and $\mathcal{D}^+$. Suppose g^- has a larger margin than g on $\mathcal{D}^-$. Since the support vectors are the same in $\mathcal{D}$ and $\mathcal{D}^-$, the margins between the two classes are also the same. Thus, if g^- has a larger margin than g on $\mathcal{D}^-$, then it also has a larger margin on $\mathcal{D}$, which is a contradiction since g is the maximum margin classifier in that set. Similarly, suppose g^+ has a larger margin than g on $\mathcal{D}^+$, then by the same logic as before, it also has a larger margin on $\mathcal{D}$, which is again a contradiction. Thus, since the maximum-margin classifier is unique, and the fact that g^- and g^+ cannot have a larger margins on $\mathcal{D}^-$ and $\mathcal{D}^+$, respectively, it follows that $g = g^- = g^+$. Therefore, g is still the maximum margin classifier when we add or remove a point.

3 Problem 3

In this problem, we are asked to argue whether hard-margin SVM is always better than soft-margin SVM in a linearly separable case. In general, hard-margin SVM is not always better than soft-margin SVM. It is important to note that hard-margin SVM can be thought of as a subset of soft-margin SVM. The main problem with hard-margin SVM is that it is much more prone to overfitting than the soft-margin variant. This is because in the hard-margin case, any violations of the margin are not allowed. Consider the following figure to illustrate the point.

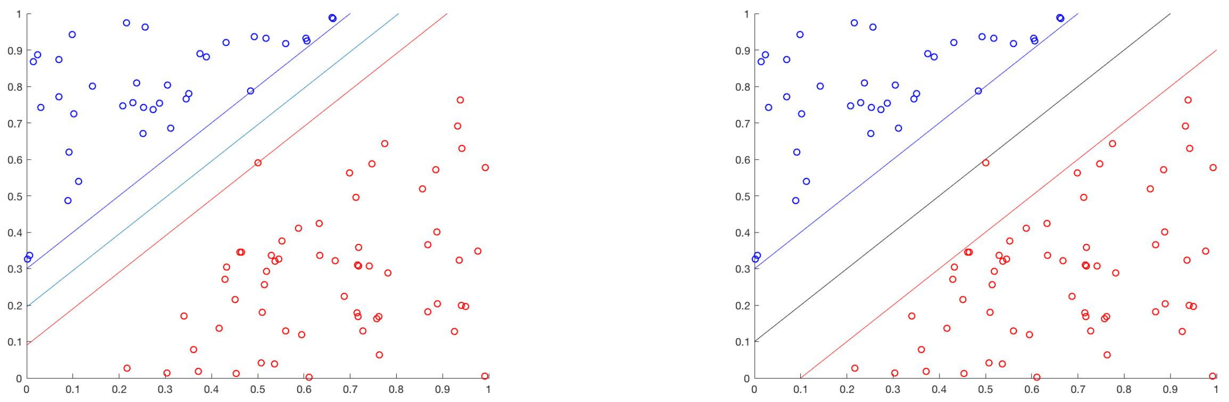


Figure 2: Hard-margin SVM (left) and soft-margin SVM (right) in a simple linear classifier case

Assuming the points are distributed as shown above, we consider the case where we want a linear separator to distinguish the two classes. We notice that there is an outlier in the red class that seems to be distant from the rest of the points in the same class, however, the data set in general is still linearly separable. If we train a separator using hard-margin SVM, we obtain the figure on the left above. Hard-margin does not recognize the outlier and so produces a classifier (light blue line in the middle) which is noticeably suboptimal and may have poor generalization performance. If we train a soft-margin SVM, however, we can control the value of the constant C to dictate essentially how sensitive the separator should be to any margin violations. With the correct choice of C, the soft-margin SVM may ignore the outlier and produce a much better-fitting separator, shown in the above figure to the right, that has superior generalization performance than in the hard-margin case as it is more immune to noise. Thus, using hard-margin SVM is not always better than using soft-margin SVM because the hard-margin variant may overfit the training data and manifest poor test performance.

4 Problem 4

In this problem, we return to the digit classification problem faced in Assignment 1. Specifically, we seek to improve upon the results of the Perceptron algorithm by using SVM to determine a classifier. The programs in this section are written in MATLAB.

4.1 Large-Margin Separator Using SVM

In this part, we aim to learn a large-margin separator using SVM to distinguish between digits 2 and 8. We begin this process by filtering the training set for data points describing either of these two digits, using a function I wrote called `filter1`. Then, I assign labels 1 to digit 2 and -1 to digit 8.

Having set up the matrices containing the input data and output labels, I am now ready to train the large-margin SVM to separate the two classes. To do this, I employ MATLAB's built-in function `fitcsvm`, which in its most basic use takes in input the input data matrix and the label matrix. I also specify the kernel function to be used, in this case the Gaussian (RBF) Kernel.

Operating under the assumption that we have no prior knowledge regarding the linear separability of the training data, it is safest to train a soft-margin SVM, which entails specifying the constraint C. In addition, the Gaussian Kernel has the form

$$K(x, x^\top) = e^{-\gamma ||x - x^\top||^2}$$

which requires a value for γ to be specified. Cross-validation comes into play here to determine the optimal values of γ and C. The function `fitcsvm` can take as argument `('OptimizeHyperparameters', 'auto')`, which automatically computes the optimal values for γ (i.e. 1/(Kernel Scale)) and C (i.e. Box Constraint). This configuration of the `fitcsvm` function is quite useful, as it attempts to minimize the cross-validation loss by varying the specified parameters (γ & C in this case). Specifically, the function chooses random values for kernel scale and box constraint and measures the cross-validation loss (specified as 10-fold) for this choice. It does this by default 30 times, and iteratively uses these measurements to model the true function describing the variation of cross-validation loss with kernel scale and box constraint, and ultimately reports the the values that achieve the minimum. The function automatically generates a 3D plot which shows the estimated variation of the objective function value (which is the cross-validation misclassification rate) against Kernel Scale and Box Constraints. The plot is shown below, along with the full configuration of the `fitcsvm`.

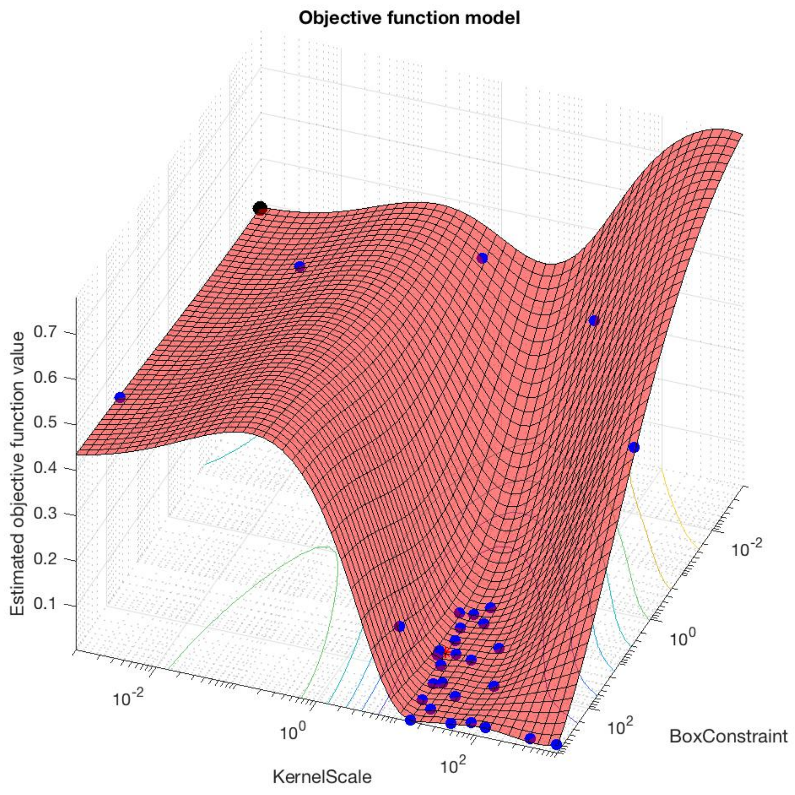


Figure 3: Validation error versus Kernel Scale ($1/\gamma$) and Box Constraint (C)

```
SVMModel = fitcsvm(rawTrain(:,2:end), trainLabels, 'Standardize', false,...,
    'KernelFunction', 'rbf','OptimizeHyperparameters', 'auto',...,
    'HyperparameterOptimizationOptions',...,
    struct('SaveIntermediateResults', true,...
    'Kfold',10));
```

In this configuration, the optimal value of kernel scale is around 11, while the box constraint ideal value varies greatly in the range of around 50-1000 (the training and test erros are quite similar for these, however, so it does not seem that the optimal value of C is unique or accurate). Using the values generated by cross-validation, I obtain a training error of 0, as was the case with the Perceptron. However, the superior performance of the SVM classifier becomes evident when examining the test errors of the two classifiers. From Assignment 1, we achieve a test error of 0.0549 using the Perceptron algorithm. Using the SVM classifier, we obtain a test error of 0.0302, a 45% reduction in test error.

4.2 Reducing Dimensionality through Principal Component Analysis

In this part of the problem, we aim to improve the performance of the classifier by reducing the dimension of the data set using a technique known as Principal Component Analysis (PCA). Essentially, we use the PCA algorithm and apply it to the training set, and the algorithm returns a matrix $V = (v_1, v_2, ..., v_d)$, where each column of the matrix represents the coefficients for a principal component, and the columns are in decreasing order relative to their singular values. Where $x \in \mathbb{R}^d$ is the input vector in question, we use $(v_1^\top x, v_2^\top x, ..., v_k^\top x)$ as the k-dimensional approximation of x, for $1 \leq k \leq d$.

Having now understood the mechanism for compressing the dimensionality of the data, it is useful to employ it with the aim of improving the performance of the classifier. We are given the matrix X which contains the input data, where every row corresponds to a data point. In order to obtain the k-dimensional approximation of X, we can simply compute $Z = X \times [v_1, v_2, ..., v_k]$, where $Z \in \mathbb{R}^{n \times k}$.

The task given in the problem is to write a program that, for any given k, trains a soft-margin SVM on the data for digits 2 and 8 using the k-dimensional approximation of the input data. In order to do this, the program begins by transforming the original data set X to Z with the specified value of k. I also transform the test set to its k-dimensional approximation. Next, I use MATLAB's built-in `fitcsvm` function to train a soft-margin SVM on the transformed training set, similarly to Part A. I compare the predicted labels against the actual labels assigned in order to calculate training and testing errors in a binary sense.

In particular, I am interested in determining the variation of the various errors with k. Thus, I calculate the training, test, and validation errors achieved by each k-dimensional approximation. Cross validation is essentially applied here to determine the optimal value of k which manifests the lowest validation error. Now, ideally, the values of the constants γ and C need to be cross-validated for each k. However, this becomes quite computationally expensive given that there are 256 values for k to cross-validate. As a result, I use the average values that were obtained for γ and C in the first part of the problem, specifically 0.1 and 1000, respectively (remember, the optimal value of C was not finely tuned). In the submitted code, however, the code for computing optimal values for γ and C is commented, and can be uncommented if the cross-validation of γ and C is wanted. The actual variation of the various error metrics with k is shown in Figure 4.

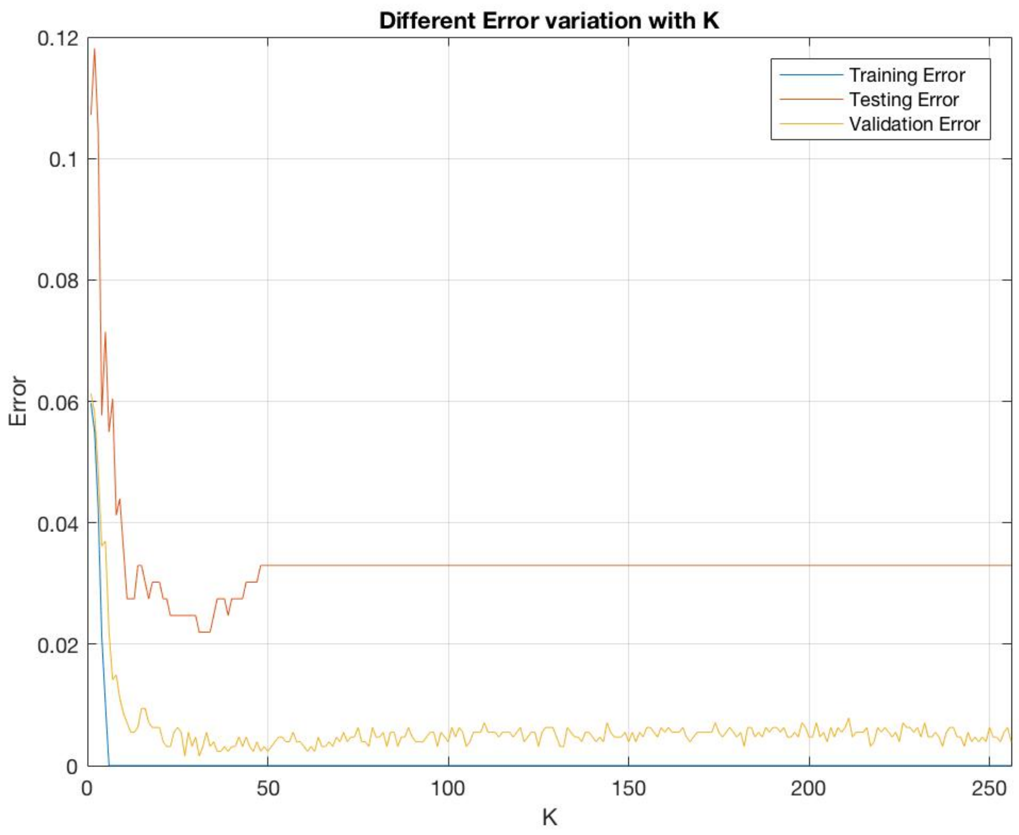


Figure 4: Variation of training, test, and validation error with k

We instantly notice a trend in the various errors, namely the sharp decrease in error as k increases for small values of k. Examining the validation error, we notice that it reaches its minimum at $k = 41$, indicating that this is the optimal value of k for this data set, and on average, the optimal value of $k_{optimal} \in [36, 42]$. We also note that the test error achieves its minimum of 0.0247 for $k = 31, 32, 33, 34$. Examining the training error, we note that it decreases to 0 and remains at the value starting from $k = 7$, indicating that the training data becomes linearly separable at that point.

We can make further observations about the variation of the errors. If we first examine the test error, we notice that there is a general decrease in test error until it reaches it minimum, followed by a slight increase and then a plateau. This higher test error for higher values of k may be attributed to a looser bound on the absolute difference between training and test error, due to the higher dimensionality of the input features. This may be rectified by increasing the number of training points such that this bound may decrease. If we consider training, then there is no need for additional data, since the error goes to 0 quite rapidly and stays there. In terms of validation error, we notice the significant decrease in error, followed by a very slight average increase in the value. Thus, as with test error, increasing the number of training points here may prove beneficial, however the effect would be quite minor.

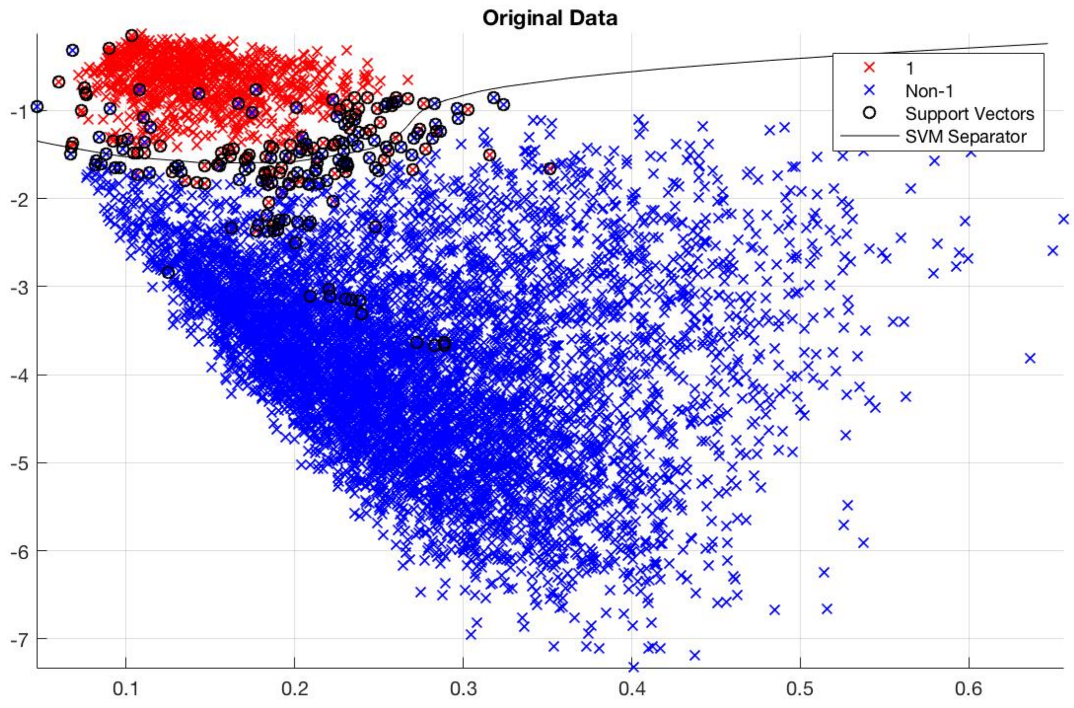
4.3 Comparing Digit 1 Against All Others

In this part of the problem, we shift our attention away from the 2 & 8 digit classification problem, and instead try to distinguish between digit 1 and all others in a 2D space. Specifically, we train a soft-margin SVM on two different data sets: the hand-crafted feature vectors provided and the 2-dimensional PCA approximation of the data. I show my findings using scatter-plots.

I begin by assigning labels to the input data, where a digit 1 is assigned a label 1 and all other digits are assigned -1. I obtain the 2-dimensional approximation of the training data, by employing the results of the PCA analysis conducted in the previous part. The 256-dimensional training data is again represented by matrix X, and the 2-dimensional approximation is determined similarly to the previous section, namely by calculating $Z = X \times [v_1, v_2]$. The same transformation is applied to the test data.

I begin by training an SVM classifier on the hand-crafted features, following which I train a different SVM classifier on the 2-dimensional approximation of the data (I again use the `('OptimizeHyperparameters', 'auto')` to obtain the classifier that uses the optimal values of γ and C). Figure 5 shows the resultant scatter plots, along with the decision boundaries in either case. Qualitatively, we can note that there appears to be perhaps a bigger margin separating the two classes in the case of the 2-dimensional approximation.

It is vital to examine the training and testing behaviors of the two models. In the program run corresponding to Figure 5, the first model obtained a training and test error of 0.0108 and 0.0189, respectively. The second model obtained significantly better results, achieving training and test errors of 0.0032 and 0.0075. Thus, it appears that the 2-dimensional approximation of the data produces significantly better results than the hand-crafted features.



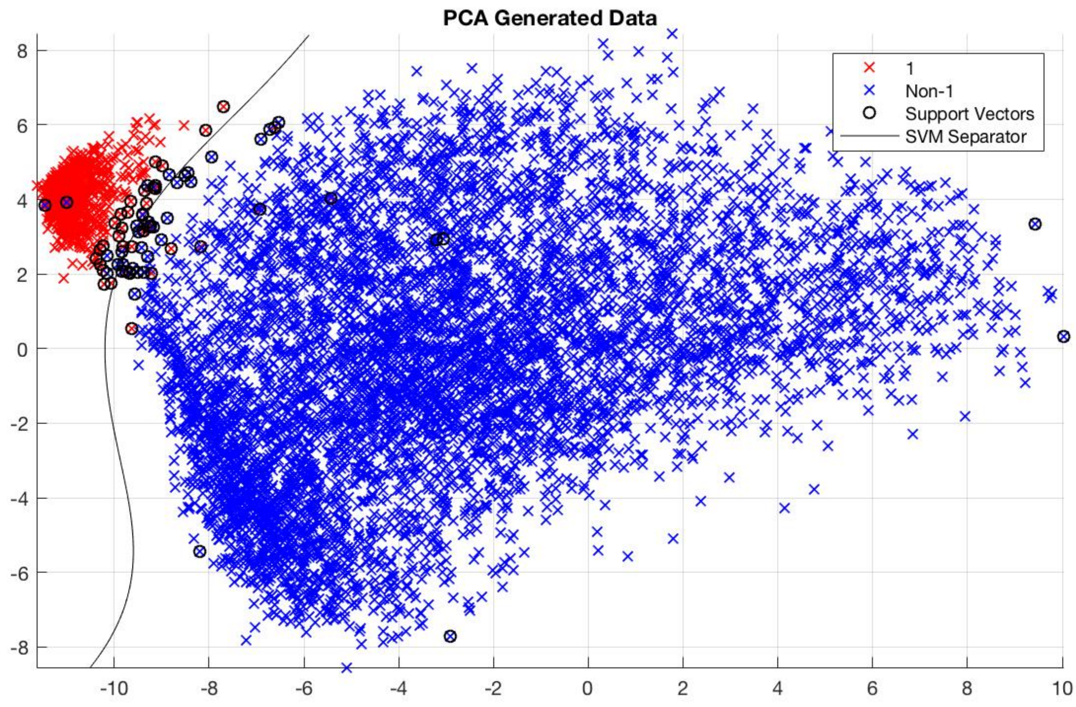


Figure 5: Scatter plots of hand-crafted features (top) and 2-dimensional approximation (bottom) and their corresponding decision boundaries