ECE 273: Convex Optimization **Spring 2018**

Final Project Report

Osman Cihan Kilinc (A53245359) - Kazim Ergun (A53248281)

1 Introduction

A support vector machine (SVM) is a supervised learning technique from the field of machine learning applicable to both classification and regression. From the author of 'Support Vector Networks' [1], the function of SVMs are described as follows: "The machine conceptually implements the following idea: input vectors are non-linearly mapped to a very high-dimensional feature space. In this feature space a linear decision surface is constructed.". This linear decision is constructed by solving an optimization problem, thus this project will focus on exploring the construction of the problem and further solving it with Convex Optimization algorithms.
The history of Support Vector Machines goes back to 1936 when R.A Fisher [2] suggested the first algorithm for pattern recognition, Linear Discriminant Analysis(LDA). The fundamentals for the SVM continued to build up with Aronszajn's(1950) 'Theory of Reproducing Kernels' [3], and the foundation of classification using linear discriminants consolidated with Rosenblatt's(1957) Perceptron Classifier [4]. The field of 'statistical learning theory' began with Vapnik and Chervonenkis (1974) [5] and SVMs can be said to have started when statistical learning theory was developed further with Vapnik (1979) [6]. Rooted from this statistical learning theory the original Support Vector Machines are developed based on Structural Risk Minimization again by Vapnik et al.(1995) [1].
In this paper, considering the historical background,we will follow the fundamentals that lead to the creation of SVM algorithms, then build the convex optimization problem and solve it using CVX Toolbox [7]. Particularly, we first talk about the concepts: Risk Minimization, Linear Discriminants, Kernels, Duality and KKT Conditions and put it all together to construct SVM algorithm. Then, we consider linearly separable data case and formulate a Hard-Margin SVM solution (via dual form), and review Soft-Margin SVM for not linearly separable case. Finally, using "kernel trick", we show how to non-linearly map the input space into a very high dimensional feature space and solve non-linear classification problems. The Hard-Margin SVM, Soft-Margin SVM and Kernel SVM are tested on both synthetic data and MNIST Handwritten Digits Dataset [8].

2 Preliminary

In this section we discuss the preliminary mathematical concepts that SVMs are build upon. We believe these are important to review before moving on to SVMs because they will provide insight on things that we have to justify later on. In order to have a clear and consistent explanation, we focus on binary classification case.
A classification problem has two types of variables, $x_i \in \mathbb{R}^d$, the vector of observations(features) in the world, and $y_i \in \mathbb{R}^d$ state(class) vector of the world. The set of all samples in this world (or dataset) constructs the observation set $\mathbf{X} \in \mathbb{R}^{nxd}$ and state set $\mathbf{Y} \in \mathbb{R}^{nxd}$. The task of our machine learning algorithm for classification is to find a mapping $x \rightarrow y$ that achieves smallest number of misclassification. To quantify the misclassification error, we define a loss $L[y, f(x, \alpha)]$ where $f(x, \alpha)$ is the function of mapping parametrized by α.

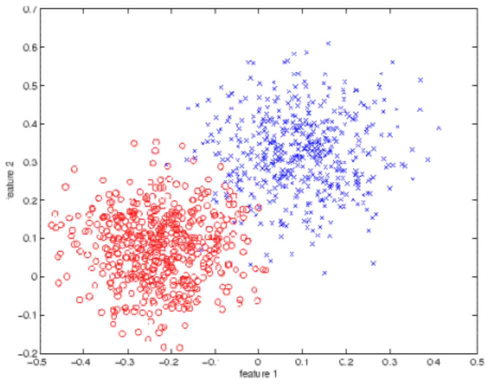


Figure 1: An Example Binary Classification Problem in 2D Space

2.1 Risk Minimization

After quantifying the error by the loss function, the goal of achieving smallest number of misclassifications can be restated as the goal to find a set of parameters α that minimizes the expected value of the loss, i.e the 'Risk'.

$$\begin{aligned} R(\alpha) &= \mathbb{E}\, L(y, f(x,\alpha)) \\ &= \int L(y, f(x,\alpha)) P(x,y)\, dxdy \end{aligned} \tag{1}$$

In literature, this is called the 'Actual Risk', however it is not always possible to calculate since it requires the knowledge of joint probability distribution $P(x,y)$. Therefore, the 'emprical risk', $R_{emp}(\alpha)$ is defined to measure the error rate on the training set for a fixed and finite number of observations.

$$R_{emp}(\alpha) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i,\alpha)) \tag{2}$$

There is an inequality relation between the actual risk and the emprical risk which was given by Vapnik(1995) [6]. The Equation (3) puts a bound on the actual risk in terms of the emprical risk, the number of observations, and a parameter $\eta \in [0,1]$, which is the probability associated with loss function such that it takes two different values with probabilities η and $1-\eta$. Also, there is a parameter h called the 'Vapnik Chernonenkis (VC)' dimension, where details for it can be found in [6].

$$R(\alpha) = R_{emp}(\alpha) + \sqrt{\frac{h(log(2n/h) - log(\eta/4)}{n}} \tag{3}$$

To find the best mapping function $f(x,\alpha)$, we do *emprical risk minimization* that gives us the model with minimum error over the dataset X.

$$R^* = \inf_{f(\alpha):X\to Y} R_{emp}(\alpha) \tag{4}$$

2.2 Linear Discriminant

In classical Bayes Decision Theory, the classification is made with a prior assumption on class densities. However, estimating these densities then deriving the boundary is not always a good strategy. Density estimation is an 'ill-posed' problem, since a slight change in problem conditions may result in an arbitrarily large change in result. Therefore, directly working on the decision function, which creates the boundary that discriminates the classes is a better idea. In discriminant learning, we first propose a parametric family of decision boundaries, then pick the element in this family that produces the best classifier. If we take our decision boundary between two classes as a hyperplane, such that $w^T x + b = 0$, w being the normal to the plane and b being the bias term, the decision function can be expressed as:

$$f^*(x) = \begin{cases} 0, \text{ if } w^T x + b > 0 \\ 1, \text{ if } w^T x + b < 0 \end{cases} \tag{5}$$

Going back to simple binary classification example, if we plot the decision boundary as in Figure 2. we can make a geometric interpretation. The hyperplane $w^T x + b$ divides the spaces into two half-spaces. The boundary is the plane with normal w and has a distance to origin $d = b/||w||$. Furthermore, a data point x_i has a distance $|w^T x_i + b|/||w||$ to the decision boundary. Any point on the decision boundary will have $w^T x + b = 0$, on the side w points to will have $w^T x + b > 0$ (blue points), and on the side the opposite of w points to will have $w^T x + b < 0$ (red points).

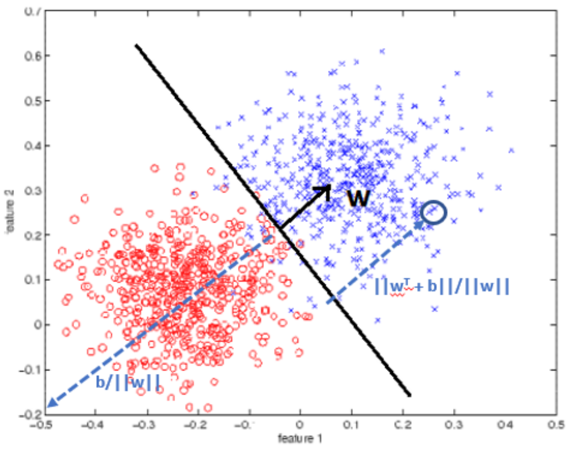


Figure 2: Linear Discriminant

The one handy trick that SVM use, as well as other discriminant learning algorithms is that having class labels $y \in -1, 1$ instead of $y \in 0, 1$.Then the decision function becomes:

$$f^*(x) = \begin{cases} -1, \text{ if } w^T x + b > 0 \\ 1, \text{ if } w^T x + b < 0 \end{cases} \Rightarrow f^* = sgn[w^T x + b] \tag{6}$$

This enables us to have a simple *necessary and sufficient* condition for having *zero* empirical risk given the training set $D = (x_1, y_1), (x_2, y_2, ..., (x_n, y_n)$ is linearly separable. The condition is

$$y_i(w^T x + b) > 0, \forall i \tag{7}$$

2.3 Kernels

If a problem is non-linear, instead of trying to fit a non-linear model, one can map the problem from the input space to a new (higher-dimensional) space (called the feature space) by doing a non-linear transformation using suitably chosen basis functions and then use a linear model in the feature space. This is known as the 'kernel trick'. Learning a linear boundary in a higher dimensional space will be equivalent to learning a non-linear boundary in the current space. Doing this, we can avoid implementing high complexity classifiers. A linear model in the feature space corresponds to a non-linear model in the input space. The need for kernels are very natural since no model in real world is truly linear. A feature space transformation followed by a linear discriminant is highly useful and we can see the examples of this in many machine learning algorithms (e.g. Neural Networks, SVMs).
If we introduce a mapping $\phi : X \to Z$ such that $dim(Z) > dim(X)$ we can achieve linear separability for some $k = dim(Z)$. However, instead of applying this mapping to each data point as $\phi(x)$, we can leverage the dot-product form that occurs in most machine learning algorithms. For example, in dot-product form we can express the binary classifier and its feature space transformed version as:

$$\begin{aligned} &y_i(\sum_{j=1}^{n} \alpha_j y_j x_j^T x_i + b) \\ &y_i(\sum_{j=1}^{n} \alpha_j y_j \phi(x_j)^T \phi(x_i) + b) \end{aligned} \tag{8}$$

Then we can apply the 'kernel-trick', where we call $K(x, z) = \phi(x)^T \phi(z)$. This means that we do not have to apply the transformation on each data point one by one and compute their dot-product, instead we can simply define the kernel $K(x, z)$ directly. In practice, we do not care about the actual mapping function $\phi(x)$ that generates our kernel, we generally pick a kernel from library of know kernels. Examples for these kernels can be: 1) the linear kernel, $K(x, z) = x^T z$, 2) gaussian kernel, $K(x, z) = e^{\frac{||x-z||^2}{\sigma}}$ or 3) the polynomial kernel, $K(x, z) = (1 + x^T z)^k$.

3 Support Vector Machines

The methods presented in the previous sections, namely risk minimization, forming optimal separating hyperplanes, and the kernel technique are elegantly combined in a learning algorithm known as Support Vector Machines. SVMs go one step further from the necessary and sufficient condition for zero empirical risk given in Equation (7) and works on the *margin*, that is the distance from the boundary to the closest point. The margin is defined as:

$$\gamma = \min_i \frac{|w^T x_i + b|}{||w||} \tag{9}$$

Maximizing the margin defined in Equation(9) is ill-defined in the sense that γ does not change if both w and b are scaled by a factor λ. Therefore, we need some sort of normalization and this can be done by arbitrarily selecting some normalization on w, e.g. choosing $||w|| = 1$. The SVM algorithm works with a more convenient normalization and makes $|w^T x + b| = 1$ for the closest point, i.e. $\min_i |w^T x_i + b|$ under which $\gamma = 1/||w||$. Then the SVM is the classifier that maximizes the margin under these constraints:

$$\begin{aligned} \underset{\mathbf{w,b}}{\text{minimize}} \quad & \frac{1}{2}||w||^2 \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1, \ \forall i \end{aligned} \tag{10}$$

In the following sections, we provide the derivations for the linear hard margin SVM, linear soft margin SVM and nonlinear SVM with kernel trick. Going forward with the mathematical solution, we use the Lagrangian Dual for the formulations and find a solution satisfying Karush-Kuhn-Tucker. Since the problem for SVMs is convex (a convex objective function, with constraints which give a convex feasible region), and for convex problems (if the regularity condition holds), the Karush-Kuhn-Tucker conditions are necessary and sufficient, thus solving the SVM problem is equivalent to finding a solution to the KKT conditions.

3.1 Hard Margin

Hard Margin SVM is the linear 'vanilla' version of the SVM which is given in the Equation (10). This version of SVM is only applicable for the linearly separable case. Switching to the dual of our primal problem in (10), we introduce Lagrangian multipliers $\alpha_i, \forall i$, for each of the inequality constraints. The dual is:

$$g(\alpha) = \inf_{\alpha \geq 0} \mathcal{L}(w, b, \alpha) \tag{11}$$

where the Lagrangian $\mathcal{L}$ is given by

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_i \alpha_i [y_i(w^T x_i + b) - 1] \tag{12}$$

This is a convex quadratic problem, therefore $\mathcal{L}$ is minimized at the point where gradient of $\mathcal{L}$ with respect to w, b, and α_i vanishes.

$$\begin{aligned} \nabla_w \mathcal{L} = 0 \quad &\Longleftrightarrow w^* = \sum_i \alpha_i y_i x_i \\ \nabla_b \mathcal{L} = 0 \quad &\Longleftrightarrow \sum_i y_i \alpha_i = 0 \end{aligned} \tag{13}$$

Plugging back (13.1) and (13.2) to Lagrangian we get

$$\mathcal{L}(w^*, b, \alpha) = -\frac{1}{2} \sum_{ij} \alpha i \alpha j y_i y_j x_i^T x_j + \sum_i \alpha_i \tag{14}$$

This wraps up the dual problem formulation and it is given by:

$$\begin{aligned} \underset{\alpha \geq 0}{\text{maximize}} \quad & -\frac{1}{2} \sum_{ij} \alpha i \alpha j y_i y_j x_i^T x_j + \sum_i \alpha_i \\ \text{subject to} \quad & \sum_i \alpha_i y_i = 0 \end{aligned} \tag{15}$$

Once problem in Equation (15), the vector $w^* = \sum_i \alpha_i y_i x_i$ is the normal to the maximum margin plane. However, while w^* is explicitly determined, the dual solution does not determine the optimal b^*, since b drops off when we take derivatives. There are various ways to choose b, one of them is to average over all points on the margin. The points located on the margin are called *support vectors*, and can be found using KKT conditions. From the KKT conditions, an inactive constraint has zero Lagrange multiplier α_i, i.e $\alpha_i > 0$ and $y_i(w^T x_i + b^*) = 1$ or $\alpha_i = 0$ and $y_i(w^T x_i + b^*) \geq 1$. Hence, $\alpha_i > 0$ only for points $|w^T x_i + b^*| = 1$, which are those who lie at a distance equal to the margin. Remembering that the linear discriminant decision rule is:

$$\begin{aligned} f(x) &= sgn[w^T x_i + b^*] \\ &= sgn[\sum_i y_i \alpha_i^* x_i^T x + b^*] \\ &= sgn[\sum_{i \in SV} y_i \alpha_i^* x_i^T x + b^*] \end{aligned} \tag{16}$$

where $SV = \{i | \alpha_i^* > 0\}$ is the set of support vectors. Since the points with $\alpha_i = 0$ have no influence on the decision, we can simply throw away all other points.

3.1.1 Implementation

We solved the dual form of the optimization problem in Equation (15) using the CVX Toolbox [7] in MATLAB. The function below takes the training set X and labels Y, returns the SVM model.

```
function model = hardmargin_svm(X,Y)

% Hard Margin SVM

    K=X*X'; % Dot-product kernel
    N = size(X,1);

    cvx_begin
        cvx_precision best
        variable a_m(N);
        minimize (0.5.*quad_form(Y.*a_m,K) - ones(N,1)'*(a_m));
        subject to
            a_m ≥ 0;
            Y'*(a_m) == 0;
    cvx_end
% Removing data points which are not on the margin boundary
    a_m(a_m<10^-5)=0;
    X_m = X(a_m>0,:);
    Y_m = Y(a_m>0);
    a_m = a_m(a_m>0);
    K=X*X_m';
    b_m=mean(Y-K*(a_m.*Y_m));
%Return the SVM train model
    model = getModel(X_m, Y_m, a_m, b_m, 0, "hard")
end
```

3.2 Soft Margin

If the hard margin SVM is applied to a non-separable data, it won't find a feasible solution. However, a separable case is rarely the case in practice so we need to extend the SVM to the non-separable case. When classes overlap, we cannot enforce a margin that separates two classes completely, but we can enforce a *soft margin*. This can be done by introducing slack variables. Instead of solving the problem in Equation (10), we solve the problem

$$\begin{aligned} \underset{\mathbf{w},\xi,\mathbf{b}}{\text{minimize}} \quad & \frac{1}{2}||w||^2 \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i, \ \forall i \\ & \xi_i \geq 0, \forall i \end{aligned} \tag{17}$$

However, the problem is not very well defined in the sense that by making ξ_i arbitrarily large, any w can satisfy the constraints. Hence, we add a regularizing term to the objective function that penalizes large ξ_i. The intuition behind this is as $/xi_i$ grows, the amount of error for that point grows. One way we can assign this regularization term is by using the L1-norm of ξ (there are other varieties of soft-margin SVM, i.e using L2-norm). With the addition the regularization term, the new objective function is

$$\underset{\mathbf{w},\xi,\mathbf{b}}{\text{minimize}} \quad \frac{1}{2}||w||^2 + C\sum_i \xi i \tag{18}$$

The term $\sum_i \xi i$ also can be thought as an upper bound on the number of training errors because for an error to occur, ξ_i must exceed 1. C is a constant parameter chosen by user to adjust the importance of giving penalty to errors. Similar to the hard margin SVM, we solve this convex optimization problem by using its Lagrangian dual. The Lagrangian and gradient equations following the optimality condition is

$$\begin{aligned}
&\mathcal{L}(w,b,\xi,\alpha,r) = \frac{1}{2}||w||^2 + C\sum_i \xi_i + \sum_i \alpha_i[1-\xi_i - y_i(w^T x_i + b)] - \sum_i r_i\xi_i \\
&\nabla_w \mathcal{L} = 0 \iff w^* = \sum_i \alpha_i y_i x_i \\
&\nabla_b \mathcal{L} = 0 \iff \sum_i y_i\alpha_i = 0 \\
&\nabla_{\xi_i} \mathcal{L} = 0 \iff C - \alpha_i - r_i = 0
\end{aligned} \tag{19}$$

Plugging back the results of gradient equations, we have

$$\begin{aligned}
&\mathcal{L}(w^*,b,\xi,\alpha,r) = -\frac{1}{2}\sum_{ij} \alpha i\alpha j y_i y_j x_i^T x_j + \sum_i \alpha_i \\
&\textit{with constraint}\alpha_i = C - r_i
\end{aligned} \tag{20}$$

The only difference from hard margin SVM is that here we have an extra constraint $\alpha_i = C - r_i \forall i$. Since r_i are Lagrange multipliers, $r_i \geq 0, \forall i$ which results in $\alpha_i \leq C$. Thus, the dual form with a small modification over hard margin SVM dual becomes

$$\begin{aligned}
\underset{\alpha\geq 0}{\text{maximize}} \quad & -\frac{1}{2}\sum_{ij} \alpha i\alpha j y_i y_j x_i^T x_j + \sum_i \alpha_i \\
\text{subject to} \quad & \sum_i \alpha_i y_i = 0 \\
& 0 \leq \alpha_i \leq C
\end{aligned} \tag{21}$$

From KKT conditions we may have some deductions: 1) $r_i > 0 \Leftrightarrow \xi_i = 0$ or $r_i = 0 \Leftrightarrow \xi_i$, 2) $\alpha i[1-\xi_i - y_i(w^T x_i + b)] = 0$. Since $\alpha_i = C - r_i$ it implies that: $\xi_i = 0$ *and* $\alpha_i < C$ so we have $y_i(w^T x_i + b) = 1$ (i.e. x_i on the margin), or $\xi_i > 0$ *and* $\alpha_i = C$ so we have $y_i(w^T x_i + b) = 1 - \xi_i$ (i.e. x_i is an outlier). Also,as before the decision rule is $f(x) = sgn[\sum_{i\in SV} y_i\alpha_i^* x_i^T x + b^*]$ where $SV = \{i|\alpha_i^* > 0\}$ is the set of support vectors. We can remove the points x_i which are outliers according to the condition explained.

3.2.1 Implementation

The implementation of soft margin SVM looks very similar to the previous case.

```
function model = softmargin_svm(X,Y,C,sigma)

% Soft Margin SVM (Using Soft dual for 1 Norm for slack variables)

K=X*X'; % Dot-product kernel

N = size(X,1);

cvx_begin
    cvx_precision best
    variable a_m(N);
    minimize (0.5.*quad_form(Y.*a_m,K) - ones(N,1)'*(a_m));
    subject to
        a_m ≥ 0;
        Y'*(a_m) == 0;
        a_m ≤ C; % New Constraint
cvx_end
% Remove outliers
a_m(a_m<10^-5)=0;
X_m = X(a_m>0,:);
```

```
Y_m = Y(a_m>0);
a_m = a_m(a_m>0);

K=X*X_m';

b_m = mean(Y-K*(a_m.*Y_m));
% Return model
model = getModel(X_m, Y_m, a_m, b_m, sigma, "soft");

end
```

3.3 Nonlinear Classification with Kernel Trick

The linear SVM's can be easily converted to be able to handle the cases where the data is non-linearly distributed. This is done via 'kernel trick' which we have discussed in Section 2.3. This trick exploits the fact that the decision function of SVM uses 'dot-product' in both training and decision function phases. If we introduce a mapping $\phi : X \rightarrow Z$ such that $dim(Z) > dim(X)$ we can achieve linear separability for some $k = dim(Z)$, where k can go to infinity. However, instead of applying this mapping to each data point as $\phi(x_i)$, we can leverage the dot-product and we can apply the 'kernel-trick', where we call $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$. We would only need to use the kernel in the training algorithm, and would never need to explicitly even know what ϕ is. The dual problem for the SVM with kernel trick is similar to the soft margin SVM, with dot-products replaced by kernels. We skip the dual problem formulation since it is the same as soft margin SVM.

$$\begin{aligned} \underset{\alpha\geq 0}{\text{maximize}} \quad & -\frac{1}{2}\sum_{ij} \alpha i \alpha j y_i y_j K(x_i, x_j) + \sum_i \alpha_i \\ \text{subject to} \quad & \sum_i \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \end{aligned} \tag{22}$$

Also, the decision function should be kernelized.

$$f(x) = sgn[\sum_{i\in SV} y_i \alpha_i^* K(x_i, x) + b^*] \tag{23}$$

3.3.1 Implementation

In our implementation for nonlinear SVM, we use a gaussian kernel, so called Radial Basis Function (RBF). The RBF kernel is

$$K(x_i, x_j) = e^{\frac{||x_i - x_j||^2}{\sigma}} \tag{24}$$

We modify our code to apply this kernel both for training and decision phases.

```
function model = nonlinear_svm(X,Y,C,sigma)

% Nonlinear SVM with RBF Kernel
N = size(X,1);
K = zeros(N,N);
for i=1:N % RBF kernel
    for j = 1:N
            K(i,j) = exp(-1*sum((X(i,:)-X(j,:)).^2)/(2*(sigma)^2));
    end
end

```

```
cvx_begin
    cvx_precision best
    variable a_m(N);
    minimize (0.5.*quad_form(Y.*a_m,K) - ones(N,1)'*(a_m));
    subject to
        a_m ≥ 0;
        Y'*(a_m) == 0;
        a_m ≤ C;
cvx_end

a_m(a_m<10^-5)=0;
X_m = X(a_m > 0,:);
Y_m = Y(a_m > 0);
a_m = a_m(a_m > 0);

K=zeros(length(X(:,1)),length(X_m(:,1)));
for i=1:length(X(:,1)) % RBF kernel
    for j = 1:length(X_m(:,1))
            K(i,j) = exp(-1*sum((X(i,:)-X_m(j,:)).^2)/(2*(sigma)^2));
    end
end

b_m = mean(Y-K*(a_m.*Y_m));
% Return the SVM trainmodel
model = getModel(X_m, Y_m, a_m, b_m, sigma, "nonlinear");

end
```

4 Testing

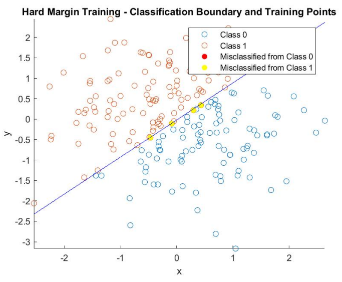
The testing done for the classification task on two datasets. First dataset is a binary class two dimensional set created synthetically. The second dataset is MNIST Handwritten Digits Dataset [8]. Synthetic dataset is generated for both linearly separable and non-separable classes.
Synthetic data is generated with following code segment:

```
%% Create a Basic Problem:
X = randn(500,2);
Y = double((X(:,1) > X(:,2)))*2-1;
X = zscore(X);
sigma = 1;
X_test = X(201:end,:);
X = X(1:200,:);
Y_test = Y(201:end);
Y = Y(1:200);
```

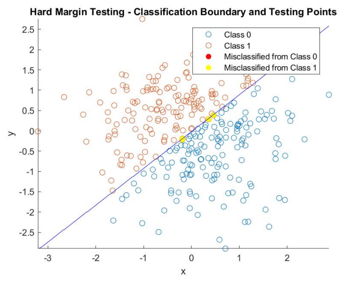
5 Results

5.1 Hard Margin

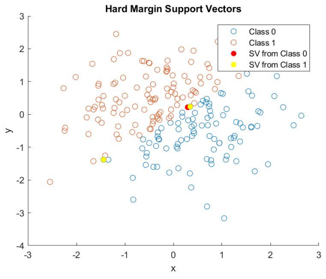
The data generated for hard margin SVM is linearly separable. The support vectors (the data points left after removing the points that does not lie on the margin) are shown in the third plot.



(a) Training Set



(b) Test Set



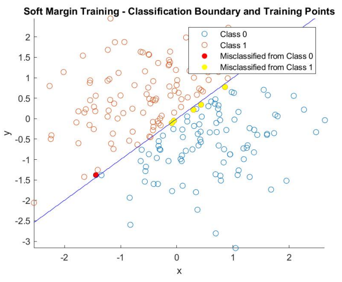
(c) Support Vectors

Figure 3: *Hard Margin SVM*

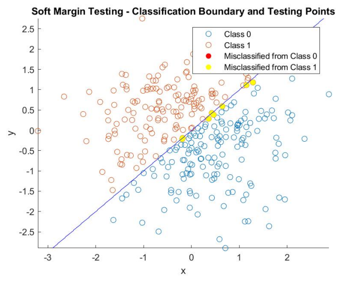
5.2 Soft Margin

5.2.1 Synthetic Data

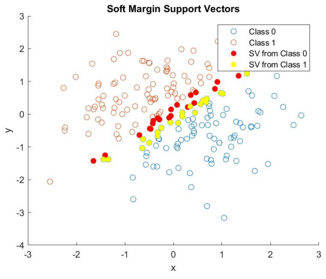
The data generated for soft margin SVM is not linearly separable, thus we can't have a line perfectly separating two classes. This time the support vectors are the points left within a 'soft' margin as shown in the third plot.



(a) Training Set



(b) Test Set



(c) Support Vectors

Figure 4: *Soft Margin SVM*

5.2.2 MNIST Dataset

Using MNIST dataset, we tried our algorithm for classification of two digits, namely '6' and '8'. Accuracy for this classification is **89.6%**. Support vectors for both classes are shown in the Figure 5 and Figure 6.

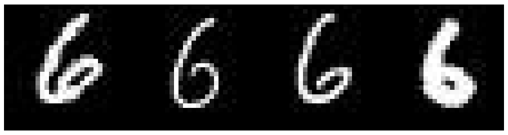
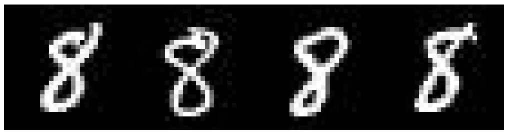

(a) SV1 (b) SV2 (c) SV3 (d) SV4

Figure 5: *Support Vectors for Class '6'*

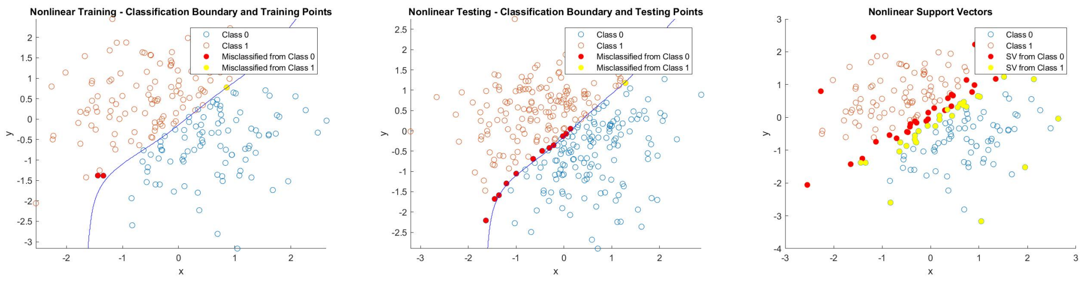


(a) SV1 (b) SV2 (c) SV3 (d) SV4

Figure 6: *Support Vectors for Class '8'*

5.3 Nonlinear Classification with Kernel Trick

For nonlinear classification we used Radial Basis Function kernel, thus the decision boundary is not linear.



(a) Training Set (b) Test Set (c) Support Vectors

Figure 7: *Nonlinear Kernelized SVM*

6 Conclusion

In this project, we first reviewed the mathematical concepts that build a foundation for the SVMs. Then, we investigated hard-margin and soft-margin formulations of linear SVMs. Particularly, we showed how to solve soft-margin SVMs via dual formulation, and justified how the dual problem will give the optimal solution of primal form. Moreover, we discussed how to solve non-linear classification via SVMs using the kernel trick. For all the cases, we identified which data points we can remove without affecting the SVM solution. Finally, implemented our own SVM algorithms using CVX Toolbox on MATLAB, and tested them on both generated synthetic data and real datasets. That being said, overall this project helped us apply our convex optimization knowledge to construct and solve an ubiquitous problem in statistical learning field.

7 How To Run the Code

The MATLAB codes provided are written in multiple .m files. The functions for Hard Margin SVM. Soft Margin SVM, and Nonlinear SVM are separated. To run the test script for synthetic data, you can use **'test_plot.m'** file. To run the script for MNIST data, you can use **'test_mnist.m'** file. The scripts also provide accuracy result for the **'libsvm'** library to have a comparison.

References

[1] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[2] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of human genetics*, vol. 7, no. 2, pp. 179–188, 1936.

[3] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American mathematical society*, vol. 68, no. 3, pp. 337–404, 1950.

[4] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[5] V. N. Vapnik and A. Y. Chervonenkis, "Ordered risk minimization. I," *Avtomat. i Telemeh.*, no. 8, pp. 21–30, 1974.

[6] V. Vapnik, *The nature of statistical learning theory.* Springer science & business media, 2013.

[7] M. Grant and S. Boyd, *CVX: Matlab software for disciplined convex programming, version 2.1*, `http://cvxr.com/cvx`, Mar. 2014.

[8] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: `http://yann.lecun.com/exdb/mnist/`.