Lisa Meng
Shagun Gupta
INF 522
Homework 6

# REPORT

## Part 1: Implementation

The Quadratic Programming solver we used was a library function from CVXOPT, a software package for convex optimization based on the Python programming language.

```
from cvxopt import matrix, solvers
```

Although Pandas was used to read/import the data files, NumPy array was the data structure used throughout the functions because the CVXOPT library requires CVXOPT matrices, and NumPy arrays and CVXOPT matrices are compatible and exchange information using the Array Interface.

Functions were defined to optimize the code and separate different parts of the code. In addition to optimization, C, the constant representing hardness of margin, was set to $1x10_5$ . We decided on a large C value because for very large C, the margin is hard, and points cannot lie in it. For smaller C, the margin is softer, and can grow to encompass some points. We also wanted to be consistent as $C = 1x10_5$ was also used in Part 2 when implementing the SVM algorithm with library functions. And of course, the software package CVXOPT optimized our code by solving the quadratic equation using Lagrange multiplier**.**

The greatest challenge we faced was researching and thoroughly understanding how to use CVXOPT and how it's functions work.

*Part (a) - Linearly Separable Data*

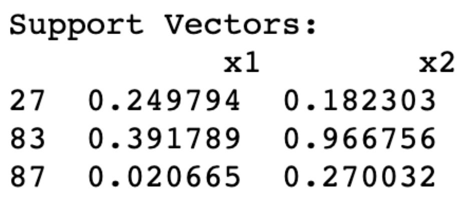
First, we defined a function to calculate the α's, using the data points (X), their labels (y), and C as parameters. This function used the Quadratic Programming solver in the CVXOPT library to solve for the α's using the method of Lagrange multiplier. Using the α's that we calculated as a parameter, we were able to solve for w and b, using these equations listed below:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

$$b = \underset{i \in N}{\text{average}} \frac{1}{y_i} - \mathbf{w}^T \mathbf{x}_i$$

```
w = [ 7.25005631 -3.86188926]
b = -0.10698733460882415
```

We then determined the set of support vectors by finding the indices such that $\alpha_i > 0$.

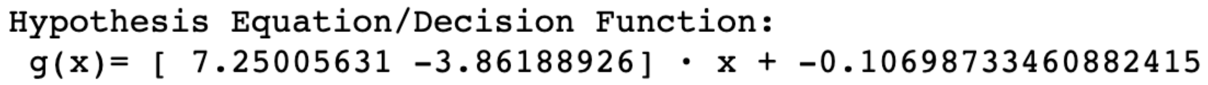


```
Support Vectors:
           x1        x2
27  0.249794  0.182303
83  0.391789  0.966756
87  0.020665  0.270032
```

With the w and b we solved for, we are able to find the equation of the hyperplane line, which takes the form of:

General equation is $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ OR $\mathbf{w} \cdot \mathbf{x} + b = 0$

Thus the equation of our line is:



```
Hypothesis Equation/Decision Function:
 g(x)= [ 7.25005631 -3.86188926] · x + -0.10698733460882415
```

*Part (b) - Non-Linearly Separable Data*

The RBF Kernel was used to implement the Kernel Function. The formula used is:

$$K(X_1, X_2) = exponent(-\gamma \|X_1 - X_2\|^2)$$

||X1 — X2 || = Euclidean distance between X1 & X2

Gamma was calculated using the following formula:

$$\boldsymbol{\gamma} = \tfrac{1}{2} * \{\boldsymbol{\sigma}_2(x_1) + \boldsymbol{\sigma}_2(x_2)\}$$

Upon research it was discovered that standard values of gamma is 0.1. The higher the value of gamma, the higher the chances of over-fitting. The gamma value sets the width of the bell-shaped curve. A technique called normalisation was used to calculate the value of gamma. It prevents scaling because it is calculated on the basis of X.

The support vectors obtained are:

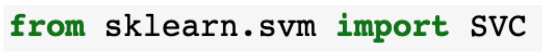
```
Support Vectors:
             x1         x2
5    -8.474228   5.156216
36  -10.260969   2.073918
51    1.339331 -10.290988
55    9.679177   4.375954
71   -9.467609   2.361395
94   -6.800023  -7.023843
95    9.901435  -0.314831
```

The final equation obtained is:

```
Hypothesis Equation/Decision Function:
 g(x)= [-21.94332854  -3.11296292] · x + -170.9016698899817
```

## Part 2: Software Familiarization

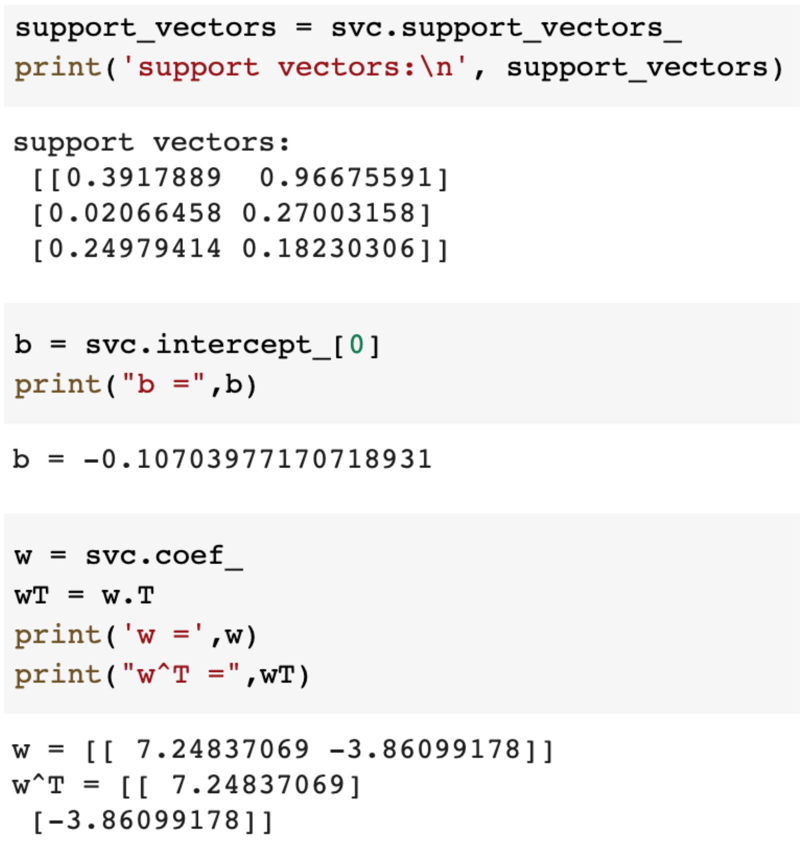
After thorough research, we decided to use the SKLearn library because it had the most relevant and efficient functions for the SVM algorithm.



```
from sklearn.svm import SVC
```

*Part (a) - Linear Separable Data*

Since our data set is linearly separable, when using the SKLearn library, we set the kernel to “linear” and C to 1x10$^{5}$. Note that we want the margin to be hard, so no data points lay within the margins, thus we set C to a high value. Using the library functions, we were easily able to determine w, b, and the support vectors.



```
support_vectors = svc.support_vectors_
print('support vectors:\n', support_vectors)
```

```
support vectors:
 [[0.3917889  0.96675591]
 [0.02066458 0.27003158]
 [0.24979414 0.18230306]]
```

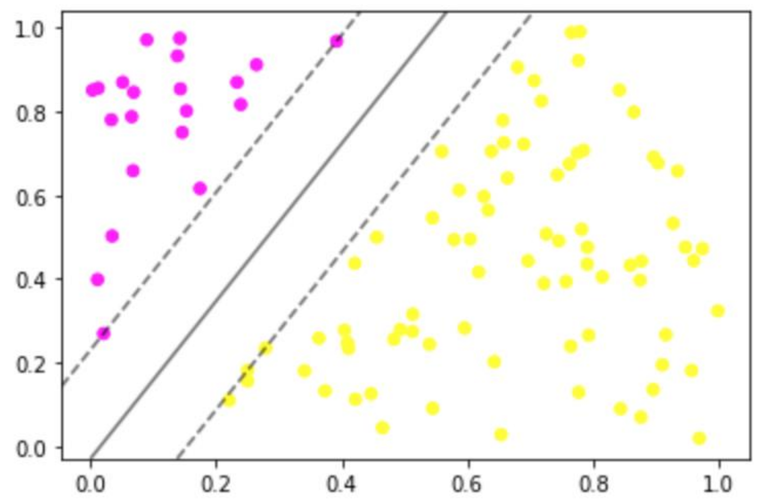
```
b = svc.intercept_[0]
print("b =",b)
```

```
b = -0.10703977170718931
```

```
w = svc.coef_
wT = w.T
print('w =',w)
print("w^T =",wT)
```

```
w = [[ 7.24837069 -3.86099178]]
w^T = [[ 7.24837069]
 [-3.86099178]]
```

Using matplotlib, we also created a function to visualize the equation of the line and the margins against the data points.



With the w and b we obtained from the library functions, we can determine the equation of the line:

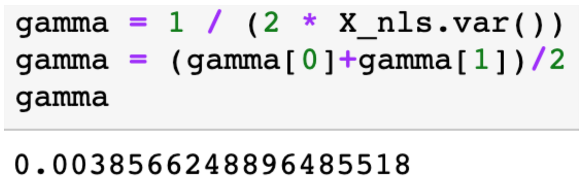
```
Hypothesis Equation/Decision Function:
 g(x)= [[ 7.24837069]
 [-3.86099178]] x + -0.10703977170718931
```

Because our code from scratch has minimal lines and returned the same answer as using a library, we believe that our code does not require further improvement in terms of effectiveness and accuracy.

*Part (b) - Non-Linear Separable Data*

The data set for Part(b) is non-linearly separable, thus when using the SKLearn library, we set the kernel to "rbf" and C to 1x10$_5$. Again, we want the margin to be hard, so no data points lay within the margins, thus we set C to a high value. We used RBF, Radial Basis Function, as our kernel function because after experimenting with the other functions (polynomial and sigmoid), RBF provided the best results.

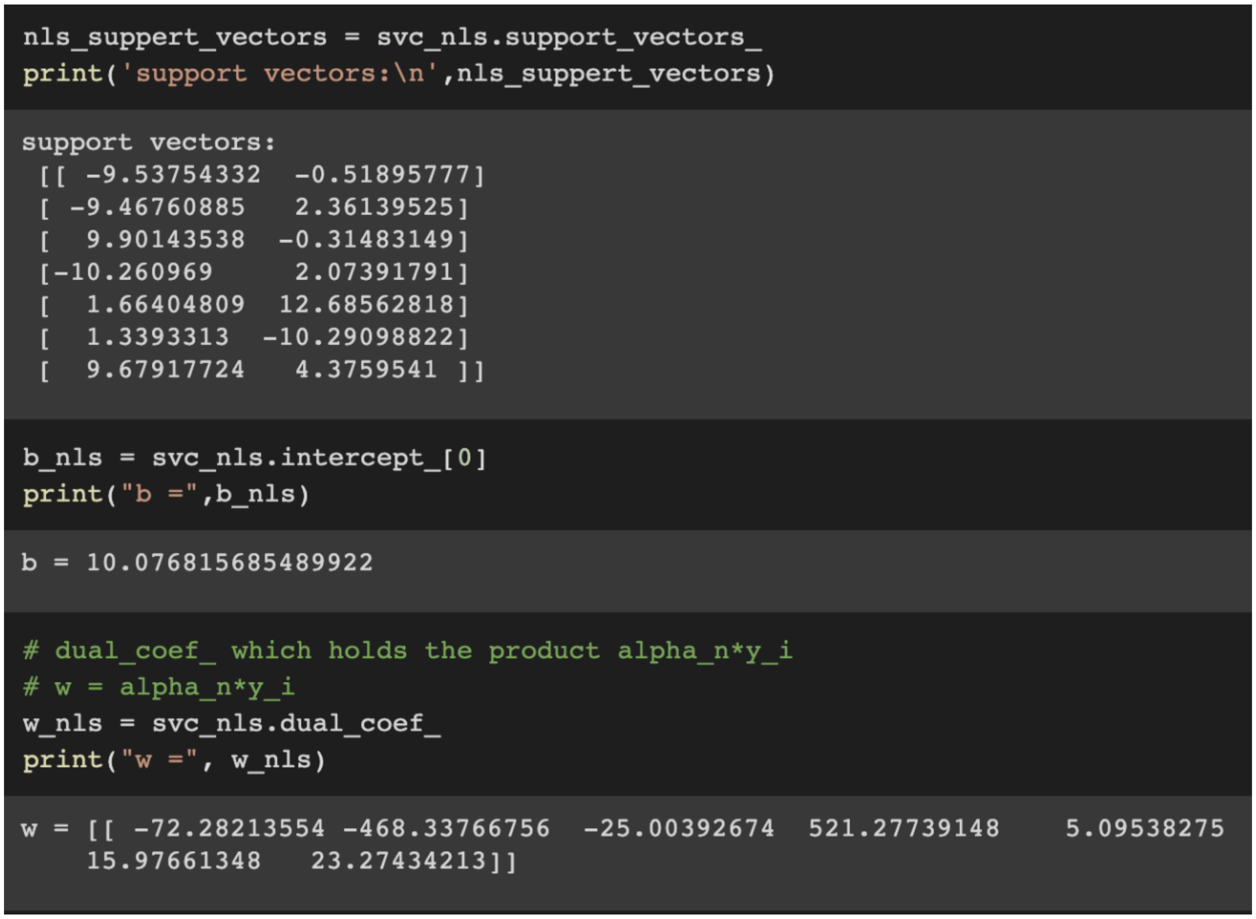
We also set the gamma parameter to "scale" because it provided the nicest results over "auto", the default setting. If gamma is set to "auto", it uses 1 / n_features as the value of gamma. If gamma is set to "scale", it uses 1 / (n_features * X.var()). The gamma parameter sets the width of the bell-shaped curve. The larger the value of gamma, the narrower will be the bell and small values of gamma will yield wide bells. In other words, as the value of gamma increases, the model gets overfits and as the value of gamma decreases, the model underfits. Thus we decided to use "scale" for the gamma parameter because it was a smaller value than "auto", which provided wider curves. For specificity, we calculated the value of gamma when set to "scale":



```
gamma = 1 / (2 * X_nls.var())
gamma = (gamma[0]+gamma[1])/2
gamma
```

```
0.0038566248896485518
```

Using the library functions, we were easily able to determine w, b, and the support vectors for the non-linear separable data. Note that w for non-linear separable data equals to $\alpha_n$*$y_i$. The dimension of w for non-linear kernel is arbitrary can be assumed to go as high as infinity.



```
nls_suppert_vectors = svc_nls.support_vectors_
print('support vectors:\n',nls_suppert_vectors)
```

```
support vectors:
 [[ -9.53754332  -0.51895777]
 [ -9.46760885   2.36139525]
 [  9.90143538  -0.31483149]
 [-10.260969     2.07391791]
 [  1.66404809  12.68562818]
 [  1.3393313  -10.29098822]
 [  9.67917724   4.3759541 ]]
```

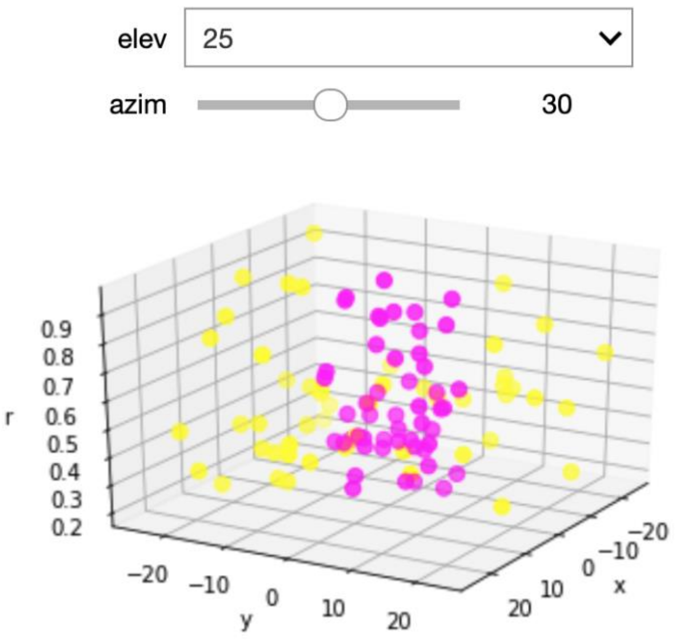
```
b_nls = svc_nls.intercept_[0]
print("b =",b_nls)
```

```
b = 10.076815685489922
```

```
# dual_coef_ which holds the product alpha_n*y_i
# w = alpha_n*y_i
w_nls = svc_nls.dual_coef_
print("w =", w_nls)
```

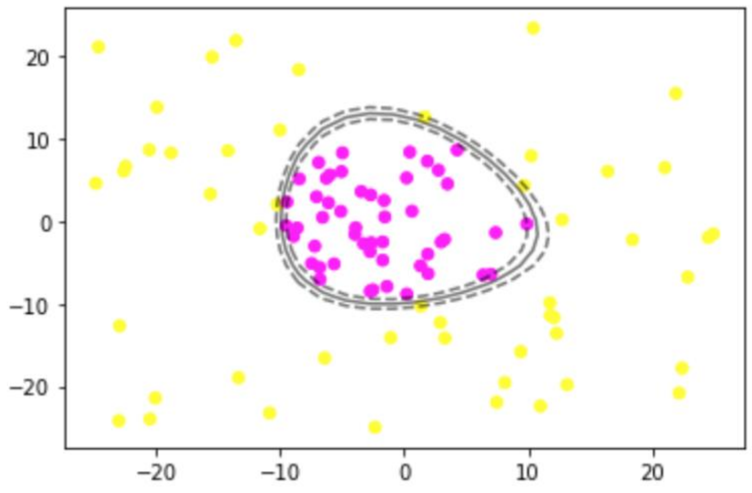
```
w = [[ -72.28213554 -468.33766756  -25.00392674  521.27739148    5.09538275
    15.97661348   23.27434213]]
```

Using ipywidgets and mpl_toolkits, we created a function to plot an interactive 3D visualization of the equation of the curve and margins against the non-linearly separable data points.


elev 25

azim 30

With the matplotlib function from Part 2(a), we can also visualize the curve and margins in 2D:



The Kernel method/trick can be used in the SVM algorithm for non-linearly separable data because it provides an easy and efficient way of mapping to a higher dimensional space. Based on this method, the equation of the curve generally takes this form:

$$f(\vec{x}) = \text{sign}(\sum_i \alpha_i y_i K(\vec{x}_i, \vec{x}) + b)$$

With the w and b we obtained from the library functions, we can determine the equation of the curve as:

```
Hypothesis Equation/Decision Function:
 g(x)= [[ -72.28213554 -468.33766756  -25.00392674  521.27739148
5.09538275 15.97661348   23.27434213]] K(x_n,x) + 10.076815685489922
```

`where K(x_n,x) is the RBF kernel function used.`

$$K(X_1, X_2) = exponent(-\gamma \|X_1 - X_2\|^2)$$

||X1 — X2 || = Euclidean distance between X1 & X2

Due to the difference in the implementation of the algorithm by the library (which uses a lot of parameter tuning and optimization) and does not have a restriction on the number of dimensions it can extend to (ours had 2) and our implementation which is just a modification .

**Part 3: Applications**

1. Face detection – SVMs classify parts of the image as a face and non-face and create a square boundary around the face.
2. Text and hypertext categorization – SVMs allow Text and hypertext categorization for both inductive and transductive models. They use training data to classify documents into different categories. It categorizes on the basis of the score generated and then compares with the threshold value.
3. Classification of images – Use of SVMs provides better search accuracy for image classification. It provides better accuracy in comparison to the traditional query-based searching techniques.
4. Bioinformatics – It includes protein classification and cancer classification. We use SVM for identifying the classification of genes, patients on the basis of genes and other biological problems.
5. Protein fold and remote homology detection – Apply SVM algorithms for protein remote homology detection.
6. Handwriting recognition – We use SVMs to recognize handwritten characters used widely.
7. Generalized predictive control(GPC) – Use SVM based GPC to control chaotic dynamics with useful parameters.

**Contributions:**

- ➢ For Part 1, Lisa and Shagun collectively discussed the concepts of Support Vector Machines and the steps to implement the algorithm for linear separable data and non-linear separable data. We decided to initially code the algorithms individually. From the individual work, we then picked the best code to submit, which was Lisa’s code for Part 1(a) and Shagun’s code for Part 1(b).
- ➢ For Part 2, Lisa researched and implemented the SKLearn library for the SVM algorithm for both the linear and nonlinear separable datasets.
- ➢ Part 3 was researched and completed by Shagun.

- The report was compiled by Lisa and Shagun.