GNR652 Assisgnment 2

Rishabh Ramteke - 170070046

February 17, 2019

1 Introduction

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

2 Preliminary

A classification problem has two types of variables, $x_i \epsilon R^d$, the vector of observations(features) in the world, and $y_i \epsilon R^d$ state(class) vector of the world. The set of all samples in this world (or dataset) constructs the observation set X ϵ R^{nxd} and state set Y ϵ 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, α)] where f(x, α) is the function of mapping parametrized by α.

3 Theory

3.1 Linear SVMs

- Training data $\{\mathbf{x}_i, y_i\}$ $i = 1, \ldots, l$, $\mathbf{x}_i \in R^n$, and $y_i \in \{-1, 1\}$
- On a separating hyperplane: $\mathbf{w}^T\mathbf{x} + b = 0$, where

  - w normal to the hyperplane
  - $\frac{|b|}{\|\mathbf{w}\|}$ is the distance to origin
  - $\|\mathbf{w}\|$ Euclidean norm of $\mathbf{w}$

- d_+, d_- shortest distances from labeled points to hyperplane
- Define margin $m = d_+ + d_-$
- Task: find the separating hyperplane that maximizes m
- Key point: Maximizing the margin minimizes the VC dimension

3.2 Computation

- For the separating plane:

$$\begin{align} \mathbf{w}^T\mathbf{x}_i + b \geq +1, \quad y_i = +1 \tag{1} \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1, \quad y_i = -1 \tag{2} \\ \equiv \tag{3} \\ y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \geq 0, \qquad \forall i \tag{4} \end{align}$$

- For the closest points the equalities are satisfied, so:

$$d_+ + d_- = \frac{|1-b|}{\|w\|} + \frac{|-1-b|}{\|w\|} = \frac{2}{\|w\|} \tag{5}$$

3.3 Switching to Lagrangian

- One coefficient per train sample
- The constraints easier to handle
- Training data appears only in dot products
- Great for applying the kernel trick later on

3.4 Lagrangian Form

- Minimize

$$L_P = \frac{\|w\|^2}{2} - \sum_{i=1}^{l} \alpha_i y_i (\mathbf{x}_i \mathbf{x}_i + b) + \sum_{i=1}^{l} \alpha_i \tag{6}$$

- Convex quadratic programming problem with the dual: maximize

$$L_D = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j (\mathbf{w}^T \mathbf{x}_j) \tag{7}$$

3.5 The Support Vectors

- The points with $\alpha_i > 0$ are the support vectors
- Solution only depends on them
- All others have $\alpha_i = 0$ and can be moved arbitrarily far from the decision hyperplane, or removed

3.6 Testing

- Once the hyperplane is found:

$$\hat{y} = (\mathbf{w}^T \mathbf{x} + b) \tag{8}$$

4 Implementing the SVM algorithm (Hard margin)

4.1 Aim and Methods

- The dataset is for credit card fraud. It is a binary classification task (labels 0 and 1).
- Since the labels are 0 and 1 , we need to change them to -1 and 1 respectively as these are preferred in SVM.
- Now, since the data is highly biased (500 '1' labels among 2 lakh '-1' labels), thus I will consider 100 '+1' and 100 '-1'.

- In the dual formulation based SVM, we convert the primal Lagrangian to the dual formulation which is a convex quadratic optimization problem with linear constraints. We apply a convex optimization function ***"cvxopt.solvers.qp"*** (available in python) to solve the α values (the lagrangian multipliers).
- Note that if there are N data points in the training set, α will be a vector of 1 x N (corresponding to the points). Many of these N α values will be 0 and the ones corresponding to support vector will be non-zero.
- We need to convert the dual formula in the matrix notation.
- "cvxopt.solvers.qp" requires a particular form for the optimisation problem. So need to convert the matrix based dual formula into the one "cvxopt" accepts. It will give output $\alpha$from which we can calculate w and b parameter.
- Check the testing accuracy based on the % of correctly classified samples in the test set.

4.2 Linearly separable, binary classification

General steps to solve the SVM problem are the following:

- Create P where $H_{i,j} = y^{(i)}y^{(j)} < x^{(i)}x^{(j)} >$
- Calculate $\mathbf{w} = \sum_i^m y^{(i)}\alpha_i x^{(i)}$
- Determine the set of support vectors S by finding the indices such that $\alpha_i > 0$
- Calculate the intercept term using b $= y^{(s)} - \sum_{m \in S} \alpha_m y^{(m)} < x^{(m)}x^{(s)} >$
- For each new point x classify according to y' $= sign(w^T x' + b)$

4.3 Re-writing the problem in an appropriate format

Since we will solve this optimization problem using the CVXOPT library in python we will need to match the solver's API which, according to the documentation is of the form:

- $$\begin{aligned} &\min \frac{1}{2}x^T P x + q^T x \\ s.t.\ \ &Gx \leq h \\ &Ax = b \end{aligned}$$

- With API
cvxopt.solvers.qp(P, q[, G, h[, A, b[, solver[, initvals]]]])

- the dual problem is expressed as:
$$\max_\alpha \sum_i^m \alpha_i - \frac{1}{2}\sum_{i,j}^m y^{(i)}y^{(j)}\alpha_i\alpha_j < x^{(i)}x^{(j)} >$$

- Let H be a matrix such that $H_{i,j} = y^{(i)}y^{(j)} < x^{(i)}x^{(j)} >$ then the
$$\max_\alpha \sum_i^m \alpha_i - \frac{1}{2}\alpha^T \mathbf{H}\alpha$$
optimization becomes: $s.t.\ \alpha_i \geq 0$
$$\sum_i^m \alpha_i y^{(i)} = 0$$

- We convert the sums into vector form and multiply both the objective and the constraint by 1 which turns this into a minimization problem and reverses the inequality
$$\begin{aligned} &\min_\alpha \frac{1}{2}\alpha^T \mathbf{H}\alpha - 1^T\alpha \\ &s.t.\ -\alpha_i \leq 0 \\ &s.t.\ y^T\alpha = 0 \end{aligned}$$

- We are now ready to convert our numpy arrays into the cvxopt format, using the same notation as in the documentation this gives

- P:=H a matrix of size mm q:=1 a vector of size m1 G:=diag[1] a diagonal matrix of -1s of size mm h:=0 a vector of zeros of size m1 A:=y the label vector of size m1 b:=0 a scalar

4.4 Computing the matrix H in vectorized form

Consider the simple example with 2 input samples
$\{x^{(1)}, x^{(2)}\} \in R^2$ which are two dimensional vectors. i.e. $x^{(1)} = (x_1^{(1)}, x_2^{(1)})^T$
$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \end{bmatrix}$

We now proceed to creating a new matrix X' where each input sample x is multiplied by the corresponding output label y. This can be done easily in Numpy using vectorization and padding. $X' = \begin{bmatrix} x_1^{(1)}y^{(1)} & x_2^{(1)}y^{(1)} \\ x_1^{(2)}y^{(2)} & x_2^{(2)}y^{(2)} \end{bmatrix}$

Finally we take the matrix multiplication of X and its transpose giving $H = X'X'^T$
$H = X'X'^T = \begin{bmatrix} x_1^{(1)}y^{(1)} & x_2^{(1)}y^{(1)} \\ x_1^{(2)}y^{(2)} & x_2^{(2)}y^{(2)} \end{bmatrix} \begin{bmatrix} x_1^{(1)}y^{(1)} & x_1^{(2)}y^{(2)} \\ x_2^{(1)}y^{(1)} & x_2^{(2)}y^{(2)} \end{bmatrix}$

5 Implementation in Python

5.1 Importing data and dividing into training and test set

```
#Importing Libraries
import numpy as np
import csv
import matplotlib.pyplot as plt
from numpy.linalg import inv
from cvxopt import matrix as cvxopt_matrix #Importing with custom names to avoid i
from cvxopt import solvers as cvxopt_solvers

#Importing data
filepath ="/home/rishabh/Downloads/set1.csv"
File = np.genfromtxt(filepath, delimiter=',') #converting csv to numpy
# X_1 = File[:98, 0:30]
# X_2 = File[98:, 0:30]
np.random.shuffle(File) #shuffle data in File

#defining training and testing set
X_train = File[:180, 0:30]
```

```
Y_train = File[:180, [30]]
X_test = File[180:, 0:30]
Y_test = File[180:, [30]]

#Data shapes
print("\n####  Data Shapes  ####\n")
print("MyData_shape =",File.shape)
print("X_train_shape =",X_train.shape,"Y_train_shape =",Y_train.shape)
print("X_test_shape =",X_test.shape,"X_test_shape =",Y_test.shape)
```

5.2 CVXOPT solver and resulting α

```
# Initializing values and computing H.
m, n = X_train.shape
y = Y_train.reshape(-1, 1) * 1.
X_dash = y * X_train
H = np.dot(X_dash, X_dash.T) * 1.

# Converting into cvxopt format
P = cvxopt_matrix(H)
q = cvxopt_matrix(-np.ones((m, 1)))
G = cvxopt_matrix(-np.eye(m))
h = cvxopt_matrix(np.zeros(m))
A = cvxopt_matrix(y.reshape(1, -1))
b = cvxopt_matrix(np.zeros(1))

# Setting solver parameters
cvxopt_solvers.options['show_progress'] = False
cvxopt_solvers.options['abstol'] = 1e-10
cvxopt_solvers.options['reltol'] = 1e-10
cvxopt_solvers.options['feastol'] = 1e-10

# Run solver
sol = cvxopt_solvers.qp(P, q, G, h, A, b)
alphas = np.array(sol['x']) #alphas are langrange multipliers
```

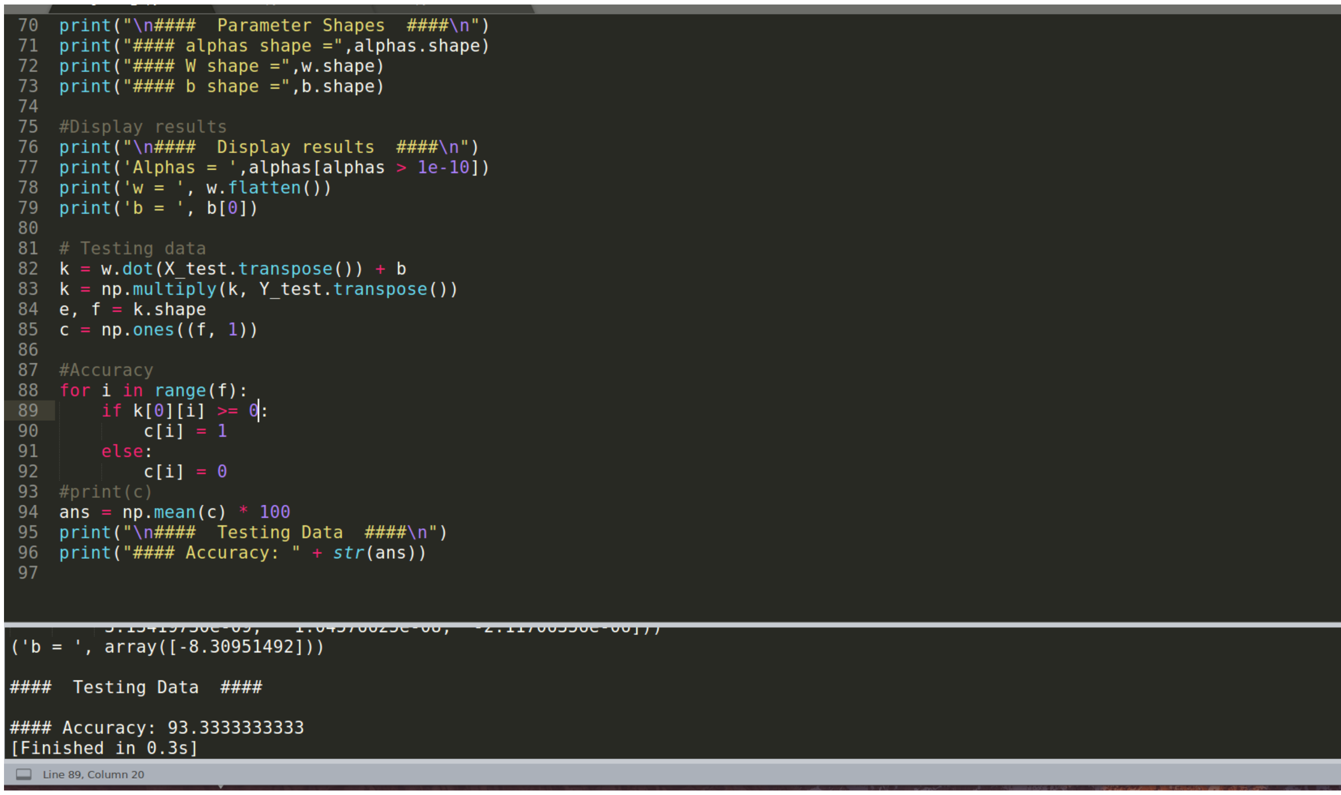
5.3 Compute w and b parameters

```
#Computing W parameter
w = np.multiply(alphas, Y_train)
w = w.transpose().dot(X_train)

#computing b parameter

#Method 1
# l = w.dot(X_1.transpose())
# z = w.dot(X_2.transpose())
# c1 = l.min(1)
# c2 = z.max(1)
# b = (-0.5) * (c1 + c2)

#Method 2
S = (alphas > 1e-40).flatten()
b = y[S] - np.dot(X_train[S], w.T)

#parameter shapes
print("\n####  Parameter Shapes  ####\n")
print("#### alphas shape =",alphas.shape)
print("#### W shape =",w.shape)
print("#### b shape =",b.shape)

#Display results
print("\n####  Display results  ####\n")
print('Alphas = ',alphas[alphas > 1e-10])
print('w = ', w.flatten())
print('b = ', b[0])
```

5.4 Testing and finding accuracy

```
    #Accuracy
for i in range(f):
    if k[0][i] >= 0:
        c[i] = 1
    else:
        c[i] = 0
#print(c)
ans = np.mean(c) * 100
print("\n####  Testing Data  ####\n")
print("#### Accuracy: " + str(ans))
```

5.5 Result



```
print("\n####  Parameter Shapes  ####\n")
print("#### alphas shape =",alphas.shape)
print("#### W shape =",w.shape)
print("#### b shape =",b.shape)

#Display results
print("\n####  Display results  ####\n")
print('Alphas = ',alphas[alphas > 1e-10])
print('w = ', w.flatten())
print('b = ', b[0])

# Testing data
k = w.dot(X_test.transpose()) + b
k = np.multiply(k, Y_test.transpose())
e, f = k.shape
c = np.ones((f, 1))

#Accuracy
for i in range(f):
    if k[0][i] >= 0:
        c[i] = 1
    else:
        c[i] = 0
#print(c)
ans = np.mean(c) * 100
print("\n####  Testing Data  ####\n")
print("#### Accuracy: " + str(ans))
```

```
('b = ', array([-8.30951492]))

####  Testing Data  ####

#### Accuracy: 93.3333333333
[Finished in 0.3s]
```

Figure 1: Accuracy for dataset 1

- Run the code ”gnr2.py” on terminal
- I got the following output :

```
####  Data Shapes  ####
```

```
('MyData_shape =', (195, 31))
('X_train_shape =', (180, 30), 'Y_train_shape =', (180, 1))
('X_test_shape =', (15, 30), 'X_test_shape =', (15, 1))

####  Parameter Shapes  ####

('#### alphas shape =', (180, 1))
('#### W shape =', (1, 30))
('#### b shape =', (180, 1))

####  Display results  ####

('Alphas = ', array([  2.01143781e-08,   3.37634087e-10,   1.22603959e-08,
         1.18265155e-10,   3.59732974e-08,   2.10351144e-10,
         6.98866585e-10,   1.26638211e-10]))
('w = ', array([  2.68387380e-04,   7.71822025e-08,   9.82697081e-08,
        -2.21030168e-07,   1.03360261e-07,   1.20294132e-07,
        -4.78593969e-08,   4.98864112e-08,  -1.56416389e-08,
        -3.83184446e-08,  -1.01015505e-07,   9.27079146e-08,
        -1.71764013e-07,   4.69967942e-08,  -2.05841178e-07,
        -2.01301313e-08,   1.20356548e-07,   2.31395762e-07,
         1.26432946e-07,  -1.17474635e-07,  -5.45262617e-09,
        -1.34794104e-08,  -3.61113112e-08,  -2.68412413e-08,
        -6.93643875e-08,   6.11404393e-08,   2.61866582e-08,
         3.13419730e-09,   1.04576625e-08,  -2.11706356e-06]))
('b = ', array([-8.30951492]))

####  Testing Data  ####

#### Accuracy: 93.3333333333
[Finished in 0.3s]
```

- **Got an accuracy of 93.3 % for dataset 1**

Similarly I got the outputs for the following datasets:

dataset	Accuracy
set1	93.33
set2	93.33
set3	86.67
set4	73.33
set 5	73.33

Note: Everytime i run the code, the dataset file gets shuffled due the code np.random.shuffle. So i get different accuracies each time