IE406 Machine Learning

Lab Assignment - 5

Group 39

201901201: Sharvil Sheth

201901216: Prabhav Shah

201901301: Mohil Desai

201901462: Dhaval Vaidya

201901464: Vishvesh Patel

0.0.1 1.Perform SVM on iris dataset.

[12]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[13]:
```
from sklearn import datasets
iris = datasets.load_iris()
```

(a) Use sklearn SVM classifier and perform classification on dataset.

[14]:
```
X = iris.data
Y = iris.target
```

[15]:
```
from sklearn.model_selection import train_test_split
X_train , X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.
↪3,random_state=5)
```

[16]:
```
from sklearn.svm import SVC
model = SVC()
model.fit(X_train,Y_train)
predicted = model.predict(X_test)
```

[17]:
```
from sklearn import metrics
metrics.accuracy_score(Y_test,predicted)
```

[17]:
```
0.9777777777777777
```

(b) normalize the data and then perform same experiment on normalized data

[18]:
```
from sklearn.preprocessing import MinMaxScaler
min_max = MinMaxScaler()
X_normalised = min_max.fit_transform(X)
```

[19]:
```
from sklearn.model_selection import train_test_split
X_train , X_test, Y_train, Y_test = train_test_split(X_normalised,Y,test_size␣
↪= 0.3,random_state=5)
```

[20]:
```
model = SVC()
model.fit(X_train,Y_train)
predicted = model.predict(X_test)
```

[21]:
```
from sklearn import metrics
metrics.accuracy_score(Y_test,predicted)
```

[21]: 0.9555555555555556

(c) use the given SVM kernels and perform svm classification.

[22]:
```
from sklearn import svm
```

I. Linear

[23]:
```
linear_svc = svm.SVC(kernel='linear').fit(X_train,Y_train)
pred_lin = linear_svc.predict(X_test)
```

[24]:
```
metrics.accuracy_score(Y_test,pred_lin)
```

[24]: 0.9333333333333333

II. poly

[25]:
```
poly_svc = svm.SVC(kernel='poly').fit(X_train,Y_train)
pred_poly = poly_svc.predict(X_test)
```

[26]:
```
metrics.accuracy_score(Y_test,pred_poly)
```

[26]: 0.9555555555555556

III. bf

[27]:
```
rbf_svc = svm.SVC(kernel='rbf').fit(X_train,Y_train)
pred_rbf = rbf_svc.predict(X_test)
```

[28]:
```
metrics.accuracy_score(Y_test,pred_rbf)
```

[28]: 0.9555555555555556

IV. Sigmoid

[29]:
```
sigmoid_svc = svm.SVC(kernel='sigmoid').fit(X_train,Y_train)
pred_sig = sigmoid_svc.predict(X_test)
```

[30]:
```
metrics.accuracy_score(Y_test,pred_sig)
```

[30]: 0.3333333333333333

V. Precomputed

[34]:
```
X_sq_train = np.dot(X_train, X_train.T)
X_sq_test = np.dot(X_test, X_train.T)
precomputed_svc = svm.SVC(kernel='precomputed').fit(X_sq_train,Y_train)
pred_comp = precomputed_svc.predict(X_sq_test)
```

[35]:
```
metrics.accuracy_score(Y_test,pred_comp)
```

[35]: 0.9333333333333333

0.0.2 2.Perform SVM on https://drive.google.com/file/d/13nw-uRXPY8XIZQxKRNZ3yYlho-CYm_Qt

[36]:
```
import pandas as pd
data = pd.read_csv('bill_authentication.csv')
```

[37]:
```
X = data.iloc[:,:-1]
Y = data.iloc[:,-1]
```

(a) Use sklearn SVM classifier and perform classification on dataset.

[38]:
```
from sklearn.model_selection import train_test_split
X_train , X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.
↪3,random_state=5)
```

[39]:
```
from sklearn.svm import SVC
model = SVC()
model.fit(X_train,Y_train)
predicted = model.predict(X_test)
```

[40]:
```
from sklearn import metrics
metrics.accuracy_score(Y_test,predicted)
```

[40]: 1.0

(b) normalize the data and then perform same experiment on normalized data

[41]:
```
from sklearn.preprocessing import MinMaxScaler
min_max = MinMaxScaler()
X_normalised = min_max.fit_transform(X)
```

[42]:
```
from sklearn.model_selection import train_test_split
X_train , X_test, Y_train, Y_test = train_test_split(X_normalised,Y,test_size␣
↪= 0.3,random_state=5)
```

[43]:
```
model = SVC()
model.fit(X_train,Y_train)
predicted = model.predict(X_test)
```

[44]:
```
from sklearn import metrics
metrics.accuracy_score(Y_test,predicted)
```

[44]: 1.0

(c) use the given SVM kernels and perform svm classification.

[45]:
```
from sklearn import svm
```

VI. Linear

[46]:
```
linear_svc = svm.SVC(kernel='linear').fit(X_train,Y_train)
pred_lin = linear_svc.predict(X_test)
```

[47]:
```
metrics.accuracy_score(Y_test,pred_lin)
```

[47]:
```
0.9830097087378641
```

VII. poly

[48]:
```
poly_svc = svm.SVC(kernel='poly').fit(X_train,Y_train)
pred_poly = poly_svc.predict(X_test)
```

[49]:
```
metrics.accuracy_score(Y_test,pred_poly)
```

[49]:
```
1.0
```

VIII. bf

[50]:
```
rbf_svc = svm.SVC(kernel='rbf').fit(X_train,Y_train)
pred_rbf = rbf_svc.predict(X_test)
```

[51]:
```
metrics.accuracy_score(Y_test,pred_rbf)
```

[51]:
```
1.0
```

IX. Sigmoid

[52]:
```
sigmoid_svc = svm.SVC(kernel='sigmoid').fit(X_train,Y_train)
pred_sig = sigmoid_svc.predict(X_test)
```

[53]:
```
metrics.accuracy_score(Y_test,pred_sig)
```

[53]:
```
0.5388349514563107
```

X. precomputed

[55]:
```
X_sq_train = np.dot(X_train, X_train.T)
X_sq_test = np.dot(X_test, X_train.T)
precomputed_svc = svm.SVC(kernel='precomputed').fit(X_sq_train,Y_train)
pred_comp = precomputed_svc.predict(X_sq_test)
```

[56]:
```
metrics.accuracy_score(Y_test,pred_comp)
```

[56]:
```
0.9830097087378641
```

0.0.3 3. A Implement a function for hard margin SVM in primal form using cvxpy.

[3]:
```
import cvxpy as cp
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[14]:
```
from sklearn.svm import SVC
```

[ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

[ ]:
```
plt.style.use('ggplot')
```

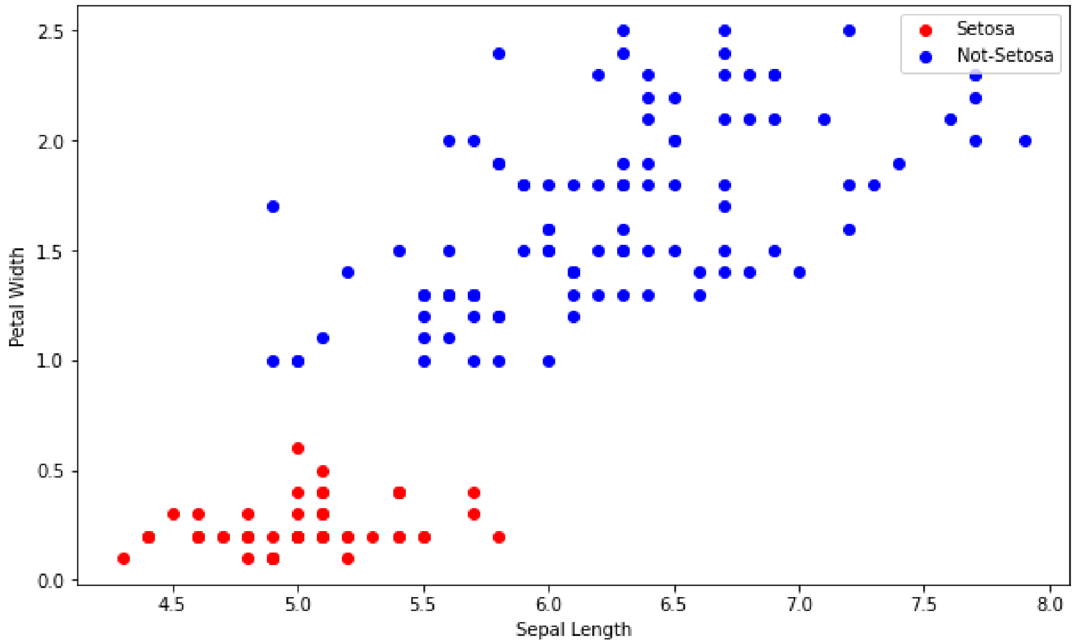
[5]:
```
#Reading Iris data-set
df = pd.read_csv('Iris.csv')
df.head()
```

[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

B. implementation on the IRIS dataset.

[7]:
```
#Data preperation
#Dropping SepalWidthCm and PetalLengthCm column
iris = df.drop(['SepalWidthCm', 'PetalLengthCm'], axis=1)

#Making Setosa = 1 and Non-Setosa = -1
iris['Species'] = iris['Species'].apply(lambda x: 1 if x == 'Iris-setosa'␣
↪else -1)

#Getting X and Y
X = np.array(iris.drop(['Id', 'Species'], axis=1))
Y = np.array(iris['Species'])

#Data visualisation
plt.figure(figsize=(10,6))
c1 = plt.scatter(X[Y==1, 0], X[Y==1, 1], c = 'r')
c2 = plt.scatter(X[Y==-1, 0], X[Y==-1, 1], c = 'b')
plt.legend((c1,c2),('Setosa','Not-Setosa'))
plt.xlabel('Sepal Length')
plt.ylabel('Petal Width')
plt.show()
```

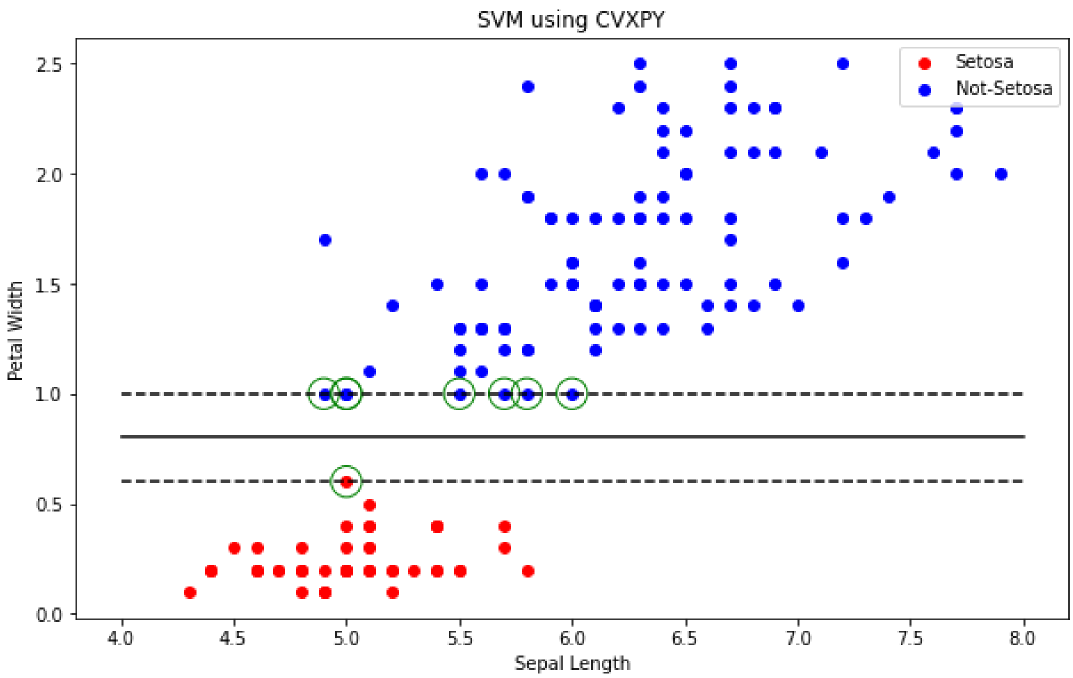


C. Hard Margin Classifier using CVXPY

[9]:
```
#Defining thetas
theta = cp.Variable(2)
theta0 = cp.Variable(1)

#Defining objective function
obj_fun = cp.Minimize(0.5*cp.square(cp.norm(theta)))

#Defining constraint
constraints = []
for i in range(150):
  constraints.append(1 - Y[i]*(theta@X[i] + theta0)<=0)

#Defining hard margin problem
prob = cp.Problem(obj_fun, constraints)
prob.solve()

#Optimal values of theta and theta0
Theta0 = theta0.value
Theta1 = theta.value[0]
Theta2 = theta.value[1]

print('Optimal value of \u03F4 = ', theta.value)
print('Optimal value of \u03F40 = ', theta0.value)

#Data visualisation
plt.figure(figsize=(10,6))
c1 = plt.scatter(X[Y==1, 0], X[Y==1, 1], c = 'r')
c2 = plt.scatter(X[Y==-1, 0], X[Y==-1, 1], c = 'b')
```

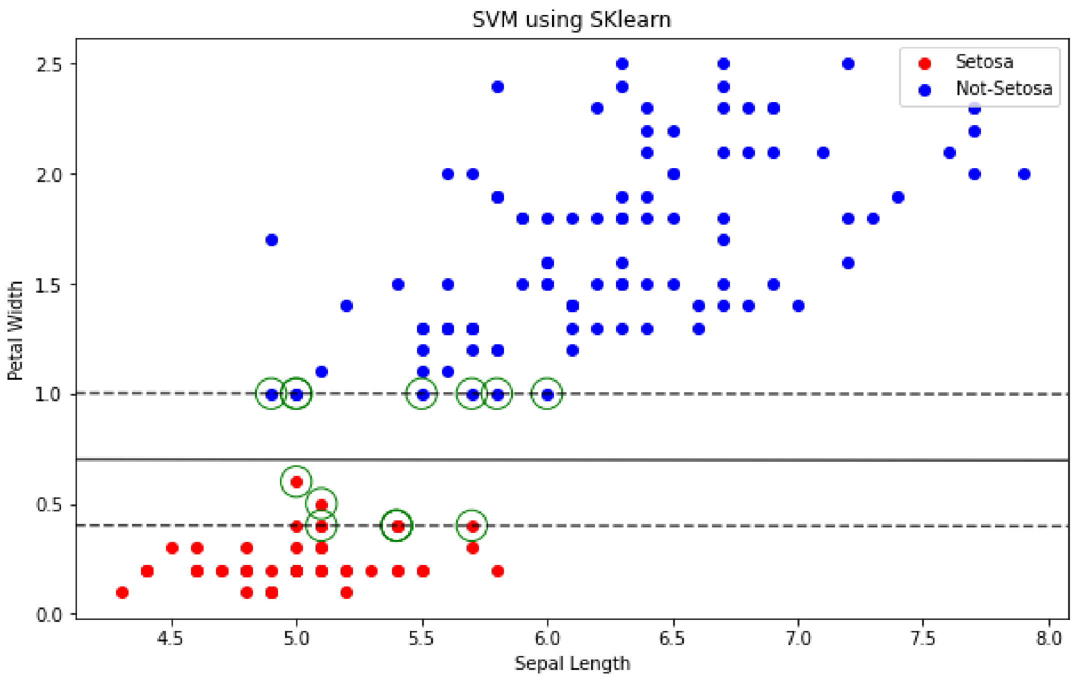
```
plt.legend((c1,c2), ('Setosa','Not-Setosa'))
plt.title('SVM using CVXPY')
plt.xlabel('Sepal Length')
plt.ylabel('Petal Width')

x_axis = np.linspace(4, 8, 1000)

hyperplane = -Theta0 - (Theta1*x_axis)

#Plotting hyperplane
plt.plot(x_axis, hyperplane/Theta2,'k')

#Plotting margins
plt.plot(x_axis, (1 + hyperplane)/Theta2,'k--')
plt.plot(x_axis, (-1 + hyperplane)/Theta2,'k--')

#Plotting support vectors
for i in range(150):
  temp = Y[i]*((X[i]@theta.value) + theta0.value)
  if(np.round(temp, 2)==1):
    plt.scatter(X[i][0],X[i][1], facecolors='none', s = 300, linewidth=1, marker = 'o',edgecolor = 'g')

plt.show()
```

```
Optimal value of  =  [-2.46457378e-10 -5.00000000e+00]
Optimal value of 0 =  [4.]
```



D. Plotting decision boundary using SKlearn's SVM

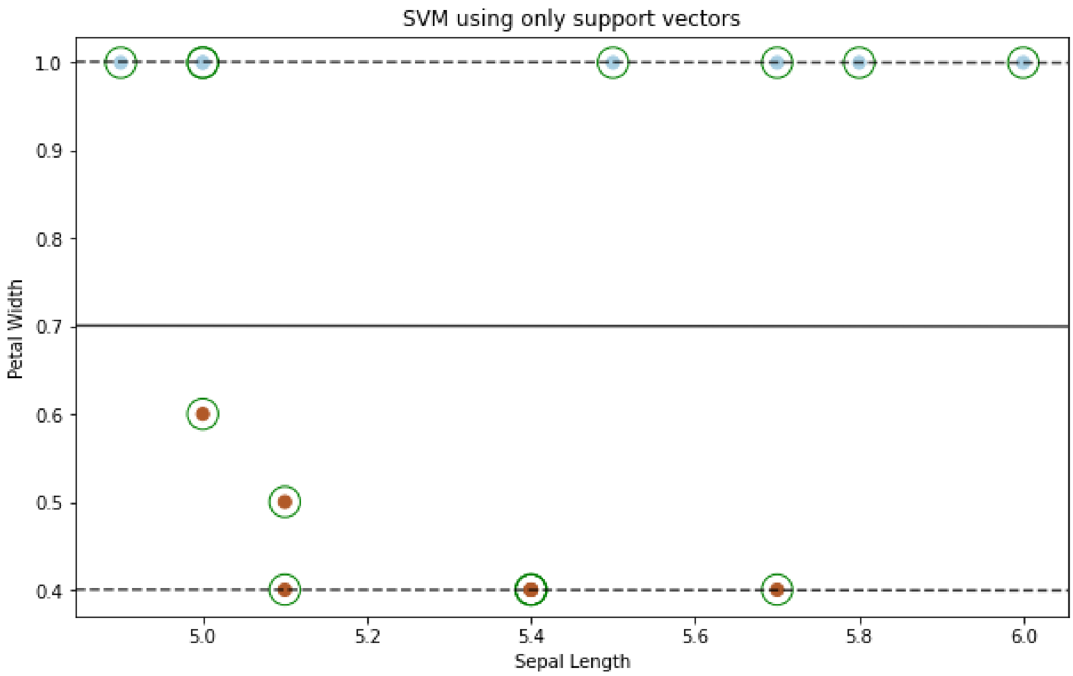
```
[11]: #Making Setosa = 1 and Non-Setosa = 0
Y = np.where(Y==1, 1, 0)

#Training SVM
SVM = SVC(kernel='linear')
SVM.fit(X, Y)

#Data visualisation
plt.figure(figsize=(10,6))
c1 = plt.scatter(X[Y==1, 0], X[Y==1, 1], c = 'r')
c2 = plt.scatter(X[Y==0, 0], X[Y==0, 1], c = 'b')
plt.legend((c1,c2), ('Setosa','Not-Setosa'))
plt.title('SVM using SKlearn')
plt.xlabel('Sepal Length')
plt.ylabel('Petal Width')

ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

x = np.linspace(xlim[0], xlim[1], 30)
y = np.linspace(ylim[0], ylim[1], 30)
y1, x1 = np.meshgrid(y, x)
xy = np.vstack([x1.ravel(), y1.ravel()]).T

P = SVM.decision_function(xy).reshape(x1.shape)

#Plotting decision boundary and margins
ax.contour(x1, y1, P, colors='k',levels=[-1, 0, 1], alpha=0.75,␣
 ↪linestyles=['--', '-', '--'])

#Plotting support vectors
ax.scatter(SVM.support_vectors_[:, 0], SVM.support_vectors_[:, 1], s=300,␣
 ↪linewidth=1, facecolors='none', edgecolors='g')

ax.set_xlim(xlim)
ax.set_ylim(ylim)

plt.show()
```



E. throwing away all the points except the support vectors

[13]:
```
#Getting support vectors
X_sv = SVM.support_vectors_

Y_sv = []
for i in SVM.support_:
  Y_sv.append(Y[i])

SVM_sv = SVC(kernel='linear')
SVM_sv.fit(X_sv, Y_sv)

#Plotting support vectors
plt.figure(figsize=(10,6))
plt.scatter(X_sv[:, 0], X_sv[:, 1], c=Y_sv, s=50, cmap=plt.cm.Paired)
plt.title('SVM using only support vectors')
plt.xlabel('Sepal Length')
plt.ylabel('Petal Width')

ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T

P_sv = SVM_sv.decision_function(xy).reshape(XX.shape)
```

```
#Plotting decision boundary and margins
ax.contour(XX, YY, P_sv, colors='k', levels=[-1, 0, 1], alpha=0.75,␣
↪linestyles=['--', '-', '--'])

ax.scatter(SVM_sv.support_vectors_[:, 0], SVM_sv.support_vectors_[:, 1],␣
↪s=300, linewidth=1, facecolors='none', edgecolors='g')

plt.show()
```



Conclusion : Removing all points other than support vector, decision boundary does not change since the support vectors are the data points closest to the decision surface. Hence, if the support vectors are same then the desicion boundary wont change.

[ ]:

```
```