Politecnico di Torino

01SQIOV

Artificial Intelligence and Machine Learning

Homework 2:
SVMs

Prof. Barbara Caputo
Prof. Paolo Russo

Nour Saeed
Politecnico di Torino
Department of Computer Engineering, Cinema, & Mechatronics
Mechatronics Engineering
S250409@studenti.polito.it

January 5, 2019

Abstract

This report is a required deliverable for the course of AIML. It deals with Support Vector Machines. The code used to implement this is available with the report. Each section of the code is commented with the corresponding homework step.

Please note that each split is done at random. Result may vary from test to test.

Setup

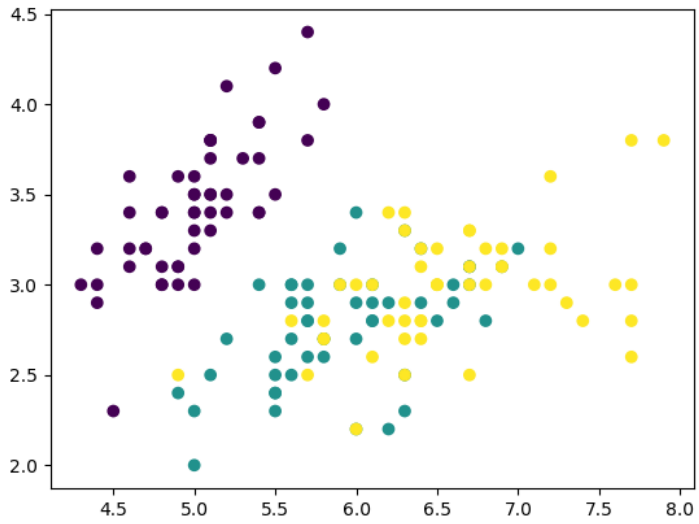
For this homework, I used PyCharm as my IDE and created a virtual environment in which I added all required libraries.
In this homework, we will use the iris dataset provided in the Scikit Learn library.
Let's first import the required libraries and load the data set.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, svm
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import accuracy_score,confusion_matrix

# Load Iris dataset
iris=datasets.load_iris()
```

We select the first two dimensions and split data into train, validation and test sets in proportion 5:2:3, as follows:

```
# Select the first two dimensions
x=iris.data[:,:2]
y=iris.target
print('Dataset shape: '+ str(x.shape))

plt.scatter(x[:,0],x[:,1],c=y)
plt.savefig('./Output/1.png')
plt.show()

# Randomly split data into train, validation and test sets

X_tv, X_test, y_tv, y_test = train_test_split(
    x, y, test_size=0.3
)
X_train, X_val, y_train, y_val = train_test_split(
    X_tv, y_tv, test_size=2/7
)
```

We keep the merged train and validation set *X_tv and y_tv* as we will use them later on. The code above also display our data. Note that each color signifies a class.



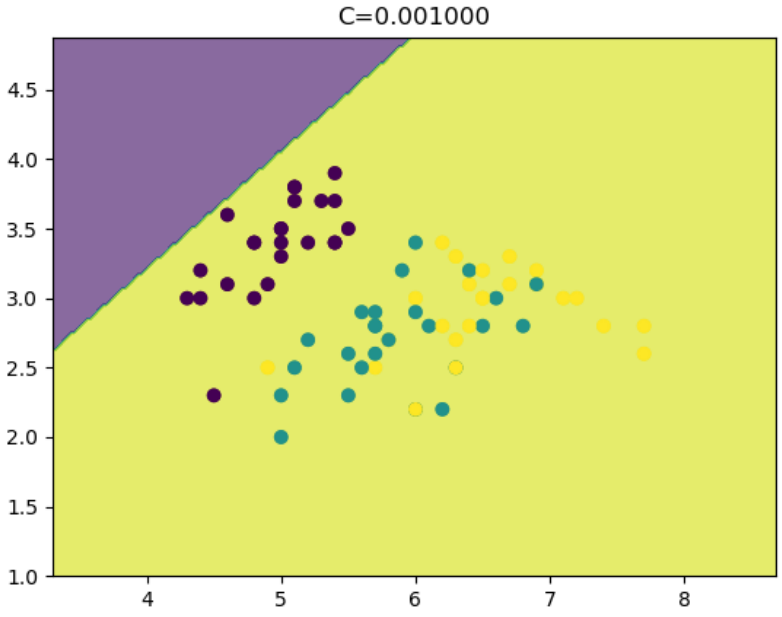
Linear Support Vector Machines

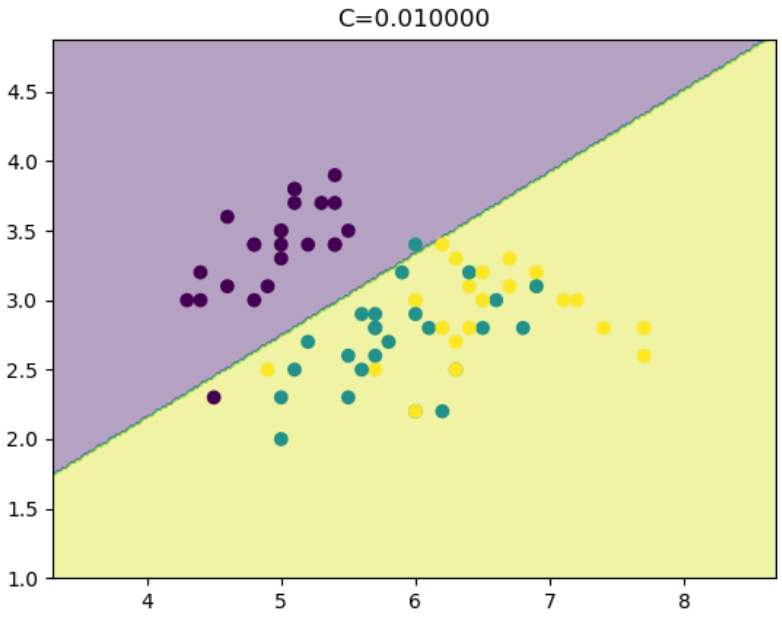
We now have to loop over the value of C of the SVM, train it and do some plots. The code below does that.

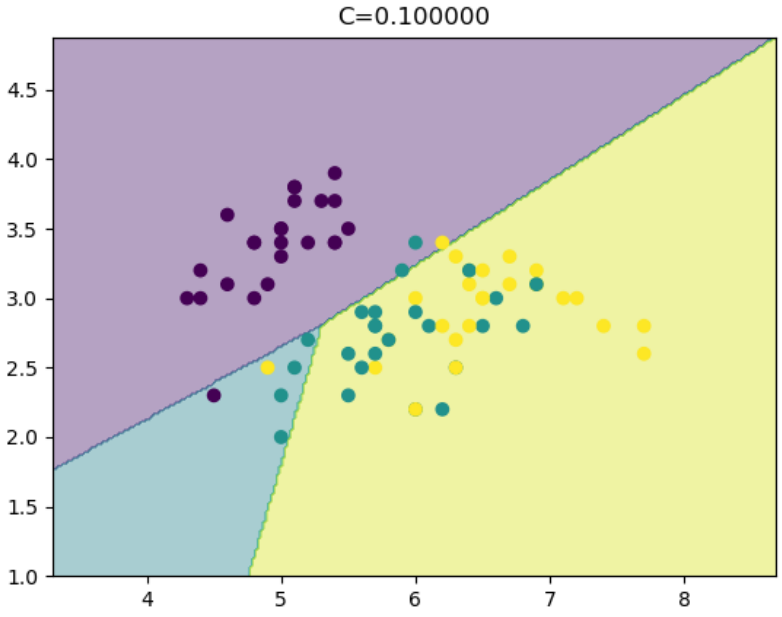
```
# Training, Plotting, and Evaluation using linear SVM


x_min, x_max = X_train[:,0].min() - 1, X_train[:,0].max() + 1
y_min, y_max = X_train[:,1].min() - 1, X_train[:,1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

Cs= np.float_power(10,np.arange(-3,4,1))
acc=[]
for c in Cs:
    linear=svm.LinearSVC(C=c,max_iter=1000000000,multi_class='
    ovr')
    linear.fit(X_train,y_train)
    Z = linear.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.4)
    plt.scatter(X_train[:,0],X_train[:,1],c=y_train)
    plt.title("C=%f" %c)
```

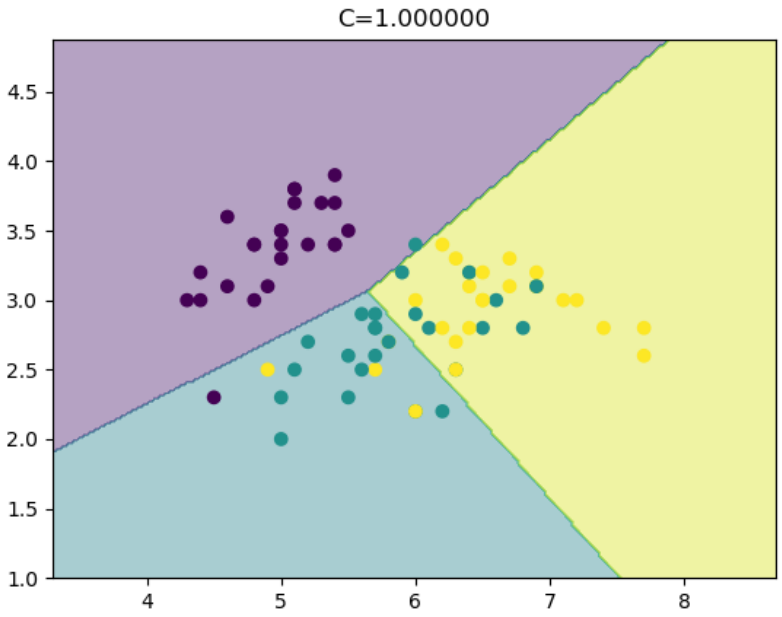
```
    plt.savefig('./Output/'+str(c)+'.png')
    plt.show()

    y_valP=linear.predict(X_val)
    acc.append(accuracy_score(y_val,y_valP))
```

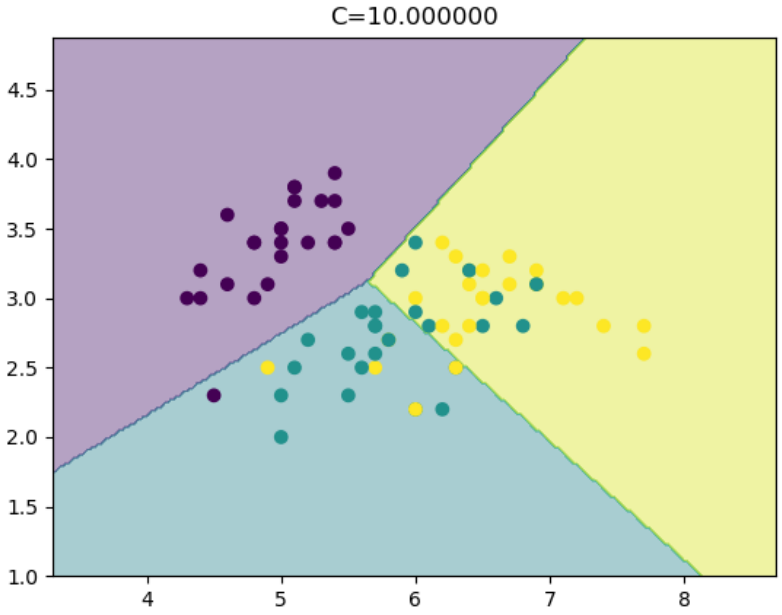
Plotting the decision boundaries for each value of C, we get:

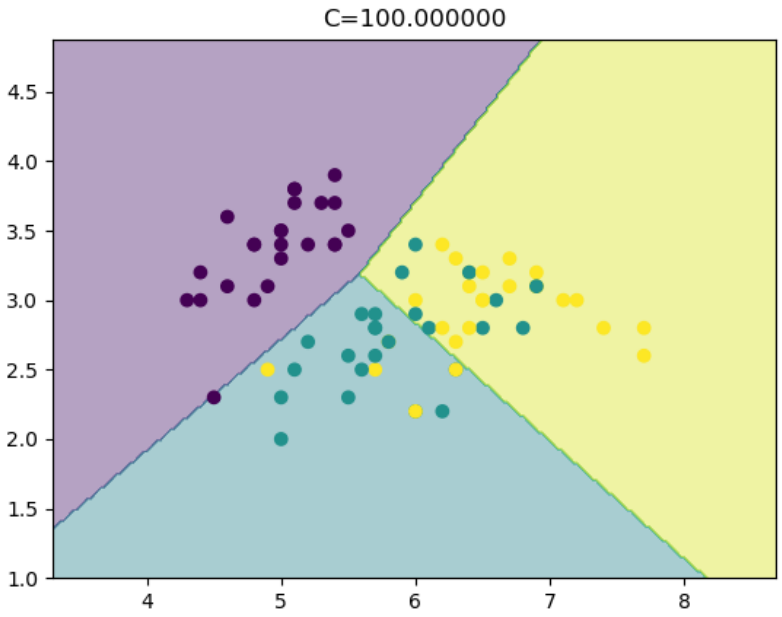


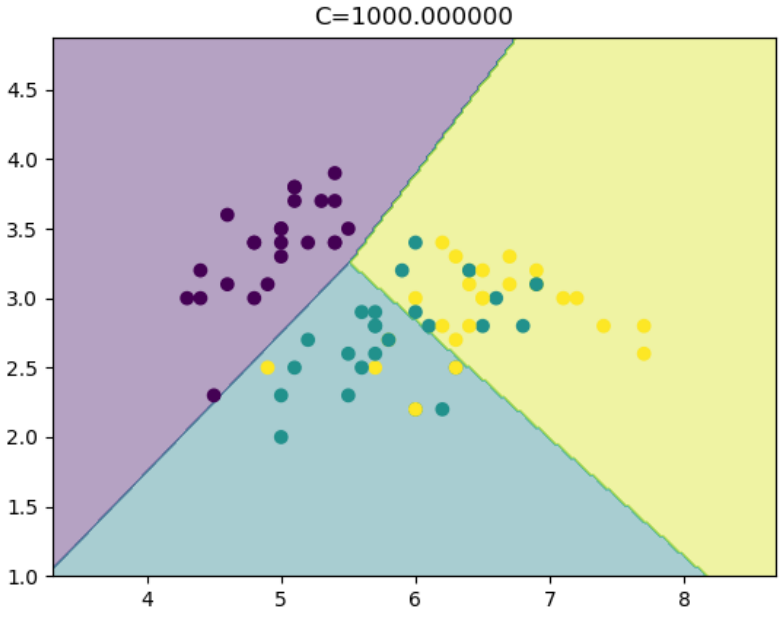

C=0.010000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


C=0.100000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


C=1.000000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


C=10.000000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

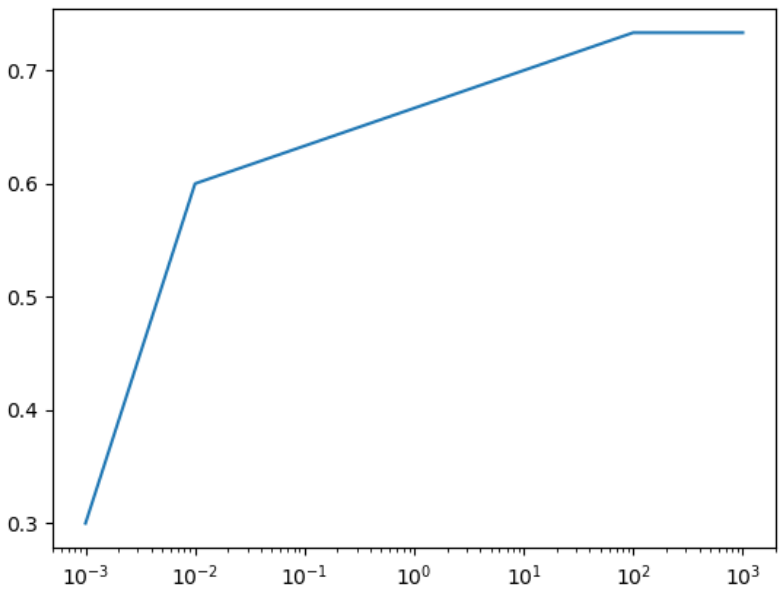

C=100.000000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


C=1000.000000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

We used the "one vs rest" in the seperaion.
Notice how for low values of C, we cannot separate the data. The linear SVM tries to find a good separation with low C but fails at finding a good one. Notice how the green class is squished between the purple boundary and the yellow one.
As we increase the value of C, thus allowing a little bet more error, The boundaries become more visible. To better see the effect of C on our classifier, let's plot the accuracy wrt values of C on the validation.

```
plt.plot(Cs,acc)
plt.xscale(value='log')
plt.savefig('./Output/Cs_acc.png')
plt.show()
```

The resulting plot is:



We could see that in our case, it reaches a maximum on the validation set at almost C=100. The code below automatically selects the lowest C with the highest accuracy. That way we allow the minimum mistakes but for maximum accuracy. We then test the SVM on our Test set

```
#Test Set Evaluation
C=Cs[acc.index(np.max(acc))] #Returns lowest C with highest
    accuracy
linear = svm.LinearSVC(C=C, max_iter=1000000000, multi_class='
    ovr')
linear.fit(X_train, y_train)
y_testP=linear.predict(X_test)
print("The accuracy of the classifier is: ")
linScore=accuracy_score(y_test,y_testP)
print(linScore)
```

The result was:

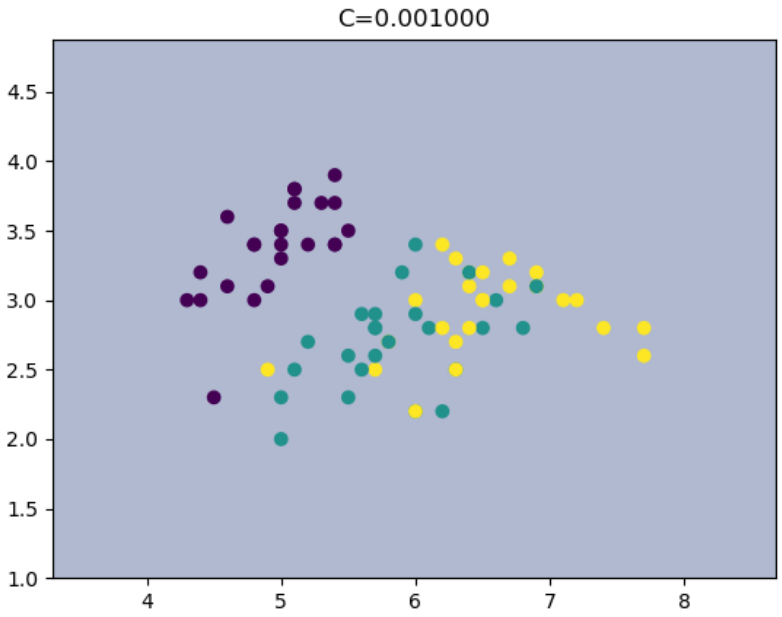
```
The accuracy of the classifier is:
0.822222222222222
```

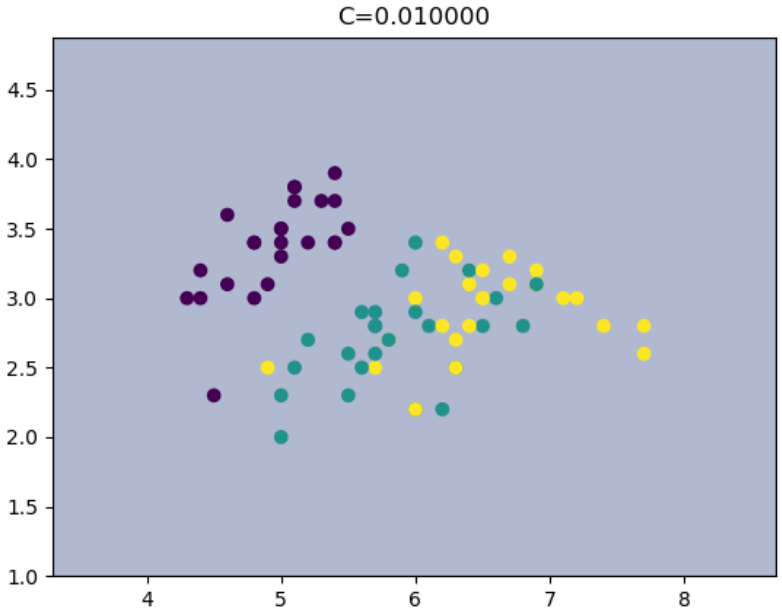
SVM with RBF Kernel

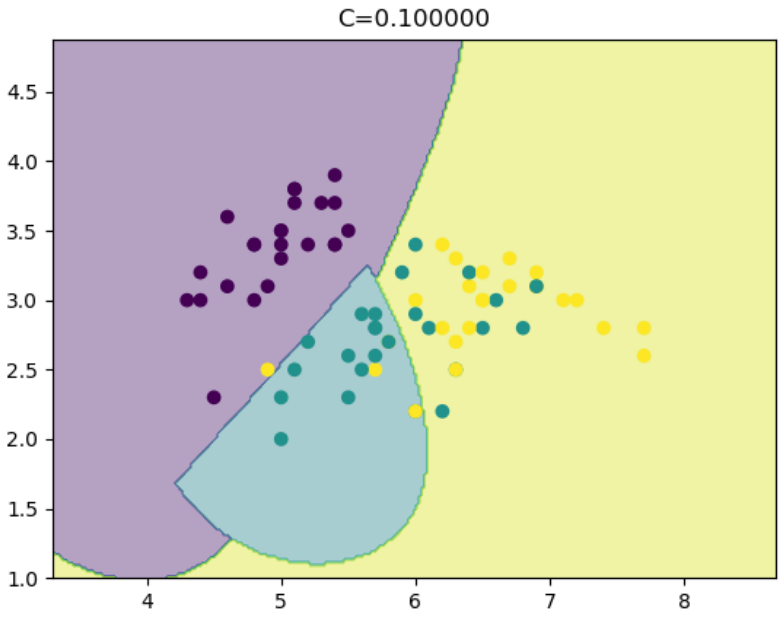
Doing the same as before, we just change the kernel from a linear one to the RBF kernel. Below is the code the loops over values of C with gamma set to auto. It also plots the decision boundaries in each case.

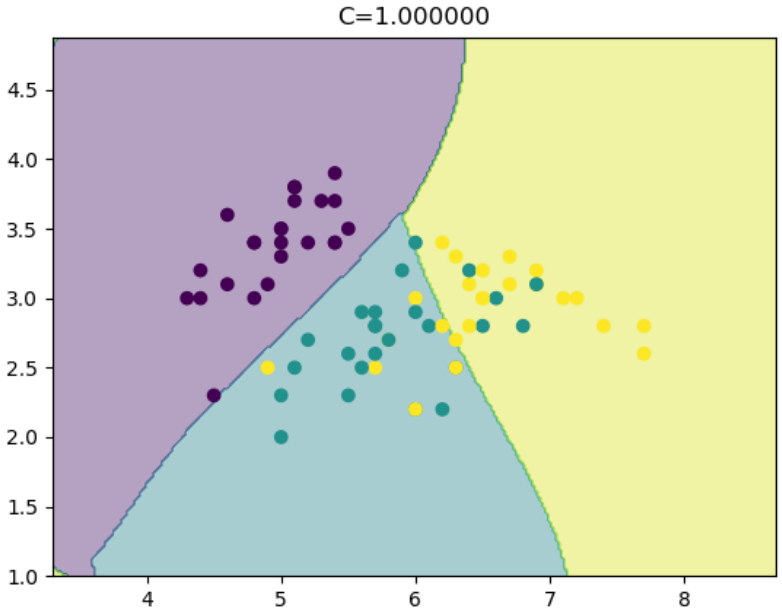
```
# RBF Kernel
acc=[]
for c in Cs:
    rbf=svm.SVC(C=c,kernel='rbf',gamma='auto')
    rbf.fit(X_train,y_train)
    Z = rbf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.4)
    plt.scatter(X_train[:,0],X_train[:,1],c=y_train)
    plt.title("C=%f" %c)
    plt.savefig('./Output/'+str(c)+'RBF.png')
    plt.show()

    y_valP=rbf.predict(X_val)
    acc.append(accuracy_score(y_val,y_valP))
```

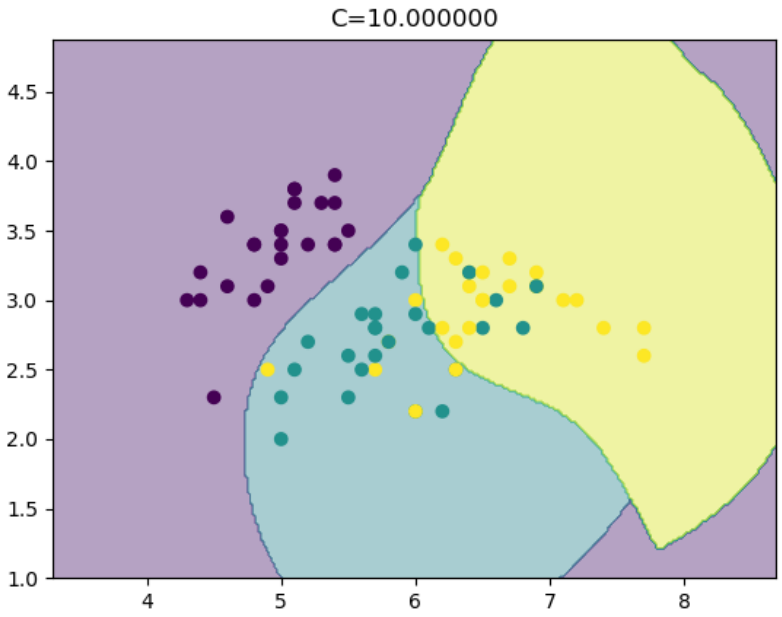
Plotting the decision boundaries for each value of C, we get:

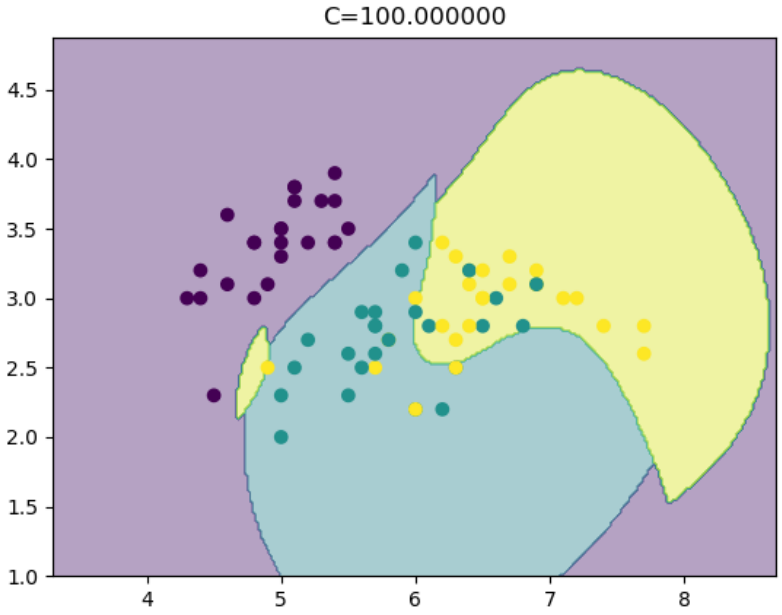

C=0.001000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

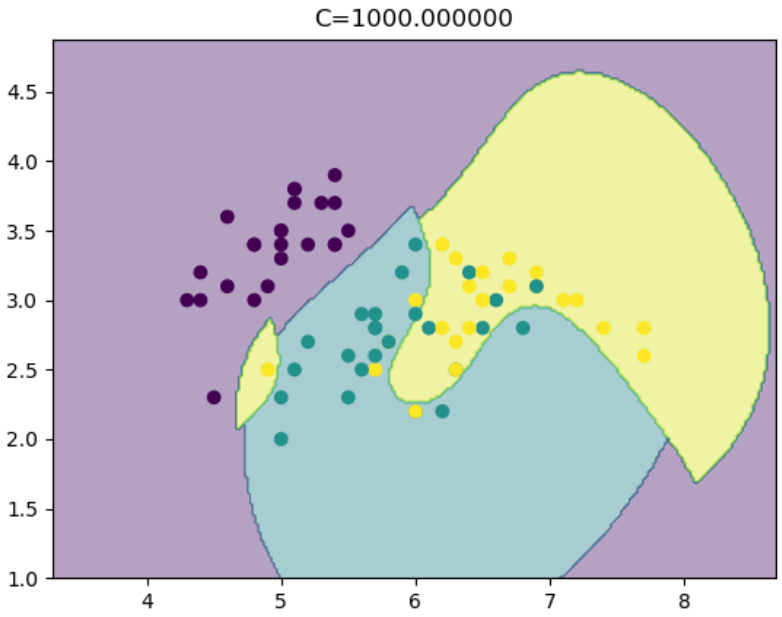

C=0.010000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


C=0.100000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

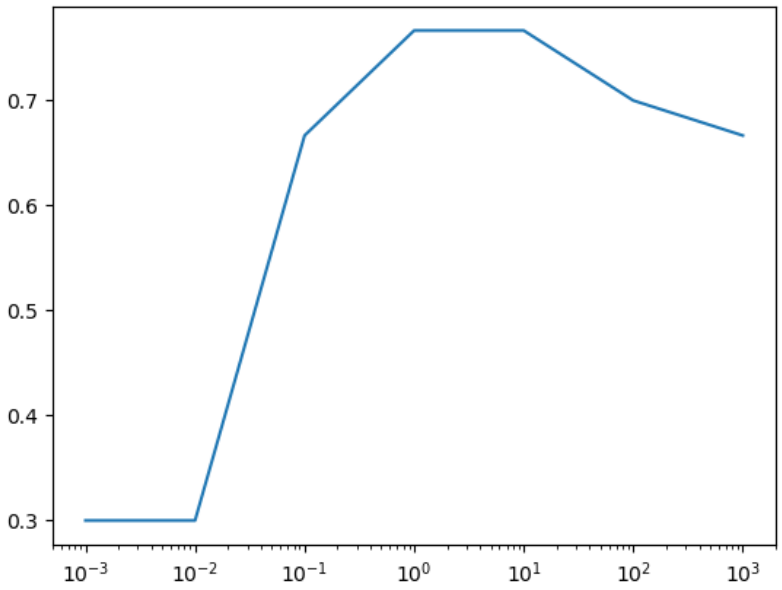

C=1.000000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


C=10.000000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8


C=100.000000
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8



Notice how for the low values of C, we weren't able to find a separator. As we increased the C, the decision boundaries started to form. For very high values of C, the model seems overfitted. We plot the accuracy on the validation set:



Look how the accuracy decreases at high Cs.
We test on our test set. We get the following output:

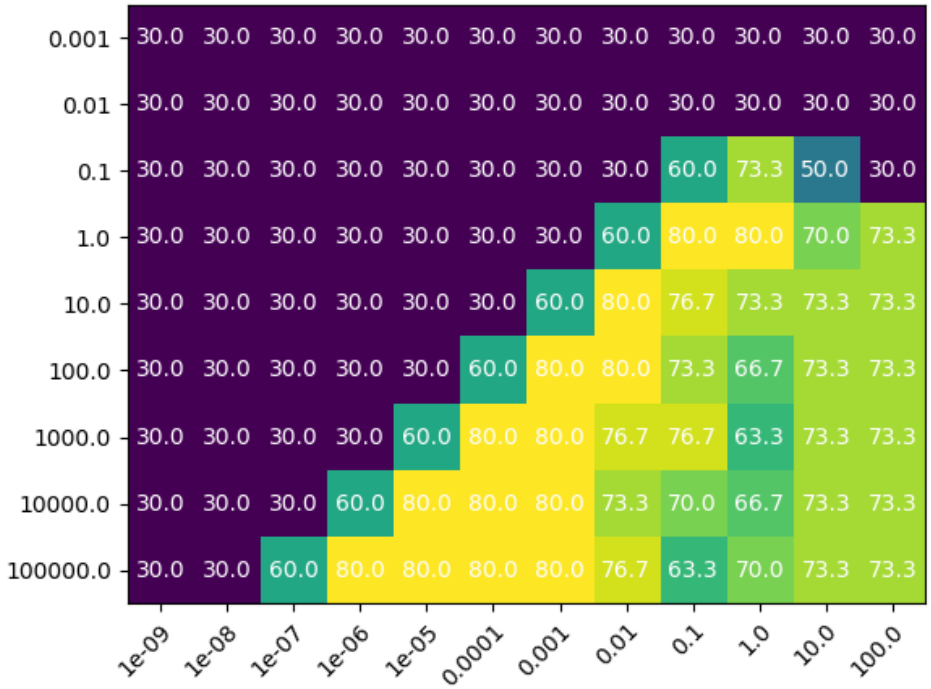
```
The accuracy of the classifier is:
0.8444444444444444
```

There was an increase in the accuracy and this was expected. Since this data isn't explicitly linearly separable, a linear SVM wouldn't be the best choice.
Now we want to do a grid search. The code below does that. We vary C from 0.001 to 10000 and gamma from 1e-9 to 100.

```
# Grid Search for both gamma and C

Cs= np.float_power(10,np.arange(-3,6,1))
Gs= np.float_power(10,np.arange(-9,3,1))
acc=np.zeros((len(Cs),len(Gs)))
```

```
for i,c in enumerate(Cs):
    for j,g in enumerate(Gs):
        rbf=svm.SVC(C=c,kernel='rbf',gamma=g)
        rbf.fit(X_train,y_train)
        y_valP=rbf.predict(X_val)
        acc[i,j]=accuracy_score(y_val,y_valP)

fig, ax = plt.subplots()
im = ax.imshow(acc)
ax.set_xticks(np.arange(len(Gs)))
ax.set_yticks(np.arange(len(Cs)))
ax.set_xticklabels(Gs)
ax.set_yticklabels(Cs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
for i in range(len(Cs)):
    for j in range(len(Gs)):
        text = ax.text(j, i, "{0:0.1f}".format(acc[i,j]*100),
                       ha="center", va="center", color="w")
plt.xlabel("Gamma")
plt.ylabel("C")
plt.savefig('./Output/gridSearch.png')
plt.show()
```

The output of the grid search is:



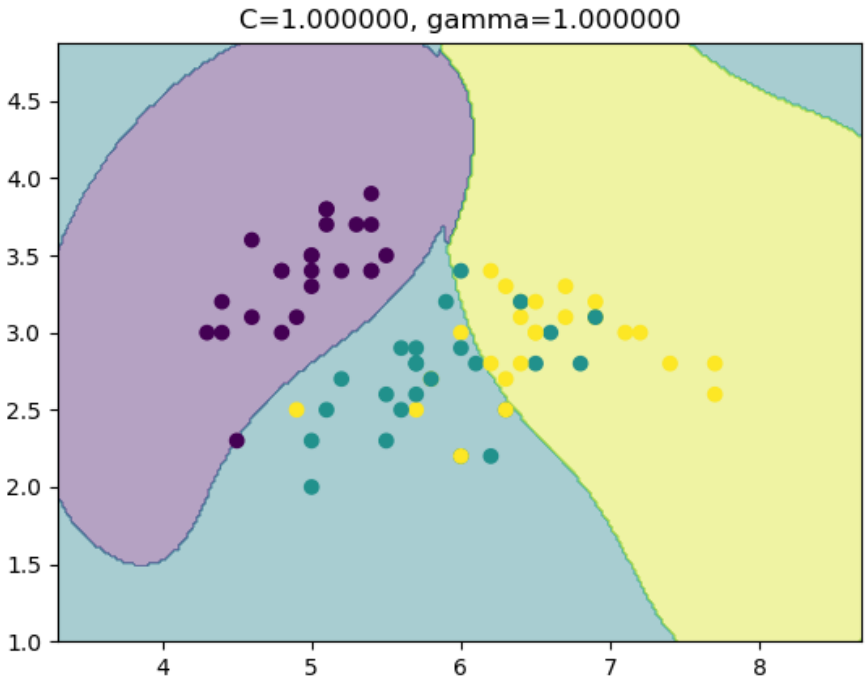
We now exact the values of C and gamma that correspond to the maximum accuracy. We then train the model

```
i,j=np.where(acc == np.max(acc))

Cf=Cs[i][0]
Gf=Gs[j][0]
rbf=svm.SVC(C=Cf,kernel='rbf',gamma=Gf)
rbf.fit(X_train,y_train)
```

The accuracy on the test set:

```
The accuracy of the classifier is:
0.8444444444444444
```

The resulting decision boundary is:



K-Fold

The merged training and validation set is already available. We do a grid search on the SVM with cross validation

```
# Grid search with CV

Cs= np.float_power(10,np.arange(-3,6,1))
Gs= np.float_power(10,np.arange(-9,3,1))
kf = KFold(n_splits=5)
kf.get_n_splits(X_tv,y_tv)
scores = np.empty((len(Cs), len(Gs), kf.n_splits))

for i, cvar in enumerate(Cs):
    for j, gvar in enumerate(Gs):
        for k,[train, test] in enumerate(kf.split(X_tv,y_tv)):
            rbf=svm.SVC(C=cvar,gamma=gvar)
            rbf.fit(X_tv[train], y_tv[train])
            scores[i,j,k] = rbf.score(X_tv[test],y_tv[test])
```

We then average the accuracy over the folds and plot the grid using the following code:

```
# This averages the value over each fold and plots the grid
average=np.zeros((len(Cs),len(Gs)))

for i in range(1,len(Cs)):
    for j in range(1,len(Gs)):
        average[i,j]=sum(scores[i,j,:])/kf.n_splits


fig, ax = plt.subplots()
im = ax.imshow(average)
ax.set_xticks(np.arange(len(Gs)))
ax.set_yticks(np.arange(len(Cs)))
ax.set_xticklabels(Gs)
ax.set_yticklabels(Cs)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
for i in range(len(Cs)):
    for j in range(len(Gs)):
        text = ax.text(j, i, "{0:0.1f}".format(average[i,j
   ]*100+5),
                       ha="center", va="center", color="w")
plt.xlabel("Gamma")
plt.ylabel("C")
plt.savefig('./Output/gridSearchNew.png')
plt.show()


# Max Accuracy values for C and Gamma
i,j=np.where(average == np.max(average))
Cf2=i[0,]
Gf2=j[0,]
```

The final accuracy is around 66%. Even though at certain folds the max value of accuracy was 85%, at certain folds it was less. This is due to the K-fold splitting of the data. For certain folds, and depending on the training data for that fold, the accuracy would certainly change. So, though the accuracy, it is somewhat acceptable in sense that it is dependant on the train data and test data.