PROJECT 3 - MIDTERM
(b) Create a binary classifier to predict the quality test result using Support Vector Machines

Problem Statement: The task is to develop a SVM binary classifier using the SVM functions from the Scikit-learn library to predict whether each capacitor in the test set will pass QC

Method of approach: I declared all of the dependencies required for the program's execution at the start. The training dataset then is read and transformed to a pandas dataframe. After that, I had to remove the dataset's first line, which contained the number of rows and features. After that, I had to partition the dataset into X_train and y_train dataframes for input and output, respectively. For the testing dataset, the same approach was used throughout, and the inputs and labels were classified into X_test and y_test dataframes. To begin the execution, I used sklearn to import stander scaler, which standardizes a feature by subtracting the mean and then scaling to unit variance. The svm model is imported from sklearn and performed using the RBF kernel technique after scaling is completed. When the data set is linearly inseparable, it is recommended to use kernel functions like RBF. The scatter plot in the provided issue statement shows that the dataset is linearly inseparable, signaling that training the model with the kernel strategy is a good choice. Finally, the confusion matrix has been imported from sklearn, and the corresponding accuracy, precision, recall, and f1 values have been calculated, with the model's final accuracy seeming to be 0.84 or 84 percent. plt.contour (referred at: https://matplotlib.org/3.5.1/api/_as_gen/matplotlib.pyplot.contourf.html) and np.meshgrid (referred at: https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html) methods are used to visualize the plot, as stated in the 9th march lecture.

The final values are:

Test Data:
Confusion Matrix:
[[17 0]
[5 11]]
Accuracy for test set = 84 percent
Precision for test set = 100 percent
Recall for test set = 77 percent
F1 for test set = 87 percent

Output and Screenshots:

jupyter Ravi_Adithya_P3_SVM Last Checkpoint: 4 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

```
TP = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
TN = cm[1,1]
accuracy=((TP+TN)/(TP+TN+FP+FN))*100
precision=(TP/(TP+FP))*100
recall=(TP/(TP+FN))*100
f1=2*(1/((1/precision)+(1/recall)))
print('\nTest Data:\n')
print("Confusion Matrix:")
print(cm)
print('Accuracy for test set = %d percent' %accuracy)
print('Precision for test set = %d percent' %precision)
print('Recall for test set = %d percent' %recall)
print('F1 for test set = %d percent' %f1)
```

```
Test Data:

Confusion Matrix:
[[17  0]
 [ 5 11]]
Accuracy for test set = 84 percent
Precision for test set = 100 percent
Recall for test set = 77 percent
F1 for test set = 87 percent
```

In [58]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 0.01, stop = X_set[:, 0].max() + 0.01, step = 0.1),
                     np.arange(start = X_set[:, 1].min() - 0.01, stop = X_set[:, 1].max() + 0.01, step = 0.1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
```

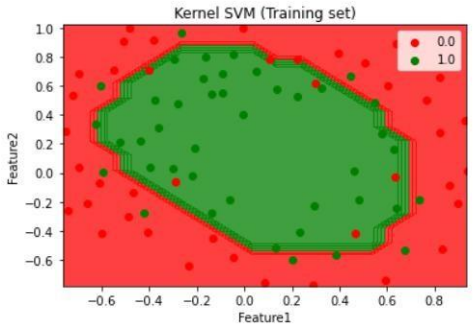
jupyter Ravi_Adithya_P3_SVM Last Checkpoint: 4 hours ago (autosaved)

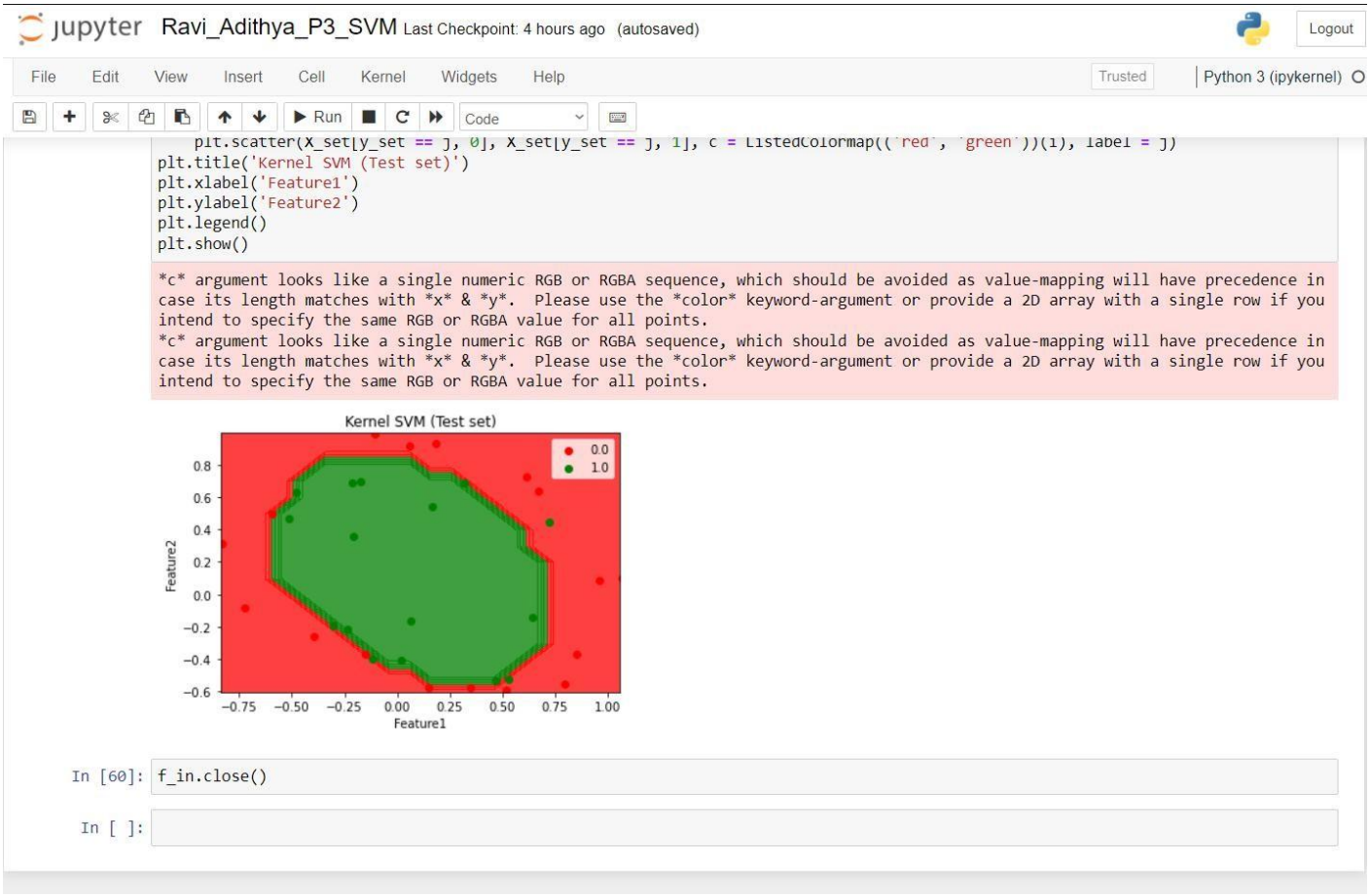
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

```
                     np.arange(start = X_set[:, 1].min() - 0.01, stop = X_set[:, 1].max() + 0.01, step = 0.1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Kernel SVM (Training set)')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```





Source code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

f = input("Enter the name of your training dataset file: ")
f_in= open(f, "r")

df=pd.read_csv(f,delimiter='\t', header=None, names=['Feature1','Feature2','Outcome'])
df = df.iloc[1: , :]
X_train = df.iloc[:, :-1].values
y_train = df.iloc[:, -1].values
f_in.close()

f = input("Enter the name of the testing dataset file:\t")
f_in = open(f,"r")
df=pd.read_csv(f,delimiter='\t', header=None, names=['Feature1','Feature2','Outcome'])
df = df.iloc[1: , :]
X_test = df.iloc[:, :-1].values
```

```
y_test = df.iloc[:, -1].values

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0, C=1.0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
TP = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
TN = cm[1,1]
accuracy=((TP+TN)/(TP+TN+FP+FN))*100
precision=(TP/(TP+FP))*100
recall=(TP/(TP+FN))*100
f1=2*(1/((1/precision)+(1/recall)))
print('\nTest Data:\n')
print("Confusion Matrix:")
print(cm)
print('Accuracy for test set = %d percent' %accuracy)
print('Precision for test set = %d percent' %precision)
print('Recall for test set = %d percent' %recall)
print('F1 for test set = %d percent' %f1)

from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 0.01, stop = X_set[:, 0].max()
+ 0.01, step = 0.1),
                     np.arange(start = X_set[:, 1].min() - 0.01, stop = X_set[:,
1].max() + 0.01, step = 0.1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
```

```
        plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red',
'green'))(i), label = j)
plt.title('Kernel SVM (Training set)')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.legend()
plt.show()

from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 0.01, stop = X_set[:, 0].max()
+ 0.01, step = 0.1),
                                np.arange(start = X_set[:, 1].min() - 0.01, stop = X_set[:,
1].max() + 0.01, step = 0.1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(),
X2.ravel()]).T)).reshape(X1.shape),
                     alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
        plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red',
'green'))(i), label = j)
plt.title('Kernel SVM (Test set)')
plt.xlabel('Feature1')
plt.ylabel('Feature2') plt.legend()
plt.show()
```