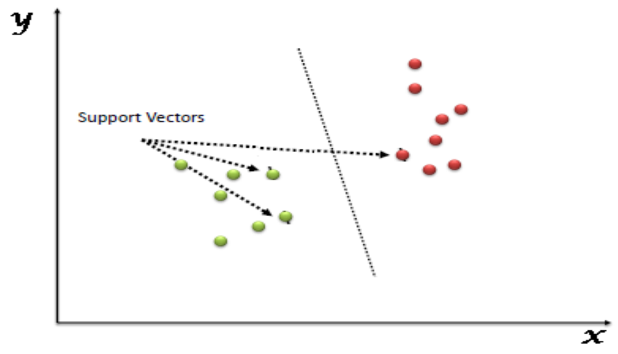
Experiment No. 5

Title: To implement SVM

Lab Objective: To implement an appropriate machine learning model for the given application. LO1

Theory: "Support Vector Machine" (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).



Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/ line).

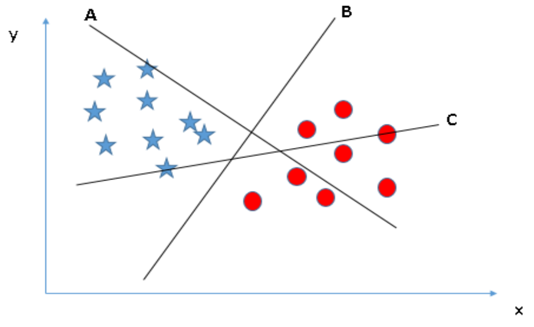
You can look at support vector machines and a few examples of their working here.

How does it work?

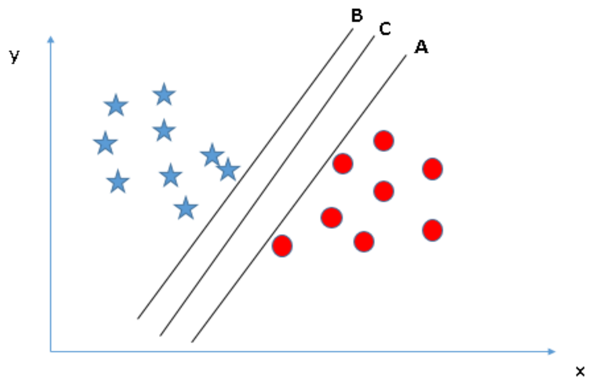
Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now the burning question is "How can we identify the right hyper-plane?". Don't worry, it's not as hard as you think!

Let's understand:

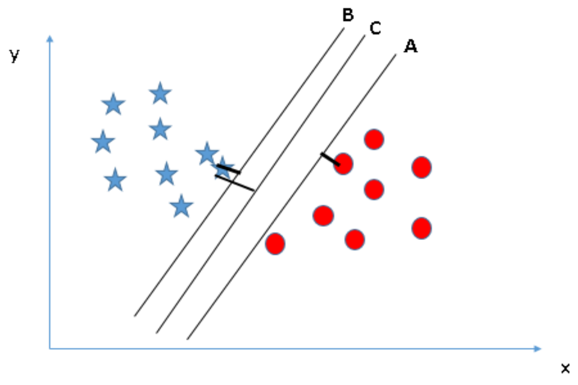
- **Identify the right hyper-plane (Scenario-1):** Here, we have three hyper-planes (A, B, and C). Now, identify the right hyper-plane to classify stars and circles.



- You need to remember a thumb rule to identify the right hyper-plane: "Select the hyper-plane which segregates the two classes better". In this scenario, hyper-plane "B" has excellently performed this job.
- **Identify the right hyper-plane (Scenario-2):** Here, we have three hyper-planes (A, B, and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?

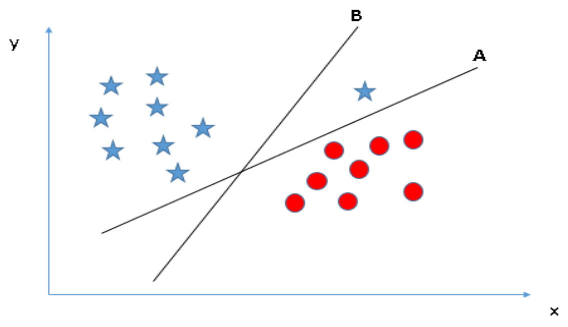


Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**.



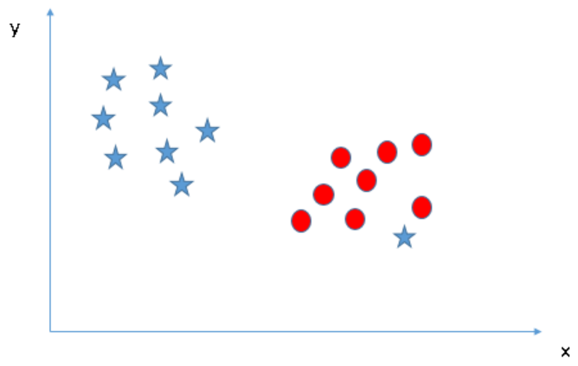
Let’s look at the below snapshot:
Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

- **Identify the right hyper-plane (Scenario-3):**Hint: Use the rules as discussed in previous section to identify the right hyper-plane

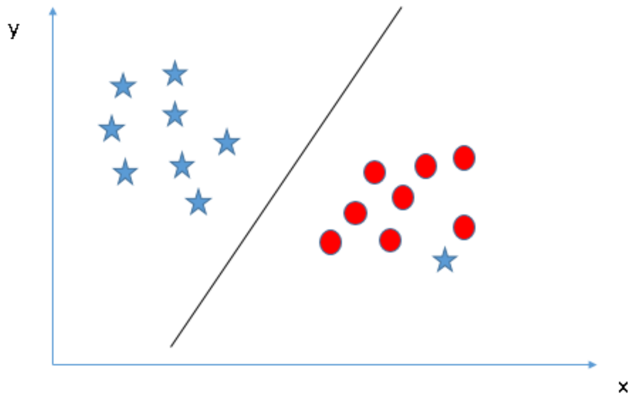


Some of you may have selected the hyper-plane **B** as it has higher margin compared to **A.** But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**

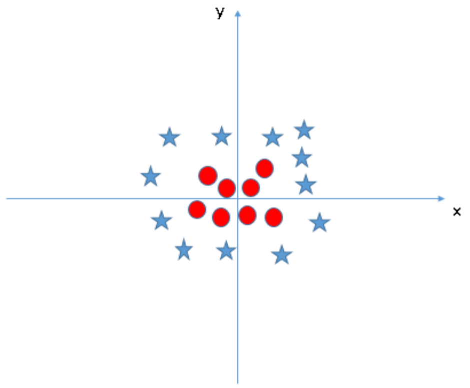
- **Can we classify two classes (Scenario-4)?:** Below, I am unable to segregate the two classes using a straight line, as one of the stars lies in the territory of other(circle) class as an outlier.



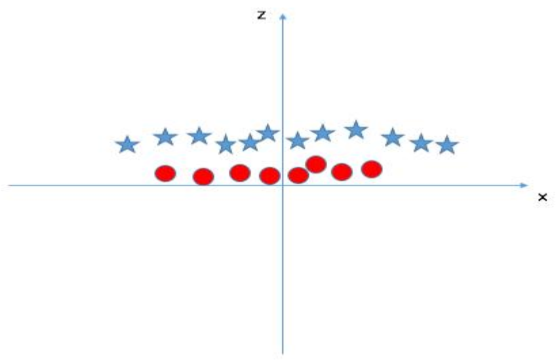
As I have already mentioned, one star at other end is like an outlier for star class. The SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin. Hence, we can say, SVM classification is robust to outliers.



- **Find the hyper-plane to segregate to classes (Scenario-5):** In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.



- SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature z=x^2+y^2. Now, let's plot the data points on axis x and z:

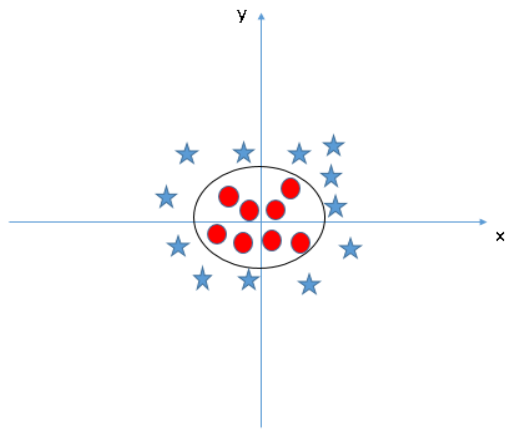


In above plot, points to consider are:

- All values for z would be positive always because z is the squared sum of both x and y
- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.

In the SVM classifier, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, the SVM algorithm has a technique called the **kernel trick**. The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined.

When we look at the hyper-plane in original input space it looks like a circle:



Prerequisite Software and Command:

- **Python 3 and above**
- **Pip install numpy**

- **Pip install pandas**
- **Pip install matplotlib**
- **Pip install sklearn**

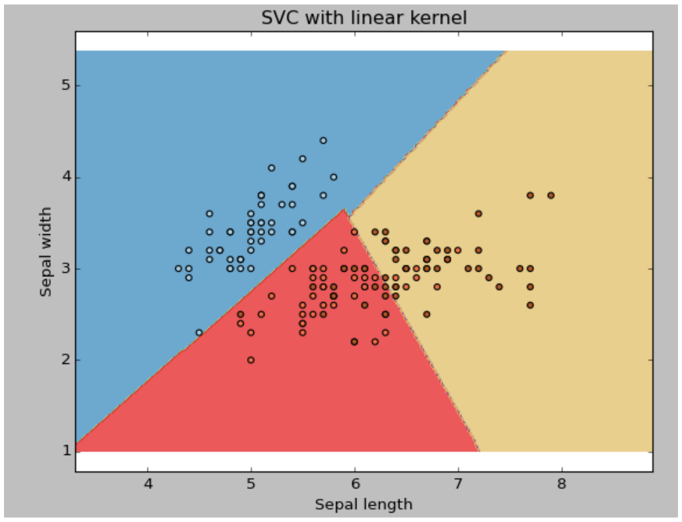
(These above command should be run only once)

Program Code:

SVM Code: Have a linear SVM kernel

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features. We could
 # avoid this ugly slicing by using a two-dim dataset
y = iris.target
# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=1,gamma='auto').fit(X, y)
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
```

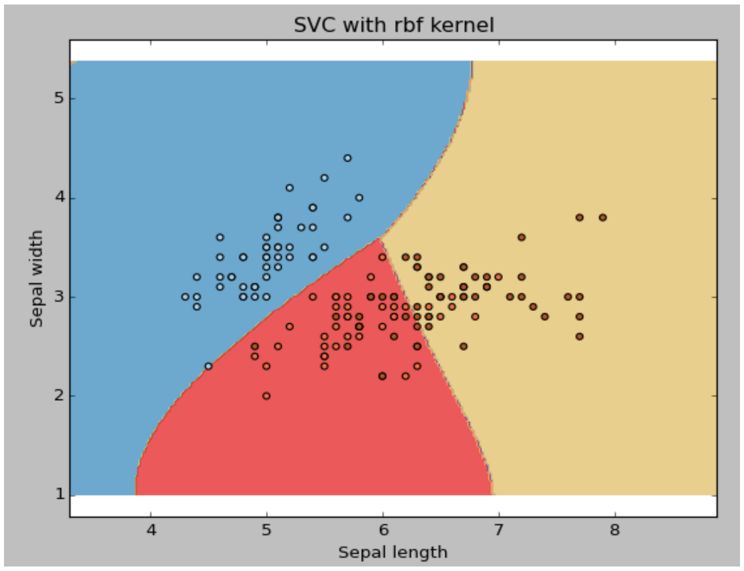
```
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 np.arange(y_min, y_max, h))
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
```



Example: Use SVM rbf kernel

Change the kernel type to rbf in below line and look at the impact.

```
svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)
```

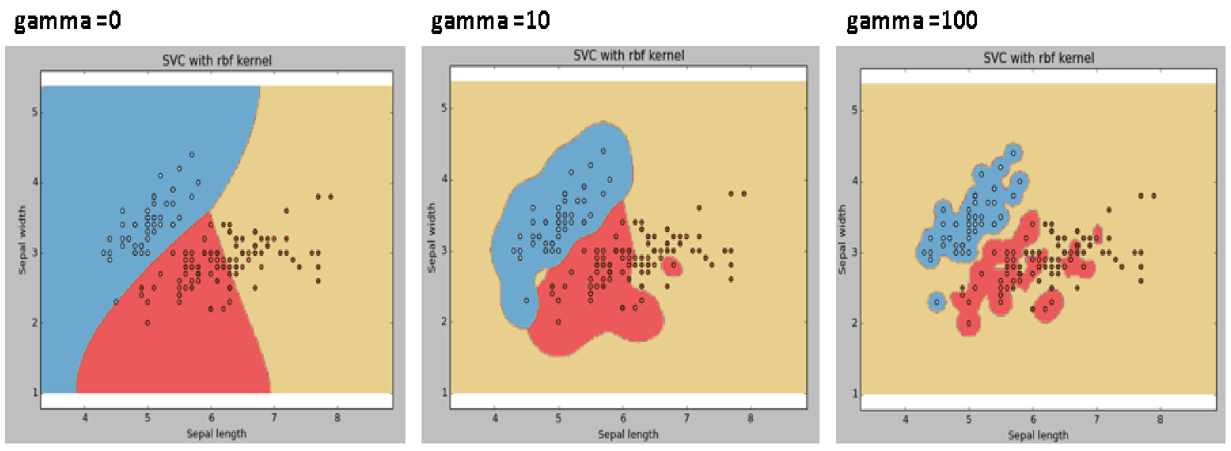


I would suggest you go for linear SVM kernel if you have a large number of features (>1000) because it is more likely that the data is linearly separable in high dimensional space. Also, you can use RBF but do not forget to cross-validate for its parameters to avoid over-fitting.

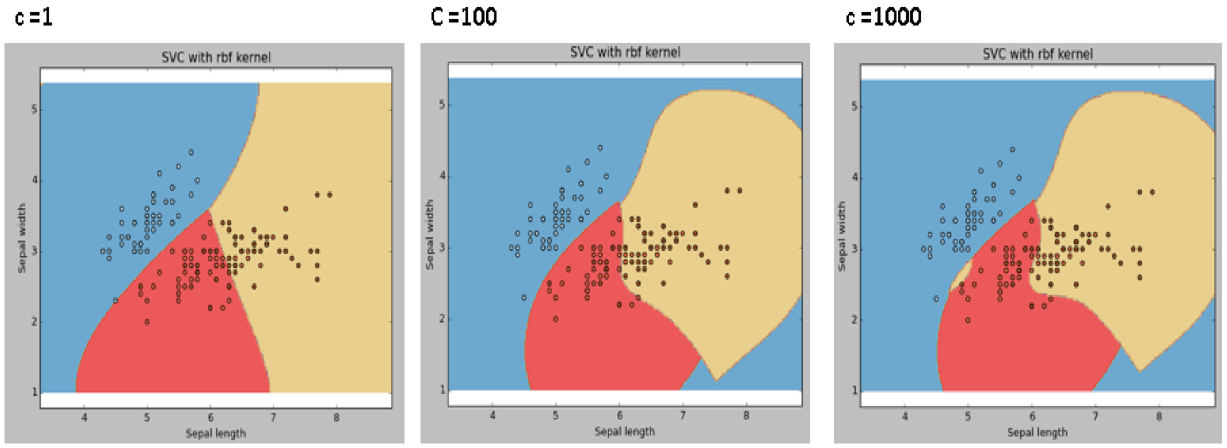
gamma: Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. Higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.

Example: Let’s difference if we have gamma different gamma values like 0, 10 or 100.

svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)

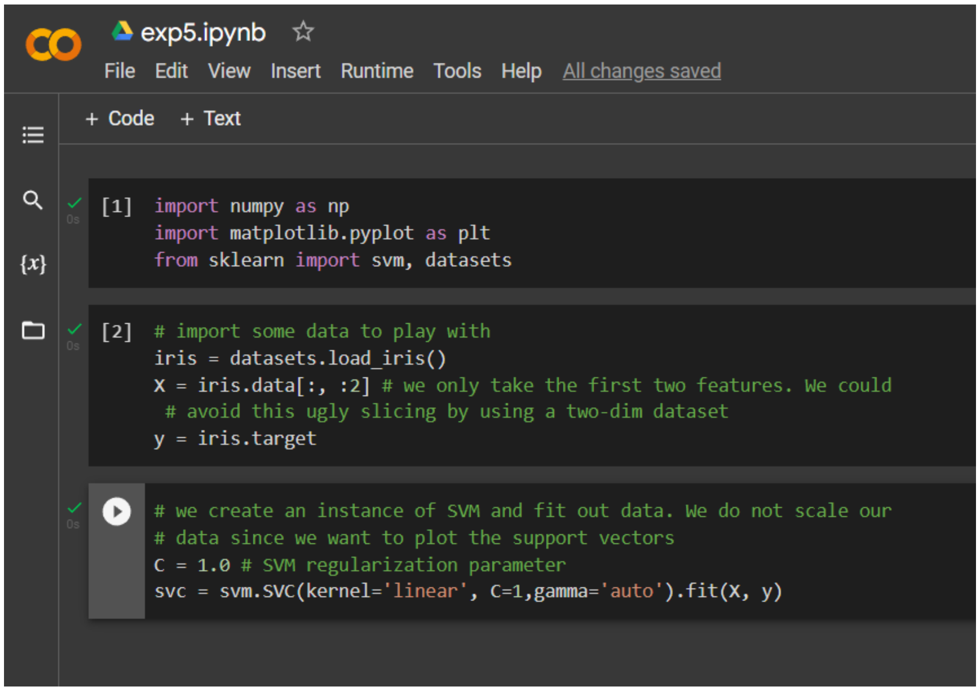


C: Penalty parameter C of the error term. It also controls the trade-off between smooth decision boundaries and classifying the training points correctly.



We should always look at the cross-validation score to have effective combination of these parameters and avoid over-fitting.

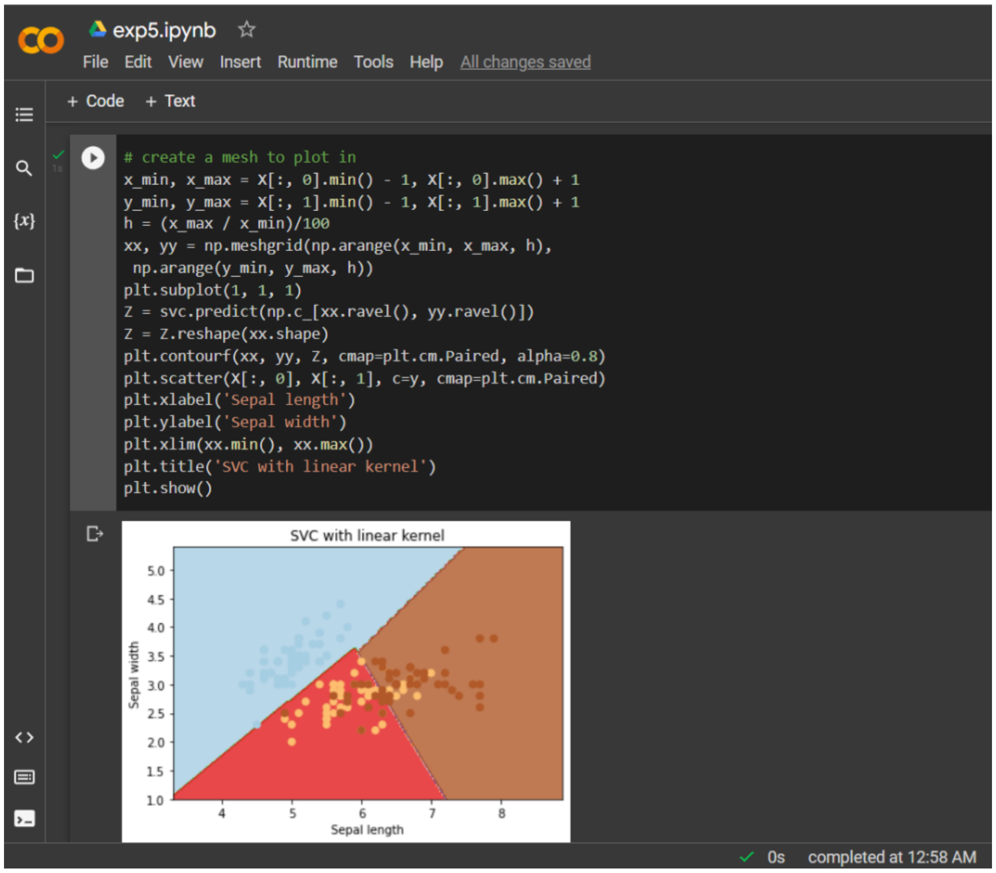
IMPLEMANTATION:



```
[1] import numpy as np
    import matplotlib.pyplot as plt
    from sklearn import svm, datasets
```

```
[2] # import some data to play with
    iris = datasets.load_iris()
    X = iris.data[:, :2] # we only take the first two features. We could
     # avoid this ugly slicing by using a two-dim dataset
    y = iris.target
```

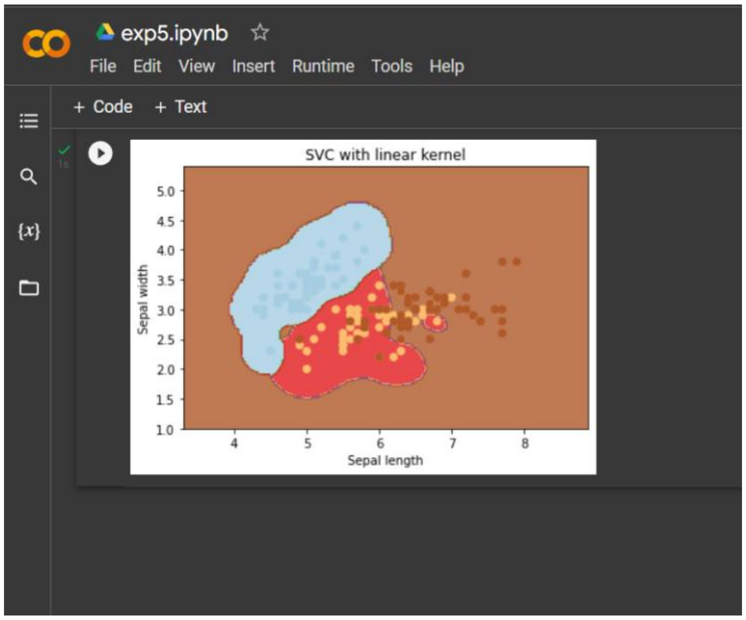
```
# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=1,gamma='auto').fit(X, y)
```



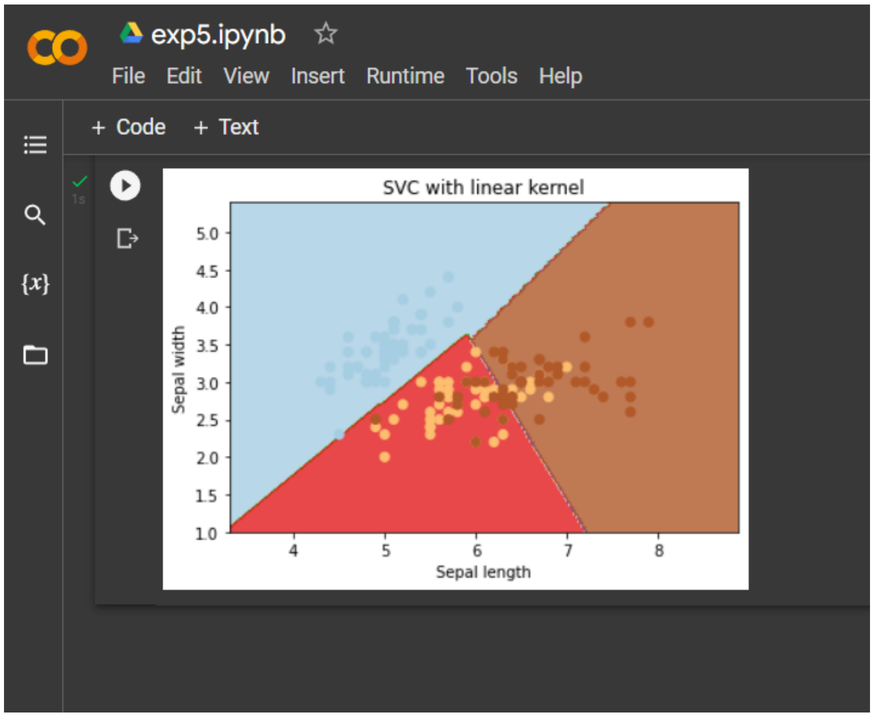
```
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 np.arange(y_min, y_max, h))
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
```

PROGRAM OUTPUT:

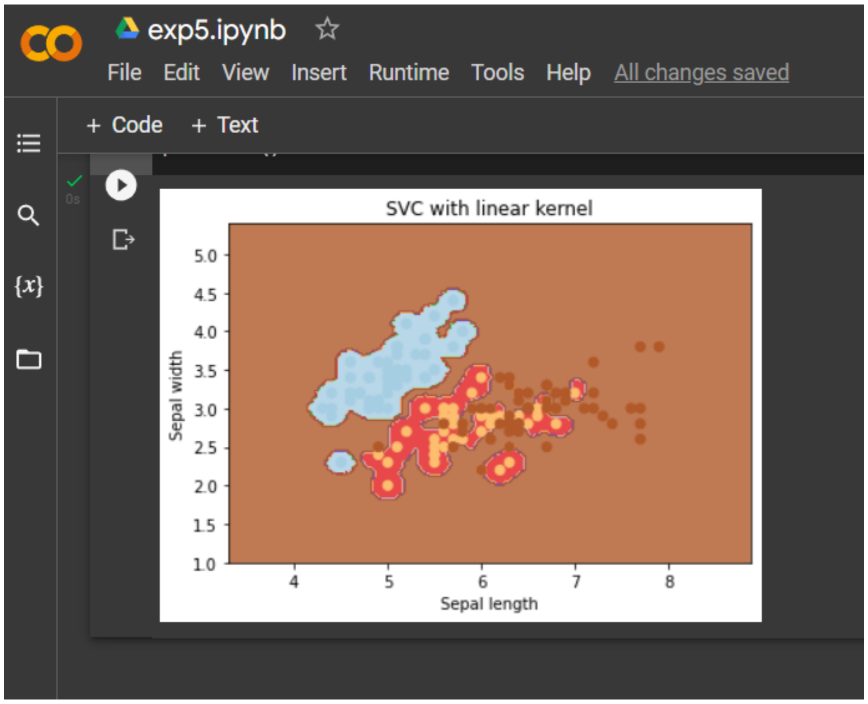
svc = svm.SVC(kernel='rbf', C=1,gamma=10).fit(X, y)



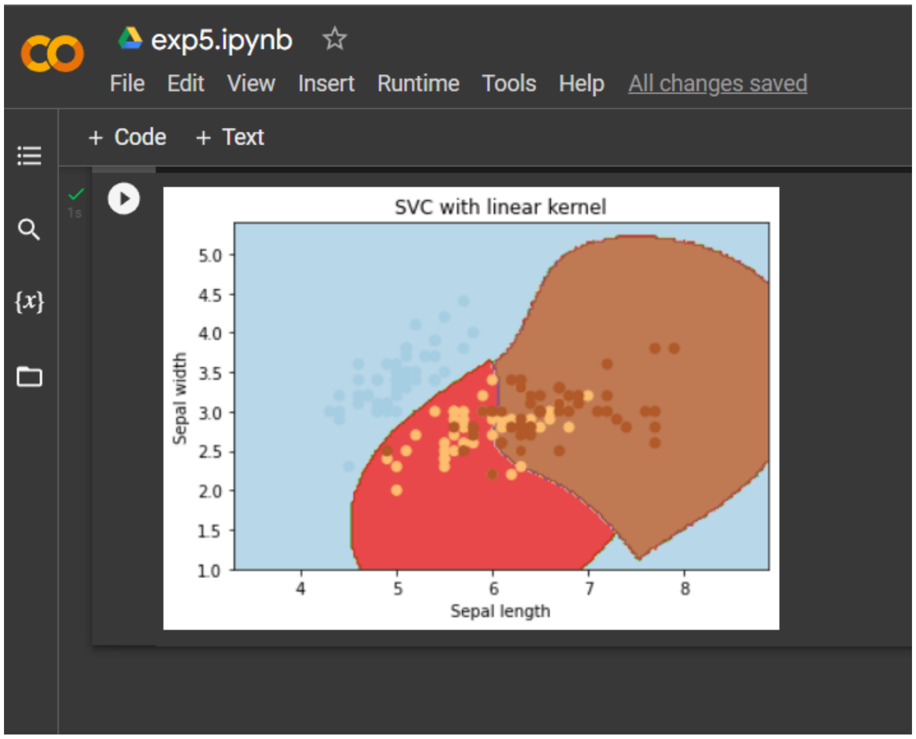
svc = svm.SVC(kernel='linear', C=1,gamma=100).fit(X, y)



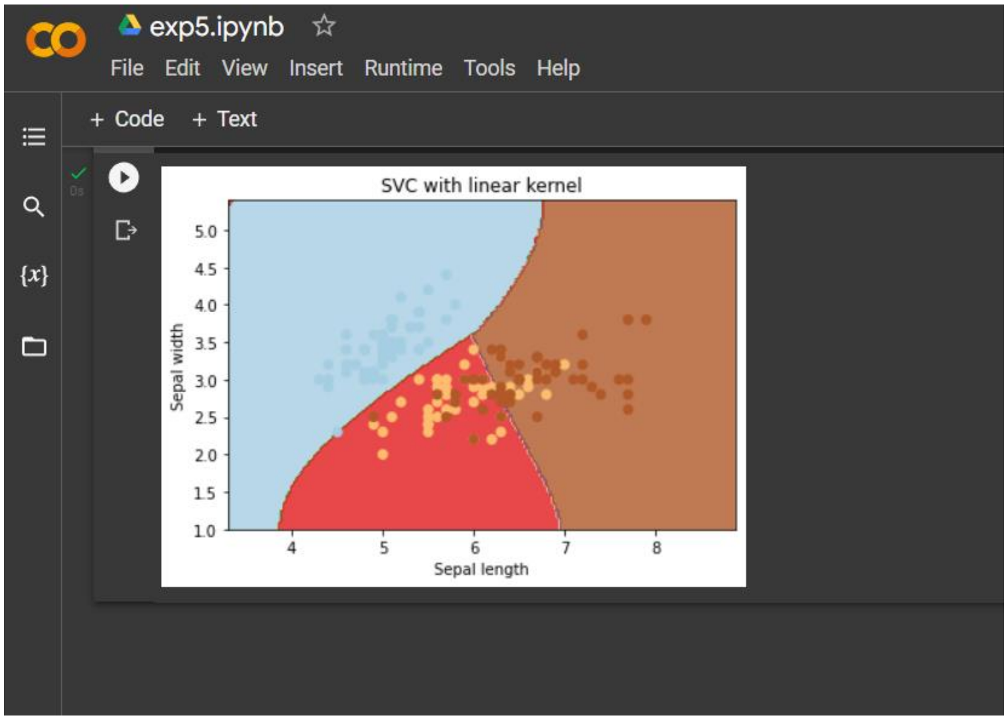
svc = svm.SVC(kernel=rbf, C=1,gamma=100).fit(X, y)



svc = svm.SVC(kernel='rbf', C=100,gamma='auto').fit(X, y)



svc = svm.SVC(kernel='rbf', C=1,gamma='auto').fit(X, y)



Conclusion: SVM implemented with experimental model on given data set of iris file for prediction of sepals length and width.