Here I m going to run Support Vector machine with different kernels(linear,gaussian,polynomial) and also tune the various parameters such as C ,gamma and degree to find out the best performing model .

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline
```

Reading the comma separated values file into the dataframe

In [2]:

```
df = pd.read_csv('voice.csv')
df.head()
```

Out[2]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.e
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.8933
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.8921
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.8463
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.9633
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.9719

5 rows × 21 columns

Checking the correlation between each feature

In [3]:

```
1 df.corr()
```

Out[3]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt
meanfreq	1.000000	-0.739039	0.925445	0.911416	0.740997	-0.627605	-0.322327	-0.316036
sd	-0.739039	1.000000	-0.562603	-0.846931	-0.161076	0.874660	0.314597	0.346241
median	0.925445	-0.562603	1.000000	0.774922	0.731849	-0.477352	-0.257407	-0.243382
Q25	0.911416	-0.846931	0.774922	1.000000	0.477140	-0.874189	-0.319475	-0.350182
Q75	0.740997	-0.161076	0.731849	0.477140	1.000000	0.009636	-0.206339	-0.148881
IQR	-0.627605	0.874660	-0.477352	-0.874189	0.009636	1.000000	0.249497	0.316185
skew	-0.322327	0.314597	-0.257407	-0.319475	-0.206339	0.249497	1.000000	0.977020
kurt	-0.316036	0.346241	-0.243382	-0.350182	-0.148881	0.316185	0.977020	1.000000
sp.ent	-0.601203	0.716620	-0.502005	-0.648126	-0.174905	0.640813	-0.195459	-0.127644
sfm	-0.784332	0.838086	-0.661690	-0.766875	-0.378198	0.663601	0.079694	0.109884
mode	0.687715	-0.529150	0.677433	0.591277	0.486857	-0.403764	-0.434859	-0.406722
centroid	1.000000	-0.739039	0.925445	0.911416	0.740997	-0.627605	-0.322327	-0.316036
meanfun	0.460844	-0.466281	0.414909	0.545035	0.155091	-0.534462	-0.167668	-0.194560
minfun	0.383937	-0.345609	0.337602	0.320994	0.258002	-0.222680	-0.216954	-0.203201
maxfun	0.274004	-0.129662	0.251328	0.199841	0.285584	-0.069588	-0.080861	-0.045667
meandom	0.536666	-0.482726	0.455943	0.467403	0.359181	-0.333362	-0.336848	-0.303234
mindom	0.229261	-0.357667	0.191169	0.302255	-0.023750	-0.357037	-0.061608	-0.103313
maxdom	0.519528	-0.482278	0.438919	0.459683	0.335114	-0.337877	-0.305651	-0.274500
dfrange	0.515570	-0.475999	0.435621	0.454394	0.335648	-0.331563	-0.304640	-0.272729
modindx	-0.216979	0.122660	-0.213298	-0.141377	-0.216475	0.041252	-0.169325	-0.205539

Checking whether there is any null values

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
meanfreq    0
sd          0
median      0
Q25         0
Q75         0
IQR         0
skew        0
kurt        0
sp.ent      0
sfm         0
mode        0
centroid    0
meanfun     0
minfun      0
maxfun      0
meandom     0
mindom      0
maxdom      0
dfrange     0
modindx     0
label       0
dtype: int64
```

In [5]:

```
df.shape
```

Out[5]:

```
(3168, 21)
```

In [6]:

```
print("Total number of labels: {}".format(df.shape[0]))
print("Number of male: {}".format(df[df.label == 'male'].shape[0]))
print("Number of female: {}".format(df[df.label == 'female'].shape[0]))
```

```
Total number of labels: 3168
Number of male: 1584
Number of female: 1584
```

Thus we can see there are equal number of male and female labels

In [7]:

```
df.shape
```

Out[7]:

```
(3168, 21)
```

There are 21 features and 3168 instances.

Separating features and labels

In [8]:

```
X=df.iloc[:, :-1]
X.head()
```

Out[8]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.e
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.8933
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.8921
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.8463
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.9633
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.9719

Converting string value to int type for labels

In [9]:

```
from sklearn.preprocessing import LabelEncoder
y=df.iloc[:,-1]

# Encode label category
# male -> 1
# female -> 0

gender_encoder = LabelEncoder()
y = gender_encoder.fit_transform(y)
print(y)
```

```
[1 1 1 ... 0 0 0]
```

Data Standardisation

Standardization refers to shifting the distribution of each attribute to have a mean of zero and a standard deviation of one (unit variance). It is useful to standardize attributes for a model. Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data.

In [10]:

```
# Scale the data to be between -1 and 1
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
```

Splitting dataset into training set and testing set for better generalisation

In [11]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1
```

Running SVM with default hyperparameter.

In [12]:

```
from sklearn.svm import SVC
from sklearn import metrics
svc=SVC() #Default hyperparameters
svc.fit(X_train,y_train)
y_pred=svc.predict(X_test)
print('Accuracy Score:')
print(metrics.accuracy_score(y_test,y_pred))

```

```
Accuracy Score:
0.9763406940063092
```

Default Linear kernel

In [13]:

```
svc=SVC(kernel='linear')
svc.fit(X_train,y_train)
y_pred=svc.predict(X_test)
print('Accuracy Score:')
print(metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy Score:
0.9779179810725552
```

Default RBF kernel

In [14]:

```
svc=SVC(kernel='rbf')
svc.fit(X_train,y_train)
y_pred=svc.predict(X_test)
print('Accuracy Score:')
print(metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy Score:
0.9763406940063092
```

We can conclude from above that svm by default uses rbf kernel as a parameter for kernel

Default Polynomial kernel¶

In [15]:

```
svc=SVC(kernel='poly')
svc.fit(X_train,y_train)
y_pred=svc.predict(X_test)
print('Accuracy Score:')
print(metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy Score:
0.9589905362776026
```

Polynomial kernel is performing poorly.The reason behind this maybe it is overfitting the training dataset

Performing K-fold cross validation with different kernels

CV on Linear kernel

In [23]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
svc=SVC(kernel='linear')
scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy') #cv is cross validation
print(scores)
```

```
[0.91167192 0.97160883 0.97160883 0.97791798 0.95899054 0.9873817
 0.99369085 0.97791798 0.95253165 0.99367089]
```

We can see above how the accuracy score is different everytime.This shows that accuracy score depends upon how the datasets got split.

In [24]:

```
print(scores.mean())
```

```
0.9696991175178692
```

In K-fold cross validation we generally take the mean of all the scores.

In [26]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
svc=SVC(kernel='rbf')
scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy') #cv is cross validation
print(scores)
```

```
[0.93375394 0.95583596 0.96845426 0.96214511 0.96529968 0.99684543
 0.99053628 0.98422713 0.91455696 0.99367089]
```

In [27]:

```
print(scores.mean())
```

```
0.9665325639899376
```

CV on Polynomial kernel

In [29]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
svc=SVC(kernel='poly')
scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy') #cv is cross validation
print(scores)
```

```
[0.89274448 0.94952681 0.93059937 0.92744479 0.94952681 0.99369085
 0.98422713 0.96529968 0.87974684 0.9778481 ]
```

In [30]:

```
print(scores.mean())
```

```
0.9450654873617378
```

When K-fold cross validation is done we can see different score in each iteration.This happens because when we use train_test_split method,the dataset get split in random manner into testing and training dataset.Thus it depends on how the dataset got split and which samples are training set and which samples are in testing set.

With K-fold cross validation we can see that the dataset got split into 10 equal parts thus covering all the data into training as well into testing set.This is the reason we got 10 different accuracy score.

Taking all the values of C and checking out the accuracy score with kernel as linear.

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

Thus for a very large values we can cause overfitting of the model and for a very small value of C we can cause underfitting.Thus the value of C must be chosen in such a manner that it generalised the unseen data well

In [31]:

```
C_range=list(range(1,26))
acc_score=[]
for c in C_range:
    svc = SVC(kernel='linear', C=c)
    scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy')
    acc_score.append(scores.mean())
print(acc_score)
```

```
[0.9696991175178692, 0.969068202691371, 0.969068202691371, 0.96906820269137
1, 0.9693836601046201, 0.9693836601046201, 0.969068202691371, 0.968752745278
1215, 0.9684372878648724, 0.9684372878648724, 0.9684372878648724, 0.96843728
78648724, 0.9681208321686698, 0.9681208321686698, 0.9681208321686698, 0.9681
208321686698, 0.9678043764724673, 0.9678043764724673, 0.9678043764724673, 0.
9678043764724673, 0.9681208321686698, 0.968436289581919, 0.968436289581919,
0.9681198338857164, 0.9681198338857164]
```

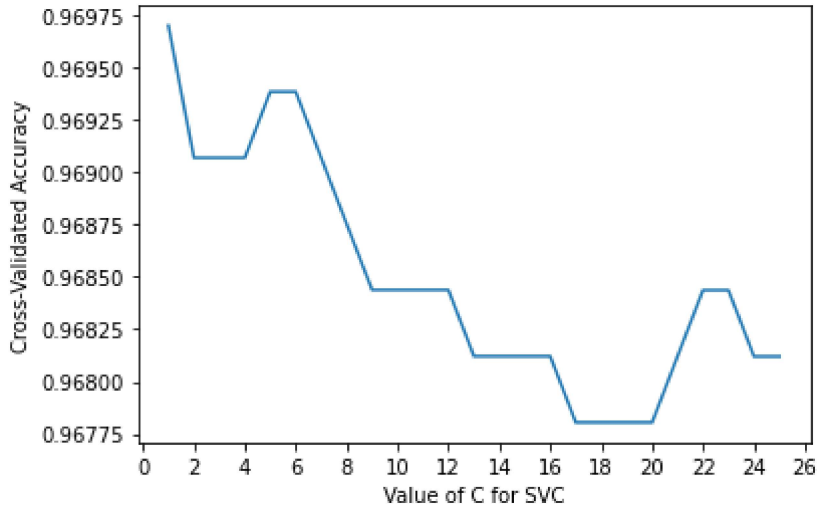
In [32]:

```
import matplotlib.pyplot as plt
%matplotlib inline


C_values=list(range(1,26))
# plot the value of C for SVM (x-axis) versus the cross-validated accuracy (y-axis)
plt.plot(C_values,acc_score)
plt.xticks(np.arange(0,27,2))
plt.xlabel('Value of C for SVC')
plt.ylabel('Cross-Validated Accuracy')
```

Out[32]:

```
Text(0, 0.5, 'Cross-Validated Accuracy')
```



From the above plot we can see that accuracy has been close to 97% for C=1 and C=6 and then it drops around 96.8% and remains constant.

Let us look into more detail of what is the exact value of C which is giving us a good accuracy score

In [33]:

```
C_range=list(np.arange(0.1,6,0.1))
acc_score=[]
for c in C_range:
    svc = SVC(kernel='linear', C=c)
    scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy')
    acc_score.append(scores.mean())
print(acc_score)
```

```
[0.9706474863235236, 0.9693846583875733, 0.9693836601046201, 0.9693836601046
201, 0.9696991175178692, 0.9696991175178692, 0.9693836601046201, 0.969698119
2349158, 0.9693826618216667, 0.9696991175178692, 0.9696991175178692, 0.96969
91175178692, 0.9696991175178692, 0.9690672044084174, 0.9693836601046201, 0.9
693836601046201, 0.9693836601046201, 0.9693836601046201, 0.969068202691371,
0.969068202691371, 0.9693836601046201, 0.9693836601046201, 0.96906820269137
1, 0.969068202691371, 0.969068202691371, 0.969068202691371, 0.96906820269137
1, 0.969068202691371, 0.969068202691371, 0.969068202691371, 0.96906820269137
1, 0.969068202691371, 0.969068202691371, 0.969068202691371, 0.96906820269137
1, 0.969068202691371, 0.969068202691371, 0.969068202691371, 0.96906820269137
1, 0.969068202691371, 0.969068202691371, 0.969068202691371, 0.96938366010462
01, 0.9693836601046201, 0.9693836601046201, 0.9693836601046201, 0.9693836601
046201, 0.969068202691371, 0.969068202691371, 0.969068202691371, 0.969068202
691371, 0.9693836601046201, 0.9693836601046201, 0.9693836601046201, 0.969383
6601046201]
```

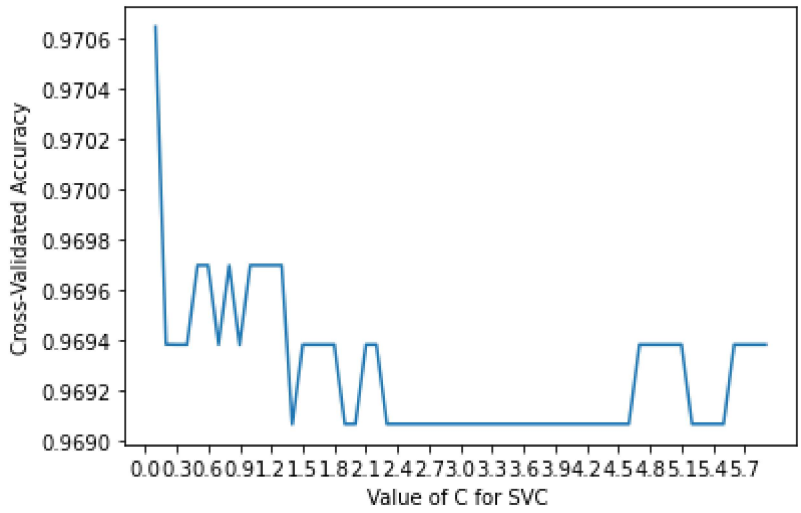
In [34]:

```
import matplotlib.pyplot as plt
%matplotlib inline

C_values=list(np.arange(0.1,6,0.1))
# plot the value of C for SVM (x-axis) versus the cross-validated accuracy (y-axis)
plt.plot(C_values,acc_score)
plt.xticks(np.arange(0.0,6,0.3))
plt.xlabel('Value of C for SVC ')
plt.ylabel('Cross-Validated Accuracy')
```

Out[34]:

```
Text(0, 0.5, 'Cross-Validated Accuracy')
```



Accuracy score is highest for C=0.1.

Taking kernel as rbf and taking different values gamma

Technically, the gamma parameter is the inverse of the standard deviation of the RBF kernel (Gaussian function), which is used as similarity measure between two points. Intuitively, a small gamma value define a Gaussian function with a large variance. In this case, two points can be considered similar even if are far from each other. In the other hand, a large gamma value means define a Gaussian function with a small variance and in this case, two points are considered similar just if they are close to each other

In [35]:

```
gamma_range=[0.0001,0.001,0.01,0.1,1,10,100]
acc_score=[]
for g in gamma_range:
    svc = SVC(kernel='rbf', gamma=g)
    scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy')
    acc_score.append(scores.mean())
print(acc_score)
```

```
[0.888240226809887, 0.9551820868106857, 0.9681168390368565, 0.96368745757297
44, 0.9061883560276325, 0.6016421754582119, 0.49905362776025236]
```

In [36]:

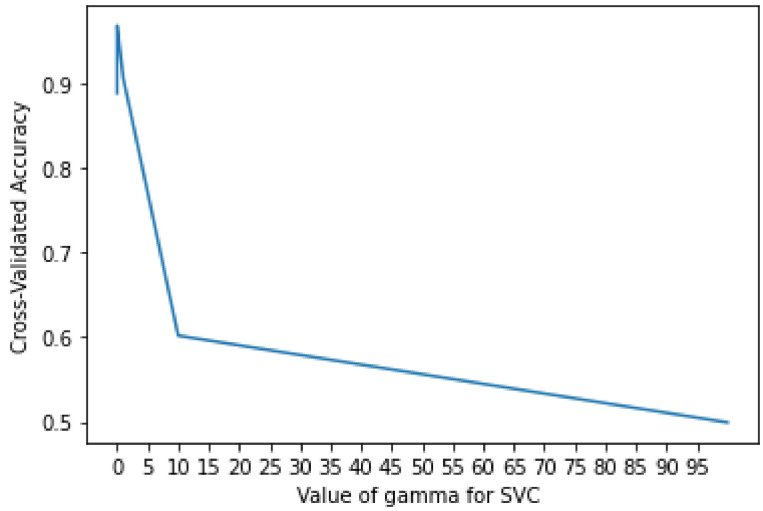
```
import matplotlib.pyplot as plt
%matplotlib inline

gamma_range=[0.0001,0.001,0.01,0.1,1,10,100]

# plot the value of C for SVM (x-axis) versus the cross-validated accuracy (y-axis)
plt.plot(gamma_range,acc_score)
plt.xlabel('Value of gamma for SVC ')
plt.xticks(np.arange(0.0001,100,5))
plt.ylabel('Cross-Validated Accuracy')
```

Out[36]:

```
Text(0, 0.5, 'Cross-Validated Accuracy')
```



We can see that for gamma=10 and 100 the kernel is performing poorly.We can also see a slight dip in accuracy score when gamma is 1.Let us look into more details for the range 0.0001 to 0.1.

In [37]:

```
gamma_range=[0.0001,0.001,0.01,0.1]
acc_score=[]
for g in gamma_range:
    svc = SVC(kernel='rbf', gamma=g)
    scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy')
    acc_score.append(scores.mean())
print(acc_score)
```

```
[0.888240226809887, 0.9551820868106857, 0.9681168390368565, 0.96368745757297
44]
```

In [38]:

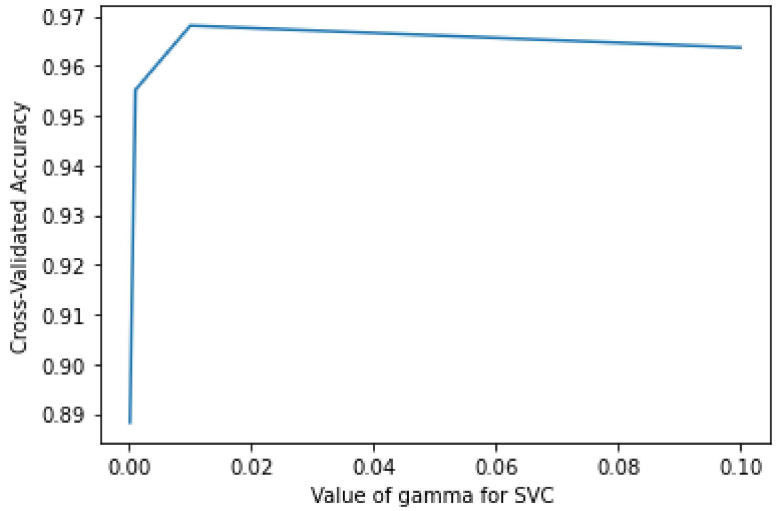
```
import matplotlib.pyplot as plt
%matplotlib inline

gamma_range=[0.0001,0.001,0.01,0.1]

# plot the value of C for SVM (x-axis) versus the cross-validated accuracy (y-axis)
plt.plot(gamma_range,acc_score)
plt.xlabel('Value of gamma for SVC ')
plt.ylabel('Cross-Validated Accuracy')
```

Out[38]:

```
Text(0, 0.5, 'Cross-Validated Accuracy')
```



The score increases steadily and raches its peak at 0.01 and then decreases till gamma=1.Thus Gamma should be around 0.01.

Let us look into more detail for gamma values

In [39]:

```
gamma_range=[0.01,0.02,0.03,0.04,0.05]
acc_score=[]
for g in gamma_range:
    svc = SVC(kernel='rbf', gamma=g)
    scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy')
    acc_score.append(scores.mean())
print(acc_score)
```

```
[0.9681168390368565, 0.9681168390368565, 0.9681148424709501, 0.9671664736652
957, 0.9665325639899376]
```

In [40]:

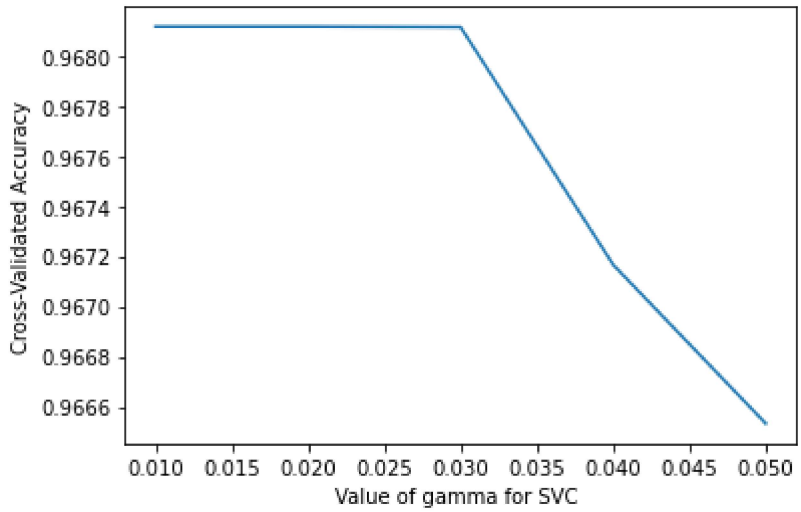
```
import matplotlib.pyplot as plt
%matplotlib inline

gamma_range=[0.01,0.02,0.03,0.04,0.05]

# plot the value of C for SVM (x-axis) versus the cross-validated accuracy (y-axis)
plt.plot(gamma_range,acc_score)
plt.xlabel('Value of gamma for SVC ')
plt.ylabel('Cross-Validated Accuracy')
```

Out[40]:

```
Text(0, 0.5, 'Cross-Validated Accuracy')
```



We can see there is constant decrease in the accuracy score as gamma value increase.Thus gamma=0.01 is the best parameter.

Taking polynomial kernel with different degree

In [41]:

```
degree=[2,3,4,5,6]
acc_score=[]
for d in degree:
    svc = SVC(kernel='poly', degree=d)
    scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy')
    acc_score.append(scores.mean())
print(acc_score)
```

```
[0.8515842750469194, 0.9450654873617378, 0.8313989937307829, 0.8661622010142
555, 0.7736463283152977]
```

In [42]:

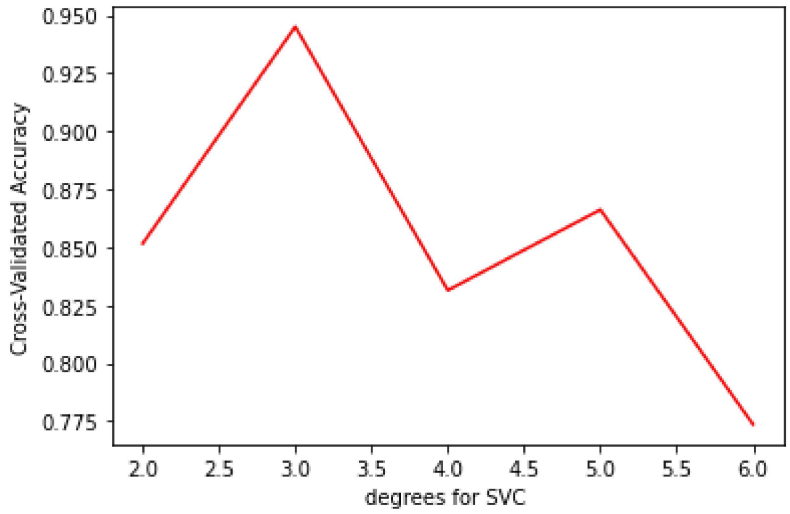
```
import matplotlib.pyplot as plt
%matplotlib inline

degree=[2,3,4,5,6]

# plot the value of C for SVM (x-axis) versus the cross-validated accuracy (y-axis)
plt.plot(degree,acc_score,color='r')
plt.xlabel('degrees for SVC ')
plt.ylabel('Cross-Validated Accuracy')
```

Out[42]:

```
Text(0, 0.5, 'Cross-Validated Accuracy')
```



Score is high for third degree polynomial and then there is drop in the accuracy score as degree of polynomial increases.Thus increase in polynomial degree results in high complexity of the model and thus causes overfitting.

Now performing SVM by taking hyperparameter C=0.1 and kernel as linear

In [43]:

```
from sklearn.svm import SVC
svc= SVC(kernel='linear',C=0.1)
svc.fit(X_train,y_train)
y_predict=svc.predict(X_test)
accuracy_score= metrics.accuracy_score(y_test,y_predict)
print(accuracy_score)
```

```
0.9747634069400631
```

With K-fold cross validation(where K=10)

In [45]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
svc=SVC(kernel='linear',C=0.1)
scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy')
print(scores)
```

```
[0.90851735 0.97160883 0.97476341 0.97791798 0.95899054 0.99053628
 0.99369085 0.97791798 0.95886076 0.99367089]
```

Taking the mean of all the scores

In [46]:

```
print(scores.mean())
```

```
0.9706474863235236
```

The accuracy is slightly good without K-fold cross validation but it may fail to generalise the unseen data.Hence it is advisable to perform K-fold cross validation where all the data is covered so it may predict unseen data well.

Now performing SVM by taking hyperparameter gamma=0.01 and kernel as rbf

In [47]:

```
from sklearn.svm import SVC
svc= SVC(kernel='rbf',gamma=0.01)
svc.fit(X_train,y_train)
y_predict=svc.predict(X_test)
metrics.accuracy_score(y_test,y_predict)
```

Out[47]:

```
0.9668769716088328
```

With K-fold cross validation(where K=10)

In [48]:

```
svc=SVC(kernel='linear',gamma=0.01)
scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
```

```
[0.91167192 0.97160883 0.97160883 0.97791798 0.95899054 0.9873817
 0.99369085 0.97791798 0.95253165 0.99367089]
0.9696991175178692
```

Now performing SVM by taking hyperparameter degree=3 and kernel as poly

In [49]:

```
from sklearn.svm import SVC
svc= SVC(kernel='poly',degree=3)
svc.fit(X_train,y_train)
y_predict=svc.predict(X_test)
accuracy_score= metrics.accuracy_score(y_test,y_predict)
print(accuracy_score)
```

```
0.9589905362776026
```

With K-fold cross validation(where K=10)

In [50]:

```
svc=SVC(kernel='poly',degree=3)
scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy')
print(scores)
print(scores.mean())
```

```
[0.89274448 0.94952681 0.93059937 0.92744479 0.94952681 0.99369085
 0.98422713 0.96529968 0.87974684 0.9778481 ]
0.9450654873617378
```

Let us perform Grid search technique to find the best parameter

In [51]:

```
from sklearn.svm import SVC
svm_model= SVC()
```

In [52]:

```
tuned_parameters = {
 'C': (np.arange(0.1,1,0.1)) , 'kernel': ['linear'],
 'C': (np.arange(0.1,1,0.1)) , 'gamma': [0.01,0.02,0.03,0.04,0.05], 'kernel': ['rbf'],
 'degree': [2,3,4] ,'gamma':[0.01,0.02,0.03,0.04,0.05], 'C':(np.arange(0.1,1,0.1)) , 'k
                    }
```

In [54]:

```
#from sklearn.grid_search import GridSearchCV
from sklearn.model_selection import GridSearchCV

model_svm = GridSearchCV(svm_model, tuned_parameters,cv=10,scoring='accuracy')
```

In [55]:

```
model_svm.fit(X_train, y_train)
print(model_svm.best_score_)
```

```
0.9569745728424264
```

In [61]:

```
print(model_svm.best_params_)
```

```
{'C': 0.9, 'degree': 3, 'gamma': 0.05, 'kernel': 'poly'}
```

In [62]:

```
y_pred= model_svm.predict(X_test)
print(metrics.accuracy_score(y_pred,y_test))
```

```
0.9589905362776026
```

In [ ]:

```

```