

# Project 7 Report

## Team members:

**Christopher Higgs**

**Supratik Chanda**

## Introduction:

### **Gender Recognition by Voice and Speech Analysis:**

This database was created to identify a voice as male or female, based upon acoustic properties of the voice and speech. The dataset consists of 3,168 recorded voice samples, collected from male and female speakers. The voice samples are pre-processed by acoustic analysis in R using the seewave and tuneR packages, with an analyzed frequency range of 0hz-280hz (human vocal range).

## Dataset:

The following acoustic properties of each voice are measured and included within the CSV:

**meanfreq: mean frequency (in kHz)**

**sd: standard deviation of frequency**

**median: median frequency (in kHz)**

**Q25: first quantile (in kHz)**

**Q75: third quantile (in kHz)**

**IQR: interquantile range (in kHz)**

**skew: skewness (see note in specprop description)**

**kurt: kurtosis (see note in specprop description)**

**sp.ent: spectral entropy**

**sfm: spectral flatness**

**mode: mode frequency**

**centroid: frequency centroid (see specprop)**

**peakf: peak frequency (frequency with highest energy)**

**meanfun: average of fundamental frequency measured across acoustic signal**

**minfun: minimum fundamental frequency measured across acoustic signal**

**maxfun: maximum fundamental frequency measured across acoustic signal**

**meandom: average of dominant frequency measured across acoustic signal**

**mindom: minimum of dominant frequency measured across acoustic signal**

**maxdom: maximum of dominant frequency measured across acoustic signal**

**dfrange: range of dominant frequency measured across acoustic signal**

**modindx: modulation index. Calculated as the accumulated absolute difference between adjacent measurements of fundamental frequencies divided by the frequency range**

**label: male or female**

## **Analysis Technique**

**Steps:**

1) At first , each X features column had dollar sign which was cleansed and the data type was changed from object to float64.

2) We then checked for any duplicate values in the dataset and removed any duplicate values

2)Then, we used SelectKBest features from feature\_selection package of scikit-learn to select which features should be taken. The SelectKBest features gives an array of score\_values in percentage and another array of p\_values . Whosoevers' score or p\_value is greater , it has a deeper effect on the Y\_label. So, we took column name "kurt"(p-0.00 and score-119.87) to "mindom"(p-0.00 and score-9.86). The score\_function are chi2 and f\_classif. We took chi2 and went ahead with the score func for finding the accuracy,precision and f1\_score and confusion\_matrix.

## Logistic Regression:

We divided the dataset into train\_set and test\_set and then performed training and testing of the data. At first, we used LabelEncoder for the Y\_Label but then we found out that without encoding also, the performance is not decreasing. Hence, we went without encoding for determining the Label.

After that, We used GridSearchCV and RandomizedSearchCV to find out which produces the most accuracy and has the least execution time. We found out that RandomizedSearchCV is the most efficient model\_selection technique and the execution time is the quickest. The following analysis is done using RandomizedSearchCV.

We used Multi-Class "OVR" and multi-class "multinomial" to find out which class is the best for generalization . Realized that One over Rest is the best approach.

OVR has the following results:

```
Precision_score: [0.976109 0.909091]
Recall Score: [0.902208 0.977918]
f_score [0.937705 0.942249]
The overall f1_score: 0.9399770790771836
Test_accuracy: 0.9810725552050473
```

Multinomial has the following results:

```
Test_accuracy: 0.5299684542586751
```

We also found out the precision, f\_score and confusion\_matrix (shown below) to find out the total number of incorrect predictions for OVR approach

```
The test dataset confusion_matrix is :
[[286 31]
 [ 7 310]]
```

## 7) The next model was SVM Linear:

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. From the above plot we can see that accuracy has been close to 97% for C=1, C=5 and C=6 and then it drops around 96.8% and remains constant.

C_values	Accuracy_values
0 1	0.9694
0 2	0.9688
0 3	0.9688
0 4	0.9688
0 5	0.9691
0 6	0.9694
0 7	0.9691
0 8	0.9688
0 9	0.9685

Once done, we used the C values with GridSearchCV and RandomizedSearchCV to find out the accuracy and f1\_score.

Surprisingly, we found out that RandomizedCV took less time than GridsearchCV. The time for GridSearchCV was 117ms and that of RandomizedSearchCV was 11ms.

We went ahead with our analysis with RandomizedSearchCV

```
grid best score: 0.968834727703236
grid best parameters: {'random_state': 45, 'kernel': 'linear', 'class_weight': {0: 1}, 'C': 6}
Execution time: 11.855389963226318 ms
```

We used predict\_proba function to find out what are the probabilities of the predicted values. We found out that there are four incorrect predictions. The figure and analysis is given in the results

## Next: SVM RBF

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**

We saw that for gamma=10 and 100 the kernel is performing poorly. We also saw a slight dip in accuracy score when gamma is 1.

Once done, we used the RandomizedSearchCV as the grid\_search model\_selection and passed gamma value=0.01 and a list of class weights inside the param\_grids as parameters. We found out that changing the class weight from {0:1} to {0:6} decreases the accuracy. Hence, stuck with class\_weight {0:1}

```
grid best parameters: {'random_state': 10, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 3}, 'C': 1.0}
grid best score: for class weight 4 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 4}, 'C': 1.0}
grid best score: for class weight 5 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 5}, 'C': 1.0}
grid best score: for class weight 6 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 6}, 'C': 1.0}
grid best score: for class weight 7 0.5189423835832676
```

The execution time was also very less and went on to find the precision, f\_score and accuracy for the model.

Also used the predict\_proba method to find out the probabilities of all predictions.

Used Confusion\_matrix to find the total number of incorrect predictions and also used boxplots to visually figure out the range of the probabilities for both the correct and the incorrect predictions

```
The confusion matrix is:
[[219  98]
 [ 70 247]]
```

## NEXT SVM POLY

Taking kernel as poly and different values of polynomial degree

We first used cross-validation to find out which polynomial degree is the best for attaining the best accuracy\_score

poly_values	Accuracy_values
2	0.8507
3	0.9458
4	0.8312
5	0.866
6	0.7748

Then, we used that degree, gamma and class weights as parameters in RandomizedSearchCV to find out the class weight for the best accuracy score.

```
grid best parameters: {'random_state': 10, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 3}, 'C': 1.0}
grid best score: for class weight 4 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 4}, 'C': 1.0}
grid best score: for class weight 5 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 5}, 'C': 1.0}
grid best score: for class weight 6 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 6}, 'C': 1.0}
grid best score: for class weight 7 0.5189423835832676
```

We also found the execution time for RandomizedSearchCV.

```
'Execution time: 28.46691608428955 ms
```

After that, we analysed the total number of incorrect predictions using confusion\_matrix

```
The test dataset confusion_matrix is :
[[314  3]
 [ 10 307]]
```

Also used the predict\_proba method to find out the probabilities of all predictions and applied boxplots to visually figure out the range of the probabilities for both the correct and the incorrect predictions

## Results:

The dataset Cleansing:

The Voice Recognition Dataset before cleansing:

Out[51]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	minfun	maxfun	meandom	mindom	maxdom	dfa
0	0.06	0.06	0.03	0.02	0.09	0.08	12.86	274.40	0.89	0.49	...	0.06	0.08	0.02	0.28	0.01	0.01	0.01	\$
1	0.07	0.07	0.04	0.02	0.09	0.07	22.42	634.61	0.89	0.51	...	0.07	0.11	0.02	0.25	0.01	0.01	0.05	\$
2	0.08	0.08	0.04	0.01	0.13	0.12	30.76	1,024.93	0.85	0.48	...	0.08	0.10	0.02	0.27	0.01	0.01	0.02	\$
3	0.15	0.07	0.16	0.10	0.21	0.11	1.23	4.18	0.96	0.73	...	0.15	0.09	0.02	0.25	0.20	0.01	0.56	\$
4	0.14	0.08	0.12	0.08	0.21	0.13	1.10	4.33	0.97	0.78	...	0.14	0.11	0.02	0.27	0.71	0.01	5.48	\$

5 rows x 21 columns

The dataset after cleansing 'S' symbol and converting the object datatype to float datatype

Out[53]:

	meanfreq	sd	median	Q25	Q75	IQR	skew
0	0.05978099999999994	0.0642413	0.03202690000000004	0.0150715	0.0901933999999999	0.0751220000000001	12.8634618
1	0.0660087	0.0673100000000001	0.0402287	0.0194139	0.0926661999999999	0.0732523	22.423285399999997
2	0.0773155	0.0838294	0.0367185	0.0087011000000002	0.131908	0.123207	30.75715460000003
3	0.1512281	0.0721106000000001	0.1580112000000002	0.0965817	0.2079552999999998	0.1113734999999999	1.2328313
4	0.1351204	0.0791461	0.1246562	0.0787202	0.2060449	0.1273247	1.1011737

5 rows x 21 columns

The feature\_selection technique used is SelectKBest; chosen among SelectKBest and SelectPercentile . The function\_score tested are chi2 and f\_classify and found out the chi2 gives more distinct p\_values and score

```
SelectKBest feature_extraction_text using f_classif:Value

pvalues_: [0.000000 0.000000 0.000000 0.000000 0.000164 0.000000 0.039263 0.000001
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.083031]

scores: [406.752821 945.461378 277.588158 1121.569223 14.236082 1965.749999
4.252980 24.255365 1003.308717 463.923194 96.257909 406.752821
7228.790362 60.282137 90.228036 119.959108 125.110999 126.024161
121.457858 3.006445]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]

SelectKBest feature_extraction_text using chi2:Value

pvalues_: [0.181666 0.060081 0.177517 0.000185 0.851080 0.000000 0.000001 0.000000
0.189369 0.000000 0.066400 0.181666 0.000060 0.440885 0.579711 0.000000
0.002470 0.000000 0.000000 0.619374]

scores: [1.783946 3.535149 1.818288 13.977196 0.035247 26.337745 24.329583
11987.597927 1.722517 31.246850 3.369843 1.783946 16.094352 0.593977
0.306703 38.460229 9.162780 297.819019 290.272018 0.246748]

SelectPercentile feature_extraction_text:Value

pvalues_: [0.181666 0.060081 0.177517 0.000185 0.851080 0.000000 0.000001 0.000000
0.189369 0.000000 0.066400 0.181666 0.000060 0.440885 0.579711 0.000000
0.002470 0.000000 0.000000 0.619374]

scores: [1.783946 3.535149 1.818288 13.977196 0.035247 26.337745 24.329583
11987.597927 1.722517 31.246850 3.369843 1.783946 16.094352 0.593977
0.306703 38.460229 9.162780 297.819019 290.272018 0.246748]
```

## Displayed as a dataframe

Out[56]:

	Columns	P_value	scores
7	kurt	\$0.00	\$11,987.60
17	maxdom	\$0.00	\$297.82
18	dfrange	\$0.00	\$290.27
15	meandom	\$0.00	\$38.46
9	sfm	\$0.00	\$31.25
5	IQR	\$0.00	\$26.34
6	skew	\$0.00	\$24.33
12	meanfun	\$0.00	\$16.09
3	Q25	\$0.00	\$13.98
16	mindom	\$0.00	\$9.16
1	sd	\$0.06	\$3.54
10	mode	\$0.07	\$3.37
2	median	\$0.18	\$1.82
0	meanfreq	\$0.18	\$1.78
11	centroid	\$0.18	\$1.78
8	sp.ent	\$0.19	\$1.72
13	minfun	\$0.44	\$0.59
14	maxfun	\$0.58	\$0.31
19	modindx	\$0.62	\$0.25
4	Q75	\$0.85	\$0.04

Took features from kurt to mindom whose p\_value was significant

## Logistic Regression

Grid SearchCV results and execution time(multi-class=OVR):

---

For GridSearchCV:

grid best score for train\_set: 0.968034727703236

grid best parameters for train\_set: {'C': 1.0, 'multi\_class': 'ovr', 'penalty': 'l1', 'random\_state': 5}

Execution time: 6.657193660736084 ms

Test accuracy:

Test\_accuracy: 0.9810725552050473

**Randomized SearchCV results and execution time(multi-class=OVR):**

For RandomizedSearchCV:

grid best score for train\_set: 0.968034727703236

grid best parameters for train\_set: {'random\_state': 17, 'penalty': 'l1', 'multi\_class': 'ovr', 'C': 1.0}

Execution time: 1.7632555961608887 ms

**Test accuracy:**

---

Test\_accuracy: 0.9810725552050473

Both accuracy is exactly the same . The only difference is the execution time. Randomized SearchCV has much lesser execution time. So. we went with RandomizedCV and below are the results

**The overall scores:**

---

```
Precision_score: [0.976109 0.909091]
Recall Score: [0.902208 0.977918]
f_score [0.937705 0.942249]
The overall f1_score: 0.9399770790771836
```

**The dataframe containing values with incorrect predictions and their probability.**

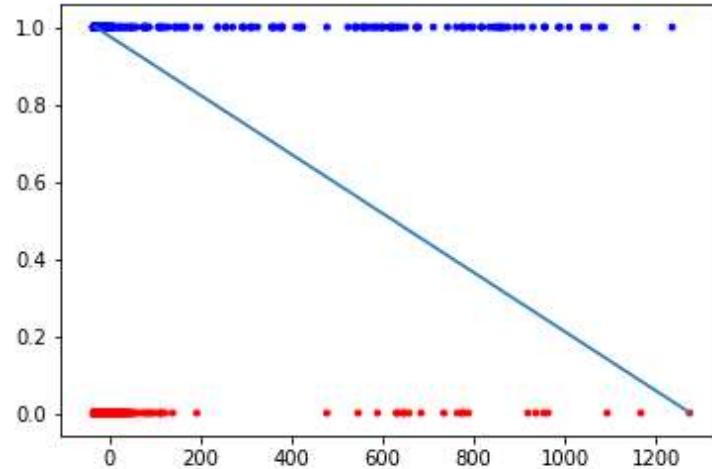
**0 = female,1=male**

Out[223]:

	Probability	Actual	Predicted	SameOrNot
10	\$0.55	0	1	1
17	\$0.73	0	1	1
20	\$0.81	0	1	1
35	\$0.63	0	1	1
57	\$0.74	0	1	1
90	\$0.74	0	1	1
136	\$0.63	0	1	1
162	\$0.51	1	0	1
249	\$0.71	0	1	1
261	\$0.80	0	1	1
268	\$0.59	0	1	1
269	\$0.71	0	1	1

**The decision boundary separating the predicted values**

Out[23]: &lt;matplotlib.collections.PathCollection at 0x2f2effebd68&gt;

**Now, Lets see the Score for multi-class=multinomial using RandomizedCV:**

```
grid best score: 0.5240726124704025
grid best parameters: {'solver': 'sag', 'random_state': 10, 'penalty': 'l2', 'multi_class': 'multinomial', 'C': 1.0}
```

**Test\_accuracy:** 0.5299684542586751

Hence, we see that OVR is the best approach for Logistic Regression

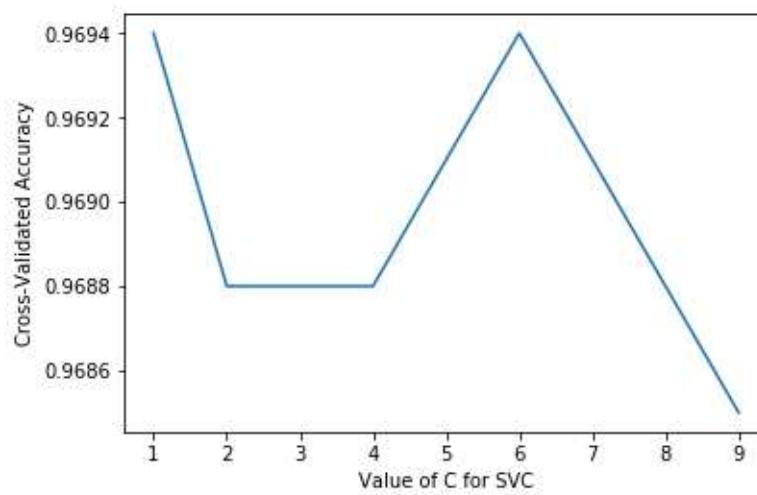
## SVM KERNEL=LINEAR

Analysing the accuracy score with different values of C

Out[64]:

C_values	Accuracy_values
0 1	0.9694
0 2	0.9688
0 3	0.9688
0 4	0.9688
0 5	0.9691
0 6	0.9694
0 7	0.9691
0 8	0.9688
0 9	0.9685

Out[73]: Text(0,0.5,'Cross-Validated Accuracy')



Going ahead with the C-Value which produces the most accuracy.Then feeding the C-value in GridSearchCV and RandomizedSearchCV:

**Scores,execution time and test\_accuracy for GridSearchCV:**

```
grid best score: 0.9696132596685083
grid best parameters: {'C': 6, 'class_weight': {0: 2}, 'kernel': 'linear', 'random_state': 5}
Execution time: 36.25111722946167 ms
```

Test\_accuracy: 0.9810725552050473

Scores, execution time and test\_accuracy for RandomizedSearchCV:

```
grid best score: 0.968034727703236
grid best parameters: {'random_state': 45, 'kernel': 'linear', 'class_weight': {0: 1}, 'C': 6}
Execution time: 11.855309963226318 ms
```

Test\_accuracy: 0.9779179810725552

The accuracy is a bit lower but the execution time is much lesser and hence, we are using RandomizedSearchCV.

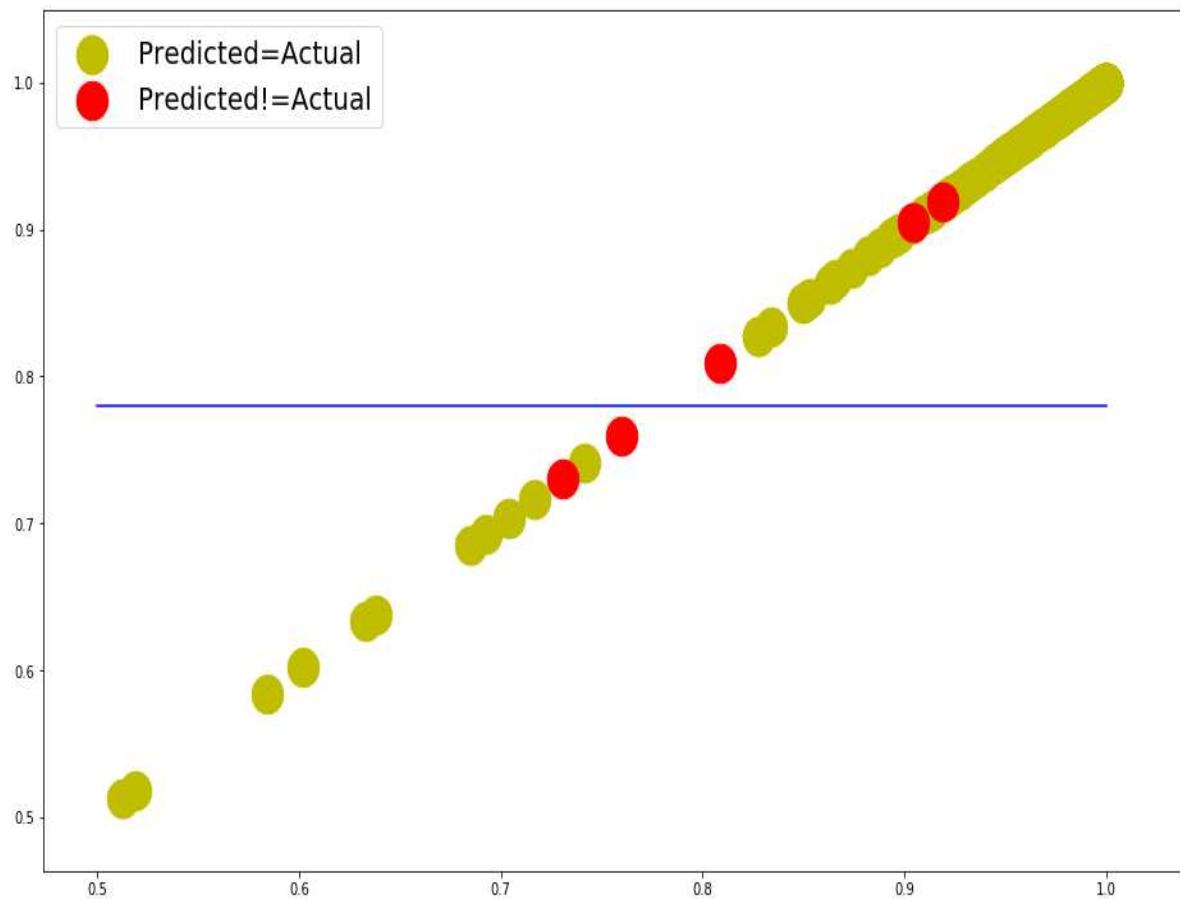
```
Precision_score: [0.984026 0.971963]
Recall Score: [0.971609 0.984227]
f_score [0.977778 0.978056]
The overall f1_score: 0.9780564263322884
```

Then we found out the shape of the dataframe containing the incorrect values and found out the total incorrect values to be 8 .

The linechart of the predict\_probabilities is shown below :

X and Y axis are the probabilities of the predicted values

&lt;matplotlib.legend.Legend at 0x282c4c8f4a8&gt;



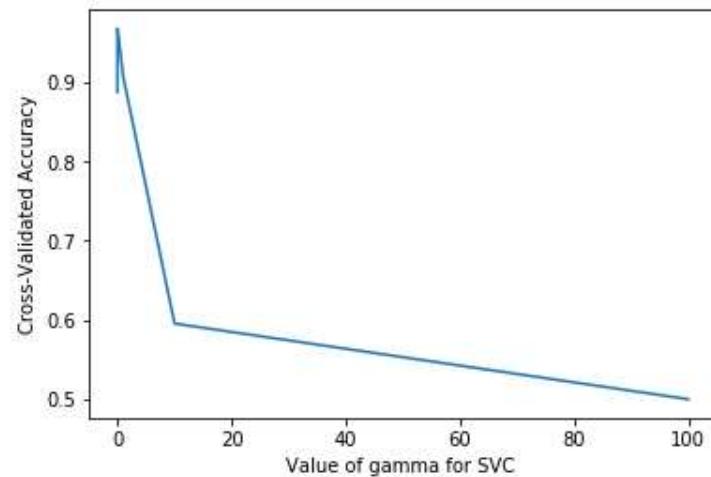
## SVM KERNEL=RBF

Analyzing the accuracy score with different values of gamma

Out[154]:

	gamma_values	Accuracy_values
0	\$0.00	0.8878
0	\$0.00	0.9552
0	\$0.01	0.9682
0	\$0.10	0.9631
0	\$1.00	0.9072
0	\$10.00	0.5955
0	\$100.00	0.5

Out[137]: Text(0,0.5,'Cross-Validated Accuracy')



We found out that gamma value 0.01 gives the highest accuracy.

Hence, we give the gamma value along with other values as parameter to RandomizedSearchCV and see the results:

```
grid best score: for class weight 0 0.5
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 0}, 'C': 1.0}
grid best score: for class weight 1 0.7052091554853985
grid best parameters: {'random_state': 13, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight 2 0.6606156274664562
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 2}, 'C': 1.0}
grid best score: for class weight 3 0.5868192580899764
grid best parameters: {'random_state': 10, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 3}, 'C': 1.0}
grid best score: for class weight 4 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 4}, 'C': 1.0}
grid best score: for class weight 5 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 5}, 'C': 1.0}
grid best score: for class weight 6 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 6}, 'C': 1.0}
grid best score: for class weight 7 0.5189423835832676
grid best parameters: {'random_state': 10, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 7}, 'C': 1.0}
grid best score: for class weight 8 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 8}, 'C': 1.0}
grid best score: for class weight 9 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 9}, 'C': 1.0}
```

We see that calss weight{0:1} gives the best prediction

Then we go ahead with class weight =0:1 and use RandomizedSearchCV:

```
grid best score: for class weight 0.7154696132596685
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'class_weight': {0: 1}, 'C': 1.0}
Execution time: 72.78458512886847 ms
```

---

**Test Accuracy:**

```
Test_accuracy: 0.7350157728706624
```

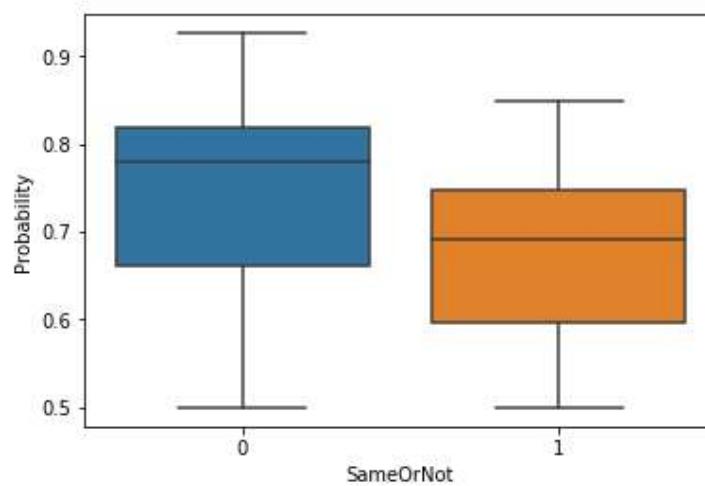
**other results:**

```
Precision_score: [0.757785 0.715942]
Recall Score: [0.690852 0.779180]
f_score [0.722772 0.746224]
The overall f1_score: 0.7462235649546827
```

Then, we used the predict\_proba method of RandomizedSearchCV to find out the probability of each predicted value.

**Boxplot showing the probability range of the incorrect and correct predicted values**

```
Out[167]: <matplotlib.axes._subplots.AxesSubplot at 0x282c4ccbf98>
```



**1 implies Actual values are not equal to Predicted Values**

**0 implies Actual values equal to Predicted Values**

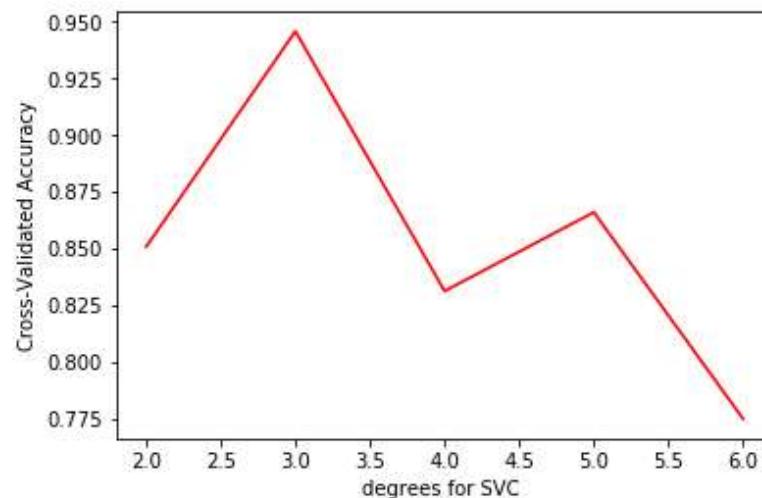
Here we see that the predicted values which are correct has a greater probability

## SVM KERNEL=POLY

Analyzing the accuracy score with different values of polynomial degree

poly_values	Accuracy_values
0	2
0	3
0	4
0	5
0	6

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



We found out that degree value 3 gives the highest accuracy.

Hence, we give the degree value along with other values as parameter to RandomizedSearchCV and see the results:

```

grid best score: for class weight 0.9494869771112865
grid best parameters: {'random_state': 5, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight 0.9526440410418311
grid best parameters: {'random_state': 45, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight 0.9522494080505131
grid best parameters: {'random_state': 45, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight 0.9526440410418311
grid best parameters: {'random_state': 45, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight 0.9534333070244673
grid best parameters: {'random_state': 5, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight 0.9538279400157853
grid best parameters: {'random_state': 13, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight 0.9538279400157853
grid best parameters: {'random_state': 45, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}

```

**Since class\_weight {0:1} has the greatest grid\_score, we go ahead with class weight =0:1 and use RandomizedSearchCV:**

---

```

grid best score: for class weight 0.9696132596685083
grid best parameters: {'random_state': 10, 'kernel': 'poly', 'degree': 1, 'class_weight': {0: 1}, 'C': 1.0}
Execution time: 23.117100954055786 ms

```

**Test Accuracy:**

---

Test\_accuracy : 0.9747634069400631

**The confusion matrix result as shown below:**

---

```

The test dataset confusion_matrix is :
[[314  3]
 [ 10 307]]

```

---

**The incorrect values total is 13.**

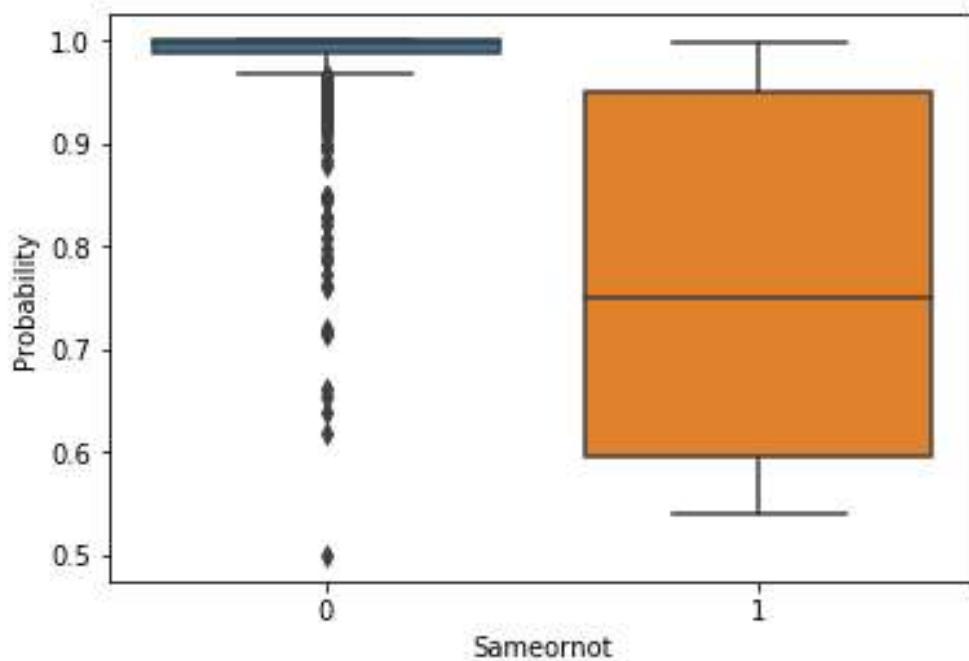
**The next dataframe is the dataframe of incorrect predicted values and their predicted probabilities of being the incorrect one**

	Probability	Actual	Predicted	Sameornot
10	\$0.87	1	0	1
58	\$0.95	1	0	1
59	\$0.73	1	0	1
185	\$0.60	0	1	1
189	\$0.65	0	1	1
253	\$0.99	1	0	1
256	\$0.59	1	0	1
267	\$0.54	1	0	1
296	\$0.58	0	1	1
487	\$0.95	1	0	1
555	\$0.75	1	0	1
578	\$1.00	1	0	1
588	\$0.88	1	0	1

Then, we used the `predict_proba` method of `RandomizedSearchCV` to find out the probability of each predicted value.

Boxplot showing the probability range of the incorrect and correct predicted values

```
<matplotlib.axes._subplots.AxesSubplot at 0x282c4ee8e48>
```



1 implies Actual values are not equal to Predicted Values

**0 implies Actual values equal to Predicted Values**

Here we see see that the predicted values which are correct has a greater probability

In [199]:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from IPython.display import display
6 from sklearn.model_selection import train_test_split
```

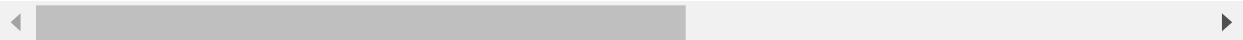
In [200]:

```
1 df =pd.read_csv('VoiceRecognition.csv')
2 df.head()
```

Out[200]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	me...
0	\$0.06	\$0.06	\$0.03	\$0.02	\$0.09	\$0.08	\$12.86	\$274.40	\$0.89	\$0.49	...	\$0.06	
1	\$0.07	\$0.07	\$0.04	\$0.02	\$0.09	\$0.07	\$22.42	\$634.61	\$0.89	\$0.51	...	\$0.07	
2	\$0.08	\$0.08	\$0.04	\$0.01	\$0.13	\$0.12	\$30.76	\$1,024.93	\$0.85	\$0.48	...	\$0.08	
3	\$0.15	\$0.07	\$0.16	\$0.10	\$0.21	\$0.11	\$1.23	\$4.18	\$0.96	\$0.73	...	\$0.15	
4	\$0.14	\$0.08	\$0.12	\$0.08	\$0.21	\$0.13	\$1.10	\$4.33	\$0.97	\$0.78	...	\$0.14	

5 rows × 21 columns



In [201]:

```
1 df.columns.values
```

Out[201]:

```
array(['meanfreq', 'sd', 'median', 'Q25', 'Q75', 'IQR', 'skew', 'kurt',
       'sp.ent', 'sfm', 'mode', 'centroid', 'meanfun', 'minfun', 'maxfun',
       'meandom', 'mindom', 'maxdom', 'dfrange', 'modindx', 'label'],
      dtype=object)
```

In [202]:

```
1 df[df.duplicated(keep='first')]
```

Out[202]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	me...
298	\$0.21	\$0.06	\$0.24	\$0.14	\$0.26	\$0.12	\$2.11	\$7.89	\$0.86	\$0.08	...	\$0.21	
2403	\$0.21	\$0.04	\$0.22	\$0.19	\$0.25	\$0.06	\$1.86	\$6.11	\$0.88	\$0.31	...	\$0.21	

2 rows × 21 columns



In [203]:

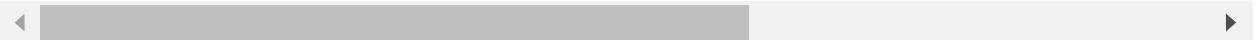
```
1 df.drop_duplicates(inplace=True)
```

In [204]: 1 df[df.duplicated(keep='first')]

Out[204]:

meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	minfun
----------	----	--------	-----	-----	-----	------	------	--------	-----	-----	----------	---------	--------

0 rows × 21 columns



In [205]: 1 df.dtypes

Out[205]:

meanfreq	float64
sd	float64
median	float64
Q25	float64
Q75	float64
IQR	float64
skew	float64
kurt	float64
sp.ent	float64
sfm	float64
mode	float64
centroid	float64
meanfun	float64
minfun	float64
maxfun	float64
meandom	float64
mindom	float64
maxdom	float64
dfrange	float64
modindx	float64
label	object
dtype:	object

In [206]:

```
1 # Removing the $ symbol from every column
2 df1 = df.iloc[:, :-1].astype('str').replace('$', '', regex=True)
```

In [207]:

```
1 df1 = df1.join(df['label'])
2 df1.head()
```

Out[207]:

	meanfreq	sd	median	Q25
0	0.05978099999999994	0.0642413	0.032026900000000004	0.0150715 0.090
1	0.0660087	0.0673100000000001	0.0402287	0.0194139 0.092
2	0.0773155	0.0838294	0.0367185	0.008701100000000002
3	0.1512281	0.07211060000000001	0.15801120000000002	0.0965817 0.207
4	0.1351204	0.0791461	0.1246562	0.0787202

5 rows × 21 columns



In [208]: 1 df1.dtypes

Out[208]:

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

In [209]: 1 df.shape

Out[209]: (3166, 21)

In [210]:

```

1 np.set_printoptions(formatter={'float_kind':'{:3f}'.format})
2 from sklearn.feature_selection import SelectKBest,chi2,SelectPercentile,f_classif
3 selValue1 = SelectKBest(f_classif,k=20).fit(df.iloc[:, :-1],df.iloc[:, :-1])
4 print('\nSelectKBest feature_extraction_text using f_classif:Value\n')
5 print('pvalues_:',selValue1.pvalues_)
6 print('nscores_:',selValue1.scores_)
7 print(selValue1.get_support(indices=True))
8 selValue2 = SelectKBest(chi2,k=20).fit(df.iloc[:, :-1],df.iloc[:, :-1])
9 print('\nSelectKBest feature_extraction_text using chi2:Value\n')
10 print('pvalues_:',selValue2.pvalues_)
11 print('nscores_:',selValue2.scores_)
12 print('\nSelectPercentile feature_extraction_text:Value\n')
13 # Used SelectPercentile feature_extraction_text
14 selValue3 = SelectPercentile(chi2,percentile=100).fit(df.iloc[:, :-1],df.iloc[:, :-1])
15 print('pvalues_:',selValue3.pvalues_)
16 print('nscores_:',selValue3.scores_)

```

SelectKBest feature\_extraction\_text using f\_classif:Value

pvalues\_:[ 0.000000 0.000000 0.000000 0.000000 0.000169 0.000000 0.039359 0.000001  
 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000  
 0.000000 0.000000 0.085898]

scores:[ 407.163482 944.502049 278.408026 1120.476092 14.178175 1962.973919  
 4.248838 24.253878 1004.510839 465.788494 96.583315 407.163482  
 7236.389947 60.013861 89.950060 119.266417 125.918148 125.028709  
 120.465472 2.951460]  
[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

SelectKBest feature\_extraction\_text using chi2:Value

pvalues\_:[ 0.181482 0.060157 0.176985 0.000186 0.851390 0.000000 0.000001 0.000000  
 0.189120 0.000000 0.065900 0.181482 0.000060 0.441775 0.580188 0.000000  
 0.002390 0.000000 0.000000 0.622505]

scores:[ 1.785445 3.533062 1.822758 13.970832 0.035098 26.317371 24.314480  
 11987.900828 1.724459 31.327900 3.382309 1.785445 16.101725 0.591669  
 0.305931 38.232633 9.222764 295.068090 287.506888 0.242363]

SelectPercentile feature\_extraction\_text:Value

pvalues\_:[ 0.181482 0.060157 0.176985 0.000186 0.851390 0.000000 0.000001 0.000000  
 0.189120 0.000000 0.065900 0.181482 0.000060 0.441775 0.580188 0.000000  
 0.002390 0.000000 0.000000 0.622505]

scores:[ 1.785445 3.533062 1.822758 13.970832 0.035098 26.317371 24.314480  
 11987.900828 1.724459 31.327900 3.382309 1.785445 16.101725 0.591669  
 0.305931 38.232633 9.222764 295.068090 287.506888 0.242363]

In [211]:

```
1 dt = pd.DataFrame({'Columns':df.columns.values[:-1], 'P_value':pd.Series(selVa
```

In [212]:

```

1 pd.options.display.float_format = '${:,.2f}'.format
2 dt.sort_values(by='scores', ascending=False, inplace=True)
3 dt

```

Out[212]:

	Columns	P_value	scores
7	kurt	\$0.00	\$11,987.90
17	maxdom	\$0.00	\$295.07
18	dfrange	\$0.00	\$287.51
15	meandom	\$0.00	\$38.23
9	sfm	\$0.00	\$31.33
5	IQR	\$0.00	\$26.32
6	skew	\$0.00	\$24.31
12	meanfun	\$0.00	\$16.10
3	Q25	\$0.00	\$13.97
16	mindom	\$0.00	\$9.22
1	sd	\$0.06	\$3.53
10	mode	\$0.07	\$3.38
2	median	\$0.18	\$1.82
0	meanfreq	\$0.18	\$1.79
11	centroid	\$0.18	\$1.79
8	sp.ent	\$0.19	\$1.72
13	minfun	\$0.44	\$0.59
14	maxfun	\$0.58	\$0.31
19	modindx	\$0.62	\$0.24
4	Q75	\$0.85	\$0.04

In [213]:

```

1 dfnew=pd.DataFrame()
2 for each in dt.Columns.values:
3     dfnew = pd.concat([dfnew,pd.DataFrame(df1[each])],axis=1)
4 dfnew = dfnew.join(pd.DataFrame(df['label']))
5 dfnew = dfnew.drop(dfnew.loc[:, 'sd':'Q75'].columns, axis=1)
6 dfnew.head()

```

Out[213]:

	kurt	maxdom	dfrange	meandom	sfm
0	274.40290550000003	0.0078125	0.0	0.0078125	0.49191779999999996 0.075122
1	634.6138545	0.0546875	0.046875	0.00901439999999999	0.5137238000000001
2	1024.927705	0.015625	0.0078125	0.00799010000000002	0.478905
3	4.1772962	0.5625	0.5546875	0.2014974	0.7272318 0.111373
4	4.333713200000001	5.484375	5.4765625	0.7128125	0.7835681

In [214]: 1 dfnew.columns.values

Out[214]: array(['kurt', 'maxdom', 'dfrange', 'meandom', 'sfm', 'IQR', 'skew', 'meanfun', 'Q25', 'mindom', 'label'], dtype=object)

## Logistics Regression: GridSearchCV

```
In [215]: 1 from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.preprocessing import LabelEncoder,StandardScaler
4 import time
5 dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
6 train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stratified=True)
7 #display(test_set.loc[:, 'kurt':'sd'])
8 lm=LogisticRegression()
9 start_time=time.time()
10 params = dict(penalty=['l1','l2'],C=[0.2,0.5,0.7,1.0],random_state=[5,10,45])
11 grid = GridSearchCV(lm,params, cv=5).fit(train_set.loc[:, 'kurt':'mindom'],train_set['label'])
12 print('For GridSearchCV:')
13 print('grid best score for train_set: ',grid.best_score_)
14 print('grid best parameters for train_set: ',grid.best_params_)
15 print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

For GridSearchCV:

```
grid best score for train_set:  0.9676145339652449
grid best parameters for train_set:  {'C': 1.0, 'multi_class': 'ovr', 'penalty': 'l1', 'random_state': 5}
Execution time: 13.599502086639404 ms
```

In [216]: 1 print('Test\_accuracy: ',grid.score(test\_set.loc[:, 'kurt':'mindom'],test\_set['label']))

Test\_accuracy: 0.9826498422712934

## Logistic Regression: RandomizedSearchCV

In [217]:

```

1 from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
2 from sklearn.linear_model import LogisticRegression
3 from sklearn.preprocessing import LabelEncoder,StandardScaler
4 import time
5 #dfnew['Label'] = (LabelEncoder().fit_transform(dfnew['Label']))
6 train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stratified=True)
7 #display(test_set.loc[:, 'kurt':'sd'])
8 lm=LogisticRegression()
9 start_time=time.time()
10 params = dict(penalty=['l1','l2'],C=[0.2,0.5,0.7,1.0],random_state=[5,10,45])
11 grid = RandomizedSearchCV(lm,params, cv=5).fit(train_set.loc[:, 'kurt':'mindom'])
12 print('For RandomizedSearchCV:')
13 print('grid best score for train_set: ',grid.best_score_)
14 print('grid best parameters for train_set: ',grid.best_params_)
15 print("Execution time: " + str((time.time() - start_time)) + ' ms')

```

For RandomizedSearchCV:

```

grid best score for train_set:  0.9640600315955766
grid best parameters for train_set:  {'random_state': 45, 'penalty': 'l1', 'multi_class': 'ovr', 'C': 0.5}
Execution time: 3.2223386764526367 ms

```

In [218]:

```

1 print('Test_accuracy: ',grid.score(test_set.loc[:, 'kurt':'mindom'],test_set['label']))

```

Test\_accuracy: 0.9810725552050473

In [219]:

```

1 from sklearn.metrics import confusion_matrix,precision_recall_fscore_support,
2 from sklearn.preprocessing import LabelEncoder
3 #dfnew['Label'] = (LabelEncoder().fit_transform(dfnew['Label']))
4 #display(test_set.loc[:, 'kurt':'sd'])
5 lm=LogisticRegression()
6 lm.fit(train_set.loc[:, 'kurt':'mindom'],train_set['label'])
7 y_pred = lm.predict(test_set.loc[:, 'kurt':'mindom'])
8 print('The test dataset confusion_matrix is :\n',confusion_matrix(test_set['label'],y_pred))

```

The test dataset confusion\_matrix is :

```

[[287  30]
 [ 8 309]]

```

In [220]:

```

1 p,r,f,s = precision_recall_fscore_support(test_set['label'], y_pred)
2 print('Precision_score: ',p,' \nRecall Score: ',r,' \nf_score',f,' \n The overall
3 .format(f1_score(test_set['label'],y_pred,average="weighted")))

```

```

Precision_score:  [0.972881 0.911504]
Recall Score: [0.905363 0.974763]
f_score [0.937908 0.942073]
The overall f1_score: 0.9399908337318668

```

In [224]:

```

1 Label_probabability = pd.Series()
2 for each in lm.predict_proba((test_set.loc[:, 'kurt':'mindom'])):
3     Label_probabability = Label_probabability.append(pd.Series(np.amax(each)))
4 Label_probabability_DTF = pd.concat([Label_probabability.reset_index(drop=True),
5                                     test_set['label'].reset_index(drop=True),
6                                     pd.Series(lm.predict(test_set.loc[:, keys=['Probability', 'Actual', 'Predicted']]))])
7 Label_probabability_DTF['SameOrNot'] = abs(Label_probabability_DTF['Actual'] - Label_probabability_DTF['Predicted']) > 0
8 Label_probabability_DTF[Label_probabability_DTF['SameOrNot'] != 0].head()
9

```

Out[224]:

	Probability	Actual	Predicted	SameOrNot
10	\$0.55	0	1	1
17	\$0.73	0	1	1
20	\$0.81	0	1	1
35	\$0.63	0	1	1
57	\$0.74	0	1	1

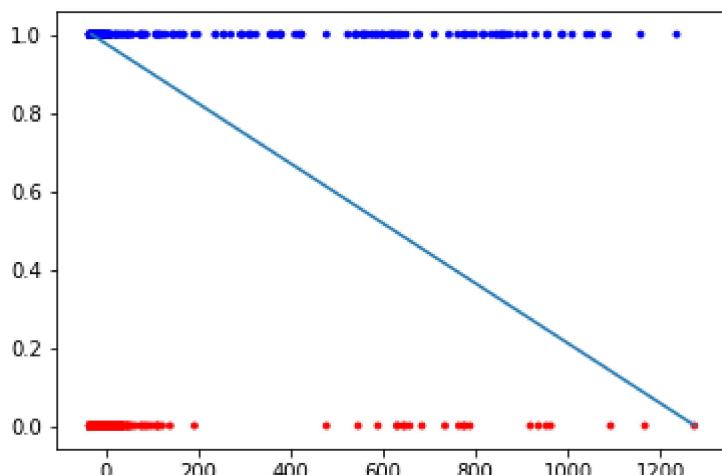
In [222]:

```

1 from sklearn.decomposition import PCA
2 from sklearn.preprocessing import MinMaxScaler
3 pca=PCA(n_components=1)
4 #scaler= StandardScaler()
5 dfnew['PCAtransformed'] = pca.fit_transform(dfnew.iloc[:, :-1])
6 #dfnew['PCAtransformed']= scaler.fit_transform(dfnew.loc[:, ['PCAtransformed']])
7 dfTransformed = dfnew.iloc[:, [-1, -2]]
8 dfTransformed = dfTransformed[['PCAtransformed', 'label']]
9 dfTransformed.sort_values(by='PCAtransformed', ascending=True, inplace=True)
10 #display(dfTransformed.tail(1))
11 X=[-30,1275]
12 Y=[1,0]
13 #Y = [-(lm.intercept_[0]+ lm.coef_[0][0]*elem)/lm.coef_[0][1] for elem in X]
14 plt.plot(X,Y)
15 color = ['r' if y_ == 0 else 'b' for y_ in dfTransformed['label']]
16 plt.scatter(dfTransformed['PCAtransformed'], dfTransformed['label'], c=color,

```

Out[222]: &lt;matplotlib.collections.PathCollection at 0x282c0593f28&gt;



In [ ]: 1

In [24]: 1 ## Using Multinomial parameter

```

In [25]: 1 ##### train_set,test_set = train_test_split(dfnew,random_state=3,test_size=0.2
2 #display(test_set.loc[:, 'kurt':'sd']))
3 import warnings
4 warnings.simplefilter('ignore')
5 lm=LogisticRegression()
6 params = dict(penalty=['l2'],C=[0.2,0.5,0.7,1.0],random_state=[5,10,45,24,17]
7             multi_class=['multinomial'])
8 grid = RandomizedSearchCV(lm,params, cv=5).fit(train_set.loc[:, 'kurt':'mindom'])
9 print('grid best score: ',grid.best_score_)
10 print('grid best paramters: ',grid.best_params_)
```

grid best score: 0.5240726124704025  
 grid best paramters: {'solver': 'sag', 'random\_state': 10, 'penalty': 'l2', 'multi\_class': 'multinomial', 'C': 1.0}

```

In [26]: 1 print('Test_accuracy: ',grid.score(test_set.loc[:, 'kurt':'mindom'],test_set['
```

Test\_accuracy: 0.5299684542586751

## SVM Linear

Type *Markdown* and *LaTeX*:  $\alpha^2$

## Using Cross Validation with Different values of C

In [64]:

```

1 from sklearn.cross_validation import cross_val_score
2 from sklearn.svm import SVC
3 from sklearn.preprocessing import StandardScaler
4 X=StandardScaler().fit_transform(df1.iloc[:, :-1])
5 C_List=pd.Series()
6 Acc_List=pd.Series()
7 DTFrame=pd.DataFrame()
8 for i in range(1,10):
9     svcCrossLinear = SVC(kernel='linear',C=i)
10    scores=cross_val_score(svcCrossLinear,X,df1.iloc[:, -1].values,scoring='ac
11    C_List = C_List.append(pd.Series(i))
12    Acc_List = Acc_List.append(pd.Series(round(scores.mean(),4)))
13 DTFrame= pd.concat([C_List,Acc_List],axis=1,keys=['C_values','Accuracy_values'
14 DTFrame['Accuracy_values'] = DTFrame['Accuracy_values'].apply(lambda x : str(
15 DTFrame

```

Out[64]:

	C_values	Accuracy_values
0	1	0.9694
0	2	0.9688
0	3	0.9688
0	4	0.9688
0	5	0.9691
0	6	0.9694
0	7	0.9691
0	8	0.9688
0	9	0.9685

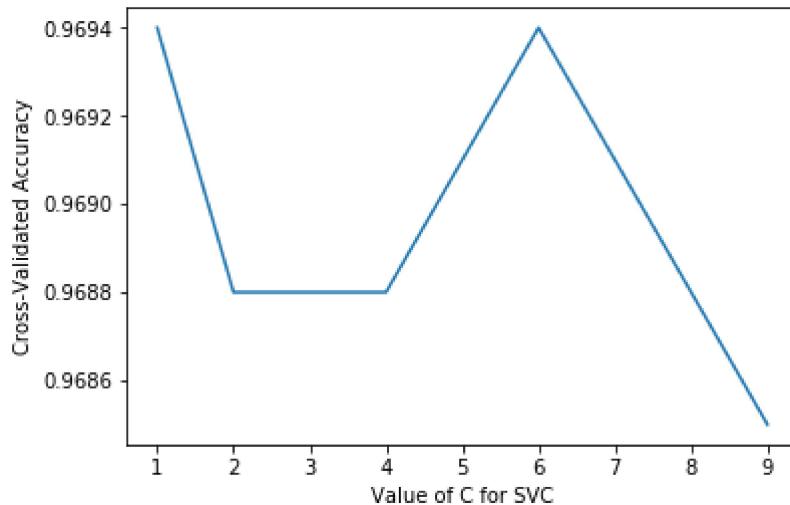
In [73]:

```

1 sns.lineplot(x=C_List,y=Acc_List)
2 plt.xlabel('Value of C for SVC')
3 plt.ylabel('Cross-Validated Accuracy')

```

Out[73]: Text(0,0.5, 'Cross-Validated Accuracy')



## Using GridSearchCV

In [152]:

```

1 from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_test_split
2 from sklearn.svm import SVC
3 from sklearn.preprocessing import LabelEncoder,StandardScaler
4 import time
5 dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
6 train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stratify=test_set.loc[:, 'kurt':'sd'])
7 #display(test_set.loc[:, 'kurt':'sd'])
8 train_set.loc[:, 'kurt':'mindom']=StandardScaler().fit_transform(train_set.loc[:, 'kurt':'mindom'])
9 test_set.loc[:, 'kurt':'mindom']=StandardScaler().fit_transform(test_set.loc[:, 'kurt':'mindom'])
10 svcLinearGrid = SVC(probability=True)
11 start_time=time.time()
12 params=dict(C=[6],kernel=['linear'],random_state=[5,10,13,45],class_weight=[{0:2},{1:1}])
13 grid=GridSearchCV(svcLinearGrid,params, cv=10).fit(train_set.loc[:, 'kurt':'mindom'])
14 print('grid best score: ',grid.best_score_)
15 print('grid best paramters: ',grid.best_params_)
16 print("Execution time: " + str((time.time() - start_time)) + ' ms')

```

C:\Users\supratik chanda\Documents\New folder (2)\lib\site-packages\pandas\core\indexing.py:543: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy> (<http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>)

```
    self.obj[item] = s
```

```
grid best score:  0.9696132596685083
grid best paramters:  {'C': 6, 'class_weight': {0: 2}, 'kernel': 'linear', 'random_state': 5}
Execution time: 36.25111722946167 ms
```

In [153]:

```

1 print('Test_accuracy: ',grid.score(test_set.loc[:, 'kurt':'mindom'],test_set['label']))

```

Test\_accuracy: 0.9810725552050473

## Using RandomizedSearchCV

In [228]:

```

1 from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
2 from sklearn.svm import SVC
3 from sklearn.preprocessing import LabelEncoder,StandardScaler
4 import time
5 dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
6 sc=StandardScaler()
7 train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stratify=dfnew['label'])
8 train_set.loc[:, 'kurt':'mindom'] =sc.fit_transform(train_set.loc[:, 'kurt':'mindom'])
9 test_set.loc[:, 'kurt':'mindom']=sc.transform(test_set.loc[:, 'kurt':'mindom'])
10 #display(test_set.loc[:, 'kurt':'sd'])
11 svcLinearRandom = SVC(probability=True)
12 start_time=time.time()
13 params=dict(C=[6],kernel=['linear'],random_state=[5,10,13,45],class_weight=[{0: 1}, {1: 6}])
14 grid=RandomizedSearchCV(svcLinearRandom,params, cv=5).fit(train_set.loc[:, 'kurt':'mindom'])
15 print('grid best score: ',grid.best_score_)
16 print('grid best parameters: ',grid.best_params_)
17 print("Execution time: " + str((time.time() - start_time)) + ' ms')

```

C:\Users\supratik chanda\Documents\New folder (2)\lib\site-packages\pandas\core\indexing.py:543: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy> (<http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>)

```

self.obj[item] = s

grid best score:  0.9691943127962085
grid best parameters:  {'random_state': 10, 'kernel': 'linear', 'class_weight': {0: 1}, 'C': 6}
Execution time: 25.42304277420044 ms

```

In [229]:

```

1 print('Test_accuracy: ',grid.score(test_set.loc[:, 'kurt':'mindom'],test_set['label']))

```

Test\_accuracy: 0.9779179810725552

In [230]:

```

1 y_pred=grid.predict(test_set.loc[:, 'kurt':'mindom'])
2 p,r,f,s = precision_recall_fscore_support(test_set['label'], y_pred)
3 print('Precision_score: ',p,' \nRecall Score:',r,' \nf_score',f, '\n The overall f1_score:',f1_score(test_set['label'],y_pred))
4 .format(f1_score(test_set['label'],y_pred)))

```

Precision\_score: [0.984026 0.971963]  
 Recall Score: [0.971609 0.984227]  
 f\_score [0.977778 0.978056]  
 The overall f1\_score: 0.9780564263322884

In [231]:

```

1 Label_probability = pd.Series()
2 svcLinearRandom.fit(train_set.loc[:, 'kurt':'mindom'], train_set['label'])
3 for each in svcLinearRandom.predict_proba((test_set.loc[:, 'kurt':'mindom'])):
4     Label_probability = Label_probability.append(pd.Series(np.amax(each)))
5 Label_probability_DTFrames = pd.concat([Label_probability.reset_index(drop=True),
6                                         test_set['label'].reset_index(drop=True)])
7                                         pd.Series(svcLinearRandom.predict(test_set), keys=['Probability', 'Actual', 'Predict'])
8
9 Label_probability_DTFrames['SameOrNot'] = abs(Label_probability_DTFrames.Actual - Label_probability_DTFrames.Predict)
10 Label_probability_DTFrames[Label_probability_DTFrames.SameOrNot != 0].shape

```

Out[231]: (8, 4)

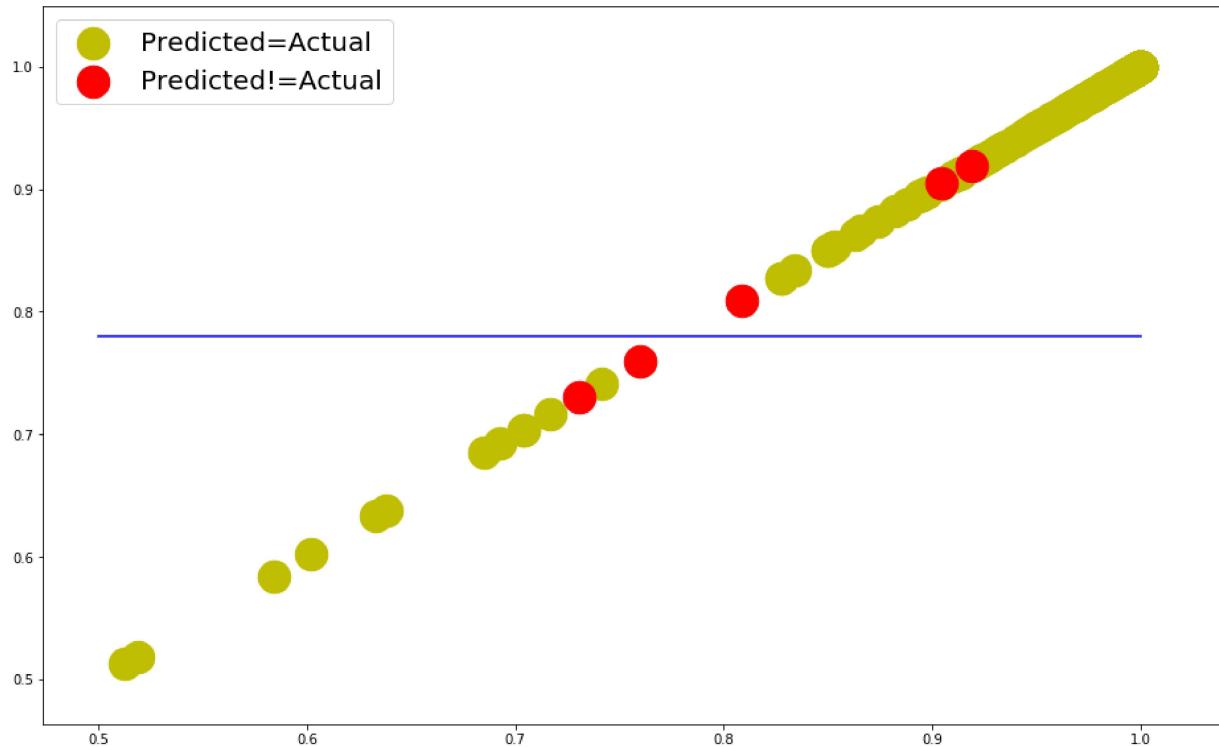
In [151]:

```

1 plt.figure(figsize=(16,10))
2 plt.scatter(Label_probability_DTFrames.query('SameOrNot == 0').Probability, Label_probability_DTFrames.query('SameOrNot == 0').Actual,
3             color='y', marker='o', s=600, label='Predicted=Actual')
4 plt.scatter(Label_probability_DTFrames.query('SameOrNot == 1').Probability, Label_probability_DTFrames.query('SameOrNot == 1').Actual,
5             color='r', marker='o', s=600, label='Predicted!=Actual')
6 plt.plot(pd.Series([0.5,1]),pd.Series([0.78,0.78]),color='b')
7 plt.legend(loc='best', fontsize=20)

```

Out[151]: &lt;matplotlib.legend.Legend at 0x282c4c8f4a8&gt;



## SVM rbf

**Taking kernel as rbf and taking different values gamma**

In [154]:

```

1 from sklearn.cross_validation import cross_val_score
2 from sklearn.svm import SVC
3 from sklearn.preprocessing import StandardScaler
4 X=StandardScaler().fit_transform(df1.iloc[:, :-1])
5 C_List=pd.Series()
6 Acc_List=pd.Series()
7 DTFrame=pd.DataFrame()
8 gamma_range=[0.0001,0.001,0.01,0.1,1,10,100]
9 for i in gamma_range:
10     svcCrossRBF = SVC(kernel='rbf',gamma=i)
11     scores=cross_val_score(svcCrossRBF,X,df1.iloc[:, -1].values,scoring='accuracy')
12     C_List = C_List.append(pd.Series(i))
13     Acc_List = Acc_List.append(pd.Series(round(scores.mean(),4)))
14 DTFrame= pd.concat([C_List,Acc_List],axis=1,keys=['gamma_values','Accuracy_values'])
15 DTFrame['Accuracy_values'] = DTFrame['Accuracy_values'].apply(lambda x : str(x))
16 DTFrame

```

Out[154]:

	gamma_values	Accuracy_values
0	\$0.00	0.8878
0	\$0.00	0.9552
0	\$0.01	0.9682
0	\$0.10	0.9631
0	\$1.00	0.9072
0	\$10.00	0.5955
0	\$100.00	0.5

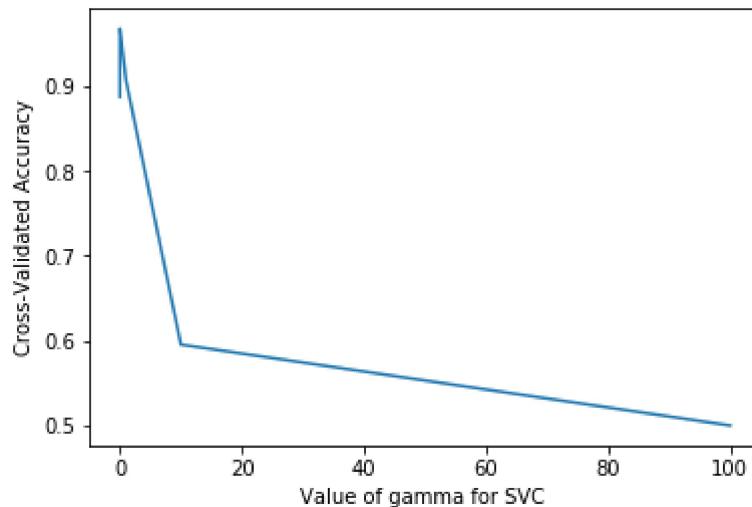
In [137]:

```

1 sns.lineplot(x=C_List,y=Acc_List)
2 plt.xlabel('Value of gamma for SVC ')
3 plt.ylabel('Cross-Validated Accuracy')

```

Out[137]: Text(0,0.5, 'Cross-Validated Accuracy')



**Taking the gamma value for highest accuracy and putting**

## that in RandomizedSearchCV

In [138]:

```

1 from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
2 from sklearn.svm import SVC
3 from sklearn.preprocessing import LabelEncoder,StandardScaler
4 import time
5 dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
6 train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stratified=True)
7 #display(test_set.loc[:, 'kurt':'sd'])
8 svc = SVC(probability=True)
9 start_time=time.time()
10 for i in range(0,10):
11     params=dict(C=[0.2,0.3,0.6,1.0],kernel=['rbf'],random_state=[5,10,13,45],
12     grid=RandomizedSearchCV(svc,params,cv=5).fit(train_set.loc[:, 'kurt':'mind'])
13     print('grid best score: for class weight '+ str(i),grid.best_score_)
14     print('grid best parameters: ',grid.best_params_)
```

```

grid best score: for class weight 0 0.5
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 0}, 'C': 1.0}
grid best score: for class weight 1 0.7052091554853985
grid best parameters: {'random_state': 13, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight 2 0.6606156274664562
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 2}, 'C': 1.0}
grid best score: for class weight 3 0.5868192580899764
grid best parameters: {'random_state': 10, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 3}, 'C': 1.0}
grid best score: for class weight 4 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 4}, 'C': 1.0}
grid best score: for class weight 5 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 5}, 'C': 1.0}
grid best score: for class weight 6 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 6}, 'C': 1.0}
grid best score: for class weight 7 0.5189423835832676
grid best parameters: {'random_state': 10, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 7}, 'C': 1.0}
grid best score: for class weight 8 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 8}, 'C': 1.0}
grid best score: for class weight 9 0.5189423835832676
grid best parameters: {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'class_weight': {0: 9}, 'C': 1.0}
```

**Since the accuracy for class\_weight 1 is the best we are going ahead with class\_weight:1**

In [155]:

```

1 from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
2 from sklearn.svm import SVC
3 from sklearn.preprocessing import LabelEncoder,StandardScaler
4 import time
5 dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
6 train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stratified=True)
7 #display(test_set.loc[:, 'kurt':'sd'])
8 svc = SVC(probability=True)
9 start_time=time.time()
10 params=dict(C=[0.2,0.3,0.6,1.0],kernel=['rbf'],random_state=[5,10,13,45],class_weight='balanced')
11 grid=RandomizedSearchCV(svc,params,cv=5).fit(train_set.loc[:, 'kurt':'mindom'])
12 print('grid best score: for class weight ',grid.best_score_)
13 print('grid best parameters: ',grid.best_params_)
14 print("Execution time: " + str((time.time() - start_time)) + ' ms')

```

grid best score: for class weight 0.7154696132596685  
 grid best parameters: {'random\_state': 5, 'kernel': 'rbf', 'class\_weight': {0: 1}, 'C': 1.0}  
 Execution time: 72.78450512886047 ms

In [128]:

```

1 print('Test_accuracy: ',grid.score(test_set.loc[:, 'kurt':'mindom'],test_set['label']))

```

Test\_accuracy: 0.7350157728706624

In [131]:

```

1 y_pred = grid.predict(test_set.loc[:, 'kurt':'mindom'])
2 p,r,f,s = precision_recall_fscore_support(test_set['label'], y_pred)
3 print('Precision_score: ',p,'Recall Score:',r,'f1_score',f,'The overall f1_score: ',f1_score(test_set['label'], y_pred))

```

Precision\_score: [0.757785 0.715942]  
 Recall Score: [0.690852 0.779180]  
 f1\_score [0.722772 0.746224]  
 The overall f1\_score: 0.7462235649546827

In [165]:

```

1 Label_probabability = pd.Series()
2 svc.fit(train_set.loc[:, 'kurt':'mindom'],train_set['label'])
3 for each in svc.predict_proba(test_set.loc[:, 'kurt':'mindom']):
4     Label_probabability = Label_probabability.append(pd.Series(np.amax(each)))
5 Label_probabability_DTF = pd.concat([Label_probabability.reset_index(drop=True),test_set['label'].reset_index(drop=True)])
6 Label_probabability_DTF['Predicted'] = pd.Series(svc.predict(test_set.loc[:, 'label']))
7 Label_probabability_DTF['Actual'] = pd.Series(test_set['label'])
8 Label_probabability_DTF['Probability'] = pd.Series(Label_probabability_DTF['Predicted'])
9 Label_probabability_DTF['SameOrNot'] = abs(Label_probabability_DTF['Actual'] - Label_probabability_DTF['Predicted'])
10 Label_probabability_DTF.head()

```

Out[165]:

	Probability	Actual	Predicted	SameOrNot
0	\$0.74	1	1	0
1	\$0.74	1	1	0
2	\$0.64	1	1	0
3	\$0.58	1	1	0
4	\$0.55	1	1	0

In [162]:

```
1 from sklearn.metrics import confusion_matrix as cm
2 svm_pred=grid.predict(test_set.loc[:, 'kurt':'mindom'])
3 matrix = cm(test_set['label'],svm_pred)
4 print('The confusion matrix is: ')
5 print(matrix)
```

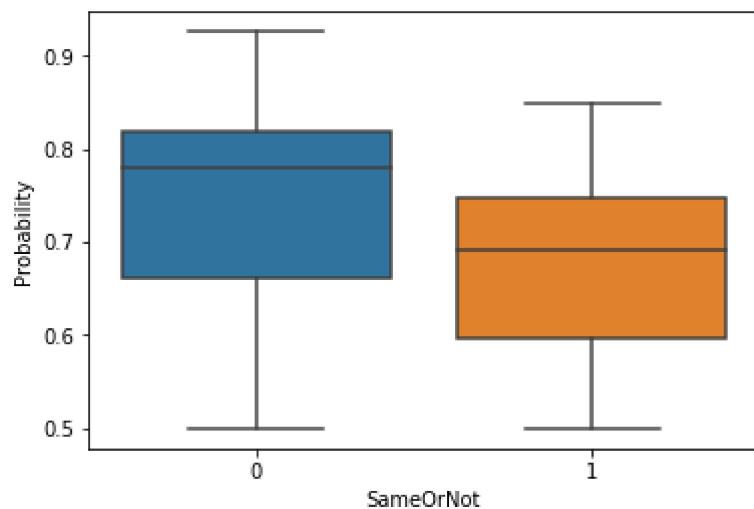
The confusion matrix is:

```
[[219  98]
 [ 70 247]]
```

In [167]:

```
1 sns.boxplot(x=Label_probability_DTFrames.SameOrNot,y=Label_probability_DTF
```

Out[167]: &lt;matplotlib.axes.\_subplots.AxesSubplot at 0x282c4ccbf98&gt;



## SVM Poly

In [172]:

```

1 from sklearn.cross_validation import cross_val_score
2 from sklearn.svm import SVC
3 from sklearn.preprocessing import StandardScaler
4 X=StandardScaler().fit_transform(df1.iloc[:, :-1])
5 C_List=pd.Series()
6 Acc_List=pd.Series()
7 DTFrame=pd.DataFrame()
8 polyList=[2,3,4,5,6]
9 for i in polyList:
10     svcCrossRBF = SVC(kernel='poly',degree=i)
11     scores=cross_val_score(svcCrossRBF,X,df1.iloc[:, -1].values,scoring='accuracy')
12     C_List = C_List.append(pd.Series(i))
13     Acc_List = Acc_List.append(pd.Series(round(scores.mean(),4)))
14 DTFrame= pd.concat([C_List,Acc_List],axis=1,keys=['poly_values','Accuracy_values'])
15 DTFrame['Accuracy_values'] = DTFrame['Accuracy_values'].apply(lambda x : str(x))
16 DTFrame

```

Out[172]:

	poly_values	Accuracy_values
0	2	0.8507
0	3	0.9458
0	4	0.8312
0	5	0.866
0	6	0.7748

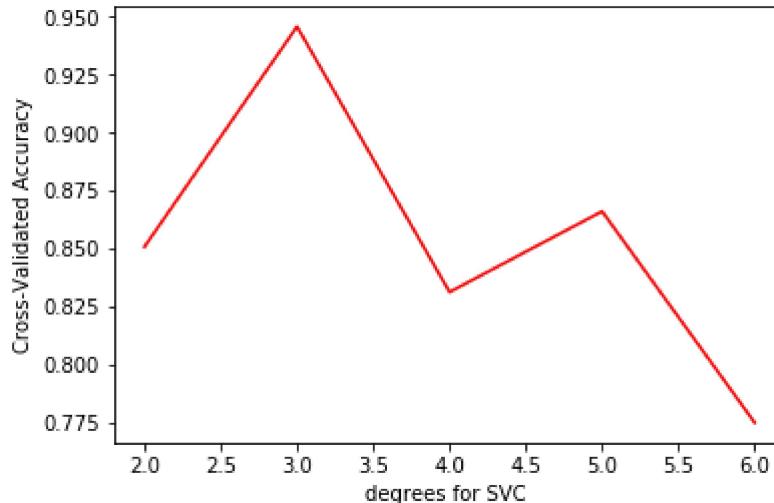
In [176]:

```

1 sns.lineplot(x=C_List,y=Acc_List,color='red')
2 plt.xlabel('degrees for SVC ')
3 plt.ylabel('Cross-Validated Accuracy')

```

Out[176]: Text(0,0.5,'Cross-Validated Accuracy')



In [175]:

```

1 from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
2 from sklearn.svm import SVC
3 from sklearn.preprocessing import LabelEncoder,StandardScaler
4 import time
5 dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
6 train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stratified=True)
7 sc=StandardScaler()
8 train_set.loc[:, 'kurt':'mindom']= sc.fit_transform(train_set.loc[:, 'kurt':'mindom'])
9 #display(test_set.loc[:, 'kurt':'sd'])
10 svc = SVC(probability=True)
11 start_time=time.time()
12 params=dict(C=[0.6,1.0],kernel=['poly'],degree=[2,3,4],random_state=[5,10,13],
13 for i in range(3,10):
14     grid=RandomizedSearchCV(svc,params,cv=i).fit(train_set.loc[:, 'kurt':'mindom'])
15     print('grid best score: for class weight ',grid.best_score_)
16     print('grid best parameters: ',grid.best_params_)
```

C:\Users\supratik chanda\Documents\New folder (2)\lib\site-packages\pandas\core\indexes.py:543: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy> (<http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>)

```

self.obj[item] = s

grid best score: for class weight  0.9494869771112865
grid best parameters: {'random_state': 5, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight  0.9526440410418311
grid best parameters: {'random_state': 45, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight  0.9522494080505131
grid best parameters: {'random_state': 45, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight  0.9526440410418311
grid best parameters: {'random_state': 45, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight  0.9534333070244673
grid best parameters: {'random_state': 5, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight  0.9538279400157853
grid best parameters: {'random_state': 13, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
grid best score: for class weight  0.9538279400157853
grid best parameters: {'random_state': 45, 'kernel': 'poly', 'degree': 3, 'class_weight': {0: 1}, 'C': 1.0}
```

In [ ]:

```
1 # 5 fold Cross validation produces the best score
```

In [180]:

```

1 from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
2 from sklearn.svm import SVC
3 from sklearn.preprocessing import LabelEncoder,StandardScaler
4 import time
5 dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
6 train_set,test_set = train_test_split(dfnew,random_state=5,test_size=0.2,stratified=True)
7 sc=StandardScaler()
8 train_set.loc[:, 'kurt':'mindom']= sc.fit_transform(train_set.loc[:, 'kurt':'mindom'])
9 #display(test_set.loc[:, 'kurt':'sd'])
10 svc1 = SVC(probability=True)
11 start_time=time.time()
12 params=dict(C=[0.6,1.0],kernel=['poly'],degree=[1,2,3,4],random_state=[5,10,1])
13 grid=RandomizedSearchCV(svc1,params, cv=5).fit(train_set.loc[:, 'kurt':'mindom'])
14 print('grid best score: for class weight ',grid.best_score_)
15 print('grid best parameters: ',grid.best_params_)
16 print("Execution time: " + str((time.time() - start_time)) + ' ms')

```

C:\Users\supratik chanda\Documents\New folder (2)\lib\site-packages\pandas\core\indexing.py:543: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy> (<http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>)

```

self.obj[item] = s

grid best score: for class weight  0.9696132596685083
grid best parameters: {'random_state': 10, 'kernel': 'poly', 'degree': 1, 'class_weight': {0: 1}, 'C': 1.0}
Execution time: 23.117100954055786 ms

```

In [181]:

```

1 test_set.loc[:, 'kurt':'mindom']=sc.transform(test_set.loc[:, 'kurt':'mindom'])
2 print('Test_accuracy: ',grid.score(test_set.loc[:, 'kurt':'mindom'],test_set['

```

Test\_accuracy: 0.9747634069400631

C:\Users\supratik chanda\Documents\New folder (2)\lib\site-packages\pandas\core\indexing.py:543: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy> (<http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>)

```
self.obj[item] = s
```

In [182]:

```

1 from sklearn.metrics import confusion_matrix,precision_recall_fscore_support,
2 from sklearn.preprocessing import LabelEncoder
3 dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
4 #display(test_set.loc[:, 'kurt':'sd'])
5 svc1.fit(train_set.loc[:, 'kurt':'mindom'],train_set['label'])
6 y_pred = svc1.predict(test_set.loc[:, 'kurt':'mindom'])
7 print('The test dataset confusion_matrix is :\n',confusion_matrix(test_set['l

```

The test dataset confusion\_matrix is :

```

[[314  3]
 [ 10 307]]

```

In [183]:

```

1 Label_probabability = pd.Series()
2 for each in svc1.predict_proba((test_set.loc[:, 'kurt':'mindom'])):
3     Label_probabability = Label_probabability.append(pd.Series(npamax(each)))
4 Label_probabability_DTF = pd.concat([Label_probabability.reset_index(drop=True),
5                                     test_set['label'].reset_index(drop=True),
6                                     pd.Series(svc1.predict(test_set.loc[:, 'kurt':'mindom']),
7                                     keys=['Probability','Actual','Predicted'])])
8 Label_probabability_DTF['Sameornot']=abs(Label_probabability_DTF.Actual-
9 Label_probabability_DTF.Predicted)

```

Out[183]:

	Probability	Actual	Predicted	Sameornot
0	\$0.99	0	0	0
1	\$0.95	1	1	0
2	\$0.65	1	1	0
3	\$1.00	0	0	0
4	\$0.99	0	0	0

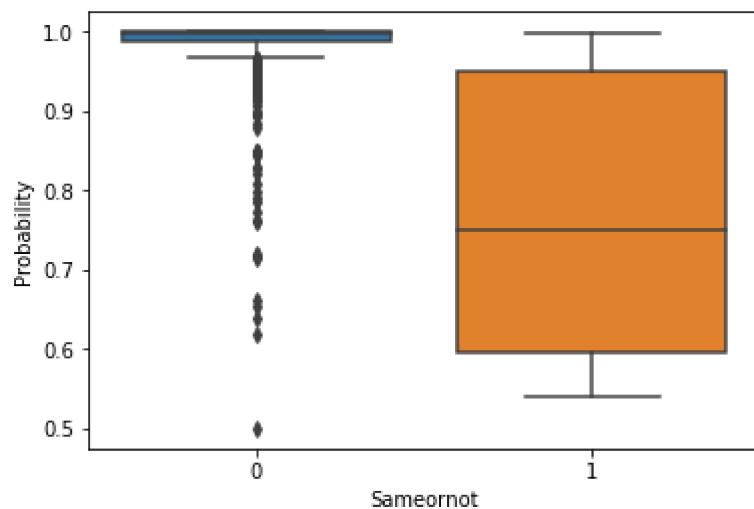
In [185]: 1 Label\_probabability\_DTFrames[Label\_probabability\_DTFrames.Sameornot !=0]

Out[185]:

	Probability	Actual	Predicted	Sameornot
10	\$0.87	1	0	1
58	\$0.95	1	0	1
59	\$0.73	1	0	1
185	\$0.60	0	1	1
189	\$0.65	0	1	1
253	\$0.99	1	0	1
256	\$0.59	1	0	1
267	\$0.54	1	0	1
296	\$0.58	0	1	1
487	\$0.95	1	0	1
555	\$0.75	1	0	1
578	\$1.00	1	0	1
588	\$0.88	1	0	1

In [184]: 1 sns.boxplot(x=Label\_probabability\_DTFrames.Sameornot,y=Label\_probabability\_DTF

Out[184]: <matplotlib.axes.\_subplots.AxesSubplot at 0x282c4ee8e48>



In [ ]: 1