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 precentage 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:  0.9688347277832236
grid best parameters:  {'random_state': 45, 'kernel': 'linear', 'class_weight': {0: 1}, 'C': 6}
Execution time: 11.855380963226318 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_selectionand passed gamma value=0.01 and a list of class weights inside the param_grids as paramaters. 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_mtarix 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	dfra
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 × 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	0.059780999999999994	0.0642413	0.032026900000000004	0.0150715	0.090193399999999999	0.07512200000000001	12.8634618
1	0.0660087	0.067310000000000001	0.0402287	0.0194139	0.092666199999999999	0.0732523	22.423285399999997
2	0.0773155	0.0838294	0.0367185	0.0087011000000000002	0.131908	0.123207	30.757154600000003
3	0.1512281	0.072110600000000001	0.15801120000000002	0.0965817	0.20795529999999998	0.11137349999999999	1.2328313
4	0.1351204	0.0791461	0.1246562	0.0787202	0.2060449	0.1273247	1.1011737

5 rows × 21 columns

The feature_selection technique used is SelectKBest; chosen among SeleckBest and SelectPrecentile . 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

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

```
grid best score:  0.5240726124704025
grid best paramters:  {'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

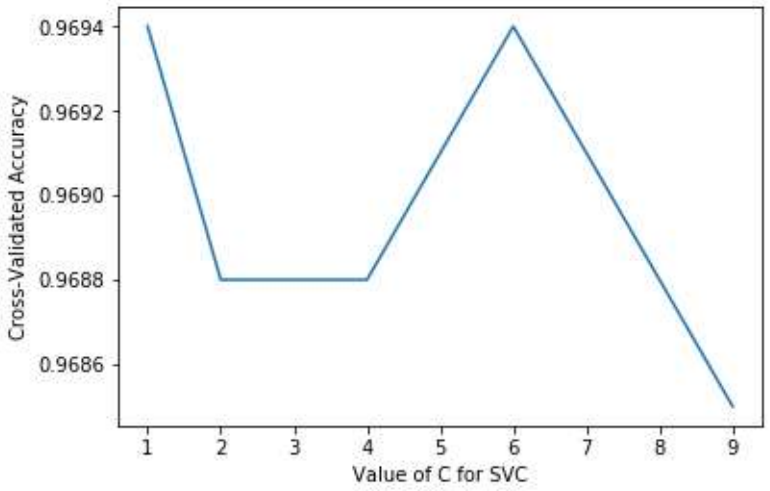
SVM KERNEL=LINEAR

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 paramters:  {'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
```

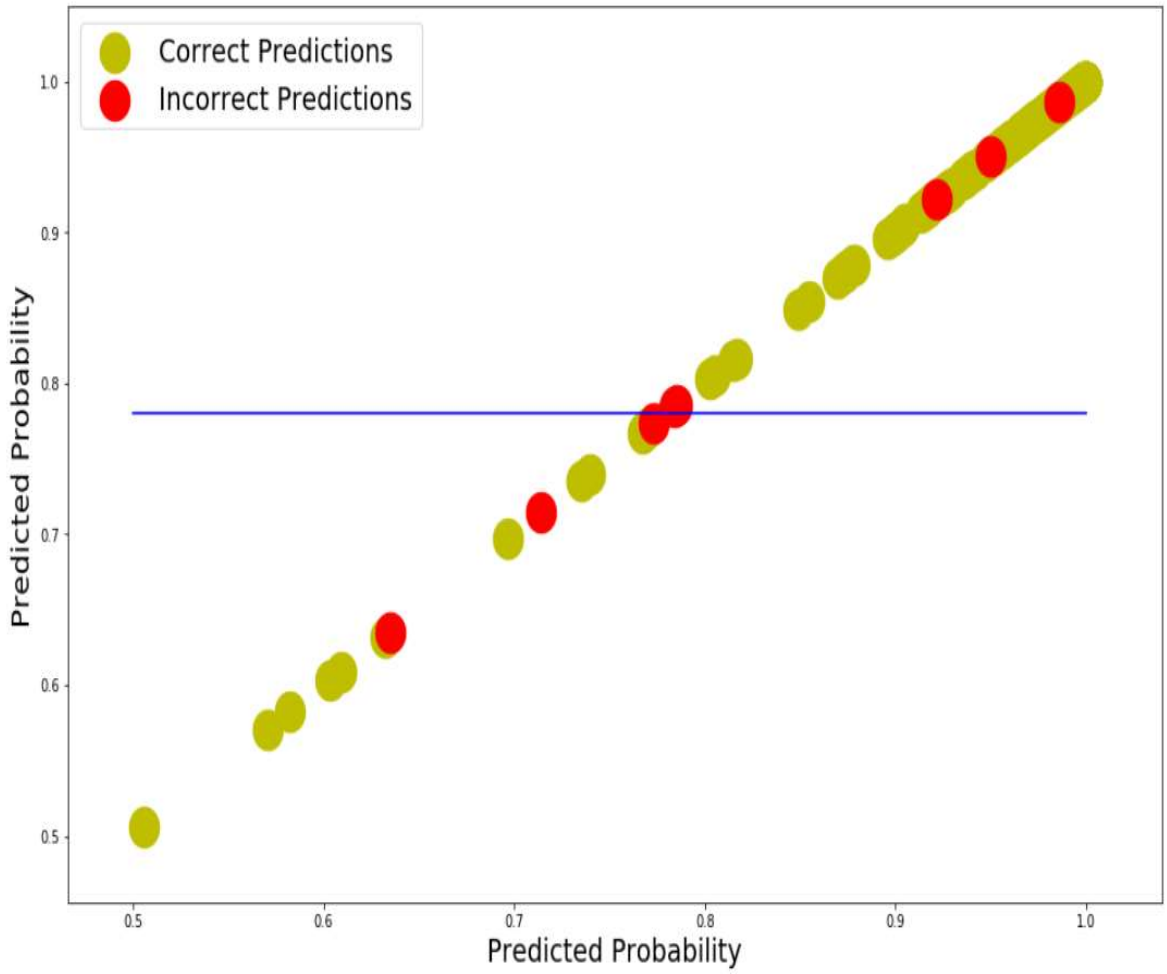
The accuracy is a bit lower but the execution time is much lesser and hence, we are using RandomizedSearchCV.

```
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_probabilties is shown below :

X and Y axis are the probabilities of the predicted values



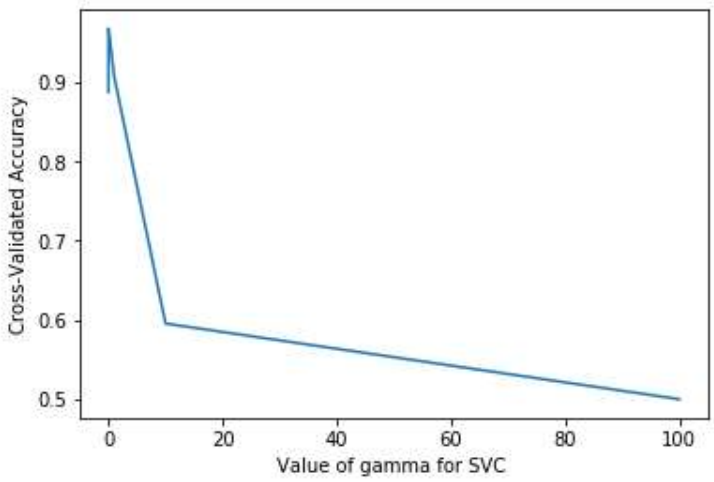
SVM KERNEL=RBF

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: 72.78450512886047 ms
```

Test Accuracy:

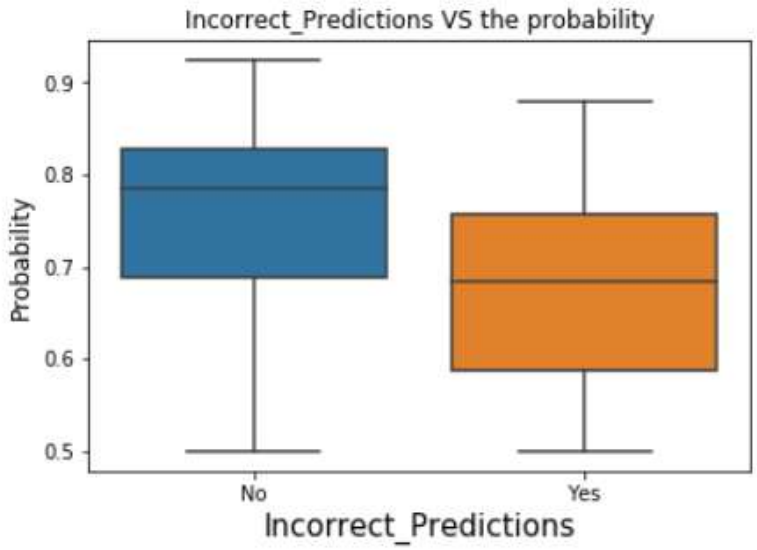
```
Test_accuracy:  0.7350157728706624
```

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 probablity of each predicted value.

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



1 implies Actual values are not equal to Predicted Values

0 implies Actual values equal to Predicted Values

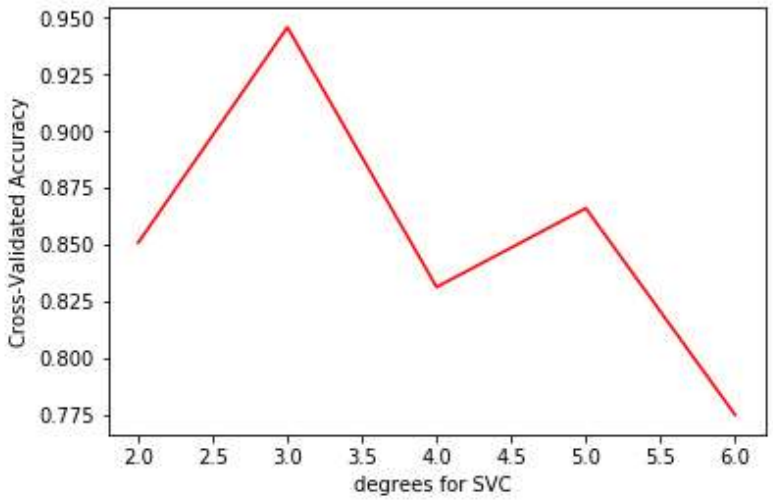
Here we see see that the predicted values which are correct has a greater probability

SVM KERNEL=POLY

Analyzing the accuracy score with different values of polynomial degree

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

```
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]]
```

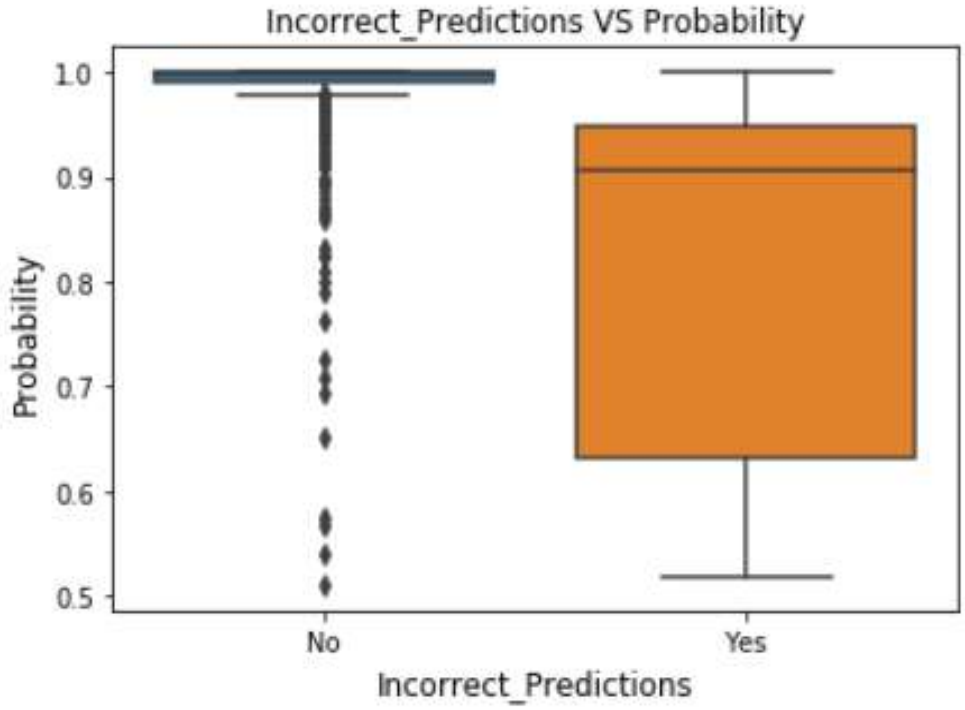
The incorrect values total is 13.

The next dataframe is the dataframe of incorrect predicted values and their predicted proabilities 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 probablity of each predicted value.

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



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]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
from sklearn.model_selection import train_test_split
```

In [200]:
```
df =pd.read_csv('VoiceRecognition.csv')
df.head()
```

Out[200]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	m
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]:
```
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]:
```
df[df.duplicated(keep='first')]
```

Out[202]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	mea
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]:
```
df.drop_duplicates(inplace=True)
```

In [204]:
```
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]:
```
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 [259]:
```
# Removing the $ symbol from every column
df1 = df.iloc[:,:-1].astype('str').replace('$','',regex=True)
```

In [260]:
```
df1= df1.join(df['label'])
df1.head()
```

Out[260]:

	meanfreq	sd	median	Q25	
0	0.059780999999999994	0.0642413	0.032026900000000004	0.0150715	0.090
1	0.0660087	0.06731000000000001	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 [261]:
```
1 df1.dtypes
```

Out[261]:
```
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]:
```
np.set_printoptions(formatter={'float_kind':'{:3f}'.format})
from sklearn.feature_selection import SelectKBest,chi2,SelectPercentile,f_cla
selValue1 = SelectKBest(f_classif,k=20).fit(df.iloc[:,:-1],df.iloc[:,-1])
print('\nSelectKBest feature_extraction_text using f_classif:Value\n')
print('pvalues_:  ',selValue1.pvalues_)
print('\nscores:    ',selValue1.scores_)
print(selValue1.get_support(indices=True))
selValue2 = SelectKBest(chi2,k=20).fit(df.iloc[:,:-1],df.iloc[:,-1])
print('\nSelectKBest feature_extraction_text using chi2:Value\n')
print('pvalues_:  ',selValue2.pvalues_)
print('\nscores:    ',selValue2.scores_)
print('\nSelectPercentile feature_extraction_text:Value\n')
# Used SelectPercentile feature_extraction_text
selValue3 = SelectPercentile(chi2,percentile=100).fit(df.iloc[:,:-1],df.iloc[
print('pvalues_:  ',selValue3.pvalues_)
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 0.0
00001
 0.000000 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 0.0
00000
 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 0.0
00000
 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]:
```
dt = pd.DataFrame({'Columns':df.columns.values[:-1],'P_value':pd.Series(selVa
```

In [212]:
```
pd.options.display.float_format = '${:,.2f}'.format
dt.sort_values(by='scores',ascending=False,inplace=True)
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]:
```
dfnew=pd.DataFrame()
for each in dt.Columns.values:
    dfnew = pd.concat([dfnew,pd.DataFrame(df1[each])],axis=1)
dfnew = dfnew.join(pd.DataFrame(df['label']))
dfnew = dfnew.drop(dfnew.loc[:,'sd':'Q75'].columns,axis=1)
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	0.009014399999999999	0.5137238000000001	
2	1024.927705	0.015625	0.0078125	0.007990100000000002	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]:
```
dfnew.columns.values
```

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

Logistics Regression: GridSearchCV

In [215]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder,StandardScaler
import time
dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stra
#display(test_set.loc[:,'kurt':'sd'])
lm=LogisticRegression()
start_time=time.time()
params = dict(penalty=['l1','l2',],C=[0.2,0.5,0.7,1.0],random_state=[5,10,45,
grid = GridSearchCV(lm,params,cv=5).fit(train_set.loc[:,'kurt':'mindom'],trai
print('For GridSearchCV:')
print('grid best score for train_set: ',grid.best_score_)
print('grid best parameters for train_set: ',grid.best_params_)
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', 'penalt
y': 'l1', 'random_state': 5}
Execution time: 13.599502086639404 ms
```

In [216]:
```
print('Test_accuracy: ',grid.score(test_set.loc[:,'kurt':'mindom'],test_set['
```

```
Test_accuracy:  0.9826498422712934
```

Logistic Regression: RandomizedSearchCV

In [217]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder,StandardScaler
import time
#dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stra
#display(test_set.loc[:,'kurt':'sd'])
lm=LogisticRegression()
start_time=time.time()
params = dict(penalty=['l1','l2',],C=[0.2,0.5,0.7,1.0],random_state=[5,10,45,
grid = RandomizedSearchCV(lm,params,cv=5).fit(train_set.loc[:,'kurt':'mindom'
print('For RandomizedSearchCV:')
print('grid best score for train_set: ',grid.best_score_)
print('grid best parameters for train_set: ',grid.best_params_)
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', 'mul
ti_class': 'ovr', 'C': 0.5}
Execution time: 3.2223386764526367 ms
```

In [218]:
```
print('Test_accuracy: ',grid.score(test_set.loc[:,'kurt':'mindom'],test_set['
```

```
Test_accuracy:  0.9810725552050473
```

In [219]:
```
from sklearn.metrics import confusion_matrix,precision_recall_fscore_support,
from sklearn.preprocessing import LabelEncoder
#dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
#display(test_set.loc[:,'kurt':'sd'])
lm=LogisticRegression()
lm.fit(train_set.loc[:,'kurt':'mindom'],train_set['label'])
y_pred = lm.predict(test_set.loc[:,'kurt':'mindom'])
print('The test dataset confusion_matrix is :\n',confusion_matrix(test_set['l
```

```
The test dataset confusion_matrix is :
 [[287  30]
 [  8 309]]
```

In [220]:
```
p,r,f,s = precision_recall_fscore_support(test_set['label'], y_pred)
print('Precision_score: ',p,'\nRecall Score:',r,'\nf_score',f,'\n The overall
      .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]:
```
Label_probabability = pd.Series()
for each in lm.predict_proba((test_set.loc[:,'kurt':'mindom'])):
    Label_probabability = Label_probabability.append(pd.Series(np.amax(each))
Label_probabability_DTFrame = pd.concat([Label_probabability.reset_index(drop
                                         test_set['label'].reset_index(drop=T
                                         pd.Series(lm.predict(test_set.loc[:,
                                        keys=['Probability','Actual','Predict
Label_probabability_DTFrame['SameOrNot']= abs(Label_probabability_DTFrame.Act
Label_probabability_DTFrame[Label_probabability_DTFrame.SameOrNot !=0].head()
```

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 [ ]:
```

```

In [24]:
```
## Using Multinomial parameter
```

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

```
grid best score:  0.5240726124704025
grid best paramters:  {'solver': 'sag', 'random_state': 10, 'penalty': 'l2', 'm
ulti_class': 'multinomial', 'C': 1.0}
```

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

```
Test_accuracy:  0.5299684542586751
```

SVM Linear

Type *Markdown* and LaTeX: α^2

Using Cross Validation with Different values of C

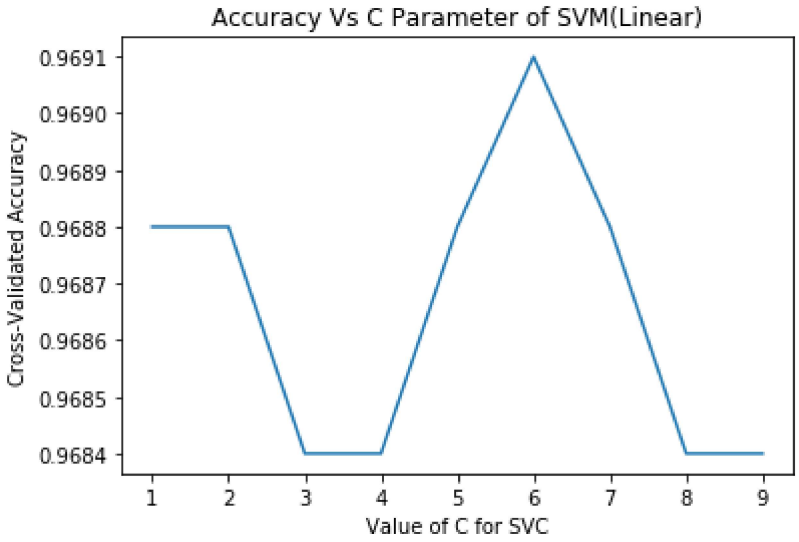
In [262]:
```
from sklearn.cross_validation import cross_val_score
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
X=StandardScaler().fit_transform(df1.iloc[:,:-1])
C_List=pd.Series()
Acc_List=pd.Series()
DTFrame=pd.DataFrame()
for i in range(1,10):
    svcCrossLinear = SVC(kernel='linear',C=i)
    scores=cross_val_score(svcCrossLinear,X,df1.iloc[:,-1].values,scoring='ac
    C_List = C_List.append(pd.Series(i))
    Acc_List = Acc_List.append(pd.Series(round(scores.mean(),4)))
DTFrame= pd.concat([C_List,Acc_List],axis=1,keys=['C_values','Accuracy_values
DTFrame['Accuracy_values'] = DTFrame['Accuracy_values'].apply(lambda x : str(
DTFrame
```

Out[262]:

	C_values	Accuracy_values
0	1	0.9688
0	2	0.9688
0	3	0.9684
0	4	0.9684
0	5	0.9688
0	6	0.9691
0	7	0.9688
0	8	0.9684
0	9	0.9684

In [263]:
```
1 sns.lineplot(x=C_List,y=Acc_List)
2 plt.xlabel('Value of C for SVC')
3 plt.ylabel('Cross-Validated Accuracy')
4 plt.title('Accuracy Vs C Parameter of SVM(Linear)')
```

Out[263]: Text(0.5,1,'Accuracy Vs C Parameter of SVM(Linear)')



Using GridSearchCV

In [269]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder,StandardScaler
import time
dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stra
#display(test_set.loc[:,'kurt':'sd'])
train_set.loc[:,'kurt':'mindom']=StandardScaler().fit_transform(train_set.loc
test_set.loc[:,'kurt':'mindom'] =StandardScaler().fit_transform(test_set.loc[
svcLinearGrid = SVC(probability=True)
start_time=time.time()
params=dict(C=[6],kernel=['linear'],random_state=[5,10,13,45],class_weight=[{
grid=GridSearchCV(svcLinearGrid,params,cv=10).fit(train_set.loc[:,'kurt':'min
print('grid best score: ',grid.best_score_)
print('grid best paramters: ',grid.best_params_)
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/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  self.obj[item] = s

grid best score:  0.9691943127962085
grid best paramters:  {'C': 6, 'class_weight': {0: 1}, 'kernel': 'linear', 'ran
dom_state': 5}
Execution time: 72.60279583930969 ms
```

In [270]:
```
print('Test_accuracy: ',grid.score(test_set.loc[:,'kurt':'mindom'],test_set['
```

```
Test_accuracy:  0.9779179810725552
```

Using RandomizedSearchCV

In [316]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder,StandardScaler
import time
import warnings
warnings.simplefilter('ignore')
dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
sc=StandardScaler()
train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stra
train_set.loc[:,'kurt':'mindom'] =sc.fit_transform(train_set.loc[:,'kurt':'mi
test_set.loc[:,'kurt':'mindom']=sc.transform(test_set.loc[:,'kurt':'mindom'])
#display(test_set.loc[:,'kurt':'sd'])
svcLinearRandom = SVC(probability=True)
start_time=time.time()
params=dict(C=[6],kernel=['linear'],random_state=[5,10,13,45],class_weight=[{
grid=RandomizedSearchCV(svcLinearRandom,params,cv=5).fit(train_set.loc[:,'kur
print('grid best score: ',grid.best_score_)
print('grid best parameters: ',grid.best_params_)
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
grid best score:  0.9691943127962085
grid best parameters:  {'random_state': 45, 'kernel': 'linear', 'class_weight':
{0: 1}, 'C': 6}
Execution time: 22.810219764709473 ms
```

In [317]:
```
print('Test_accuracy: ',grid.score(test_set.loc[:,'kurt':'mindom'],test_set['
```

```
Test_accuracy:  0.9779179810725552
```

In [318]:
```
y_pred=grid.predict(test_set.loc[:,'kurt':'mindom'])
p,r,f,s = precision_recall_fscore_support(test_set['label'], y_pred)
print('Precision_score: ',p,'\nRecall Score:',r,'\nf_score',f,'\n The overall
      .format(f1_score(test_set['label'],y_pred)))
```

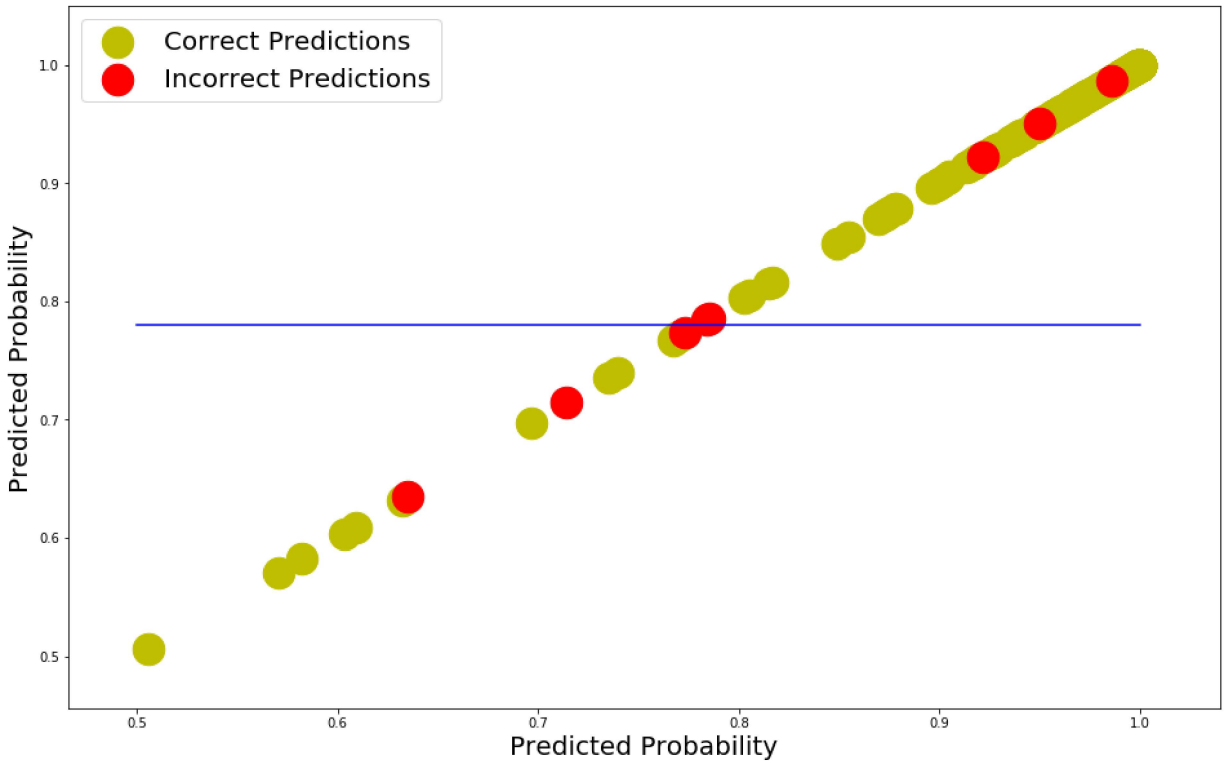
```
Precision_score:  [0.984026 0.971963]
Recall Score: [0.971609 0.984227]
f_score [0.977778 0.978056]
 The overall f1_score: 0.9780564263322884
```

In [319]:
```
Label_probabability = pd.Series()
svcLinearRandom.fit(train_set.loc[:,'kurt':'mindom'],train_set['label'])
for each in svcLinearRandom.predict_proba((test_set.loc[:,'kurt':'mindom'])):
    Label_probabability = Label_probabability.append(pd.Series(np.amax(each))
Label_probabability_DTFrame = pd.concat([Label_probabability.reset_index(drop
                                         test_set['label'].reset_index(drop=T
                                         pd.Series(svcLinearRandom.predict(te
                                        keys=['Probability','Actual','Predict
Label_probabability_DTFrame['SameOrNot']=abs(Label_probabability_DTFrame.Actu
Label_probabability_DTFrame[Label_probabability_DTFrame.SameOrNot!=0].shape
```

Out[319]: (8, 4)

In [320]:
```
plt.figure(figsize=(16,10))
plt.scatter(Label_probabability_DTFrame.query('SameOrNot == 0').Probability,L
            color='y',marker='o',s=600,label='Correct Predictions')
plt.scatter(Label_probabability_DTFrame.query('SameOrNot == 1').Probability,L
            color='r',marker='o',s=600,label='Incorrect Predictions')
plt.plot(pd.Series([0.5,1]),pd.Series([0.78,0.78]),color='b')
plt.xlabel('Predicted Probability',fontsize=20)
plt.ylabel('Predicted Probability',fontsize=20)
plt.legend(loc='best',fontsize=20)
```

Out[320]: <matplotlib.legend.Legend at 0x282c801c4a8>



SVM rbf

Taking kernel as rbf and taking different values gamma

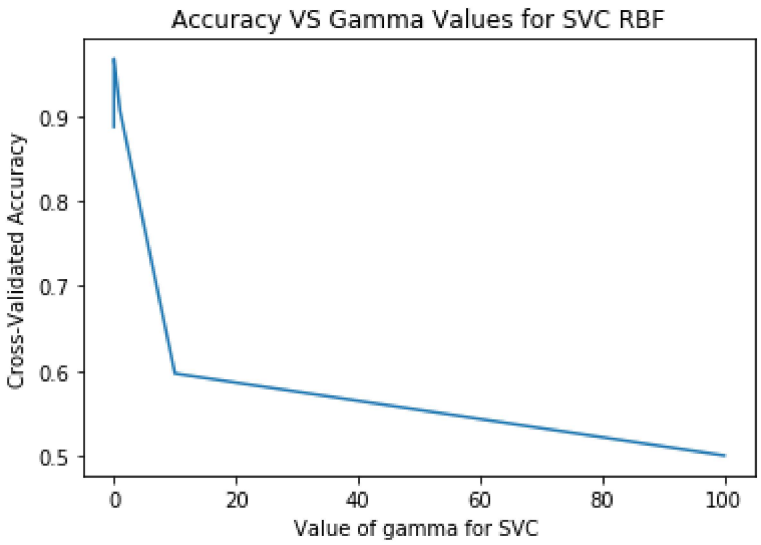
In [280]:
```
from sklearn.cross_validation import cross_val_score
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
X=StandardScaler().fit_transform(df1.iloc[:,:-1])
C_List=pd.Series()
Acc_List=pd.Series()
DTFrame=pd.DataFrame()
gamma_range=[0.0001,0.001,0.01,0.1,1,10,100]
for i in gamma_range:
    svcCrossRBF = SVC(kernel='rbf',gamma=i)
    scores=cross_val_score(svcCrossRBF,X,df1.iloc[:,-1].values,scoring='accur
    C_List = C_List.append(pd.Series(i))
    Acc_List = Acc_List.append(pd.Series(round(scores.mean(),4)))
DTFrame= pd.concat([C_List,Acc_List],axis=1,keys=['gamma_values','Accuracy_va
DTFrame['Accuracy_values'] = DTFrame['Accuracy_values'].apply(lambda x : str(
DTFrame
```

Out[280]:

	gamma_values	Accuracy_values
0	$0.00	0.8878
0	$0.00	0.9559
0	$0.01	0.9681
0	$0.10	0.9631
0	$1.00	0.9068
0	$10.00	0.5968
0	$100.00	0.5

In [289]:
```
sns.lineplot(x=C_List,y=Acc_List)
plt.xlabel('Value of gamma for SVC ')
plt.ylabel('Cross-Validated Accuracy')
plt.title('Accuracy VS Gamma Values for SVC RBF')
```

Out[289]: Text(0.5,1,'Accuracy VS Gamma Values for SVC RBF')



Taking the gamma value for highest accuracy and putting that in RandomizedSearchCV

In [282]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder,StandardScaler
import time
dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stra
#display(test_set.loc[:,'kurt':'sd'])
svc = SVC(probability=True)
start_time=time.time()
for i in range(0,10):
    params=dict(C=[0.2,0.3,0.6,1.0],kernel=['rbf'],random_state=[5,10,13,45],
    grid=RandomizedSearchCV(svc,params,cv=5).fit(train_set.loc[:,'kurt':'mind
    print('grid best score:  for class weight '+ str(i),grid.best_score_)
    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, 'cla
ss_weight': {0: 0}, 'C': 0.6}
grid best score:  for class weight 1 0.6978672985781991
grid best parameters:  {'random_state': 10, 'kernel': 'rbf', 'gamma': 0.01, 'cl
ass_weight': {0: 1}, 'C': 1.0}
grid best score:  for class weight 2 0.655608214849921
grid best parameters:  {'random_state': 10, 'kernel': 'rbf', 'gamma': 0.01, 'cl
ass_weight': {0: 2}, 'C': 1.0}
grid best score:  for class weight 3 0.5821484992101106
grid best parameters:  {'random_state': 13, 'kernel': 'rbf', 'gamma': 0.01, 'cl
ass_weight': {0: 3}, 'C': 1.0}
grid best score:  for class weight 4 0.514612954186414
grid best parameters:  {'random_state': 10, 'kernel': 'rbf', 'gamma': 0.01, 'cl
ass_weight': {0: 4}, 'C': 1.0}
grid best score:  for class weight 5 0.514612954186414
grid best parameters:  {'random_state': 45, 'kernel': 'rbf', 'gamma': 0.01, 'cl
ass_weight': {0: 5}, 'C': 1.0}
grid best score:  for class weight 6 0.514612954186414
grid best parameters:  {'random_state': 10, 'kernel': 'rbf', 'gamma': 0.01, 'cl
ass_weight': {0: 6}, 'C': 1.0}
grid best score:  for class weight 7 0.514612954186414
grid best parameters:  {'random_state': 10, 'kernel': 'rbf', 'gamma': 0.01, 'cl
ass_weight': {0: 7}, 'C': 1.0}
grid best score:  for class weight 8 0.514612954186414
grid best parameters:  {'random_state': 5, 'kernel': 'rbf', 'gamma': 0.01, 'cla
ss_weight': {0: 8}, 'C': 1.0}
grid best score:  for class weight 9 0.514612954186414
grid best parameters:  {'random_state': 45, 'kernel': 'rbf', 'gamma': 0.01, 'cl
ass_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 [283]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder,StandardScaler
import time
dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stra
#display(test_set.loc[:,'kurt':'sd'])
svc = SVC(probability=True)
start_time=time.time()
params=dict(C=[0.2,0.3,0.6,1.0],kernel=['rbf'],random_state=[5,10,13,45],clas
grid=RandomizedSearchCV(svc,params,cv=5).fit(train_set.loc[:,'kurt':'mindom']
print('grid best score:  for class weight ',grid.best_score_)
print('grid best parameters: ',grid.best_params_)
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
grid best score:  for class weight  0.7014218009478673
grid best parameters:  {'random_state': 10, 'kernel': 'rbf', 'class_weight':
{0: 1}, 'C': 1.0}
Execution time: 73.6187071800232 ms
```

In [284]:
```
print('Test_accuracy: ',grid.score(test_set.loc[:,'kurt':'mindom'],test_set['
```

```
Test_accuracy:  0.7350157728706624
```

In [285]:
```
y_pred = grid.predict(test_set.loc[:,'kurt':'mindom'])
p,r,f,s = precision_recall_fscore_support(test_set['label'], y_pred)
print('Precision_score: ',p,'\nRecall Score:',r,'\nf_score',f,'\n The overall
```

```
Precision_score:  [0.745875 0.725076]
Recall Score: [0.712934 0.757098]
f_score [0.729032 0.740741]
 The overall f1_score: 0.7407407407407407
```

In [292]:
```
_probability = pd.Series()
it(train_set.loc[:,'kurt':'mindom'],train_set['label'])
ach in svc.predict_proba((test_set.loc[:,'kurt':'mindom'])):
abel_probability = Label_probability.append(pd.Series(np.amax(each)))
_probability_DTFrame = pd.concat([Label_probability.reset_index(drop=True),
                                  test_set['label'].reset_index(drop=True),
                                  pd.Series(svc.predict(test_set.loc[:,'kurt':'m
                                 keys=['Probability','Actual','Predicted'])
_probability_DTFrame['SameOrNot']=abs(Label_probability_DTFrame.Actual- Label_
_probability_DTFrame['Incorrect_Predictions']=Label_probability_DTFrame['SameO
_probability_DTFrame.head()
```

Out[292]:

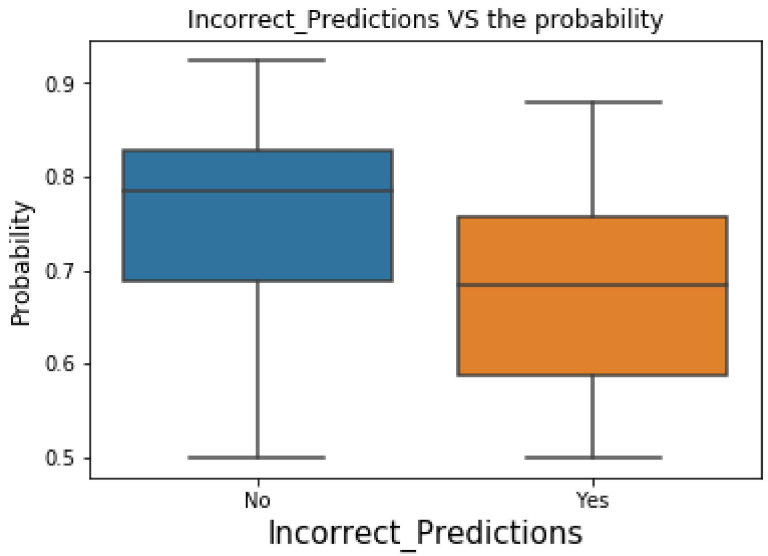
	Probability	Actual	Predicted	SameOrNot	Incorrect_Predictions
0	$0.75	1	1	0	No
1	$0.69	1	1	0	No
2	$0.78	1	1	0	No
3	$0.64	1	1	0	No
4	$0.57	1	1	0	No

In [293]:
```
from sklearn.metrics import confusion_matrix as cm
svm_pred=grid.predict(test_set.loc[:,'kurt':'mindom'])
matrix = cm(test_set['label'],svm_pred)
print('The confusion matrix is: ')
print(matrix)
```

```
The confusion matrix is:
[[226  91]
 [ 77 240]]
```

In [299]:
```
sns.boxplot(x=Label_probabability_DTFrame.Incorrect_Predictions,y=Label_proba
plt.xlabel('Incorrect_Predictions',fontsize=15)
plt.ylabel('Probability',fontsize=12)
plt.title('Incorrect_Predictions VS the probability',fontsize=12)
```

Out[299]: Text(0.5,1,'Incorrect_Predictions VS the probability')



SVM Poly

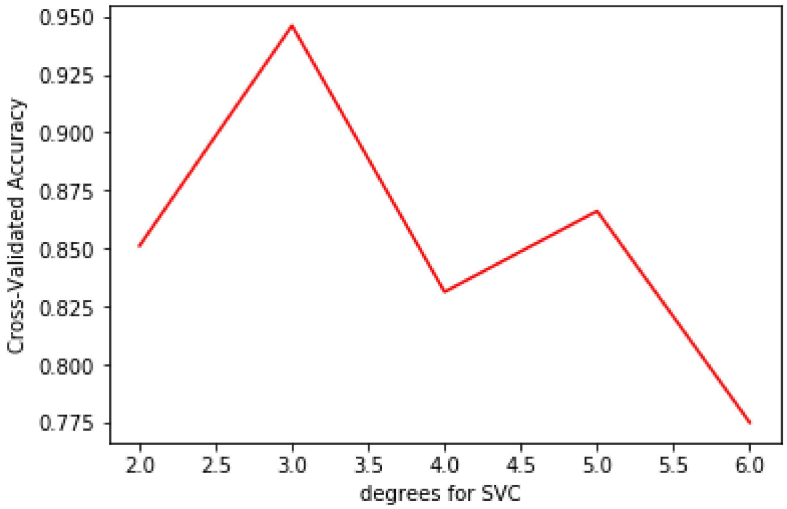
In [300]:
```
from sklearn.cross_validation import cross_val_score
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
X=StandardScaler().fit_transform(df1.iloc[:,:-1])
C_List=pd.Series()
Acc_List=pd.Series()
DTFrame=pd.DataFrame()
polyList=[2,3,4,5,6]
for i in polyList:
    svcCrossRBF = SVC(kernel='poly',degree=i)
    scores=cross_val_score(svcCrossRBF,X,df1.iloc[:,-1].values,scoring='accur
    C_List = C_List.append(pd.Series(i))
    Acc_List = Acc_List.append(pd.Series(round(scores.mean(),4)))
DTFrame= pd.concat([C_List,Acc_List],axis=1,keys=['poly_values','Accuracy_val
DTFrame['Accuracy_values'] = DTFrame['Accuracy_values'].apply(lambda x : str(
DTFrame
```

Out[300]:

	poly_values	Accuracy_values
0	2	0.8512
0	3	0.946
0	4	0.8314
0	5	0.8662
0	6	0.7752

In [301]:
```
1 sns.lineplot(x=C_List,y=Acc_List,color='red')
2 plt.xlabel('degrees for SVC ')
3 plt.ylabel('Cross-Validated Accuracy')
```

Out[301]: Text(0,0.5,'Cross-Validated Accuracy')



In [302]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder,StandardScaler
import time
dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
train_set,test_set = train_test_split(dfnew,random_state=9,test_size=0.2,stra
sc=StandardScaler()
train_set.loc[:,'kurt':'mindom']= sc.fit_transform(train_set.loc[:,'kurt':'mi
#display(test_set.loc[:,'kurt':'sd'])
svc = SVC(probability=True)
start_time=time.time()
params=dict(C=[0.6,1.0],kernel=['poly'],degree=[2,3,4],random_state=[5,10,13,
for i in range(3,10):
    grid=RandomizedSearchCV(svc,params,cv=i).fit(train_set.loc[:,'kurt':'mind
    print('grid best score:  for class weight ',grid.best_score_)
    print('grid best parameters: ',grid.best_params_)
```

```
grid best score:  for class weight  0.9451026856240127
grid best parameters:  {'random_state': 45, 'kernel': 'poly', 'degree': 3, 'cla
ss_weight': {0: 1}, 'C': 1.0}
grid best score:  for class weight  0.9494470774091627
grid best parameters:  {'random_state': 5, 'kernel': 'poly', 'degree': 3, 'clas
s_weight': {0: 1}, 'C': 1.0}
grid best score:  for class weight  0.9478672985781991
grid best parameters:  {'random_state': 10, 'kernel': 'poly', 'degree': 3, 'cla
ss_weight': {0: 1}, 'C': 1.0}
grid best score:  for class weight  0.9494470774091627
grid best parameters:  {'random_state': 10, 'kernel': 'poly', 'degree': 3, 'cla
ss_weight': {0: 1}, 'C': 1.0}
grid best score:  for class weight  0.9490521327014217
grid best parameters:  {'random_state': 10, 'kernel': 'poly', 'degree': 3, 'cla
ss_weight': {0: 1}, 'C': 1.0}
grid best score:  for class weight  0.9510268562401264
grid best parameters:  {'random_state': 45, 'kernel': 'poly', 'degree': 3, 'cla
ss_weight': {0: 1}, 'C': 1.0}
grid best score:  for class weight  0.9502369668246445
grid best parameters:  {'random_state': 45, 'kernel': 'poly', 'degree': 3, 'cla
ss_weight': {0: 1}, 'C': 1.0}
```

In [303]:
```
# 5 fold Cross validation produces the best score
```

In [304]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV,train_tes
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder,StandardScaler
import time
dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
train_set,test_set = train_test_split(dfnew,random_state=5,test_size=0.2,stra
sc=StandardScaler()
train_set.loc[:,'kurt':'mindom']= sc.fit_transform(train_set.loc[:,'kurt':'mi
#display(test_set.loc[:,'kurt':'sd'])
svc1 = SVC(probability=True)
start_time=time.time()
params=dict(C=[0.6,1.0],kernel=['poly'],degree=[1,2,3,4],random_state=[5,10,1
grid=RandomizedSearchCV(svc1,params,cv=5).fit(train_set.loc[:,'kurt':'mindom'
print('grid best score:  for class weight ',grid.best_score_)
print('grid best parameters: ',grid.best_params_)
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
grid best score:  for class weight  0.9699842022116903
grid best parameters:  {'random_state': 13, 'kernel': 'poly', 'degree': 1, 'cla
ss_weight': {0: 1}, 'C': 1.0}
Execution time: 22.586835384368896 ms
```

In [305]:
```
test_set.loc[:,'kurt':'mindom']=sc.transform(test_set.loc[:,'kurt':'mindom'])
print('Test_accuracy: ',grid.score(test_set.loc[:,'kurt':'mindom'],test_set['
```

```
Test_accuracy:  0.9747634069400631
```

In [306]:
```
from sklearn.metrics import confusion_matrix,precision_recall_fscore_support,
from sklearn.preprocessing import LabelEncoder
dfnew['label'] = (LabelEncoder().fit_transform(dfnew['label']))
#display(test_set.loc[:,'kurt':'sd'])
svc1.fit(train_set.loc[:,'kurt':'mindom'],train_set['label'])
y_pred = svc1.predict(test_set.loc[:,'kurt':'mindom'])
print('The test dataset confusion_matrix is :\n',confusion_matrix(test_set['l
```

```
The test dataset confusion_matrix is :
 [[314   3]
 [ 11 306]]
```

In [309]:
```
Label_probabability = pd.Series()
for each in svc1.predict_proba((test_set.loc[:,'kurt':'mindom'])):
    Label_probabability = Label_probabability.append(pd.Series(np.amax(each))
Label_probabability_DTFrame = pd.concat([Label_probabability.reset_index(drop
                                         test_set['label'].reset_index(drop=T
                                         pd.Series(svc1.predict(test_set.loc[
                                        keys=['Probability','Actual','Predict
Label_probabability_DTFrame['SameOrNot']=abs(Label_probabability_DTFrame.Actu
Label_probabability_DTFrame['Incorrect_Predictions']=Label_probabability_DTFr
Label_probabability_DTFrame.head()
```

Out[309]:

	Probability	Actual	Predicted	SameOrNot	Incorrect_Predictions
0	$1.00	0	0	0	No
1	$1.00	1	1	0	No
2	$0.99	1	1	0	No
3	$1.00	0	0	0	No
4	$0.99	0	0	0	No

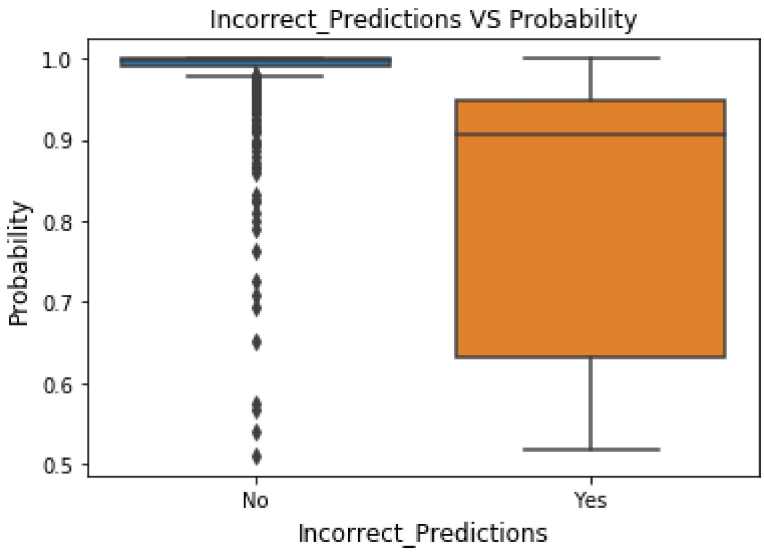
In [312]:
```
Label_probabability_DTFrame[Label_probabability_DTFrame.SameOrNot !=0]
```

Out[312]:

	Probability	Actual	Predicted	SameOrNot	Incorrect_Predictions
73	$1.00	1	0	1	Yes
130	$0.53	1	0	1	Yes
144	$0.87	1	0	1	Yes
296	$0.56	0	1	1	Yes
317	$1.00	1	0	1	Yes
349	$0.82	1	0	1	Yes
352	$0.91	0	1	1	Yes
417	$0.92	1	0	1	Yes
479	$0.99	1	0	1	Yes
524	$0.52	1	0	1	Yes
594	$0.93	1	0	1	Yes
604	$0.90	1	0	1	Yes
623	$0.57	0	1	1	Yes
626	$0.95	1	0	1	Yes

In [314]:
```
1 sns.boxplot(x=Label_probabability_DTFrame.Incorrect_Predictions,y=Label_proba
2 plt.xlabel('Incorrect_Predictions',fontsize=12)
3 plt.ylabel('Probability',fontsize=12)
4 plt.xlabel('Incorrect_Predictions',fontsize=12)
5 plt.title('Incorrect_Predictions VS Probability',fontsize=12)
```

Out[314]: Text(0.5,1,'Incorrect_Predictions VS Probability')



In [ ]:
```
1
```