SVM and its Kernal Exploration

October 28, 2017

1 Author: Pradeep Sathyamurthy

2 Date Started: Oct 15, 2017

3 Last Modified Date: Oct 28, 2017

4 Topic Focussed: SVM Kernals

5 Dataset: Voice Dataset for Gender Regonition from Kaggle

6 Introduction:

6.0.1 1. SVM is a kernal trick which can be used for both supervised and unsupervised learning.

6.0.2 2. As part of this case study I am going to apply SVM for a supervised learning as I am aware of the class labels to be classified.

6.0.3 3. Thus in this notebook I will be using the voice dataset obtained from URL sighted below to classify if the parameters for a particular instances is a male or a female

7 Objective of case study:

7.0.1 1. My main objective is to apply SVM and its different kernals and observe how the margin defined helps in improving the classification accuracy

7.0.2 2. I will try to tune different parameters in Kernal and choose the best tuning parameter wrt SVM to classify the dataset

7.0.3 3. I will also apply different classification techniques and compare the results obtained from these with result obtained from SVM classifier

8 Steps involved in this case study

8.0.1 1. Data Manipulation

8.0.2 2. Setting a benchmark accuracy for classifiers using Raw Data & Naive Bayes

8.0.3 3. Exploratory Data Analysis

8.0.4 4. Data Munging and Partition

8.0.5 5. Validating the cleaned dataset with benchmark accuracy obtained

8.0.6 6. Core Model Building - Applying Different Kernals for SVM

6.1. Linear Kernal SVM

6.2. RBF Kernal SVM

6.3. Polynomial Kernal SVM

6.4. Sigmoidal Kernal SVM

8.0.7 7. Perfomance Evaluation on Different Kernals for SVM with 10-fold cross validation

7.1. Evaluation on Linear Kernal SVM

7.2. Evaluation on RBF Kernal SVM

7.3. Evaluation on Polynomial Kernal SVM (is not in this notebook as computation time was high)

7.4. Evaluation on Sigmoidal Kernal SVM

8.0.8 8. Parameter tuning on Different Kernals for SVM with 10-fold cross validation

8.1. Tuning on Linear Kernal SVM

8.2. Tuning on RBF Kernal SVM

8.4. Tuning on Polynomial Kernal SVM

8.0.9 9. Choosing best Kernals Parameters with grid search

8.0.10 10. Visualization of kernal Margin and boundries considereing on two columns meanfun & sp.ent

8.0.11 11. Building a Decision Tree

8.0.12 12. Building a KNN model

8.0.13 13. Comparing individual classifier results

8.0.14 14. Ensemble Learning

8.0.15 15. Reporting and Discussing final results

9 Dataset URL:

http://www.primaryobjects.com/2016/06/22/identifying-the-gender-of-a-voice-using-machine-learning/

10 Importing Packages:

```
In [1]: import pandas as pd # for data handling
        import numpy as np # for data manipulation
        import sklearn as sk
        from matplotlib import pyplot as plt # for plotting
        from sklearn.preprocessing import LabelEncoder # For encoding class variables
        from sklearn.model_selection import train_test_split # for train and test split
```

```
from sklearn.svm import SVC # to built svm model
from sklearn import svm # inherits other SVM objects
from sklearn import metrics # to calculate classifiers accuracy
from sklearn.model_selection import cross_val_score # to perform cross validation
from sklearn.preprocessing import StandardScaler # to perform standardization
from sklearn.model_selection import GridSearchCV # to perform grid search for all classi
from sklearn import tree # to perform decision tree classification
from sklearn import neighbors # to perform knn
from sklearn import naive_bayes # to perform Naive Bayes
from sklearn.metrics import classification_report # produce classifier reports
from sklearn.ensemble import RandomForestClassifier # to perform ensemble bagging - rand
from sklearn.ensemble import AdaBoostClassifier # to perform ensemble boosting
% matplotlib inline
```

In [2]:
```
%pwd
```

Out[2]: 'D:\\Courses\\CSC529 - Python\\Case_Study2\\final'

In [3]:
```
%ls
```

```
 Volume in drive D is DATA
 Volume Serial Number is 3048-DECC

 Directory of D:\Courses\CSC529 - Python\Case_Study2\final

28-10-2017  13:24    <DIR>          .
28-10-2017  13:24    <DIR>          ..
28-10-2017  13:24    <DIR>          .ipynb_checkpoints
28-10-2017  13:24           681,491 SVM and its Kernal Exploration.ipynb
26-08-2016  09:29         1,065,381 voice.csv
               2 File(s)      1,746,872 bytes
               3 Dir(s)  487,571,648,512 bytes free
```

11 Step-1: Data Manipulation

11.0.1 Reading Data:

In [4]:
```
# Reding the data as pandas dataframe
data_raw = pd.read_csv('voice.csv',sep=',')
data_raw.shape
```

Out[4]: (3168, 21)

In [5]:
```
# Verifying if all records are read
data_raw.head(3)
```

Out[5]:
```
   meanfreq        sd    median       Q25       Q75       IQR       skew  \
0  0.059781  0.064241  0.032027  0.015071  0.090193  0.075122  12.863462
```

```
1  0.066009  0.067310  0.040229  0.019414  0.092666  0.073252  22.423285
2  0.077316  0.083829  0.036718  0.008701  0.131908  0.123207  30.757155

          kurt    sp.ent       sfm  ...   centroid   meanfun    minfun  \
0   274.402906  0.893369  0.491918  ...   0.059781  0.084279  0.015702
1   634.613855  0.892193  0.513724  ...   0.066009  0.107937  0.015826
2  1024.927705  0.846389  0.478905  ...   0.077316  0.098706  0.015656

     maxfun   meandom    mindom    maxdom   dfrange   modindx  label
0  0.275862  0.007812  0.007812  0.007812  0.000000  0.000000   male
1  0.250000  0.009014  0.007812  0.054688  0.046875  0.052632   male
2  0.271186  0.007990  0.007812  0.015625  0.007812  0.046512   male

[3 rows x 21 columns]
```

In [6]:
```python
# having the headers handy
columns = data_raw.columns
print(columns)
```

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

11.0.2 Data Types of Features:

In [7]:
```python
# Data type
df = pd.DataFrame(data_raw.dtypes,columns=['Data Type'])
df = df.reset_index()
df.columns = ['Attribute Name','Data Type']
df
```

Out[7]:

	Attribute Name	Data Type
0	meanfreq	float64
1	sd	float64
2	median	float64
3	Q25	float64
4	Q75	float64
5	IQR	float64
6	skew	float64
7	kurt	float64
8	sp.ent	float64
9	sfm	float64
10	mode	float64
11	centroid	float64
12	meanfun	float64
13	minfun	float64
14	maxfun	float64

```
15        meandom   float64
16         mindom   float64
17         maxdom   float64
18        dfrange   float64
19        modindx   float64
20          label    object
```

11.0.3 Checking for Missing Values:

```
In [9]: # Checking for any missing values in data and other junk values if any
        if data_raw.isnull() is True:
            print('There are missing records')
        else:
            print('No missing records')
```

```
No missing records
```

11.0.4 Seperating Independent and Target Variables:

```
In [46]: # let us seperate the independent and dependent variables seperately
         data_x = data_raw[columns[0:20]].copy()
         data_y = data_raw[columns[-1]].copy()
         print('Independent var: \n',data_x.head(3),'\n')
         print('Dependent var: \n',data_y.head(3))
```

```
Independent var:
    meanfreq        sd    median       Q25       Q75       IQR       skew  \
0  0.059781  0.064241  0.032027  0.015071  0.090193  0.075122  12.863462
1  0.066009  0.067310  0.040229  0.019414  0.092666  0.073252  22.423285
2  0.077316  0.083829  0.036718  0.008701  0.131908  0.123207  30.757155

          kurt    sp.ent       sfm  mode  centroid   meanfun    minfun  \
0   274.402906  0.893369  0.491918   0.0  0.059781  0.084279  0.015702
1   634.613855  0.892193  0.513724   0.0  0.066009  0.107937  0.015826
2  1024.927705  0.846389  0.478905   0.0  0.077316  0.098706  0.015656

     maxfun   meandom    mindom    maxdom   dfrange   modindx
0  0.275862  0.007812  0.007812  0.007812  0.000000  0.000000
1  0.250000  0.009014  0.007812  0.054688  0.046875  0.052632
2  0.271186  0.007990  0.007812  0.015625  0.007812  0.046512

Dependent var:
 0    male
1    male
2    male
Name: label, dtype: object
sample values of target values:
```

```
[1 1 1]
```

11.0.5 Target Variable Encoding:

```
In [ ]: # encoding the target variable from categorical values to binary form
        encode_obj = LabelEncoder()
        data_y = encode_obj.fit_transform(data_y)
        print('sample values of target values:\n',data_y[0:3])
```

11.0.6 Inference:

1. All independent variables are continuous in nature

2. While the target variables seems binary in nature of typr str

3. There are totally 3168 rows with 21 columns

4. There are no missing values in any of the record.

12 Step-2: Setting a benchmark accuracy for classifiers using Raw Data & Naive Bayes

```
In [47]: # Let us do a 80-20 split
         test_x_train,test_x_test,test_y_train,test_y_test = train_test_split(data_x,data_y,trai
```

```
In [48]: nbclf = naive_bayes.GaussianNB()
         nbclf = nbclf.fit(test_x_train, test_y_train)
         nbpreds_test = nbclf.predict(test_x_test)
         print('Accuracy obtained from train-test split on training data is:',nbclf.score(test_x
         print('Accuracy obtained from train-test split on testing data is:',nbclf.score(test_x_
```

```
Accuracy obtained from train-test split on training data is: 0.876479873717
Accuracy obtained from train-test split on testing data is: 0.869085173502
```

```
In [49]: test_eval_result = cross_val_score(nbclf, data_x, data_y, cv=10, scoring='accuracy')
         print('Accuracy obtained from 10-fold cross validation on actual raw data is:',test_eva
```

```
Accuracy obtained from 10-fold cross validation on actual raw data is: 0.856713239392
```

12.0.1 Inference:

1. Naive Bayes is a naive method which uses the probablistic theory to classify a target table

2. Since, it has a fast computation power in training a data and testing it, we can use it as a base method to validate our dataset

3. Accuracy obtained from this can be set as a bench mark for any classifier that we will start to work going forward

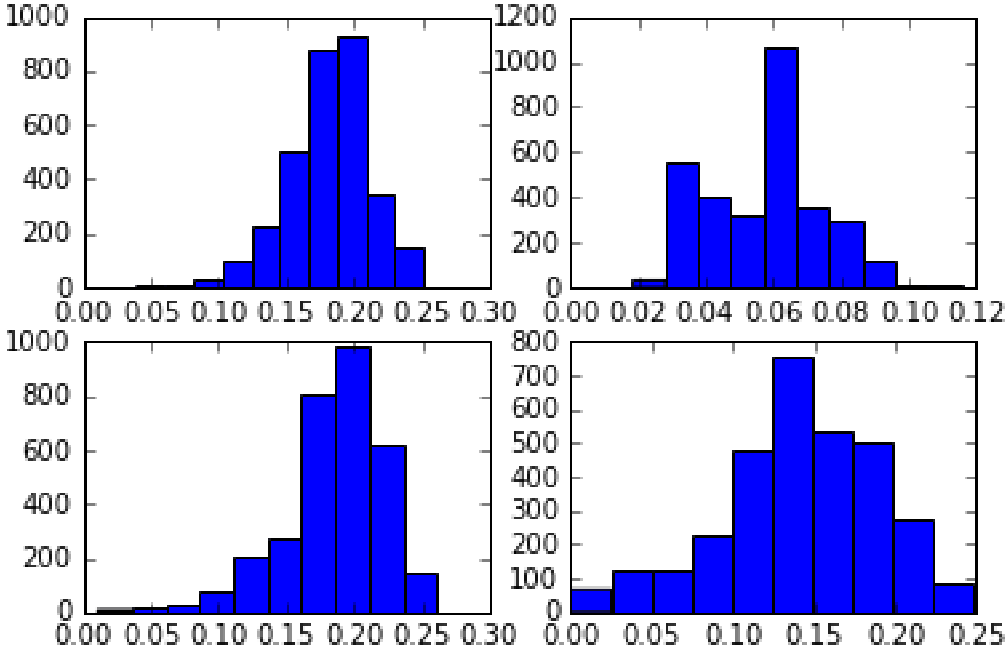
4. Using the raw data and classifying the dataset with Naive implementation with cross validation i obtained an accuracy of 0.85671

5. Thus, any data clean up we do further or any classifier model we build should not decrease the accuracy that we obtained here and it must always yeald a high or atleast an accuracy equal to 0.85671, else we will discard the data cleaning done or classifier built to classify the target variable.

13 Step-3: Exploratory Data Analysis (EDA)

```
In [50]: ### plotting the independent variables
         plt.subplot(221)
         plt.hist(data_x['meanfreq'])
         plt.subplot(222)
         plt.hist(data_x['sd'])
         plt.subplot(223)
         plt.hist(data_x['median'])
         plt.subplot(224)
         plt.hist(data_x['Q25'])
```

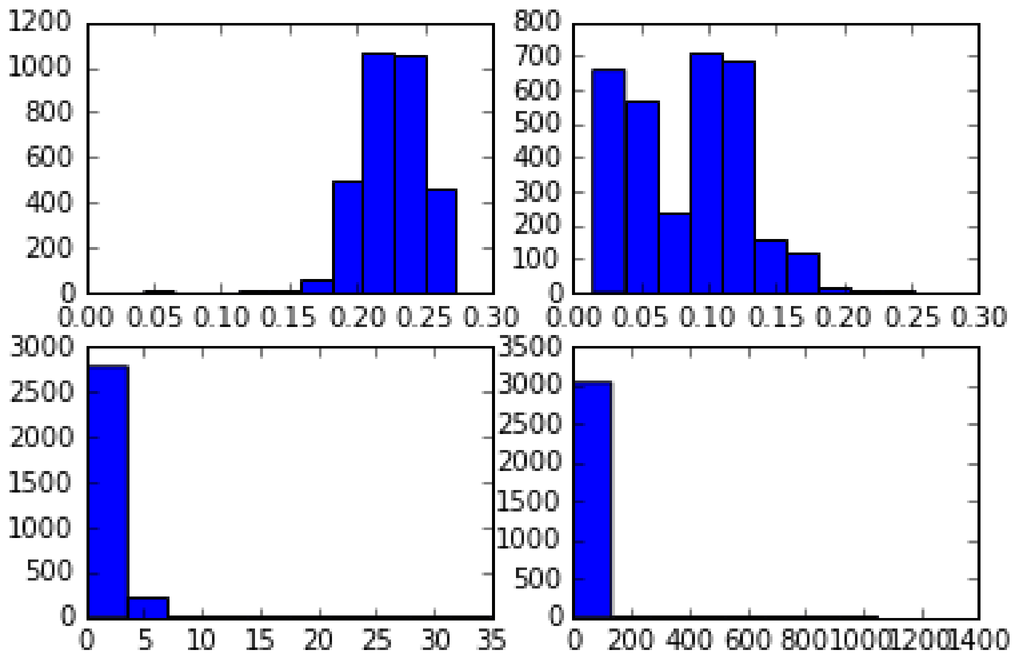
```
Out[50]: (array([  67.,  121.,  127.,  227.,  479.,  755.,  532.,  500.,  271.,   89.]),
          array([  2.28758170e-04,   2.49405762e-02,   4.96523943e-02,
                   7.43642124e-02,   9.90760304e-02,   1.23787848e-01,
                   1.48499667e-01,   1.73211485e-01,   1.97923303e-01,
                   2.22635121e-01,   2.47346939e-01]),
          <a list of 10 Patch objects>)
```



1. Variables meanfreq, sd, median, Q25 are normally distributed

In [51]:
```
plt.subplot(221)
plt.hist(data_x['Q75'])
plt.subplot(222)
plt.hist(data_x['IQR'])
plt.subplot(223)
plt.hist(data_x['skew'])
plt.subplot(224)
plt.hist(data_x['kurt'])
```

Out[51]:
```
(array([ 3037.,    23.,    13.,    13.,    20.,    20.,    21.,    12.,
            5.,     4.]),
 array([    2.06845549,   132.82289868,   263.57734187,   394.33178505,
          525.08622824,   655.84067143,   786.59511462,   917.34955781,
         1048.10400099,  1178.85844418,  1309.61288737]),
 <a list of 10 Patch objects>)
```




```
In [52]: print('Mean and Median value for Q75 is: ',[data_x.Q75.mean(), data_x.Q75.median()])
         print('Mean and Median value for IQR is: ',[data_x.IQR.mean(), data_x.IQR.median()])
```

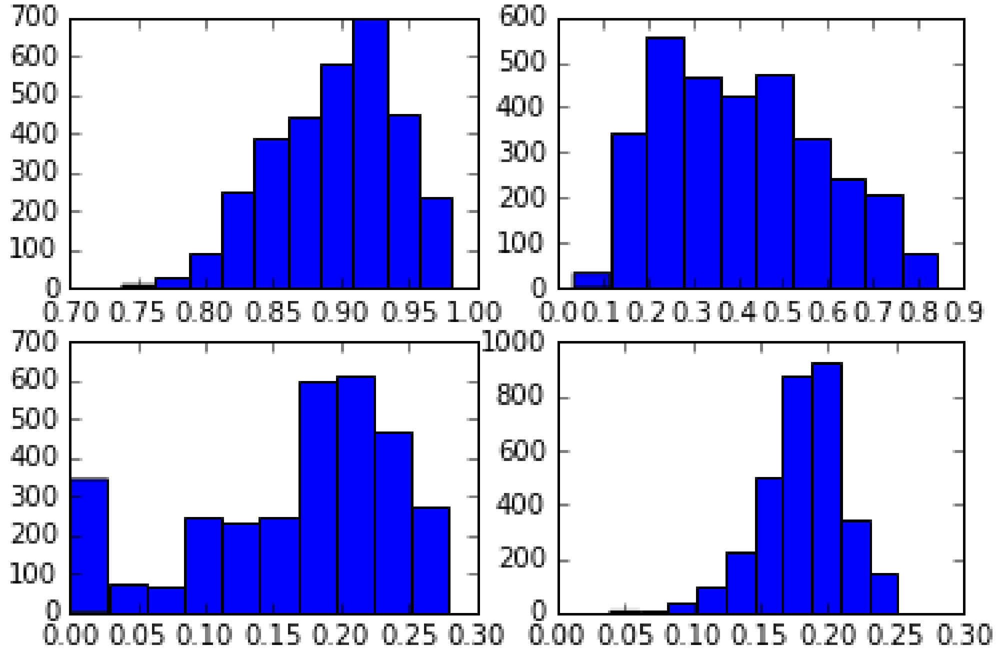
```
Mean and Median value for Q75 is:  [0.22476496141497235, 0.22568421491103252]
Mean and Median value for IQR is:  [0.0843093709296532, 0.09427995391705071]
```

1. From above visualization and summary stats we can say Q75 is normally distributed

2. While IQR, skew and kurt are skewed to right

```
In [53]: plt.subplot(221)
         plt.hist(data_x['sp.ent'])
         plt.subplot(222)
         plt.hist(data_x['sfm'])
         plt.subplot(223)
         plt.hist(data_x['mode'])
         plt.subplot(224)
         plt.hist(data_x['centroid'])
```

```
Out[53]: (array([   4.,    8.,   35.,  101.,  225.,  502.,  876.,  925.,  347.,  145.]),
          array([ 0.03936334,  0.06053938,  0.08171543,  0.10289147,  0.12406751,
                  0.14524355,  0.16641959,  0.18759563,  0.20877168,  0.22994772,
                  0.25112376]),
          <a list of 10 Patch objects>)
```

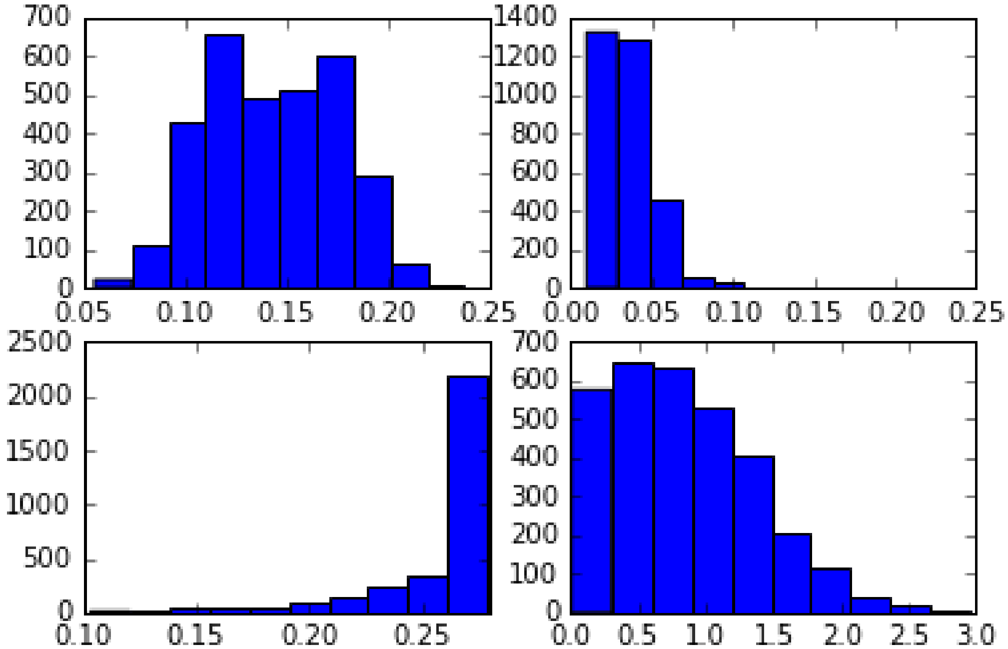


```
In [54]: print('Mean and Median value for Mode is: ',[data_x['mode'].mean(), data_x['mode'].medi

Mean and Median value for Mode is:  [0.1652817967518845, 0.18659863945578248]
```

1. sp.ent, s.fm, centroid are normally distributed

2. While mode is skewed

```
In [55]: plt.subplot(221)
         plt.hist(data_x['meanfun'])
         plt.subplot(222)
         plt.hist(data_x['minfun'])
         plt.subplot(223)
         plt.hist(data_x['maxfun'])
         plt.subplot(224)
         plt.hist(data_x['meandom'])

Out[55]: (array([ 576.,  645.,  634.,  533.,  404.,  203.,  113.,   41.,   16.,    3.]),
          array([ 0.0078125 ,  0.30279948,  0.59778646,  0.89277344,  1.18776042,
                  1.4827474 ,  1.77773438,  2.07272135,  2.36770833,  2.66269531,
                  2.95768229]),
          <a list of 10 Patch objects>)
```

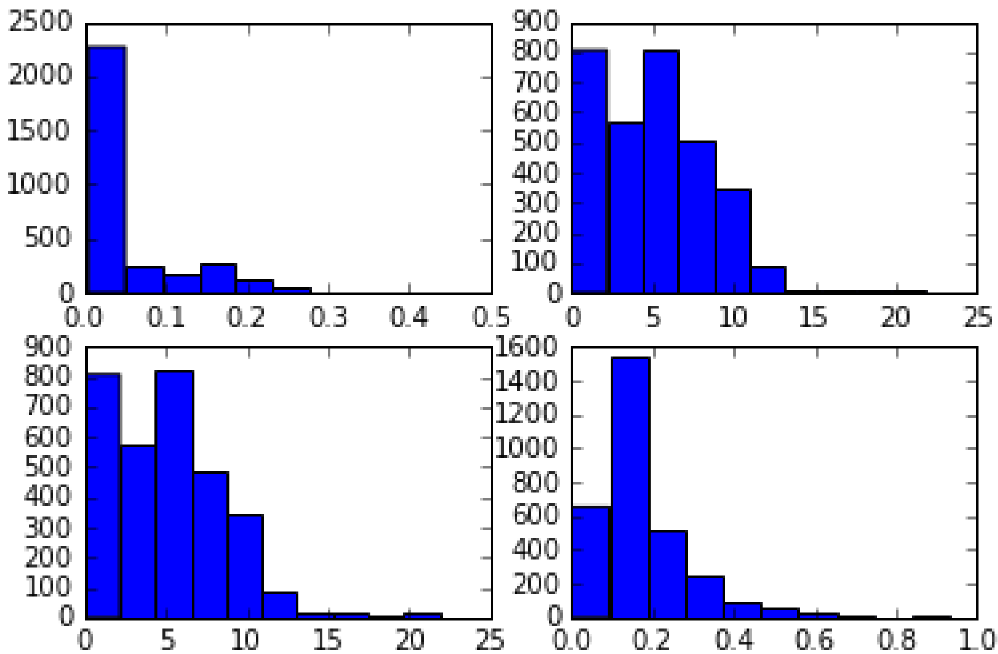


1. Variables meanfun is normally distributed

2. While variables minfun, maxfun, meandom are skewed

```
In [56]: plt.subplot(221)
         plt.hist(data_x['mindom'])
         plt.subplot(222)
         plt.hist(data_x['maxdom'])
         plt.subplot(223)
         plt.hist(data_x['dfrange'])
         plt.subplot(224)
         plt.hist(data_x['modindx'])
```

```
Out[56]: (array([  653.,  1545.,   514.,   248.,    96.,    56.,    30.,    16.,
                     4.,     6.]),
          array([ 0.        ,  0.09323741,  0.18647482,  0.27971223,  0.37294964,
                  0.46618705,  0.55942446,  0.65266187,  0.74589928,  0.83913669,
                  0.9323741 ]),
          <a list of 10 Patch objects>)
```



1. Variables modindx is normally distributed

2. While variables mindom, maxdom and dfrange are skewed

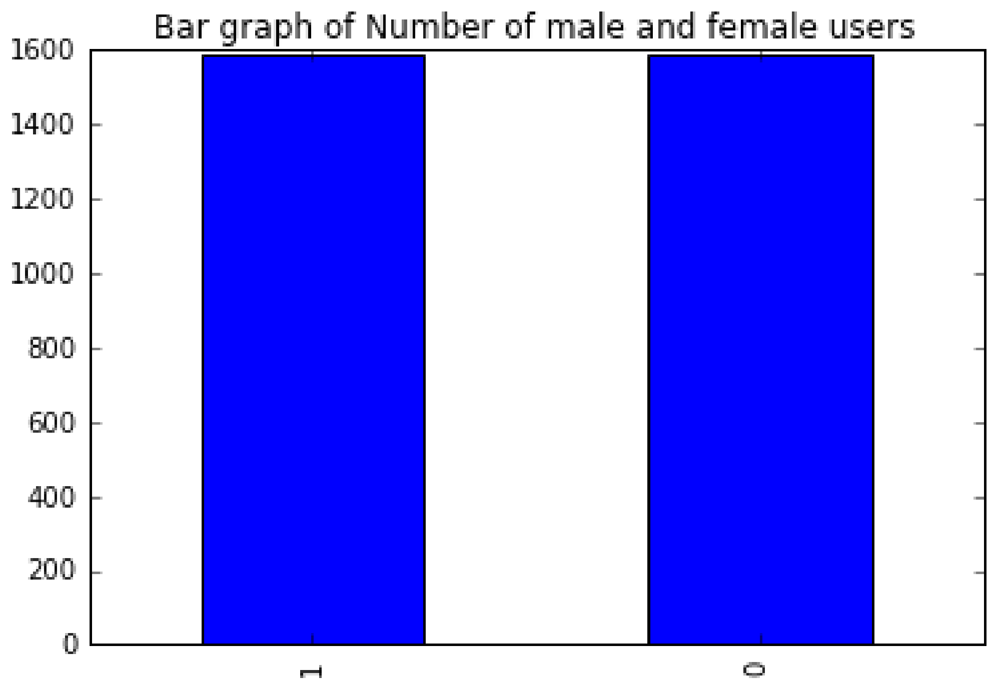
In [57]:
```
# let us do a descriptive statistics
means = data_x.describe().loc['mean']
medians = data_x.describe().loc['50%']
pd.DataFrame([means,medians], index=['mean','median'])
```

Out[57]:
```
        meanfreq        sd    median       Q25       Q75       IQR      skew  \
mean    0.180907  0.057126  0.185621  0.140456  0.224765  0.084309  3.140168
median  0.184838  0.059155  0.190032  0.140286  0.225684  0.094280  2.197101

             kurt    sp.ent       sfm      mode  centroid   meanfun    minfun  \
mean    36.568461  0.895127  0.408216  0.165282  0.180907  0.142807  0.036802
median   8.318463  0.901767  0.396335  0.186599  0.184838  0.140519  0.046110

          maxfun   meandom    mindom    maxdom   dfrange   modindx
mean    0.258842  0.829211  0.052647  5.047277  4.994630  0.173752
median  0.271186  0.765795  0.023438  4.992188  4.945312  0.139357
```

In [58]:
```
# Distribution of target variables
print(pd.Series(data_y).value_counts())
pd.Series(data_y).value_counts().plot(kind='bar', title='Bar graph of Number of male an
```

```
1    1584
0    1584
dtype: int64
```

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x247b8699550>



13.0.1 Inference:

1. Lets explain the skeweness in data from above visualization and summary stats

2. Irrespectve to viz of histogram, we can also infer those attributes with mean and median values almost equal have gaussian distribution.

3. Thus, variables meanfreq, sd, median, Q25, Q75, sp.ent, sfm, centroid, meanfun are Normally distributed

4. Variables skew, kurt, minfun, maxfun, meandom, mindom, maxdom, dfrange, midindex, IQR, mode are skewed

5. Exceptable range of voice freq for a human as per wiki is between 0.085 and 0.255KHz and hence we will remove any values from the dataset below 0.085 and above 0.255 assuming it to be a outlier based on domain knowledge

6. Our target variables (1 = Male and 0 = Female) are symmetrical in nature with equal count of 1584 records for both Male and Female

14 Step-4: Data Munging and Partition

14.0.1 Data Cleaning:

1.Exceptable range of voice freq for a human as per wiki is between 0.085 and 0.255KHz and hence we will identify the variable which has this frequncy information and remove them assuming it to be a outlier based on domain knowledge

2.In our data set meanfun is the variable which have the value of Fundamental frequency

3. As per the sitation given in wiki we can say that typical adult male will have a fundamental frequency from 85 to 180 Hz and typical adult female from 165 to 255 Hz

4. Thus, from given dataset, we will filter values based on meanfun whose values less than 0.085 and greater than 0.18 for male and values less than 0.165 and greater than 0.255 for female and consider them as outliers and remove them.

In [59]:
```
# Actual Raw Data size
data_raw.shape
```

Out[59]: (3168, 21)

In [60]:
```
# Filtering ouliers from male category
male_funFreq_outlier_index = data_raw[((data_raw['meanfun'] < 0.085) | (data_raw['meanf
                                      (data_raw['label'] == 'male')].index
male_funFreq_outlier_index = list(male_funFreq_outlier_index)
data_raw[((data_raw['meanfun'] < 0.085) | (data_raw['meanfun'] > 0.180)) & (data_raw['l
```

Out[60]: (70, 21)

In [61]:
```
# Filtering ouliers from female category
female_funFreq_outlier_index = data_raw[((data_raw['meanfun'] < 0.165) | (data_raw['mea
                                        (data_raw['label'] == 'female')].index
female_funFreq_outlier_index = list(female_funFreq_outlier_index)
data_raw[((data_raw['meanfun'] < 0.165) | (data_raw['meanfun'] > 0.255)) & (data_raw['l
```

Out[61]: (640, 21)

In [62]:
```
index_to_remove = male_funFreq_outlier_index + female_funFreq_outlier_index
len(index_to_remove)
```

Out[62]: 710

In [63]:
```
# Thus, we need to remove 710 rows from both data_x and data_y using the index obtained
# Preparing final dataset for model building
data_x = data_x.drop(index_to_remove,axis=0)
data_x.shape
```

Out[63]: (2458, 20)

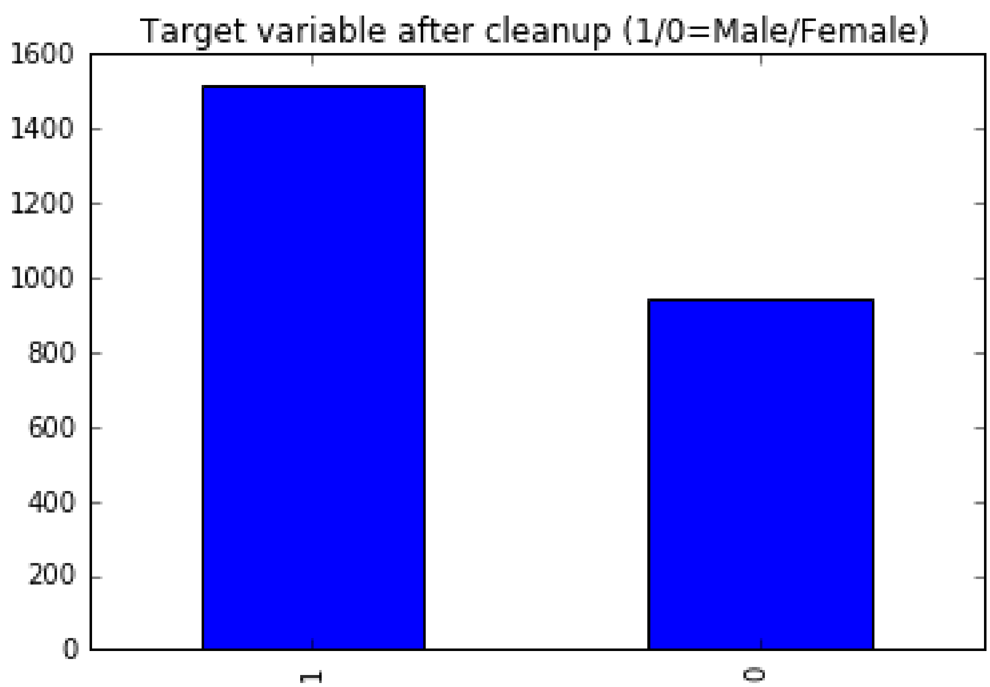
```
In [65]: # Target dataset
         data_y = pd.Series(data_y).drop(index_to_remove,axis=0)
         data_y.shape
```

Out[65]: (2458,)

```
In [69]: # Distribution of target variables after cleanup
         print(data_y.value_counts())
         data_y.value_counts().plot(kind='bar', title='Target variable after cleanup (1/0=Male/F
```

```
1    1514
0     944
dtype: int64
```

Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x247b8773748>



14.0.2 Normalization:

1. In this dataset meanfreq, median, Q25, Q75, IQR are the only variables associated with unit kHz

2. let us normalize these variables to make them unit free

3. we will apply the z-score normalization for meanfreq, median, Q25, Q75

4. we will apply min-max normalization for IQR

```
In [70]: # Z-score Normalization
         z_score_norm = lambda colname: (data_x[colname]- data_x[colname].mean())/(data_x[colnam
         min_max_norm = lambda colname: (data_x[colname]- data_x[colname].min())/(data_x[colname
```

14.0.3 Creating Partially Normalized Data

```
In [72]: data_x1 = data_x.copy()
         data_x1['z_meanfreq'] = z_score_norm('meanfreq')
         data_x1['z_median'] = z_score_norm('median')
         data_x1['z_Q25'] = z_score_norm('Q25')
         data_x1['z_Q75'] = z_score_norm('Q75')
         data_x1['Norm_IQR'] = min_max_norm('IQR')
```

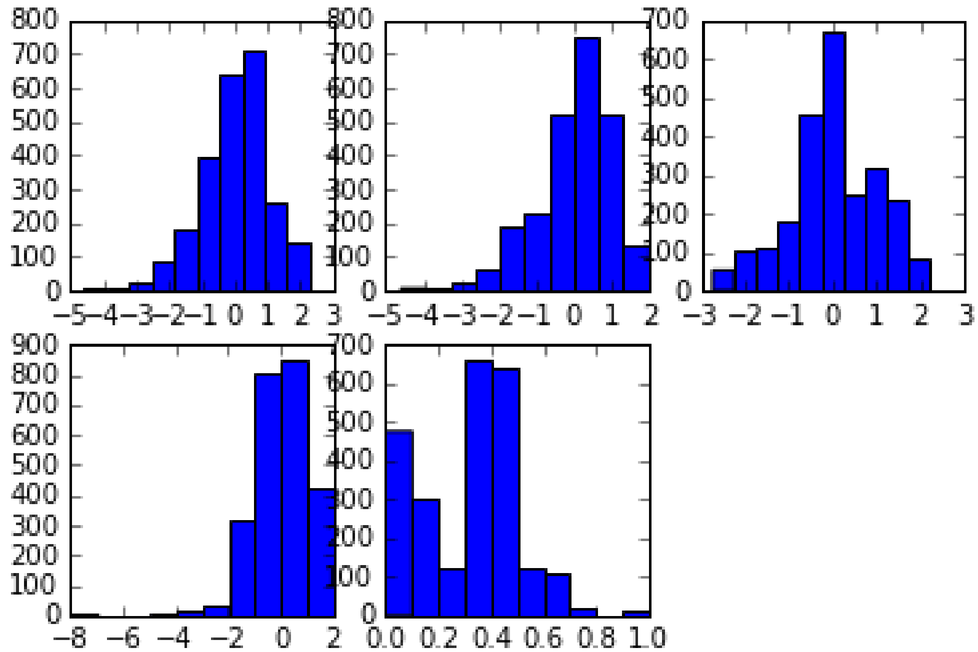
```
In [73]: # Lets now drop the original column from data_x as we have these as backup in data_raw
         data_x1 = data_x1.drop(['meanfreq','median','Q25','Q75','IQR'],axis=1)
```

```
In [74]: data_x1.head(3)
```

```
Out[74]:         sd       skew         kurt    sp.ent       sfm      mode  centroid  \
         1  0.067310  22.423285   634.613855  0.892193  0.513724  0.000000  0.066009
         2  0.083829  30.757155  1024.927705  0.846389  0.478905  0.000000  0.077316
         3  0.072111   1.232831     4.177296  0.963322  0.727232  0.083878  0.151228

             meanfun    minfun    maxfun   meandom    mindom    maxdom   dfrange  \
         1  0.107937  0.015826  0.250000  0.009014  0.007812  0.054688  0.046875
         2  0.098706  0.015656  0.271186  0.007990  0.007812  0.015625  0.007812
         3  0.088965  0.017798  0.250000  0.201497  0.007812  0.562500  0.554688

             modindx  z_meanfreq  z_median     z_Q25     z_Q75  Norm_IQR
         1  0.052632   -3.720840 -3.828504 -2.355691 -5.781683  0.246961
         2  0.046512   -3.354719 -3.920638 -2.570261 -4.094335  0.457148
         3  0.247119   -0.961373 -0.737091 -0.810072 -0.824403  0.407358
```

```
In [75]: # Plotting the normalized columns
         # we could see that z-score norm variables have mean 0 and standard deviation 1
         # And the min-max norm varibales value are confined between 0-1 and stays positive
         plt.subplot(231)
         plt.hist(data_x1['z_meanfreq'])
         plt.subplot(232)
         plt.hist(data_x1['z_median'])
         plt.subplot(233)
         plt.hist(data_x1['z_Q25'])
         plt.subplot(234)
         plt.hist(data_x1['z_Q75'])
         plt.subplot(235)
         plt.hist(data_x1['Norm_IQR'])
```

Out[75]: (array([476., 299., 126., 660., 639., 125., 107.,   16.,   1.,   9.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.]),
 <a list of 10 Patch objects>)



14.0.4 Handling Multicollinearity:

In [76]:
```
# let us see the correlation in data
corr_mat = data_x1.corr()
corr_mat
```

Out[76]:

	sd	skew	kurt	sp.ent	sfm	mode \
sd	1.000000	0.268792	0.305891	0.748671	0.841054	-0.518399
skew	0.268792	1.000000	0.978731	-0.186965	0.052345	-0.404677
kurt	0.305891	0.978731	1.000000	-0.103409	0.098550	-0.377308
sp.ent	0.748671	-0.186965	-0.103409	1.000000	0.882300	-0.345448
sfm	0.841054	0.052345	0.098550	0.882300	1.000000	-0.487812
mode	-0.518399	-0.404677	-0.377308	-0.345448	-0.487812	1.000000
centroid	-0.761064	-0.292907	-0.295208	-0.653044	-0.798872	0.703159
meanfun	-0.466995	-0.080008	-0.119739	-0.558854	-0.434546	0.305558
minfun	-0.334265	-0.174601	-0.179806	-0.319500	-0.349963	0.353356
maxfun	-0.128949	-0.034663	-0.015390	-0.157173	-0.193208	0.170668
meandom	-0.445008	-0.308871	-0.283337	-0.302235	-0.412508	0.475223
mindom	-0.371717	-0.068304	-0.105741	-0.318209	-0.312801	0.209490
maxdom	-0.447752	-0.270584	-0.248240	-0.325037	-0.410816	0.456170
dfrange	-0.441069	-0.269368	-0.246347	-0.319313	-0.405195	0.452416

modindx	0.124674	-0.130912	-0.173645	0.166691	0.190494	-0.214109
z_meanfreq	-0.761064	-0.292907	-0.295208	-0.653044	-0.798872	0.703159
z_median	-0.593734	-0.254069	-0.245334	-0.544028	-0.681039	0.710436
z_Q25	-0.864655	-0.280212	-0.311618	-0.699490	-0.787767	0.602051
z_Q75	-0.217083	-0.211600	-0.167253	-0.244650	-0.419489	0.536980
Norm_IQR	0.899810	0.214066	0.275422	0.690033	0.698088	-0.414729

	centroid	meanfun	minfun	maxfun	meandom	mindom \
sd	-0.761064	-0.466995	-0.334265	-0.128949	-0.445008	-0.371717
skew	-0.292907	-0.080008	-0.174601	-0.034663	-0.308871	-0.068304
kurt	-0.295208	-0.119739	-0.179806	-0.015390	-0.283337	-0.105741
sp.ent	-0.653044	-0.558854	-0.319500	-0.157173	-0.302235	-0.318209
sfm	-0.798872	-0.434546	-0.349963	-0.193208	-0.412508	-0.312801
mode	0.703159	0.305558	0.353356	0.170668	0.475223	0.209490
centroid	1.000000	0.474303	0.371094	0.255896	0.547437	0.252269
meanfun	0.474303	1.000000	0.345183	0.325146	0.246622	0.163801
minfun	0.371094	0.345183	1.000000	0.175142	0.305936	0.123851
maxfun	0.255896	0.325146	0.175142	1.000000	0.320966	-0.239510
meandom	0.547437	0.246622	0.305936	0.320966	1.000000	0.072956
mindom	0.252269	0.163801	0.123851	-0.239510	0.072956	1.000000
maxdom	0.524504	0.257634	0.232142	0.341457	0.801566	0.012371
dfrange	0.519982	0.254693	0.229921	0.345801	0.800298	-0.005680
modindx	-0.233599	-0.088519	0.043800	-0.393378	-0.222662	0.203165
z_meanfreq	1.000000	0.474303	0.371094	0.255896	0.547437	0.252269
z_median	0.927085	0.423266	0.336728	0.237218	0.488135	0.218869
z_Q25	0.925011	0.552781	0.336769	0.208284	0.473966	0.313023
z_Q75	0.758994	0.192549	0.236711	0.250017	0.417383	0.011247
Norm_IQR	-0.673457	-0.545740	-0.266933	-0.108211	-0.329437	-0.362716

	maxdom	dfrange	modindx	z_meanfreq	z_median	z_Q25 \
sd	-0.447752	-0.441069	0.124674	-0.761064	-0.593734	-0.864655
skew	-0.270584	-0.269368	-0.130912	-0.292907	-0.254069	-0.280212
kurt	-0.248240	-0.246347	-0.173645	-0.295208	-0.245334	-0.311618
sp.ent	-0.325037	-0.319313	0.166691	-0.653044	-0.544028	-0.699490
sfm	-0.410816	-0.405195	0.190494	-0.798872	-0.681039	-0.787767
mode	0.456170	0.452416	-0.214109	0.703159	0.710436	0.602051
centroid	0.524504	0.519982	-0.233599	1.000000	0.927085	0.925011
meanfun	0.257634	0.254693	-0.088519	0.474303	0.423266	0.552781
minfun	0.232142	0.229921	0.043800	0.371094	0.336728	0.336769
maxfun	0.341457	0.345801	-0.393378	0.255896	0.237218	0.208284
meandom	0.801566	0.800298	-0.222662	0.547437	0.488135	0.473966
mindom	0.012371	-0.005680	0.203165	0.252269	0.218869	0.313023
maxdom	1.000000	0.999837	-0.462587	0.524504	0.460816	0.468190
dfrange	0.999837	1.000000	-0.466282	0.519982	0.456893	0.462568
modindx	-0.462587	-0.466282	1.000000	-0.233599	-0.230005	-0.163478
z_meanfreq	0.524504	0.519982	-0.233599	1.000000	0.927085	0.925011
z_median	0.460816	0.456893	-0.230005	0.927085	1.000000	0.784584
z_Q25	0.468190	0.462568	-0.163478	0.925011	0.784584	1.000000

```
z_Q75       0.377937  0.377757 -0.239054    0.758994  0.742905  0.533413
Norm_IQR   -0.344284 -0.337758  0.061426   -0.673457 -0.516798 -0.885662

               z_Q75  Norm_IQR
sd         -0.217083  0.899810
skew       -0.211600  0.214066
kurt       -0.167253  0.275422
sp.ent     -0.244650  0.690033
sfm        -0.419489  0.698088
mode        0.536980 -0.414729
centroid    0.758994 -0.673457
meanfun     0.192549 -0.545740
minfun      0.236711 -0.266933
maxfun      0.250017 -0.108211
meandom     0.417383 -0.329437
mindom      0.011247 -0.362716
maxdom      0.377937 -0.344284
dfrange     0.377757 -0.337758
modindx    -0.239054  0.061426
z_meanfreq  0.758994 -0.673457
z_median    0.742905 -0.516798
z_Q25       0.533413 -0.885662
z_Q75       1.000000 -0.079667
Norm_IQR   -0.079667  1.000000
```

```
In [77]: for names in corr_mat.index:
             if len(corr_mat[(corr_mat.loc[names] > 0.9) & (corr_mat.loc[names].index != names)]
                 print('column', names,' correlates strongly with: ',corr_mat[(corr_mat.loc[name
                                                                               (corr_mat.loc[name
```

```
column skew  correlates strongly with:  Index(['kurt'], dtype='object')
column kurt  correlates strongly with:  Index(['skew'], dtype='object')
column centroid  correlates strongly with:  Index(['z_meanfreq', 'z_median', 'z_Q25'], dtype='ob
column maxdom  correlates strongly with:  Index(['dfrange'], dtype='object')
column dfrange  correlates strongly with:  Index(['maxdom'], dtype='object')
column z_meanfreq  correlates strongly with:  Index(['centroid', 'z_median', 'z_Q25'], dtype='ob
column z_median  correlates strongly with:  Index(['centroid', 'z_meanfreq'], dtype='object')
column z_Q25  correlates strongly with:  Index(['centroid', 'z_meanfreq'], dtype='object')
```

```
In [78]: corr_df = pd.DataFrame([{'Column Name':'skew', 'Correlated with':'kurt'},
                                 {'Column Name':'kurt', 'Correlated with':'skew'},
                                 {'Column Name':'centroid', 'Correlated with':['z_meanfreq', 'z_
                                 {'Column Name':'maxdom', 'Correlated with':['dfrange']},
                                 {'Column Name':'dfrange', 'Correlated with':['maxdom']},
                                 {'Column Name':'z_meanfreq', 'Correlated with':['centroid', 'z_
                                 {'Column Name':'z_median', 'Correlated with':['centroid', 'z_me
                                 {'Column Name':'z_Q25', 'Correlated with':['centroid', 'z_meanf
```

```
                            ])
corr_df
```

Out[78]:

	Column Name	Correlated with
0	skew	kurt
1	kurt	skew
2	centroid	[z_meanfreq, z_median, z_Q25]
3	maxdom	[dfrange]
4	dfrange	[maxdom]
5	z_meanfreq	[centroid, z_median, z_Q25]
6	z_median	[centroid, z_meanfreq]
7	z_Q25	[centroid, z_meanfreq]

In [79]:
```
# Thus we see high correlation exist between above variables,
# thus let us create a dataset by removing variables that create high Variance Inflatio
# Thus, removing kurt, Centroid, dfrange, z_meanfreq
data_x2 = data_x1.drop(['kurt', 'centroid', 'dfrange', 'z_meanfreq'],axis=1).copy()
data_x2.head(3)
```

Out[79]:

	sd	skew	sp.ent	sfm	mode	meanfun	minfun
1	0.067310	22.423285	0.892193	0.513724	0.000000	0.107937	0.015826
2	0.083829	30.757155	0.846389	0.478905	0.000000	0.098706	0.015656
3	0.072111	1.232831	0.963322	0.727232	0.083878	0.088965	0.017798

	maxfun	meandom	mindom	maxdom	modindx	z_median	z_Q25
1	0.250000	0.009014	0.007812	0.054688	0.052632	-3.828504	-2.355691
2	0.271186	0.007990	0.007812	0.015625	0.046512	-3.920638	-2.570261
3	0.250000	0.201497	0.007812	0.562500	0.247119	-0.737091	-0.810072

	z_Q75	Norm_IQR
1	-5.781683	0.246961
2	-4.094335	0.457148
3	-0.824403	0.407358

14.0.5 Creating Completely Normalized Dataset - All columns are normalized

In [83]:
```
# let me not do any dimentionality reduction and do z-score normalization on all indepe
xDataStdardized = StandardScaler()
xDataStdardized.fit(data_x)
data_x3 = xDataStdardized.transform(data_x).copy()
```

In [90]:
```
columns[0:20]
```

Out[90]:
```
Index(['meanfreq', 'sd', 'median', 'Q25', 'Q75', 'IQR', 'skew', 'kurt',
       'sp.ent', 'sfm', 'mode', 'centroid', 'meanfun', 'minfun', 'maxfun',
       'meandom', 'mindom', 'maxdom', 'dfrange', 'modindx'],
      dtype='object')
```

In [91]:
```
data_x3 = pd.DataFrame(data_x3, columns=columns[0:20])
data_x3.head(3)
```

```
Out[91]:    meanfreq        sd    median       Q25       Q75       IQR      skew  \
         0 -3.721597  0.544377 -3.829283 -2.356170 -5.782859 -0.397800  5.075036
         1 -3.355401  1.537541 -3.921435 -2.570784 -4.095168  0.781574  7.236841
         2 -0.961569  0.832992 -0.737241 -0.810237 -0.824571  0.502200 -0.421765

                kurt    sp.ent       sfm      mode  centroid   meanfun    minfun  \
         0  4.855087 -0.102172  0.554202 -2.145988 -3.721597 -1.004514 -1.135221
         1  7.993543 -1.089783  0.363138 -2.145988 -3.355401 -1.275017 -1.143789
         2 -0.214160  1.431488  1.725800 -1.079112 -0.961569 -1.560500 -1.036055

              maxfun   meandom    mindom    maxdom   dfrange   modindx
         0 -0.305371 -1.659948 -0.683626 -1.471597 -1.459346 -0.995844
         1  0.389022 -1.662045 -0.683626 -1.483124 -1.470873 -1.045086
         2 -0.305371 -1.266012 -0.683626 -1.321742 -1.309482  0.569030
```

14.0.6 Data Partition

```
In [92]: # Let us do a 80-20 split on raw dataset
         data_x_train,data_x_test,data_y_train,data_y_test = train_test_split(data_x,data_y,trai

In [93]: # let us do a 80-20 split on dimention reduced dataset too
         data_x2_train,data_x2_test,data_y2_train,data_y2_test=train_test_split(data_x2,data_y,t

In [94]: # let us do a 80-20 split on raw dataset which was only normalized
         data_x3_train,data_x3_test,data_y3_train,data_y3_test=train_test_split(data_x3,data_y,t

In [95]: # let us check the size
         data_x_train.shape

Out[95]: (1966, 20)

In [96]: data_x_test.shape

Out[96]: (492, 20)

In [97]: data_y_train.shape

Out[97]: (1966,)

In [98]: # let is cross check the size of dimention reduced data set too
         data_x2_train.shape

Out[98]: (1966, 16)

In [99]: data_x2_test.shape

Out[99]: (492, 16)

In [100]: # let is cross check the size of normalized raw data set too
          data_x3_train.shape

Out[100]: (1966, 20)

In [101]: data_x3_test.shape

Out[101]: (492, 20)
```

14.0.7 Inference:

1. I treated the variables with units making them unit free by standardizing them

2. z-score normalization for meanfreq, median, Q25, Q75 was done

3. min-max normalization was done for IQR variable

4. correlation between independent variables was checked to handle the multicollinearity issues

5. correlation between two variables greater than 0.9 are considered to be heavily coreelated and with respective VIF factor

6. Variables kurt, Centroid, dfrange, z_meanfreq was removed from dataset and this was maintained as a whole new dataset

7. Target variable was converted to numeric male as 1 and female as 0 using sklearn preprocessing pack n labelencoder object

8. Data partition was done based on sklearns model_selection package using train_test_split object

9. Thus I have 4 dataset treated from raw data: a.data_x_train
b.data_x_test
c.data_y_train
d.data_y_test

10. I have 4 dataset treated from raw data and dimentionality reduced: a.data_x2_train
b.data_x2_test
c.data_y2_train
d.data_y2_test

11. I have 4 dataset treated from raw data with all independent variables normalized: a.data_x3_train
b.data_x3_test
c.data_y3_train
d.data_y3_test

15 Step-5: Validating the cleaned dataset with benchmark accuracy obtained

In [102]:
```
# defining the Naive Bayes object
nbclf = naive_bayes.GaussianNB()
```

1. NB Cross Validation on Treated raw dataset

```
In [103]: # lets do a 10 fold Cross validation to make sure the accuracy obtained above
          nbclf = nbclf.fit(data_x_train, data_y_train)
          nbpreds_test = nbclf.predict(data_x_test)
          nb_eval_result1 = cross_val_score(nbclf, data_x, data_y, cv=10, scoring='accuracy')
          print('Mean accuracy with 10 fold cross validation on Naive Bayes with treated data: '
```

```
Mean accuracy with 10 fold cross validation on Naive Bayes with treated data:  0.948427662563
```

2. NB Cross Validation on Treated, partially normalized and dimension reduced dataset (This can at times help in building best SVM)

```
In [104]: # lets do a 10 fold Cross validation to make sure the accuracy obtained above
          nbclf = nbclf.fit(data_x2_train, data_y2_train)
          nbpreds_test = nbclf.predict(data_x2_test)
          nb_eval_result2 = cross_val_score(nbclf, data_x2, data_y, cv=10, scoring='accuracy')
          print('Mean accuracy with 10 fold cross validation on Naive Bayes with dimention reduc
```

```
Mean accuracy with 10 fold cross validation on Naive Bayes with dimention reduced data:  0.96957
```

3. NB Cross Validation on Treated and Completely Normalized dataset

```
In [105]: # lets do a 10 fold Cross validation to make sure the accuracy obtained above
          nbclf = nbclf.fit(data_x3_train, data_y3_train)
          nbpreds_test = nbclf.predict(data_x3_test)
          nb_eval_result3 = cross_val_score(nbclf, data_x3, data_y, cv=10, scoring='accuracy')
          print('Mean accuracy with 10 fold cross validation on Naive Bayes with Normalized data
```

```
Mean accuracy with 10 fold cross validation on Naive Bayes with Normalized data:  0.952900933653
```

15.0.1 Inference:

1. Naive bayes classifier after data tretment produce an avg accuracy of 0.95 being the data is normalized or not normalized

2. we see a significant increase in accuracy from 0.85671 to 0.952 after we clean the data

3. We see the data with dimention reduced and data which are completely normalized works better than raw treated dataset.

4. However, this can be considered as a base classifier at this point and above result makes sure that our data clean up holds good and we havent removed any influential datas from dataset.

5. This also set a new benchmark for any complex classifier that will be built further

6. Thus, accuracy of 0.95 can be set as a bench mark accuracy value for this dataset which is cleaned and processed.

7. Any model which produce accuracy less than 0.95 can be consodired as a non-efficient model for this dataset from now on

16 Step-6: Core Model Building - Applying Different Kernals for SVM

```
In [111]: def funct_svm(kernal_type,xTrain,yTrain,xTest,yTest):
              svm_obj=SVC(kernel=kernal_type)
              svm_obj.fit(xTrain,yTrain)
              yPredicted=svm_obj.predict(xTest)
              print('Accuracy Score of',kernal_type,'Kernal SVM is:',metrics.accuracy_score(yTes
              return metrics.accuracy_score(yTest,yPredicted)
```

16.0.1 6.1. Linear Kernal SVM

```
In [128]: # Partially normlized dataset
          %timeit 10
          PN_linear_result = funct_svm('linear',data_x_train,data_y_train,data_x_test,data_y_tes

100000000 loops, best of 3: 14.3 ns per loop
Accuracy Score of linear Kernal SVM is: 0.947154471545
```

```
In [129]: # Dimention reduced dataset
          %timeit 10
          DR_linear_result = funct_svm('linear',data_x2_train,data_y2_train,data_x2_test,data_y2

100000000 loops, best of 3: 13.9 ns per loop
Accuracy Score of linear Kernal SVM is: 0.941056910569
```

```
In [130]: # Completely normalized dataset
          %timeit 10
          CN_linear_result = funct_svm('linear',data_x3_train,data_y3_train,data_x3_test,data_y3

100000000 loops, best of 3: 13.9 ns per loop
Accuracy Score of linear Kernal SVM is: 0.993902439024
```

```
In [131]: linear_kernal_result = pd.DataFrame([{'Dataset':'Partially Normalized', 'Accuracy':PN_
                                               {'Dataset':'Dimention Reduced', 'Accuracy':DR_line
                                               {'Dataset':'Completely Normalized', 'Accuracy':CN_
          linear_kernal_result
```

Out[131]:

	Dataset	Accuracy
0	Partially Normalized	0.947154
1	Dimention Reduced	0.941057
2	Completely Normalized	0.993902

16.0.2 Inference:

1. I subjected 3 different dataset as explained above to a linear SVM model and I can observe that dataset which is completely normalize is performing well.

2. As part of this kernal trick, we have our hyperplane to be linear in a 20-dimentional space

3. This model exhibit a classification accuracy of 0.993902

4. Since the data is 20-dimentional, we cannot visualize if the data pocesses a linear or curved relation in feature space, we can take a domain level expertise here.

5. However, since we have none for individual analysis purpose we will try to build a model with other kernal tricks types too and see how the model behaves in classifying the gender.

16.0.3 6.2. RBF Kernal SVM

In [116]:
```
# Partially normlized dataset
%timeit 10
PN_rbf_result = funct_svm('rbf',data_x_train,data_y_train,data_x_test,data_y_test)
```

```
100000000 loops, best of 3: 14.2 ns per loop
Accuracy Score of rbf Kernal SVM is: 0.760162601626
```

In [117]:
```
# Dimention reduced dataset
%timeit 10
DR_rbf_result = funct_svm('rbf',data_x2_train,data_y2_train,data_x2_test,data_y2_test)
```

```
100000000 loops, best of 3: 14 ns per loop
Accuracy Score of rbf Kernal SVM is: 0.955284552846
```

In [118]:
```
# Completely normalized dataset
%timeit 10
CN_rbf_result = funct_svm('rbf',data_x3_train,data_y3_train,data_x3_test,data_y3_test)
```

```
100000000 loops, best of 3: 14.1 ns per loop
Accuracy Score of rbf Kernal SVM is: 0.993902439024
```

In [119]:
```
gausian_kernal_result = pd.DataFrame([{'Dataset':'Partially Normalized', 'Accuracy':PN
                                       {'Dataset':'Dimention Reduced', 'Accuracy':DR_rbf_
                                       {'Dataset':'Completely Normalized', 'Accuracy':CN_
gausian_kernal_result
```

Out[119]:

	Dataset	Accuracy
0	Partially Normalized	0.760163
1	Dimention Reduced	0.955285
2	Completely Normalized	0.993902

16.0.4 Inference:

1. RBF or Gaussian is the default kernal which SVM uses in sklearn

2. Performance of RBF kernal trick is also same as linear kernal SVM

3. I obtained a accuracy of 0.993902 for RBF Kernal using SVM for normalized dataset

4. This, shows that our voice dataset are both linearly and gaussian seperable

16.0.5 6.3. Polynomial Kernal SVM

In [120]:
```
# Partially normlized dataset
%timeit 10
PN_poly_result = funct_svm('poly',data_x_train,data_y_train,data_x_test,data_y_test)
```

```
100000000 loops, best of 3: 14 ns per loop
Accuracy Score of poly Kernal SVM is: 0.955284552846
```

In [121]:
```
# Dimentione reduced dataset
%timeit 10
DR_poly_result = funct_svm('poly',data_x2_train,data_y2_train,data_x2_test,data_y2_tes
```

```
10000000 loops, best of 3: 14 ns per loop
Accuracy Score of poly Kernal SVM is: 0.951219512195
```

In [122]:
```
# Completely normalized dataset
%timeit 10
CN_poly_result = funct_svm('poly',data_x3_train,data_y3_train,data_x3_test,data_y3_tes
```

```
100000000 loops, best of 3: 14.3 ns per loop
Accuracy Score of poly Kernal SVM is: 0.985772357724
```

In [123]:
```
poly_kernal_result = pd.DataFrame([{'Dataset':'Partially Normalized', 'Accuracy': PN_p
                                   {'Dataset':'Dimention Reduced', 'Accuracy':DR_poly
                                   {'Dataset':'Completely Normalized', 'Accuracy':CN_
poly_kernal_result
```

Out[123]:

	Dataset	Accuracy
0	Partially Normalized	0.955285
1	Dimention Reduced	0.951220
2	Completely Normalized	0.985772

16.0.6 Inference:

1. To acheive much more high accuracy, i tried using polynomial kernal too

2. I obtained an accuracy of 0.985 for polynomial kernal on normalized dataset

3. This is comparitively much less than the linear and rbf kernals

4. However, we cannot conclude this result at this stage as, our training dataset is just one single sample on which we obtained this result.

16.0.7 6.4. Sigmoidal Kernal SVM

In [124]:
```
# Partially normlized dataset
%timeit 10
PN_sigmoid_result = funct_svm('sigmoid',data_x_train,data_y_train,data_x_test,data_y_t
```

```
100000000 loops, best of 3: 13.9 ns per loop
Accuracy Score of sigmoid Kernal SVM is: 0.64837398374
```

In [125]:
```
# Dimentione reduced dataset
%timeit 10
DR_sigmoid_result = funct_svm('sigmoid',data_x2_train,data_y2_train,data_x2_test,data_
```

```
The slowest run took 200.17 times longer than the fastest. This could mean that an intermediate
100000000 loops, best of 3: 14 ns per loop
Accuracy Score of sigmoid Kernal SVM is: 0.678861788618
```

In [126]:
```
# Completely normalized dataset
%timeit 10
CN_sigmoid_result = funct_svm('sigmoid',data_x3_train,data_y3_train,data_x3_test,data_
```

```
100000000 loops, best of 3: 14 ns per loop
Accuracy Score of sigmoid Kernal SVM is: 0.831300813008
```

In [127]:
```
sigmoid_kernal_result = pd.DataFrame([{'Dataset':'Partially Normalized', 'Accuracy':PN
                                      {'Dataset':'Dimention Reduced', 'Accuracy':DR_sigm
                                      {'Dataset':'Completely Normalized', 'Accuracy':CN_
sigmoid_kernal_result
```

Out[127]:

	Dataset	Accuracy
0	Partially Normalized	0.648374
1	Dimention Reduced	0.678862
2	Completely Normalized	0.831301

16.0.8 Inference:

1. When a dataset is behaving well linearly, it is explicitly known that it doesn't work well in a sigmoidal space

2. Above result obtained is the evident for this

3. I obtained accuracy of just 0.831 with sigmoidal kernal

16.0.9 4.5. Consolidated model accuracy

In [132]:
```
kernal_result = pd.DataFrame([{'Dataset':'Completely Normalized','Kernal':'Linear', 'A
                               {'Dataset':'Completely Normalized','Kernal':'Gaussian', 'A
                               {'Dataset':'Completely Normalized','Kernal':'Polynomial',
                               {'Dataset':'Completely Normalized','Kernal':'Sigmoidal', '
                                columns=['Dataset','Kernal','Accuracy'])
kernal_result
```

Out[132]:

	Dataset	Kernal	Accuracy
0	Completely Normalized	Linear	0.993902
1	Completely Normalized	Gaussian	0.993902
2	Completely Normalized	Polynomial	0.985772
3	Completely Normalized	Sigmoidal	0.831301

16.0.10 Inference:

1. From above table it is clear that a completely normalized dataset behaves well compare to un-normalized dataset

2. I obtain a maximum accuracy due to the data treatment done, that is treating the meanfun attribute based on biological fact

3. Maximum accuracy i could acheive is 0.9939 whcih is from Linear and Gaussian Kernal using SVM

4. While the polinomial and Sigmoidal kernal doesn't seems to classify the target variable accurately and giving a low accuracy of 0.95 and 0.83 for Polynomial and Sigmoidal keransl respectively.

5. However, I cannot blindly accept this accuracy result because this is derived from one sample of training set and validated with a sample test set. In order to evaluate this model to be more robust and to ensure data doesnt overfit, I wanted to subject these model and dataset to a 10-fold cross validation and observe its result as part of next session

17 Step-7: Perfomance Evaluation on Different Kernals for SVM with 10-fold cross validation

In [138]:
```
def funct_svm_cv(kernal_type,xData,yData,k,eval_param):
    svm_obj=SVC(kernel=kernal_type)
    eval_result = cross_val_score(svm_obj, xData, yData, cv=k, scoring=eval_param)
    print(eval_param,'of each fold is:',eval_result)
```

```
        print('Mean accuracy with 10 fold cross validation for',kernal_type,' kernal SVM i
        return eval_result.mean()
```

17.0.1 7.1. Evaluation on Linear Kernal SVM

```
In [139]: # Partially normlized dataset
          %timeit 10
          PN_CV_linear_result = funct_svm_cv('linear',data_x,data_y,10,'accuracy')
```

```
100000000 loops, best of 3: 14.6 ns per loop
accuracy of each fold is: [ 0.75708502  0.94331984  0.87449393  0.97165992  0.97959184  0.987755
  0.99591837  0.99591837  0.88571429]
Mean accuracy with 10 fold cross validation for linear  kernal SVM is:  0.939145666364
```

```
In [140]: # Dimentione reduced dataset
          %timeit 10
          DR_CV_linear_result = funct_svm_cv('linear',data_x2,data_y,10,'accuracy')
```

```
100000000 loops, best of 3: 14.1 ns per loop
accuracy of each fold is: [ 0.74089069  0.91093117  0.87044534  0.951417    0.9755102   0.987755
  0.99591837  0.99183673  0.99591837  0.88571429]
Mean accuracy with 10 fold cross validation for linear  kernal SVM is:  0.930633727175
```

```
In [141]: # Completely normalized dataset
          %timeit 10
          CN_CV_linear_result = funct_svm_cv('linear',data_x3,data_y,10,'accuracy')
```

```
100000000 loops, best of 3: 14.1 ns per loop
accuracy of each fold is: [ 0.98785425  0.99595142  1.          0.95546559  1.          1.
  1.          1.          1.        ]
Mean accuracy with 10 fold cross validation for linear  kernal SVM is:  0.993927125506
```

```
In [142]: cv_linear_kernal_result = pd.DataFrame([{'Dataset':'Partially Normalized', 'Accuracy':
                                                   {'Dataset':'Dimention Reduced', 'Accuracy':DR_CV_l
                                                   {'Dataset':'Completely Normalized', 'Accuracy':CN_
          cv_linear_kernal_result
```

Out[142]:

	Dataset	Accuracy
0	Partially Normalized	0.939146
1	Dimention Reduced	0.930634
2	Completely Normalized	0.993927

17.0.2 Inference:

1. I see even with 10 fold cross validation, our linear kernal SVM is providing a high accuracy of 0.9939

2. Thus, I can consider linear Kernal SVM as one of the serious model to subject for further tuning and see if it increases the accuracy

3. From abov table it is still evident that the completely normalized dataset behaves well comparitively

17.0.3 7.2. Evaluation on RBF Kernal SVM

```
In [143]: # Partially normlized dataset
          %timeit 10
          PN_CV_rbf_result = funct_svm_cv('rbf',data_x,data_y,10,'accuracy')

100000000 loops, best of 3: 14 ns per loop
accuracy of each fold is: [ 0.61538462  0.74089069  0.65991903  0.79352227  0.85306122  0.8
  0.80816327  0.68571429  0.84897959  0.62040816]
Mean accuracy with 10 fold cross validation for rbf  kernal SVM is:  0.74260431298
```

```
In [144]: # Dimentione reduced dataset
          %timeit 10
          DR_CV_rbf_result = funct_svm_cv('rbf',data_x2,data_y,10,'accuracy')

100000000 loops, best of 3: 16 ns per loop
accuracy of each fold is: [ 0.7854251   0.90688259  0.91497976  0.92307692  0.99591837  0.967346
  0.9877551   0.98367347  0.99183673  0.89795918]
Mean accuracy with 10 fold cross validation for rbf  kernal SVM is:  0.935485416839
```

```
In [145]: # Completely normalized dataset
          %timeit 10
          CN_CV_rbf_result = funct_svm_cv('rbf',data_x3,data_y,10,'accuracy')

10000000 loops, best of 3: 14.5 ns per loop
accuracy of each fold is: [ 0.96761134  0.97165992  0.99190283  0.95546559  1.          0.979591
  1.          1.          1.          1.        ]
Mean accuracy with 10 fold cross validation for rbf  kernal SVM is:  0.986623151285
```

```
In [146]: cv_rbf_kernal_result = pd.DataFrame([{'Dataset':'Partially Normalized', 'Accuracy':PN_
                                               {'Dataset':'Dimention Reduced', 'Accuracy':DR_CV_r
                                               {'Dataset':'Completely Normalized', 'Accuracy':CN_
          cv_rbf_kernal_result
```

Out[146]:

	Dataset	Accuracy
0	Partially Normalized	0.742604
1	Dimention Reduced	0.935485
2	Completely Normalized	0.986623

17.0.4 Inference:

1. From above table, I see a slight decrease in accuracy when I subject Gaussian kernal to 10-fold cross validation

2. With out 80-20 split test set we saw an accuracy of 0.9939 however, with 10-fold CV we obtain accuracy of 0.986

3. Thus, so far we see linear kernal is behaving well consistently and there is a slight decrese with gaussian kernal

17.0.5 7.4. Evaluation on Sigmoidal Kernal SVM

```
In [147]: # Partially normlized dataset
          %timeit 10
          PN_CV_sigmoid_result = funct_svm_cv('sigmoid',data_x,data_y,10,'accuracy')
```

```
100000000 loops, best of 3: 14.1 ns per loop
accuracy of each fold is: [ 0.6194332   0.44129555  0.71255061  0.61538462  0.65306122  0.726530
  0.80816327  0.60408163  0.84081633  0.46938776]
Mean accuracy with 10 fold cross validation for sigmoid  kernal SVM is:  0.649070478394
```

```
In [148]: # Dimentione reduced dataset
          %timeit 10
          DR_CV_sigmoid_result = funct_svm_cv('sigmoid',data_x2,data_y,10,'accuracy')
```

```
100000000 loops, best of 3: 13.9 ns per loop
accuracy of each fold is: [ 0.51417004  0.69635628  0.68825911  0.8097166   0.88571429  0.710204
  0.66530612  0.62040816  0.73469388  0.51836735]
Mean accuracy with 10 fold cross validation for sigmoid  kernal SVM is:  0.684319590184
```

```
In [149]: # Completely normalized dataset
          %timeit 10
          CN_CV_sigmoid_result = funct_svm_cv('sigmoid',data_x3,data_y,10,'accuracy')
```

```
100000000 loops, best of 3: 14.1 ns per loop
accuracy of each fold is: [ 0.68825911  0.78137652  0.82995951  0.94736842  0.86122449  0.787755
  0.75918367  0.70612245  0.76734694  0.86938776]
Mean accuracy with 10 fold cross validation for sigmoid  kernal SVM is:  0.799798397092
```

```
In [150]: cv_sigmoid_kernal_result = pd.DataFrame([{'Dataset':'Partially Normalized', 'Accuracy'
                                                    {'Dataset':'Dimention Reduced', 'Accuracy':DR_CV_s
                                                    {'Dataset':'Completely Normalized', 'Accuracy':CN_
          cv_sigmoid_kernal_result
```

Out[150]:

	Dataset	Accuracy
0	Partially Normalized	0.649070
1	Dimention Reduced	0.684320
2	Completely Normalized	0.799798

17.1 Inference:

1. Like Gaussian kernal, even polynomial and sigmoidal kernals yeald less accuracy with 10 fold CV

2. I did not include the results of polynomial kernal subjected to 10 fold CV because it was consuming more time to compute

3. However, results of sigmoidal kernal is shown above and we see accuracy is dropped from 0.81 to 0.79

17.1.1 7.5. Consolidated SVM Kernal Model's Evaluation Result

```
In [152]: cv_kernal_result = pd.DataFrame([{'Dataset':'Completely Normalized','Kernal':'Linear',
                                           {'Dataset':'Completely Normalized','Kernal':'Gaussian', 'A
                                           {'Dataset':'Completely Normalized','Kernal':'Polynomial',
                                           {'Dataset':'Completely Normalized','Kernal':'Sigmoidal', '
                                            columns=['Dataset','Kernal','Accuracy'])
          cv_kernal_result
```

Out[152]:

	Dataset	Kernal	Accuracy
0	Completely Normalized	Linear	0.993927
1	Completely Normalized	Gaussian	0.986623
2	Completely Normalized	Polynomial	0.985772
3	Completely Normalized	Sigmoidal	0.799798

17.1.2 Inference:

1. From above table it is clearly evident that Linear SVM Kernal on a completely normalized datset behaves really well

2. Even with 10-fold cross validation, I obtaned an accuracy of 0.9933927 which seems consistent when compare to other kernals.

3. After linear kernal it is the Gaussian and Polynomial kernal which gives high accuracy

4. So as part of next session, we will drop Sigmoidal kernal from our further analyis as it doens't even satisfy the bench mark accuracy.

5. I will take up other 3 SVM models for performance tuning and see how the accuracychanges when we tradeoff between kernal parameters like penalty (C) and gamma in order to obtain a soft margin.

18 Step-8: Parameter tuning on Different Kernals for SVM with 5-fold cross validation - experimenting with margins

From above experimentation we see dataset which was normalized yeald a good result

Thus, for further experimentation we will use the dataset whose independent variables are normalized i.e.

data_x3 and data_y3

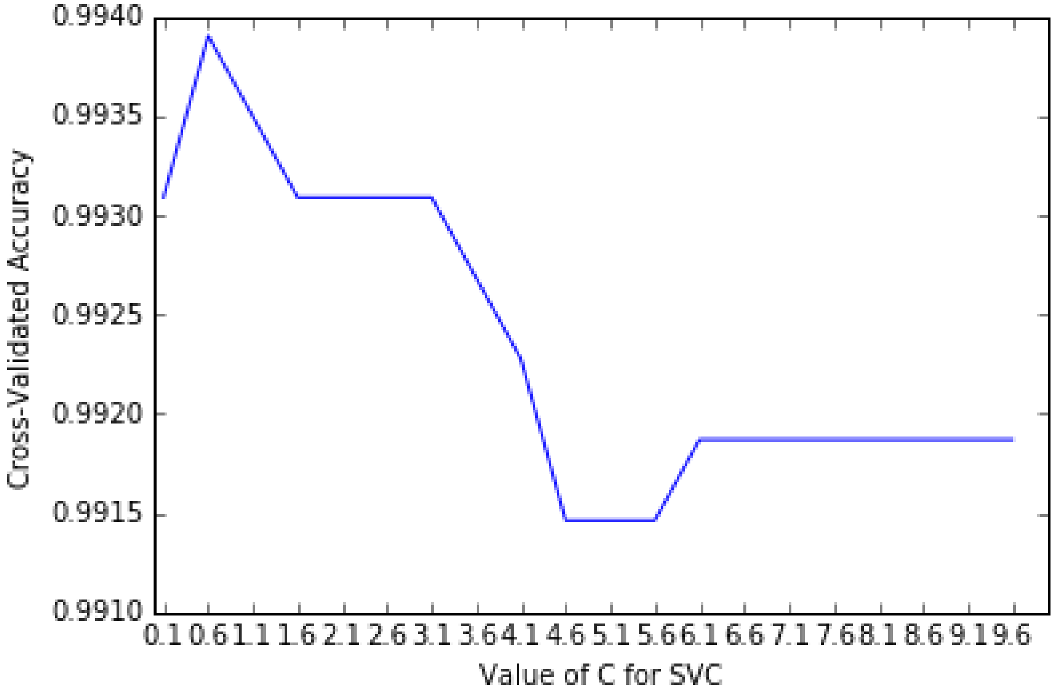
In [153]:
```
# penality parameter C is 1.0 by default in sklearn
# I would like to experiment it with multiple margins in range of c from 1 to 10
def funct_tune_svm(kernal_type,margin_val,xData,yData,k,eval_param):
    if(kernal_type=='linear'):
        svm_obj=SVC(kernel=kernal_type,C=margin_val)
    elif(kernal_type=='rbf'):
        svm_obj=SVC(kernel=kernal_type,gamma=margin_val)
    elif(kernal_type=='poly'):
        svm_obj=SVC(kernel=kernal_type,degree=margin_val)
    eval_result = cross_val_score(svm_obj, xData, yData, cv=k, scoring=eval_param)
    return eval_result.mean()
```

18.0.1 8.1. Tuning on Linear Kernal SVM

In [154]:
```
# Completely normlized dataset
accu_list = list()
for c in np.arange(0.1,10,0.5):
    result = funct_tune_svm('linear',c,data_x3,data_y,5,'accuracy')
    accu_list.append(result)
```

In [155]:
```
C_values=np.arange(0.1,10,0.5)
# plot the value of C for SVM (x-axis) versus the cross-validated accuracy (y-axis)
plt.plot(C_values,accu_list)
plt.xticks(np.arange(0.1,10,0.5))
plt.xlabel('Value of C for SVC')
plt.ylabel('Cross-Validated Accuracy')
```

Out[155]:
```
<matplotlib.text.Text at 0x247b9e5d5f8>
```



In [156]:
```
tuning_linear_svm = pd.DataFrame(columns=['Penality Parameter C', 'Accuracy'])
tuning_linear_svm['Penality Parameter C'] = np.arange(0.1,10,0.5)
tuning_linear_svm['Accuracy'] = accu_list
tuning_linear_svm
```

Out[156]:

	Penality Parameter C	Accuracy
0	0.1	0.993089
1	0.6	0.993902
2	1.1	0.993496
3	1.6	0.993089
4	2.1	0.993089
5	2.6	0.993089
6	3.1	0.993089
7	3.6	0.992683
8	4.1	0.992276
9	4.6	0.991463
10	5.1	0.991463
11	5.6	0.991463
12	6.1	0.991870
13	6.6	0.991870
14	7.1	0.991870
15	7.6	0.991870
16	8.1	0.991870
17	8.6	0.991870

```
18                    9.1  0.991870
19                    9.6  0.991870
```

18.0.2 Inference:

1. Ultimate aim in building a kernal is to find an optimum hyper plane in feature space which has maximum margin in classifying our target variable.

2. Kernal which I have built above so far in order to check the performance are those with hard margins, this is not good to be generalized as it may cause overfitting.

3. So, in this session, we will trade off between margin and Support vectors to choose an optimum boundry which will not overfit the model and at the same time deliver a high accuracy in classifying the target variable.

4. With linear kernal it is the penalty measure through which we can do some trade off

5. Above table shows the accuracy (model performance) for different values of C

6. Both from graph and above table we see 0.6 and 1.1 to be the optimum penalty measure or C value which we can treade off with in classifying the target variable.

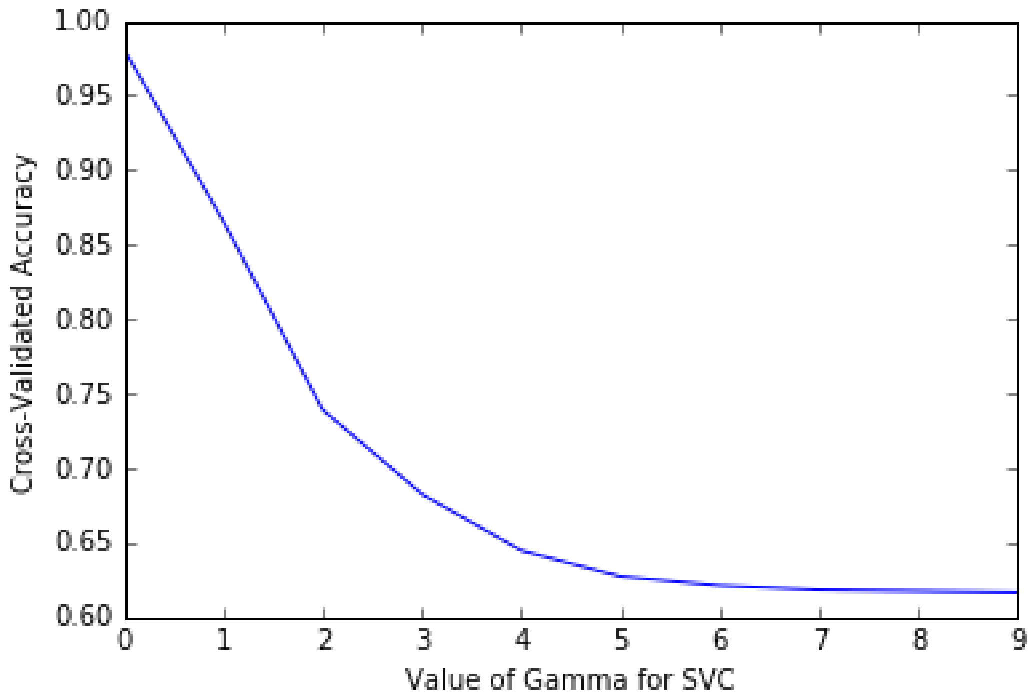
7. Even with such trade off , we obtain almost 0.9939 accuracy for linear kernal

18.0.3 8.2. Tuning on RBF Kernal SVM

In [157]:
```
# Completely normlized dataset
accu_list = list()
for c in np.arange(0.1,10,1):
    result = funct_tune_svm('rbf',c,data_x3,data_y,5,'accuracy')
    accu_list.append(result)
```

In [158]:
```
C_values=list(range(0,10))
# plot the value of C for SVM (x-axis) versus the cross-validated accuracy (y-axis)
plt.plot(C_values,accu_list)
plt.xticks(np.arange(0,10,1))
plt.xlabel('Value of Gamma for SVC')
plt.ylabel('Cross-Validated Accuracy')
```

Out[158]:
```
<matplotlib.text.Text at 0x247bbe837b8>
```




```
In [159]: tuning_rbf_svm = pd.DataFrame(columns=['Parameter Gamma', 'Accuracy'])
          tuning_rbf_svm['Parameter Gamma'] = np.arange(0.1,10,1)
          tuning_rbf_svm['Accuracy'] = accu_list

In [160]: tuning_rbf_svm
```

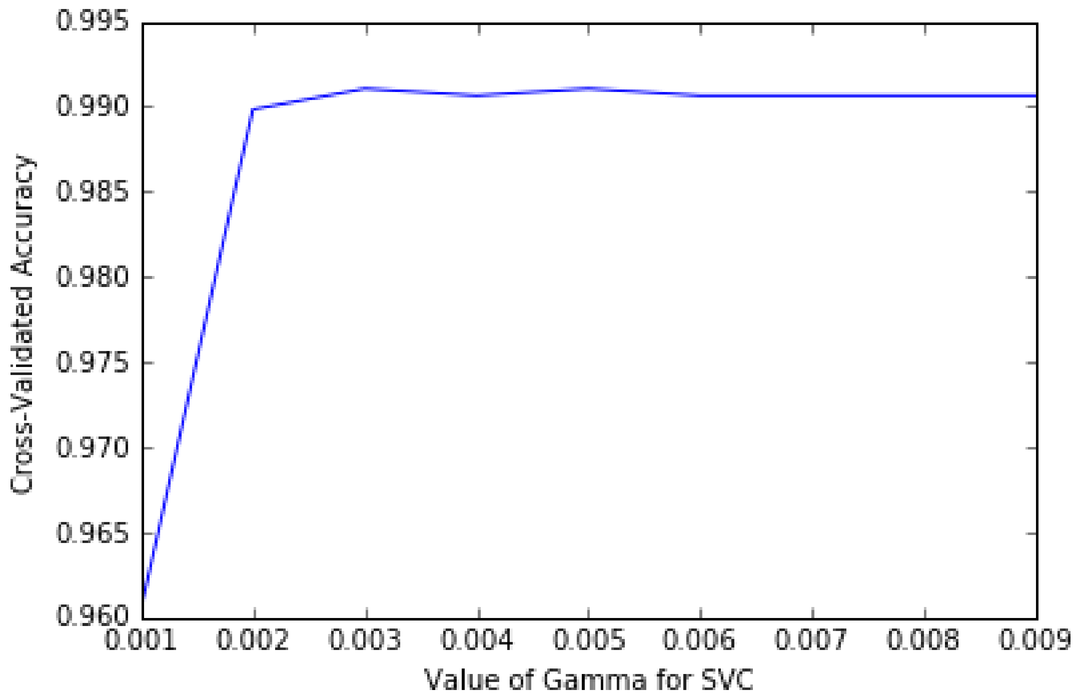
Out[160]:

	Parameter Gamma	Accuracy
0	0.1	0.981289
1	1.1	0.866114
2	2.1	0.739190
3	3.1	0.682660
4	4.1	0.644832
5	5.1	0.627340
6	6.1	0.621239
7	7.1	0.618392
8	8.1	0.617579
9	9.1	0.616765

```
In [161]: # Doing further tradeoff
          accu_list = list()
          for c in np.arange(0.001,0.01,0.001):
              result = funct_tune_svm('rbf',c,data_x3,data_y,5,'accuracy')
              accu_list.append(result)

          C_values=list(np.arange(0.001,0.01,0.001))
```

```
# plot the value of C for SVM (x-axis) versus the cross-validated accuracy (y-axis)
plt.plot(C_values,accu_list)
plt.xticks(np.arange(0.001,0.01,0.001))
plt.xlabel('Value of Gamma for SVC')
plt.ylabel('Cross-Validated Accuracy')
```

Out[161]: <matplotlib.text.Text at 0x29dc9cf3a58>



In [162]:
```
tuning_rbf_svm = pd.DataFrame(columns=['Parameter Gamma', 'Accuracy'])
tuning_rbf_svm['Parameter Gamma'] = np.arange(0.001,0.01,0.001)
tuning_rbf_svm['Accuracy'] = accu_list
tuning_rbf_svm
```

Out[162]:

	Parameter Gamma	Accuracy
0	0.001	0.960562
1	0.002	0.989837
2	0.003	0.991057
3	0.004	0.990650
4	0.005	0.991057
5	0.006	0.990650
6	0.007	0.990650
7	0.008	0.990650
8	0.009	0.990650

18.0.4 Inference:

1. In Gaussian kernal, tradeoff is done with penalty (C) along with gamma parameter

2. I first experimented with wider Gamma values ranging between 1 and 10 and obsevred Kernal started to behave bad with gamma greater than 1

3. So, I tried to find the most optimum value with in 0 and 1 and as show in above table, i obtained a maximum accuracy of 0.991 when gammal was equal to 0.03 and 0.05

4. However when compare to Linear kernal, we see rbf produce an accuracy of 0.002 times less.

5. Thus, it is quite evident again that linear kernal acts well on this dataset in classification of target variable.

18.0.5 8.3. Tuning on Polynomial Kernal SVM

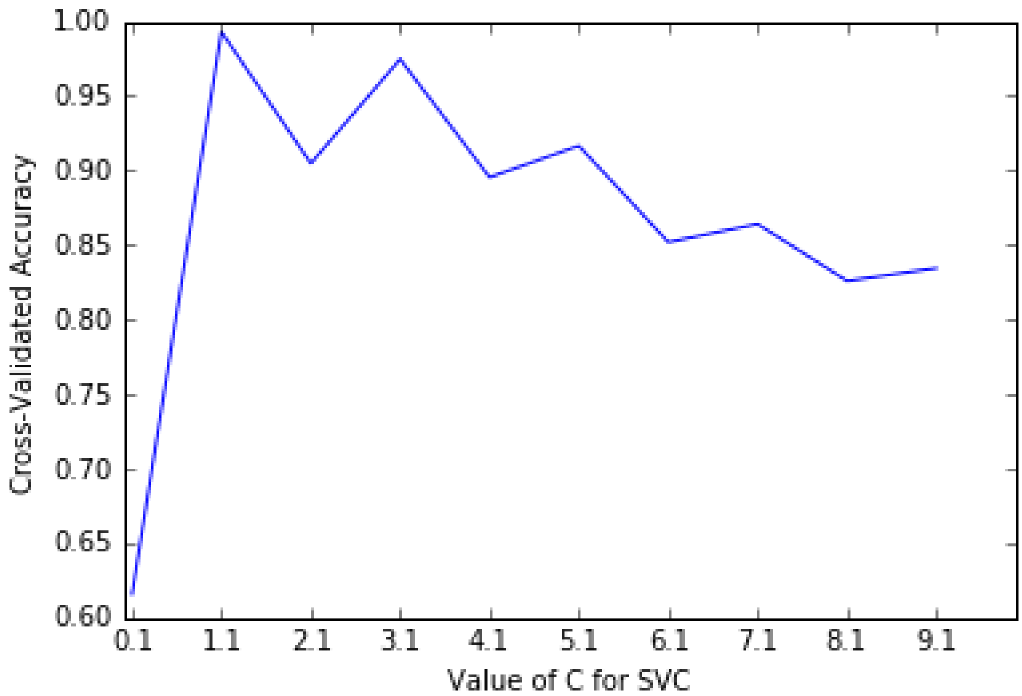
```
In [165]: # Completely normlized dataset
          accu_list = list()
          for c in np.arange(0.1,10,1):
              result = funct_tune_svm('poly',c,data_x3,data_y,5,'accuracy')
              accu_list.append(result)
```

```
In [166]: np.arange(0.1,10,1)
```

```
Out[166]: array([ 0.1,  1.1,  2.1,  3.1,  4.1,  5.1,  6.1,  7.1,  8.1,  9.1])
```

```
In [168]: C_values=list(np.arange(0.1,10,1))
          # plot the value of C for SVM (x-axis) versus the cross-validated accuracy (y-axis)
          plt.plot(C_values,accu_list)
          plt.xticks(np.arange(0.1,10,1))
          plt.xlabel('Value of C for SVC')
          plt.ylabel('Cross-Validated Accuracy')
```

```
Out[168]: <matplotlib.text.Text at 0x29dc9ddd978>
```




```
In [170]: tuning_poly_svm = pd.DataFrame(columns=['Parameter Degree', 'Accuracy'])
          tuning_poly_svm['Parameter Degree'] = np.arange(0.1,10,1)
          tuning_poly_svm['Accuracy'] = accu_list
          tuning_poly_svm
```

Out[170]:

	Parameter Degree	Accuracy
0	0.1	0.615948
1	1.1	0.993089
2	2.1	0.904793
3	3.1	0.974783
4	4.1	0.895480
5	5.1	0.916632
6	6.1	0.851958
7	7.1	0.864161
8	8.1	0.825925
9	9.1	0.834472

18.0.6 Inference:

1. Along with penalty and gamma parameter, with polynomial kernal we can trade off with degree

2. I experimented with various degree as shown above and obtained degree = 1.1 produce a high accuracy

3. Accuracy obtained by polynomial is almost same as Linear which is 0.993

4. So, to produce a final inference in choosing the best kernal we will apply a grid search in our next session and see which model and which parameter produce a high accuracy.

19 Step-9: Choosing best Kernals Parameters with grid search

```
In [171]: # Now performing SVM by taking hyperparameter C=0.1 and kernel as linear
          svc=SVC(kernel='linear',C=0.6)
          scores = cross_val_score(svc, data_x3, data_y, cv=10, scoring='accuracy')
          print(scores.mean())
```

```
0.993927125506
```

```
In [172]: # With rbf gamma value = 0.01
          svc= SVC(kernel='rbf',gamma=0.005)
          svc.fit(data_x3_train,data_y3_train)
          y_predict=svc.predict(data_x3_test)
          metrics.accuracy_score(data_y3_test,y_predict)
```

```
Out[172]: 0.99390243902439024
```

```
In [174]: np.arange(0.001,0.0,0.001)
```

```
Out[174]: array([ 0.001,  0.002,  0.003,  0.004,  0.005,  0.006,  0.007,  0.008,
                  0.009])
```

19.0.1 9.1. Choosing the best parameter

```
In [175]: # performing grid search with different tuning parameters
          svm_obj= SVC()
          grid_parameters = {
           'C': [0.1,0.6,1.1,1.6] , 'kernel': ['linear'],
           'C': [0.1,0.6,1.1,1.6] , 'gamma': [0.002,0.003,0.004,0.005], 'kernel': ['rbf'],
           'degree': [1,2,3] ,'gamma':[0.002,0.003,0.004,0.005], 'C':[0.1,0.6,1.1,1.6] , 'kernel
                            }
          model_svm = GridSearchCV(svm_obj, grid_parameters,cv=10,scoring='accuracy')
          model_svm.fit(data_x3_train, data_y3_train)
          print(model_svm.best_score_)
          print(model_svm.best_params_)
          y_pred= model_svm.predict(data_x3_test)
```

```
0.992370295015
{'gamma': 0.005, 'C': 1.6, 'kernel': 'poly', 'degree': 1}
```

```
In [176]: svm_performance = metrics.accuracy_score(y_pred,data_y3_test)
          svm_performance
```

Out[176]: 0.99390243902439024

In [177]:
```
gridSearch_kernal_result = pd.DataFrame([{'kernel': 'poly', 'gamma': 0.005, 'degree':
                                          columns=['kernel','C','gamma','degree'])
gridSearch_kernal_result
```

Out[177]:

	kernel	C	gamma	degree
0	poly	1.6	0.005	1

19.0.2 Inference:

1. I did a grid search, whcih is a structure way to obtain an optimized kernal and its parameter measures

2. From above result, I see it is the polynomial kernal with penalty measure of C=1.6 and gamma = 0.005 and with degree=1 produce a high accuracy of 0.9939 in classifying the target variable.

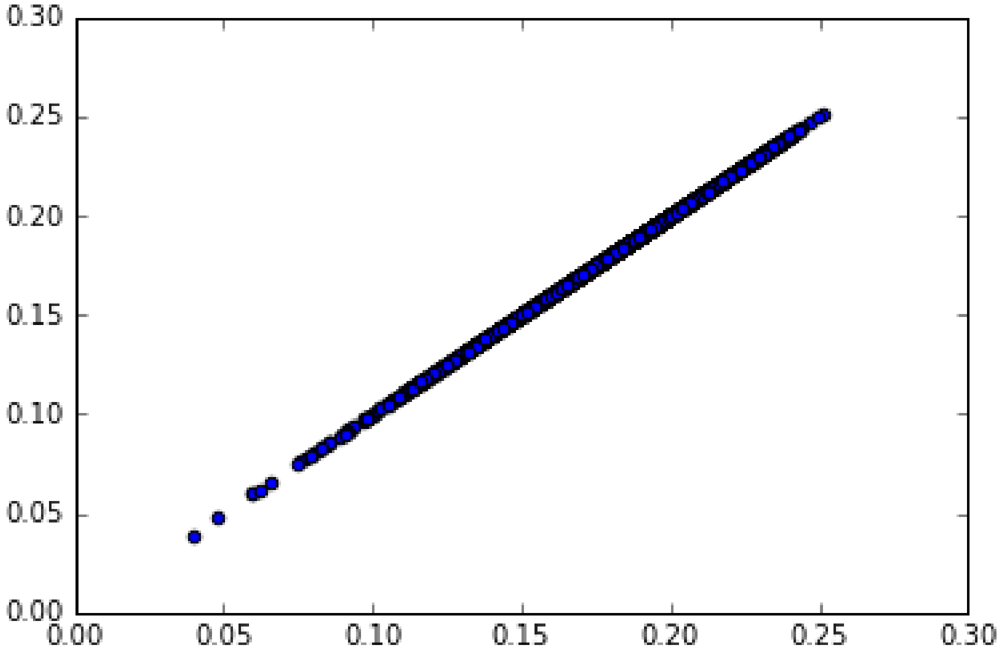
3. In this next session i have tried to visualize my margin and kernal behaviour by subjecting only 2 columns for analysis as it becomes a 2-dimentional space for visualization.

20 Step-10 Visualization of kernal Margin and boundries considereing only two columns meanfun & sp.ent to represent a 2D space

20.0.1 10.1. Choosing the best attribute to represent dataset in 2D space

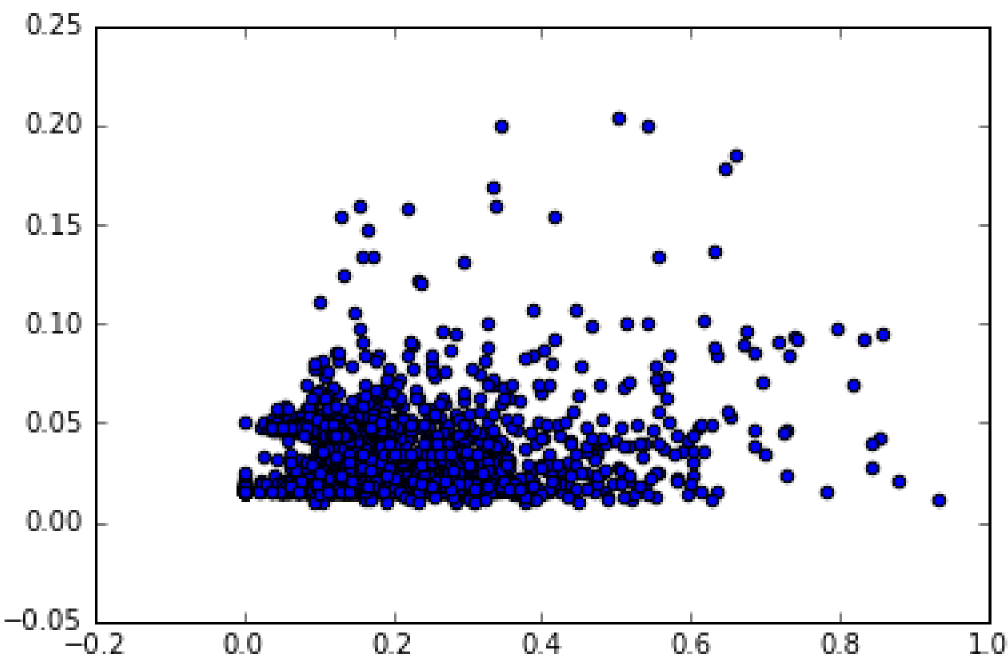
In [178]:
```
# Scatter plot with strong correlation - not useful much to represnt the distribution
plt.scatter(data_raw['meanfreq'],data_raw['centroid'])
```

Out[178]: <matplotlib.collections.PathCollection at 0x29dcb0fe320>



In [179]:
```
# Scatter plot with weak correlation - not useful much to represnt the distribution wr
plt.scatter(data_raw['modindx'],data_raw['minfun'])
```

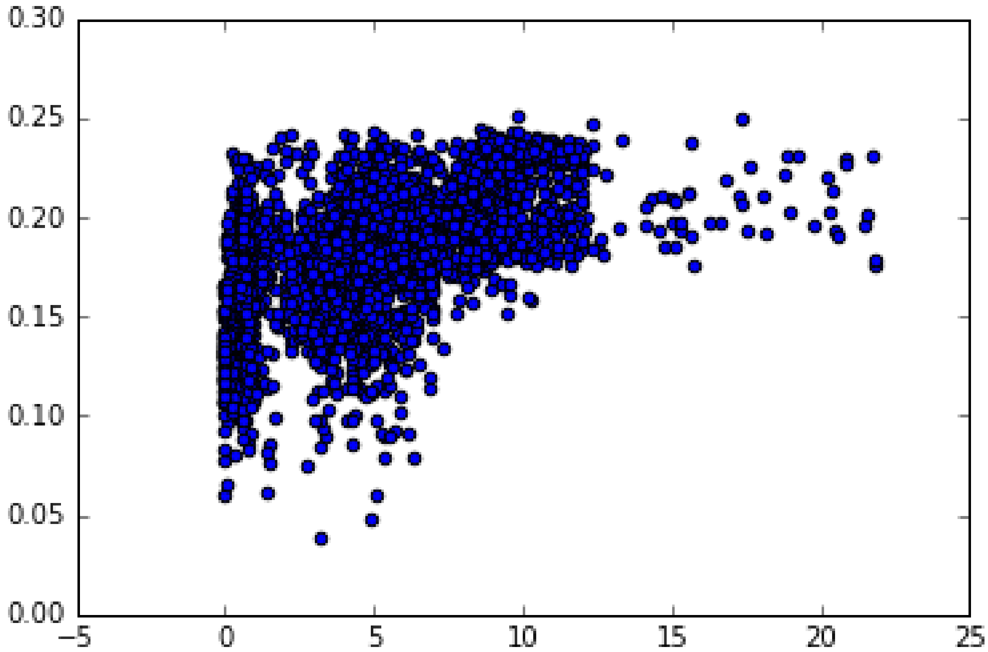
Out[179]: <matplotlib.collections.PathCollection at 0x29dcb2d5c50>



In [181]: # Scatter plot with moderate correlation - useful much to represnt the distribution wr

```
plt.scatter(data_raw['dfrange'],data_raw['centroid'])
```

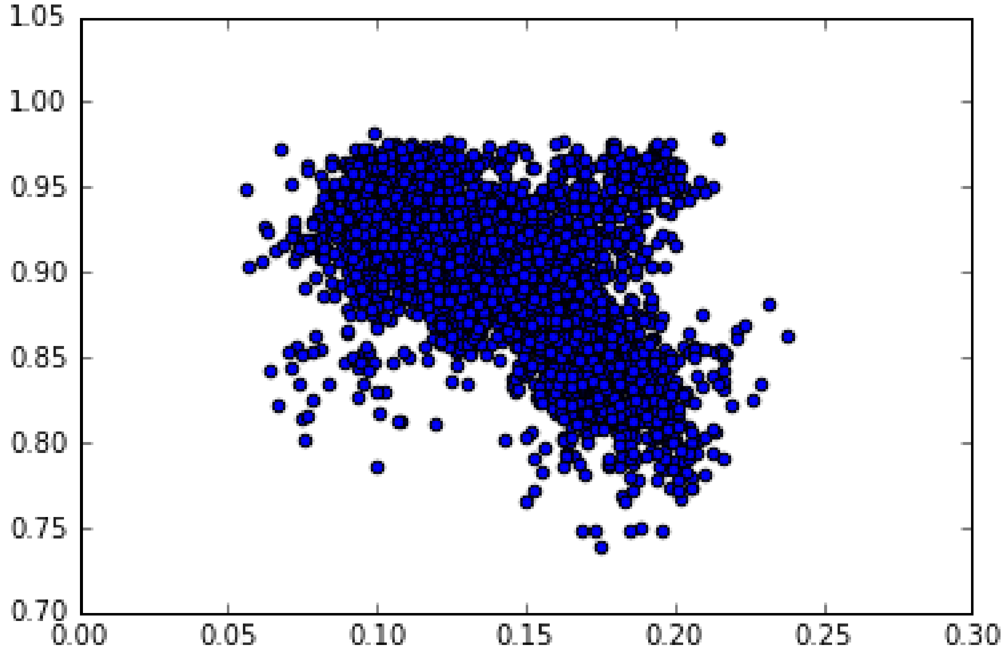
Out[181]: <matplotlib.collections.PathCollection at 0x29dc9db8048>



In [180]: # Scatter plot with moderate negative correlation - useful much to represnt the distri

```
plt.scatter(data_raw['meanfun'],data_raw['sp.ent'])
```

Out[180]: <matplotlib.collections.PathCollection at 0x29dc9fa7f98>



20.0.2 Inference:

1. After doing necessary data cleanup and model building I was able to infer that a polinomial kernal SVM with parameters C=1.6, gamma=0.005 and degree=1 plots a perfect margin in a high dimentional space to classify gender label which is our target variable

2.However, vizualizing more than two dimention is complex to represnt

3. So, I would like to choose any 2 variables from dataset through which i can represnt my margin and kernal boundries in a 2-dimentional space

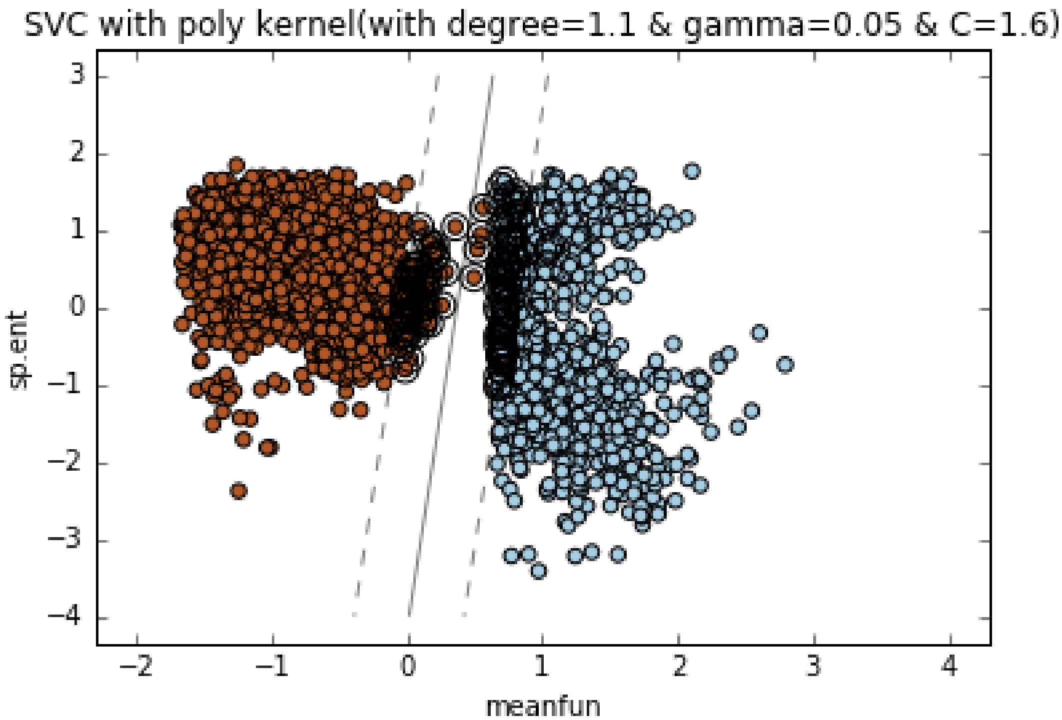
4. For this i used the correlation matrix and above scatter plot obtained above and choose two variable which is moderately correlated. As neither the strong nor the weak correlation variables might not be well represented in ourder to show the decision boundries.

5. meanfun being the most important variable for the dataset, I decided to choose it and match it with another variable which has moderate correlation with it. with 0.52 as correlation value between i choose sp.ent and meanfun to be my choise of 2-dimentional feature space.

20.0.3 10.2. Visualizing the margin modeled

```
In [185]: # import some data to play with
          X = data_x3[['meanfun','sp.ent']].copy()
          X = np.array(X)
```

```python
y = np.array(data_y)

# fit the model, don't regularize for illustration purposes
clf = SVC(kernel='poly', degree=1.1, gamma = 0.05,C=1.6)
clf.fit(X, y)

# title for the plots
title = ('SVC with poly kernel(with degree=1.1 & gamma=0.05 & C=1.6)')

plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
# plot support vectors
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='none')
ax.set_xlabel('meanfun')
ax.set_ylabel('sp.ent')
ax.set_title(title)
plt.show()
```



20.0.4 Inference:

1. meanfun being the most important variable for the dataset, I decided to choose it and match it with another variable which has moderate correlation with it. with 0.52 as correlation value between i choose sp.ent and meanfun to be my choise of 2-dimentional feature space.

2. I modeled polynomial kernal with penalty measure of C=1.6, gamma = 0.05 and degree=1 to obtain the above scatter plot.

3. When did, SVM projected my data in a 2 dimentional space and obtained an optimal margin that classifies my gender being male and female.

4. From the above figure, we can infer:

1. Orage points = Instance which are Male

2. Blue Points = Instance which are Female

3. Circled Points = Support Vectors used to obtain margin

4. Straingh Line = Hard Margin

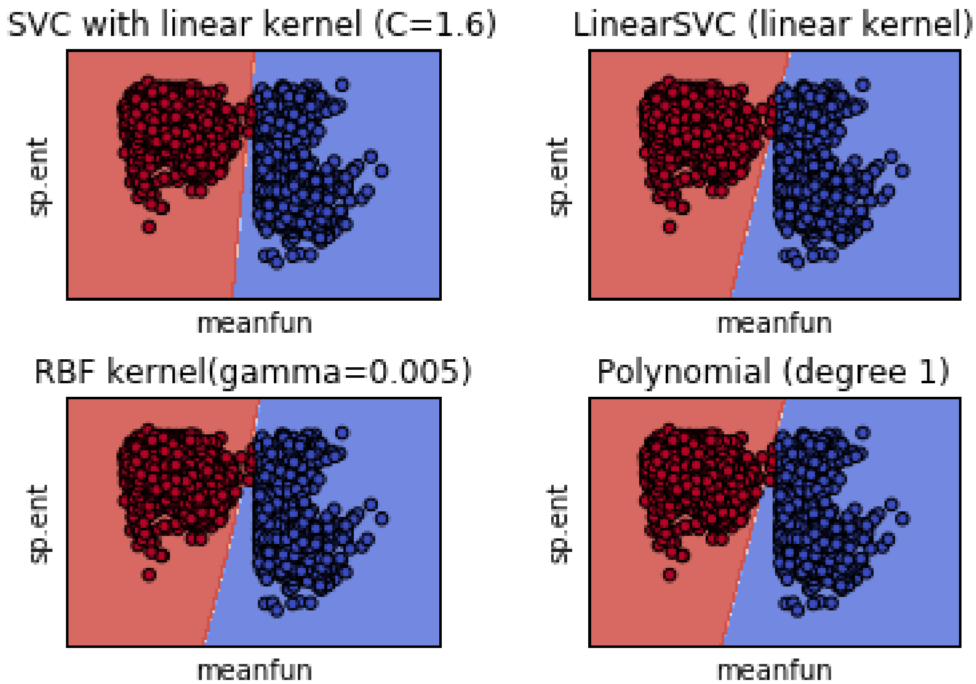
5. Dotted Lines = Soft Margin (with trade off being C=1.6, gamma=0.05 and degree=1)

5. With respective to only these two variables, meanfun and sp.ent, It is so evident that our model is not being overfit as it gives a clear distinction between two classes 'Male' and ' Female' with no complications in margins. Thus, accuracy of 0.99 can be considered to be valid enough at this point. However, this is just the visualization about margins, we will not visualize how the SVM boundy is placed in a for all our parameters in a 2D space.

20.0.5 10.3. Visualizing the Kernal boundaries

In [255]:
```
def make_meshgrid(x, y, h=.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out

# import some data to play with
X = data_x3[['meanfun','sp.ent']].copy()
```

```python
X = np.array(X)
y = np.array(data_y)

C = 1.6  # SVM regularization parameter
models = (SVC(kernel='linear', C=C),
          svm.LinearSVC(C=C),
          SVC(kernel='rbf', gamma=0.005, C=C),
          SVC(kernel='poly', degree=1, gamma=0.005, C=C))
models = (clf.fit(X, y) for clf in models)

# title for the plots
titles = ('SVC with linear kernel (C=1.6)',
          'LinearSVC (linear kernel)',
          'RBF kernel(gamma=0.005)',
          'Polynomial (degree 1)')

# Set-up 2x2 grid for plotting.
fig, sub = plt.subplots(2, 2)
plt.subplots_adjust(wspace=0.4, hspace=0.4)

X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)

for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel('meanfun')
    ax.set_ylabel('sp.ent')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)

plt.show()
```



20.0.6 Inference:

1.I still consider meanfun and sp.ent to be my favorite variables to visualize my kernal boundries in a 2D space.

2. I modeled polynomial kernal with same parameters penalty measure of C=1.6, gamma = 0.05 and degree=1 to obtain the above scatter plot.

3. When did, SVM projected my data in a 2 dimentional space and obtained above feature space with boundries that classifies gender being male and female.

4. From the above figure, we can infer:

1. Linear kernal with c=1.6 have a strict boundry

2. While in RBF kernal, the boundry is strict and also have some points misclassified

3. Polynomial kernal have a lineant boundry which are discriminative

4. From above figure, we dont see any complex boundries for polynomial and hence we need not worry about the model being over fitting

5. With respective to only these two variables, meanfun and sp.ent, It is so evident that our model is not being overfit as it gives a clear distinction between two classes 'Male' and ' Female' with no complications in margins in a feature space.

6. Thus, accuracy of 0.993 produced by Polynomial kernal can be considered to be valid enough, this means 7 out of 1000 times ther could be a misclassification. Let is see if we can minimize this error occurence by increasing the accuracy further using few ensemble learnings.

21 Step-11: Building a Decision Tree Classifier with grid search

```
In [190]: dt = tree.DecisionTreeClassifier()
          parameters = {
              'criterion': ['entropy','gini'],
              'max_depth': np.linspace(1, 20, 10),
              #'min_samples_leaf': np.linspace(1, 30, 15),
              #'min_samples_split': np.linspace(2, 20, 10)
          }
          gs = GridSearchCV(dt, parameters, verbose=0, cv=5)
          gs.fit(data_x3_train, data_y3_train)
          gs.best_params_, gs.best_score_
```

```
Out[190]: ({'criterion': 'entropy', 'max_depth': 1.0}, 0.99491353001017291)
```

```
In [191]: def measure_performance(X, y, clf, show_accuracy=True, show_classification_report=True
              y_pred = clf.predict(X)
              if show_accuracy:
                   print("Accuracy:{0:.3f}".format(metrics.accuracy_score(y, y_pred)),"\n")
              if show_classification_report:
                  print("Classification report")
                  print(metrics.classification_report(y, y_pred),"\n")

              if show_confussion_matrix:
                  print("Confussion matrix")
                  print(metrics.confusion_matrix(y, y_pred),"\n")
```

```
In [192]: dt = tree.DecisionTreeClassifier(criterion='entropy', max_depth=7)
          dt.fit(data_x3_train, data_y3_train)
          measure_performance(data_x3_test, data_y3_test, dt, show_confussion_matrix=False, show
```

```
Accuracy:0.996


Classification report
             precision    recall  f1-score   support

          0       0.99      0.99      0.99       187
          1       1.00      1.00      1.00       305

avg / total       1.00      1.00      1.00       492
```

```
In [193]: dt_performance = dt.score(data_x3_test, data_y3_test)
          dt_performance

Out[193]: 0.99593495934959353

In [261]: # lets do a 10 fold Cross validation to make sure the accuracy obtained above
          dt_eval_result = cross_val_score(dt, data_x3, data_y, cv=10, scoring='accuracy')
          print('Mean accuracy with 10 fold cross validation for Decision tree is: ',dt_eval_res

Mean accuracy with 10 fold cross validation for Decision tree is:  0.989450549451
```

21.0.1 Inference:

1. I see accuracy yealded by decision tree is 0.9894 which is less when compare to SVM classifier which was 0.993

2. We can say compare to decision tree SVM model seems more efficient

3. So, if scrutability is the requirement based on which a model needs to be built we can go ahead with decision tree model.

22 Step-12: Building a KNN with 5 nearest neighbors

```
In [198]: n_neighbors = 5
          knnclf = neighbors.KNeighborsClassifier(n_neighbors, weights='distance')
          knnclf.fit(data_x3_train, data_y3_train)

Out[198]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                     weights='distance')

In [199]: knnpreds_test = knnclf.predict(data_x3_test)

In [202]: print(knnclf.score(data_x3_test, data_y3_test))

0.99593495935


In [200]: print(classification_report(data_y3_test, knnpreds_test))

             precision    recall  f1-score   support

          0       0.99      1.00      0.99       187
          1       1.00      0.99      1.00       305
```

```
avg / total       1.00      1.00      1.00       492
```

```
In [203]: knn_performance = knnclf.score(data_x3_test, data_y3_test)
```

```
In [264]: # lets do a 10 fold Cross validation to make sure the accuracy obtained above
          knn_eval_result = cross_val_score(knnclf, data_x3, data_y, cv=10, scoring='accuracy')
          print('Mean accuracy with 10 fold cross validation for KNN is: ',knn_eval_result.mean(
```

```
Mean accuracy with 10 fold cross validation for KNN is:  0.977271750806
```

22.0.1 Inference:

1. KNN yealds an accuracy of 0.977 which is comparitive less to SVM

2. However, its accuracy touches the benchmark of 0.95 which we decided based on Naive Bayes, we can have this model for any ensemble building, etc., and it not advisable to just discard it.

2. Though KNN perform better than Naive Bayes, its accuracy is less compare to SVM

23 13. Comparing individual classifier results

```
In [204]: final_resutls = pd.DataFrame(columns=['Classifier Name', 'Performance in terms of Accu
```

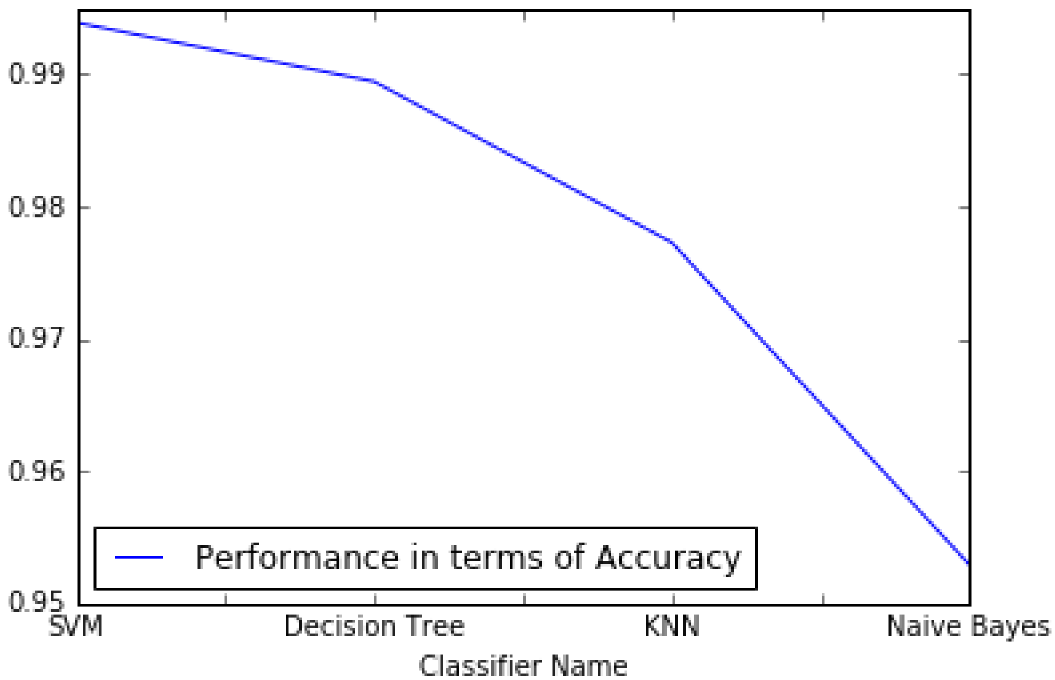
```
In [267]: final_resutls['Classifier Name'] = ['SVM','Decision Tree','KNN','Naive Bayes']
          final_resutls['Performance in terms of Accuracy'] = [svm_performance, dt_eval_result.m
                                                              knn_eval_result.mean(),nb_eval_re
```

```
In [268]: final_resutls
```

```
Out[268]:   Classifier Name  Performance in terms of Accuracy
          0             SVM                          0.993902
          1   Decision Tree                          0.989451
          2             KNN                          0.977272
          3     Naive Bayes                          0.952901
```

```
In [269]: final_resutls.plot.line(x=final_resutls['Classifier Name'])
```

```
Out[269]: <matplotlib.axes._subplots.AxesSubplot at 0x29dcb83e860>
```



23.0.1 Inference:

1. From above table and graph it seems very clear that, SVM with polynomial kernal behaves best.

2. Accuracy produces by Polynomial kernal equal to 0.993 is the highest of all cross validation results obtained from other classifiers.

3. Thus, with individual classifiers we can infer that as a individual classifier, SVM with Polynomial Kernal does a best classification wrt his voice dataset in classifying an instance as Male or Female

4. This SVM polynomial kernal tend to miss classify only 7 out of 1000 times when subjected to such dataset which is pretty good.

5. However, we will yet try to improve the accuracy further using some ensemble techniques.

24 14. Ensemble Learning

24.0.1 14.1. Bagging with Random Forest

```
In [237]: # Applying Random forest to improve the decision tree model
          from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier(criterion='entropy',max_depth=7)
rf_model = rf.fit(data_x3_train, data_y3_train)
```

```
In [241]: rfpreds_test = rf_model.predict(data_x3_test)
          rf_performance = rf_model.score(data_x3_test, data_y3_test)
```

```
In [242]: print(rf_performance)
```

```
0.997967479675
```

```
In [270]: # lets do a 10 fold Cross validation to make sure the accuracy obtained above
          rf_eval_result = cross_val_score(rf_model, data_x3, data_y, cv=10, scoring='accuracy')
          print('Mean accuracy with 10 fold cross validation for KNN is: ',rf_eval_result.mean()
```

```
Mean accuracy with 10 fold cross validation for KNN is:  0.989865322647
```

24.0.2 14.2. Boosting with Random Forest

```
In [233]: # adaboost
          adaBoost = AdaBoostClassifier()
          adaboost_model = adaBoost.fit(data_x3_train, data_y3_train)
```

```
In [243]: adboostpreds_test = adaboost_model.predict(data_x3_test)
          adaboost_performance = adaboost_model.score(data_x3_test, data_y3_test)
```

```
In [244]: print(adaboost_performance)
```

```
0.997967479675
```

```
In [271]: # lets do a 10 fold Cross validation to make sure the accuracy obtained above
          adaboost_eval_result = cross_val_score(adaboost_model, data_x3, data_y, cv=10, scoring
          print('Mean accuracy with 10 fold cross validation for KNN is: ',adaboost_eval_result.
```

```
Mean accuracy with 10 fold cross validation for KNN is:  0.993114103941
```

25 15. Reporting and Discussing the final results

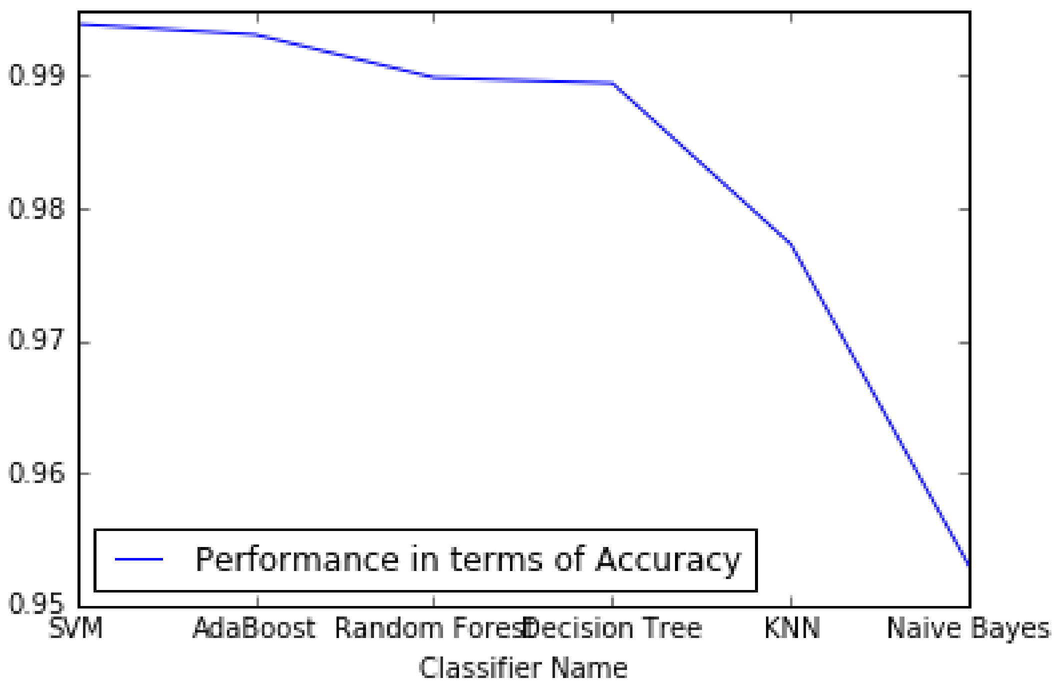
```
In [273]: final_report = pd.DataFrame(columns=['Classifier Name', 'Performance in terms of Accur
          final_report['Classifier Name'] = ['SVM','AdaBoost','Random Forest','Decision Tree','K
          final_report['Performance in terms of Accuracy'] = [svm_performance, adaboost_eval_res
                                                               dt_eval_result.mean(),
                                                               knn_eval_result.mean(),nb_eval_res
          final_report
```

Out[273]:

	Classifier Name	Performance in terms of Accuracy
0	SVM	0.993902
1	AdaBoost	0.993114
2	Random Forest	0.989865
3	Decision Tree	0.989451
4	KNN	0.977272
5	Naive Bayes	0.952901

In [274]:
```
final_report.plot.line(x=final_report['Classifier Name'])
```

Out[274]: <matplotlib.axes._subplots.AxesSubplot at 0x29dcb2690b8>



25.0.1 Inference:

1. From above table and graph it seems very clear that, SVM with polynomial kernal behaves best.

2. Accuracy produces by Polynomial kernal equal to 0.993 is the highest of all cross validation results obtained from other classifiers.

3. Thus, with individual classifiers we can infer that as a individual classifier, SVM with Polynomial Kernal does a best classification wrt his voice dataset in classifying an instance as Male or Female

4. This SVM polynomial kernal tend to miss classify only 7 out of 1000 times when subjected to such dataset which is pretty good.

5. However, we will yet try to improve the accuracy further using some ensemble techniques.

25.1 -------------------------------- End of the Book --