classification voice

Summary:

In this research, we used data-set from the Kaggle website. We have used classification algorithms and neural network algorithms to train and test data using python language. Building the best possible model, so we tried to improve the data and compare the results of the algorithms used, such as the Random Forest algorithm and some algorithms not used in previous research such as Multilayer Perceptron to find the best possible algorithm that gives the best results This project we worked on was a real challenge for our community, we have never been through this kind of analysis before, which makes it more difficult, but we've done what we can do, and this is going to be our first step in the real world of the voice systems

Introduction:

With the great technological development that we are witnessing, it has become possible to talk with machines and understand what we say through modern sound systems. Voice classification systems are gaining popularity due to their wide range of applications used in a variety of fields ranging from student exams, security services, documentation, and content-based information retrieval to criminal investigations, and search engines. However, sound systems lack many of the characteristics that specialists are still trying to develop and improve. The classification of voice by gender is one of the most important of these characteristics because of its great impact and great importance, which was shown by a group of studies, which showed that gender-based speech recognition models perform much better than sex-independent models[9]This is because determining sex, for example, in search engines gives better results according to the area of interest for each gender, They are also more able to show satisfactory results due to understanding the concerns of each unclean separately. We aim to give promising results to the classification problem in all fields. This project aims to classify voice by machine learning using classification algorithms and Neural Network Algorithm, then compare results and accuracy, improve the quality of gender voice classification.

Literature review:

As people have always been passionate about talking to machines, there was a need for a computer to identify the gender of the speakers to understand the speech and their needs according to the voice features. Hussam Allah Sheik [1] whose work aims to build a gender-based classification system applied to the GMM (Gaussian Mixture Modeling) algorithm with MFCC and SDC (where SDC fused model gave satisfactory results on Voxforge dataset). Nevertheless, when they tested on different data sets with different data languages, they were not large, and the accuracy was 80%. While Jiao [2] investigated whether there is a main effect of speakers' native language (Arabic, Korean, and Mandarin) even when speaking a second language, English, it also investigated a particular speaker-listener relationship, namely the degree of linguistic familiarity. And estimating the age of the speaker, they also found that smokers were older than non-smokers of the same ages, probably because of the effect of smoking. Acoustic characteristics of a human being, as with age estimates, researchers have found that there is a miscalculation of about ten years, as the age of younger adults has overestimated, while older people have underestimated and Accuracy in total (71%). The genetic algorithm is also used to identify a speech gender by comparing different approaches like the combination of fuzzy logic and neural network [3]. Automatic speech recognition and speaker verification can be accomplished under rather highly constrained conditions [4] an automatic gender classifier assists the development of improved male and female voice synthesizers [5, 6] in [7] an accent classification method is introduced on the top of gender classification. The most closely related work to the present one is that of Xiao et al. [8] where gender classification was incorporated in an emotional speech recognition system using a wrapper approach based on back-propagation neural networks with sequential forward selection. An accuracy of 94.65% was reported for gender classification in the Berlin dataset [9], which that research showed back-propagation neural networks for the first time studying the gender classification to express the speech in 5 emotional classes, such as anger, happiness, neutral, sadness, and surprise. The high precision it provided was one of the reasons why our team used the back-propagation algorithm-specific. This paper concentrates on a comparative study of gender and age [7] gender classification can be performed with an accuracy of 95% approximately using speech signal either from both genders or male and female separately. The accuracy for age classification is about 88%. Many recent studies have considered the problem of language identification based on various speech production features [10, 11]. A related problem that has not been explored in detail is the issue of foreign accent identification. The accent is also a challenging problem in speech recognition. It is one of the most important factors, aside from gender; it creates undesirable variability in speaker-independent speech recognition schemes. [12]An initial version of the classification algorithm classed speaker accent from among four different accents with an accuracy of 81.5% in the case of unknown text, and 88.9% assuming known text. Finally, it is shown that as accent sensitive word count increases, the ability to correctly classify accent also increases, achieving an overall classification rate of 92% among four accent classes.

Automatic identification of age and gender of a person from his/her speech has gained increasing importance in recent years because of its necessity in various commercial, medical, and forensic applications our speaking style to the person we are talking to [13]the automatic dialogue system might change the speed of its speech synthesis according to the identified user’s age [14]They[15]describes an experiment with using the Gaussian mixture models (GMM) for automatic classification of the speaker age and gender, using MFCC features and support vector machines (SVMs) with a GMM super vector the overall precision of about 75 % achieved.

3.Related Work:

There are many algorithms used to classify voice based on gender, like Linear Discriminate, Classification and Regression Trees, Decision Tree, etc. So, we used classification algorithms and neural network algorithms, some of them are used before in many types of research like Linear Discriminate, and we tried to improve accuracy by using unused algorithms like Back Propagation (Multilayer Perceptron), we think it will be better in classifying voice than other algorithms that was used before.

3.1. Algorithms

3.1.1. Random Forest:

Random forests are a group-learning method for classification (and regression) that operates by constructing a multitude of decision-making trees at training time and generating a class that is the class-learning mode for individual trees. The random forest-inducing algorithm was developed by Leo Breiman[16] and Adele Cutler[17], and "Random Forests" is their trademark. The term originated from a random decision on forests that were first proposed by Tin Kam Ho of Bell Labs in 1995. The method combines the 'bagging' idea of Breiman and the random selection of features introduced independently by Ho [18][19] and Amit and Gemanin[20] to construct a collection of decision-making trees with controlled variance. The selection of a random subset of features is an example of a random subspace method which, in Ho's formulation, is a means of implementing the classification proposed by Eugene Kleinberg.[21]

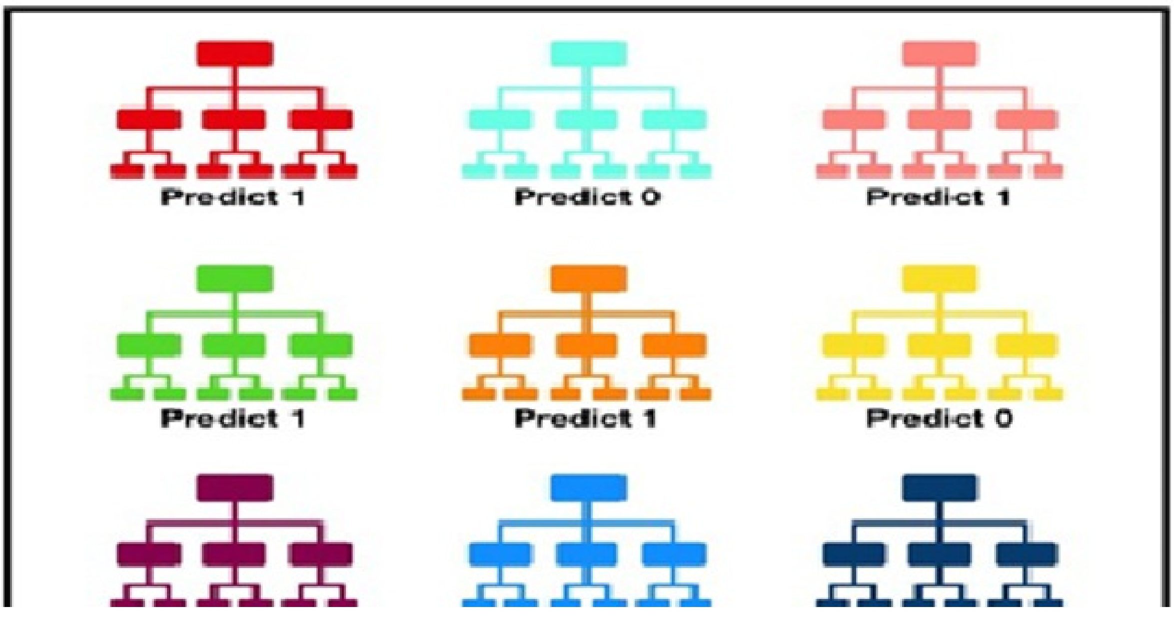


Figure1: Random Forest

3.1.2. Decision Tree (weka.classifiers.rules.DecisionTable):

Decision tables are a concise visual representation of the actions to be carried out depending on the conditions. The information expressed in the decision tables could also be represented as decision trees or as a series of if-then-else and switch-case statements in the programming language. Each decision corresponds to a variable, relationship, or predicate, the possible values of which should be listed when alternatives to the condition. Each action is a procedure or operation to be carried out, and the entries specify whether (or in what order) the action is to be carried out for a set of condition alternatives to which the entry corresponds. To make them more concise, many decision tables include alternatives that don't care about symbols in their condition. This may be a hyphen [23][24][25] or a blank hyphen[26], although the use of a blank is discouraged, as it may merely indicate that the decision table has not been finalized. [Citation Need] One of the uses of the decision tables is to identify the conditions under which certain input factors are irrelevant to the action to be taken, allowing these input tests to be skipped and thus streamlining decision-making procedures. [27]

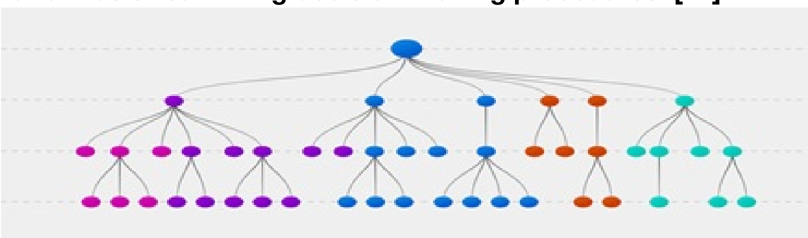


Figure2: Desicion Tree

3.1.3. Perceptron:

Perceptron is a simple binary classification algorithm, suggested by Cornell scientist Frank Rosenblatt. It helps to divide the input signals set into two parts—"yes" and "no." However, unlike many other classification algorithms, the perceptron was modeled on the essential unit of the human brain—the neuron—and has an uncanny ability to learn and solve complex problems.

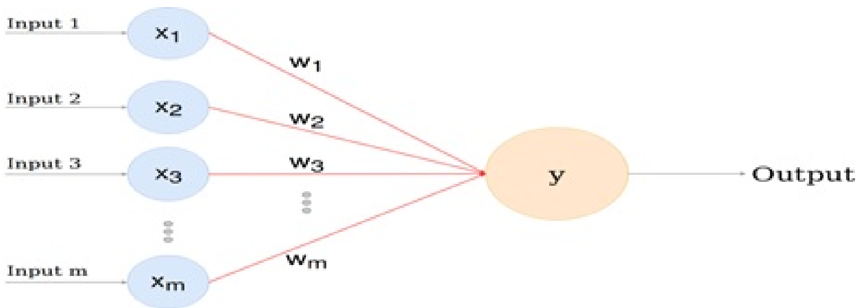
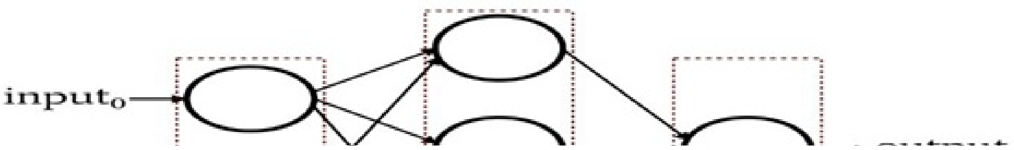


Figure3:Perceptron

3.1.4. Multilayer Perceptron:

A multilayer perceptron (MLP) is a perceptron that is paired with additional perceptron's, stacked in several layers, to solve complex problems. The diagram below shows a three-layer MLP. Also, every perceptron in the first layer to the left (the input layer) sends outputs to all the perceptron's in the second layer (the hidden layer) and all the perceptron's in the second layer send outputs to the final layer to the right (the output layer). Class MLPClassifier implements a multi-layer perceptron (MLP) algorithm that uses Back propagation to train.



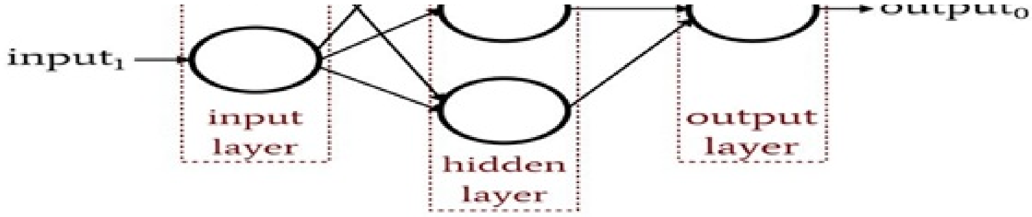


Figure4: Multilayer Perceptron

3.1.5. Support victor machine

"Support Vector Machine" (SVM) is a controlled machine learning algorithm that can be used for both classification and regression challenges. However, it is mainly used for classification problems. In the SVM algorithm, each data item is plotted as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding a hyper-plane that makes the two classes very different.

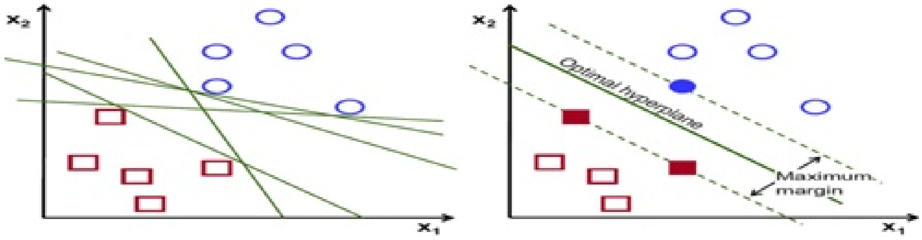


Figure5: Support victor machine

Importing

In [1]:

```
import numpy as np
np.random.seed(1337) # for reproducibility
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from keras.models import Sequential
from sklearn.model_selection import GridSearchCV
from keras.layers import Dense
from keras.optimizers import SGD, Adam, Adagrad ,RMSprop
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.neural_network import MLPClassifier
import keras.backend as K
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import Perceptron
from sklearn.neural_network import MLPClassifier
%matplotlib inline
```

4. Methodology:

Data Reading

In [40]:

```
voice = pd.read_csv('voicegender.csv',na_values=['NA'])
voice.head()
```

Out[40]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	mode	centroi
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893369	0.491918	0.000000	0.05978
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892193	0.513724	0.000000	0.06600
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.846389	0.478905	0.000000	0.07731
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.963322	0.727232	0.083878	0.15122
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.971955	0.783568	0.104261	0.13512

Gender recognition by speech and speech analysis .

This database was created to identify the voice as male or female, based on the acoustic properties of voice and speech. The data set consists of 3,168 recorded voice samples from male and female speakers.

In [3]:

```
voice.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3168 entries, 0 to 3167
Data columns (total 21 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   meanfreq  3168 non-null   float64
 1   sd        3168 non-null   float64
 2   median    3168 non-null   float64
 3   Q25       3168 non-null   float64
 4   Q75       3168 non-null   float64
 5   IQR       3168 non-null   float64
 6   skew      3168 non-null   float64
 7   kurt      3168 non-null   float64
 8   sp.ent    3168 non-null   float64
 9   sfm       3168 non-null   float64
 10  mode      3168 non-null   float64
 11  centroid  3168 non-null   float64
 12  meanfun   3168 non-null   float64
 13  minfun    3168 non-null   float64
 14  maxfun    3168 non-null   float64
 15  meandom   3168 non-null   float64
 16  mindom    3168 non-null   float64
 17  maxdom    3168 non-null   float64
 18  dfrange   3168 non-null   float64
 19  modindx   3168 non-null   float64
 20  label     3168 non-null   object 
dtypes: float64(20), object(1)
memory usage: 519.9+ KB
```

In [4]:

```
voice.describe()
```

Out[4]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent
count	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000
mean	0.180907	0.057126	0.185621	0.140456	0.224765	0.084309	3.140168	36.568461	0.895127
std	0.029918	0.016652	0.036360	0.048680	0.023639	0.042783	4.240529	134.928661	0.044980
min	0.039363	0.018363	0.010975	0.000229	0.042946	0.014558	0.141735	2.068455	0.738651
25%	0.163662	0.041954	0.169593	0.111087	0.208747	0.042560	1.649569	5.669547	0.861811
50%	0.184838	0.059155	0.190032	0.140286	0.225684	0.094280	2.197101	8.318463	0.901767
75%	0.199146	0.067020	0.210618	0.175939	0.243660	0.114175	2.931694	13.648905	0.928713
max	0.251124	0.115273	0.261224	0.247347	0.273469	0.252225	34.725453	1309.612887	0.981997

The previous info() shows some information about the data set, such as the number of records that is 3168, the number of teachers that is 21 with the class column, and 20 of the columns that are float except for the class that is the object.

While discribe() show us a set of properties for each feature describing the number of records, column mean, minimum and maximum value in data.

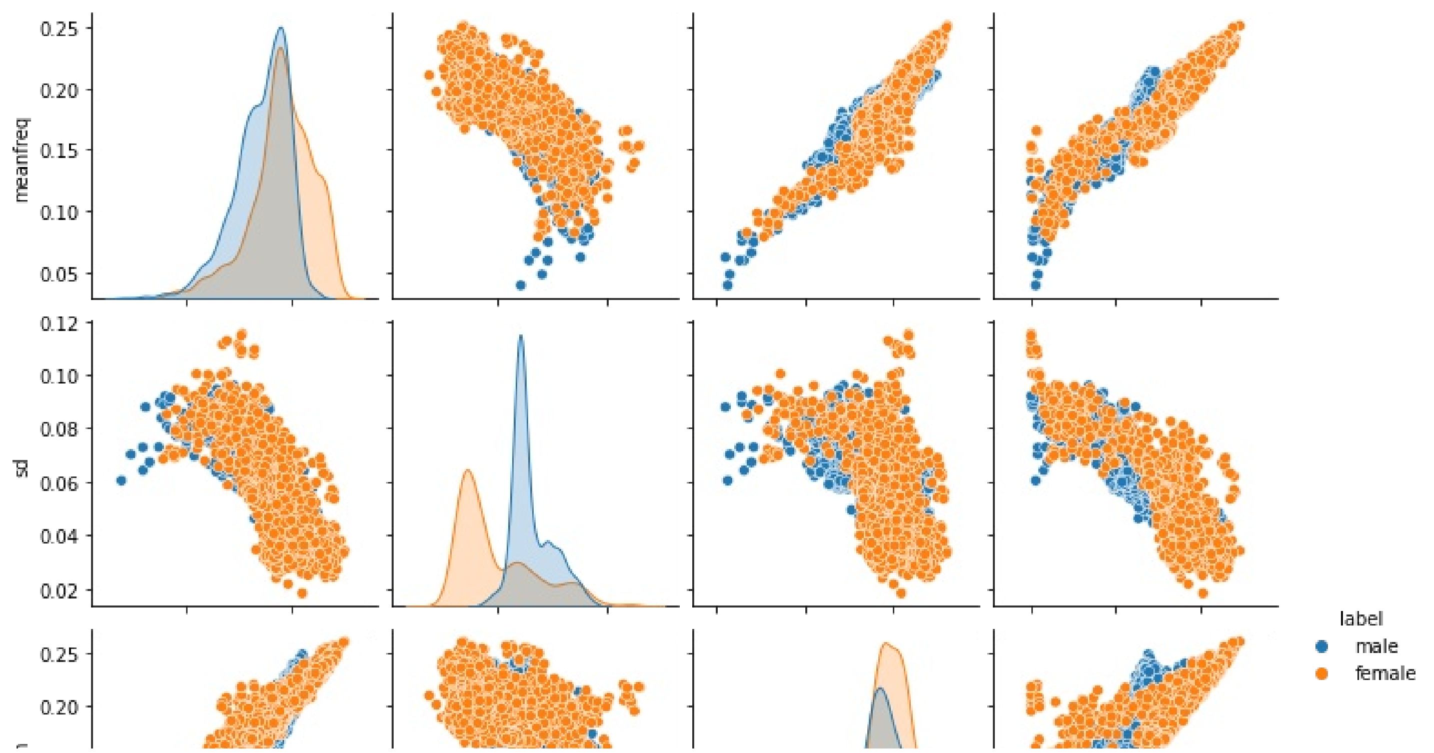
In [5]:

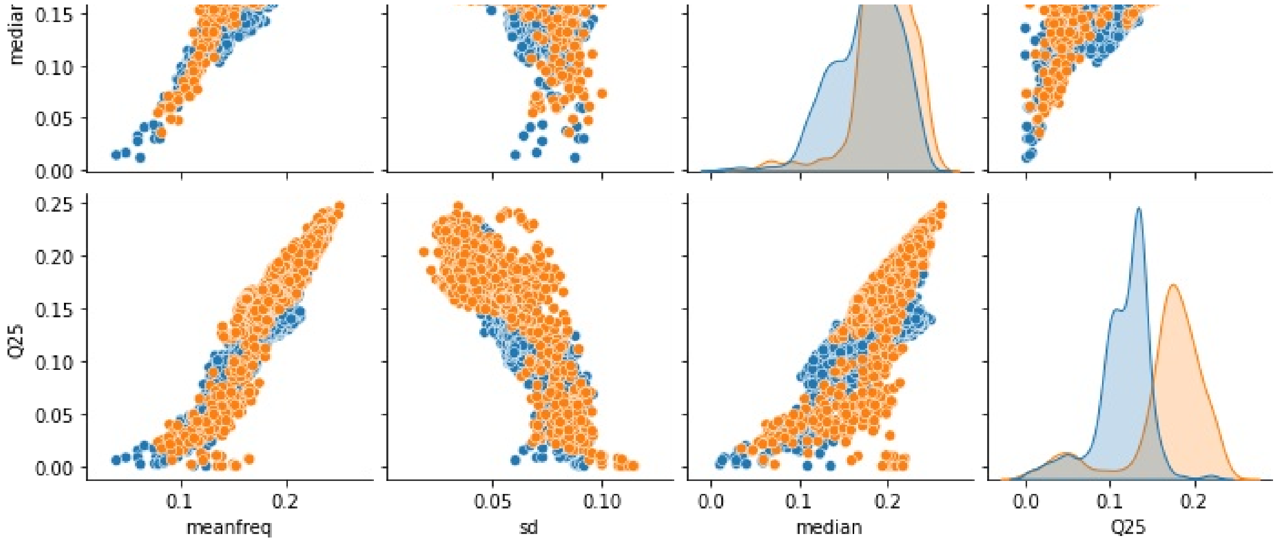
```
sns.pairplot(voice[['meanfreq','sd','median','Q25','label']], hue='label',diag_kws={'bw':
0.2})
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
```

Out[5]:

```
<seaborn.axisgrid.PairGrid at 0x7f47bf7cf198>
```





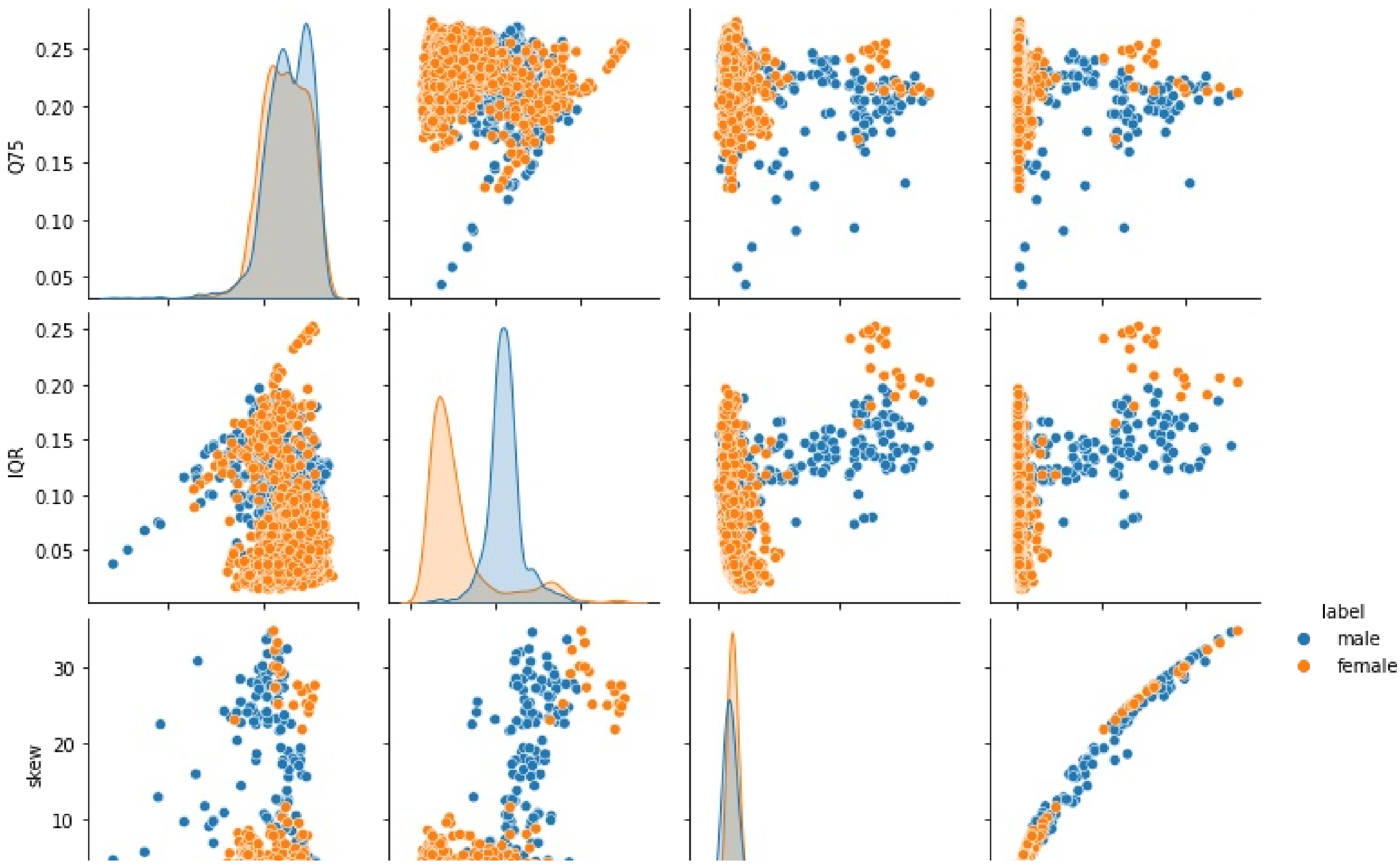
In [6]:

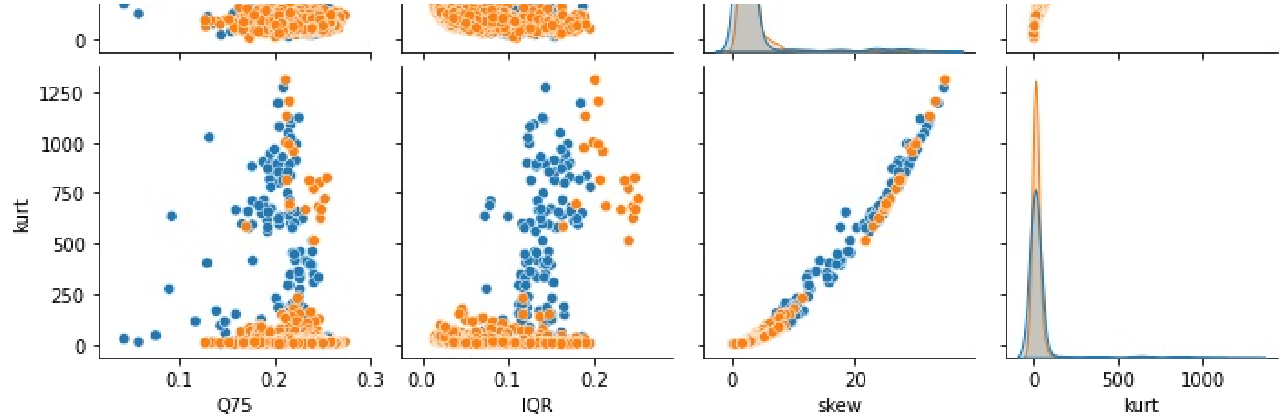
```
sns.pairplot(voice[['Q75','IQR','skew','kurt' ,'label']], hue='label',diag_kws={'bw': 0.2})
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_method`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_method`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_method`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_method`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
```

Out[6]:

```
<seaborn.axisgrid.PairGrid at 0x7f47bf4a23c8>
```





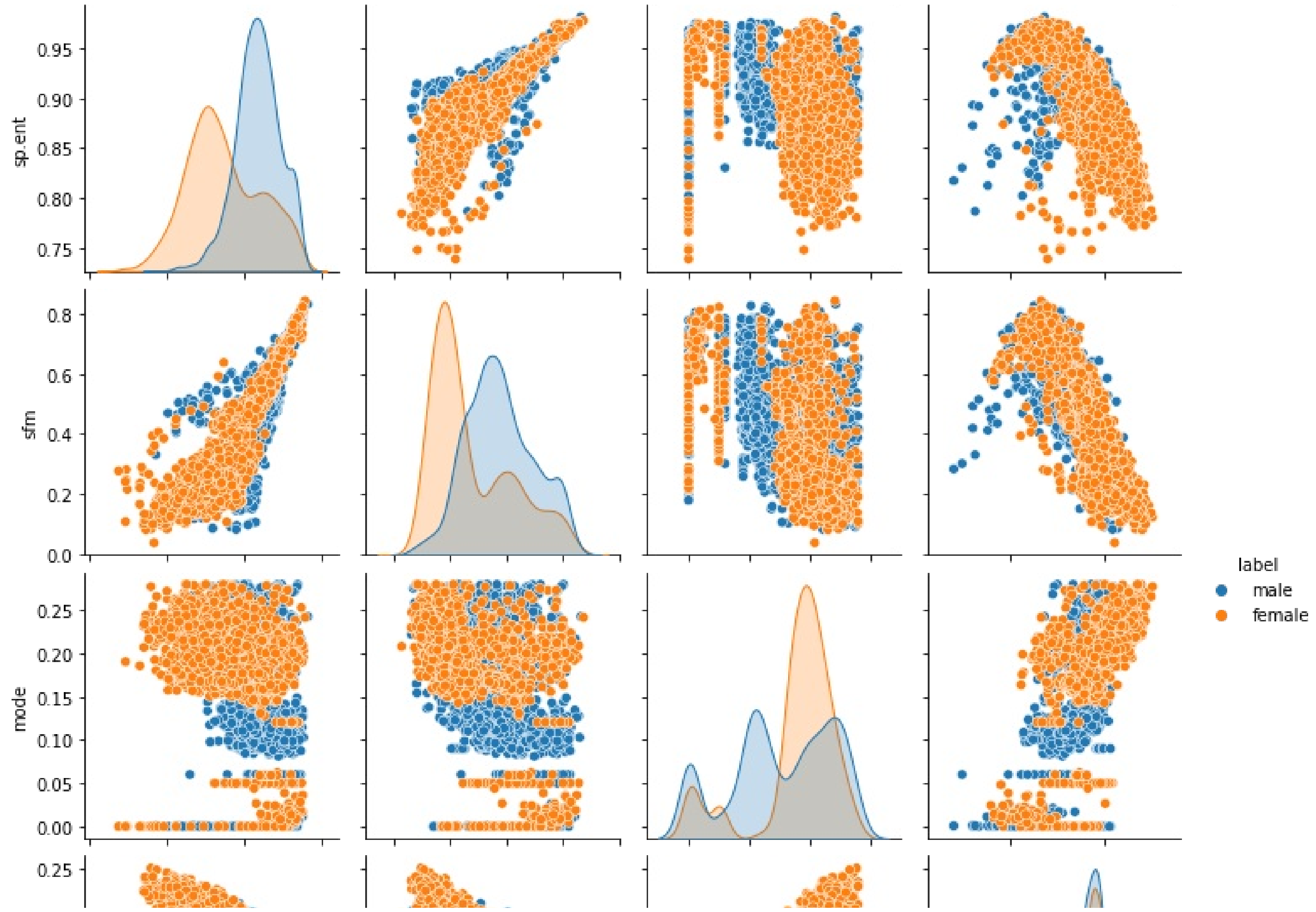
In [7]:

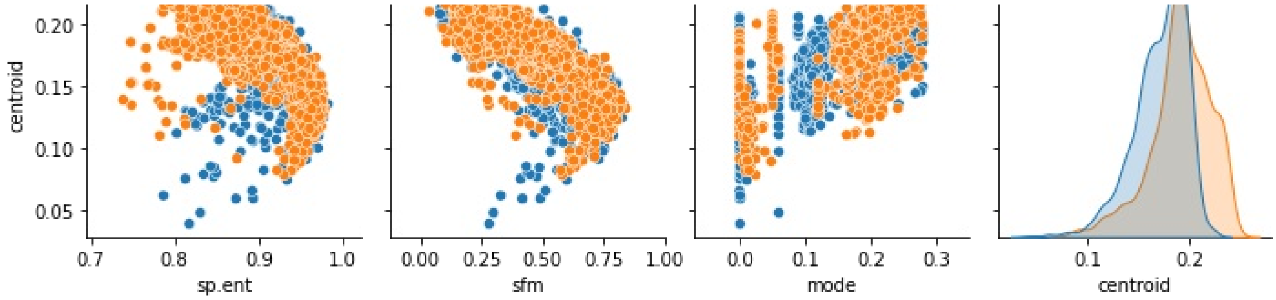
```
sns.pairplot(voice[['sp.ent','sfm','mode','centroid' ,'label']], hue='label',diag_kws={'bw': 0.2})
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_method`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_method`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_method`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The `bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_method`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
```

Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x7f47b5ed6748>
```





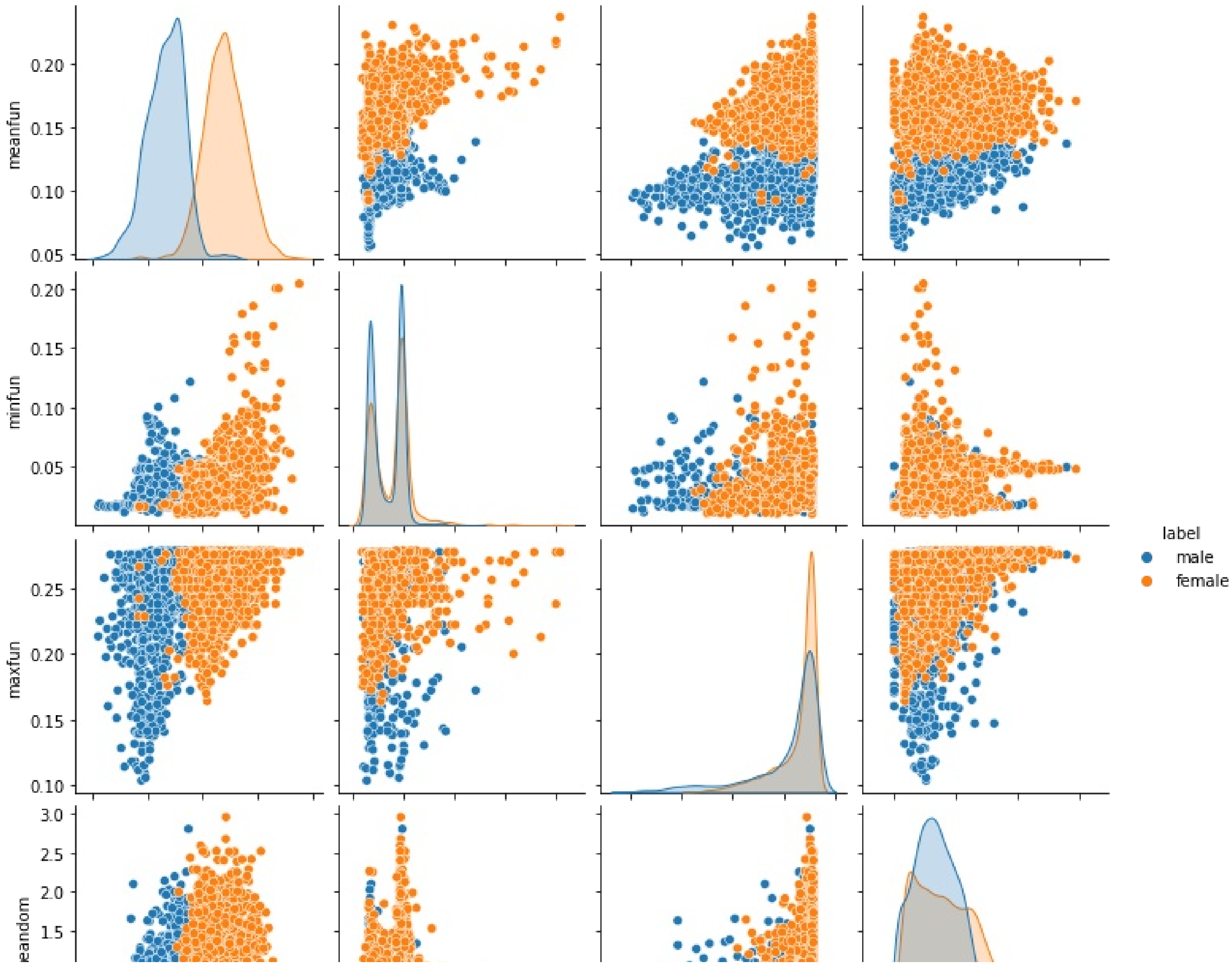
In [8]:

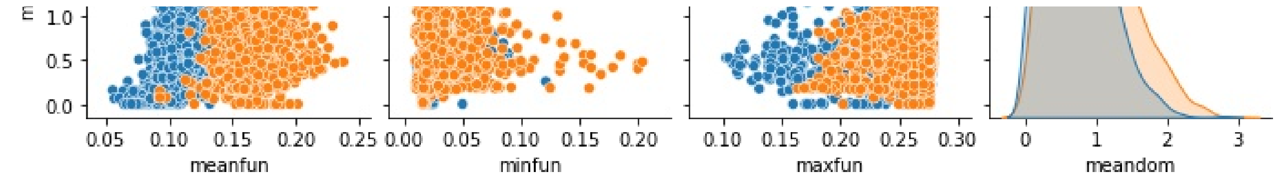
```
sns.pairplot(voice[['meanfun','minfun','maxfun','meandom','label']], hue='label',diag_kws
={'bw': 0.2})
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
```

Out[8]:

```
<seaborn.axisgrid.PairGrid at 0x7f47b34fba90>
```





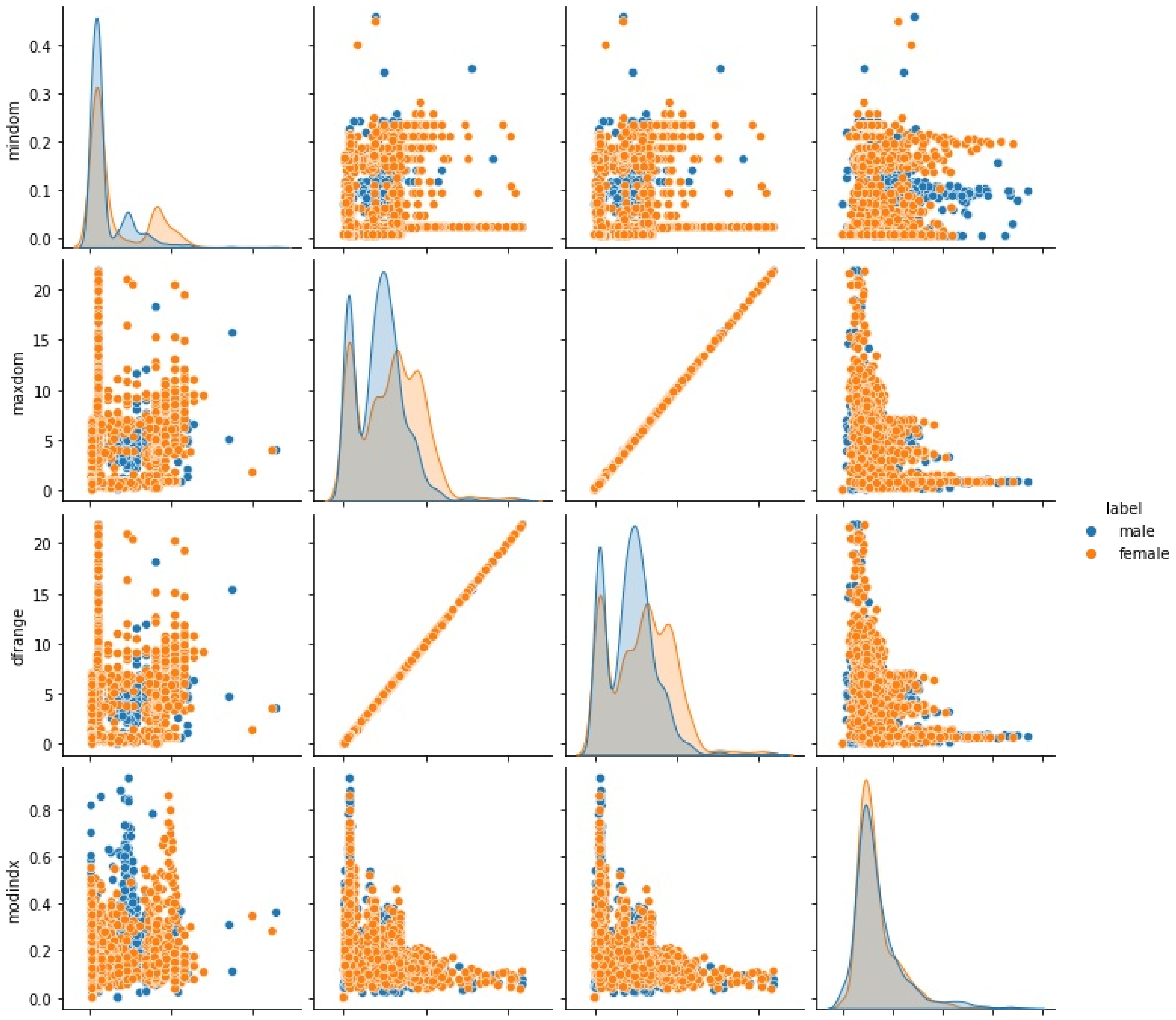
In [9]:

```
sns.pairplot(voice[['mindom','maxdom','dfrange','modindx','label']], hue='label',diag_kws
={'bw': 0.2})
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:1657: FutureWarning: The
`bw` parameter is deprecated in favor of `bw_method` and `bw_adjust`. Using 0.2 for `bw_m
ethod`, but please see the docs for the new parameters and update your code.
  warnings.warn(msg, FutureWarning)
```

Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x7f47b2eca908>
```



In [10]:

```
voice['label'].value_counts()
```

Out[10]:

```
male      1584
female    1584
Name: label, dtype: int64
```

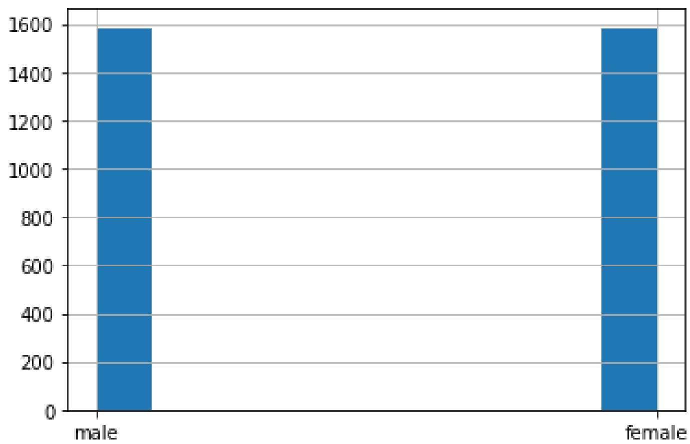
As for the class, it contains two labels Female or male, and we know from the figure below that the number of males is equal to the number of females.

In [11]:

```
voice['label'].hist()
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47b2821208>
```



Appropriate algorithms and data also help a lot in designing a model or machine learning system capable of distinguishing between different labels, and there are some steps we have to follow in order to show the best results. First, we tidy the data by looking for what might hinder the learning process, such as missing or incorrectly entered data or duplicated records and finally making sure that the class contains only two types of values. Then, pre-processing, we tried to improve the data by showing a percentage of the feature correlation and deleting highly correlated features. After that, we divided the data into training data and testing data and improved the scale of the features by using a standard scaler. Finally, we can start training and testing using classification algorithms and get the best results.

Tidying the data

As we explained earlier, there are a number of steps that need to be followed, starting with the tidying the data, and at this stage, we first checked whether there was duplication in the records because duplicate records could affect the results of the algorithms and produce unreal results.

In [42]:

```
voice.duplicated().sum()
```

Out[42]:

```
2
```

So by ‘duplicated().sum()’ command we can know how much records are duplicated, and we found two duplicate records so we deleted them via the following command.

In [43]:

In [43]:

```
voice.drop_duplicates(inplace=True, ignore_index=True)
```

Then, based on the information we saw it before in figure6, there are no missing data, but as we see know in there are some zero values that we think are 'nulls', so we've deleted those records that are estimated at 240 records.

In [44]:

```
voice.columns
```

Out[44]:

```
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')
```

In [45]:

```
voice[(voice['meanfreq'] == 0.0000 ) | (voice['sd'] == 0.0000 ) |(voice['median'] == 0.0
000)
     |(voice['Q25'] == 0.0000 )|(voice['Q75'] == 0.0000 )|(voice['IQR'] == 0.0000 )
     |(voice['skew'] == 0.0000 )|(voice['kurt'] == 0.0000 )|(voice['sp.ent'] == 0.0000 )
     |(voice['sfm'] == 0.0000)|(voice['mode'] == 0.0000 )|(voice['centroid'] == 0.0000 )
     |(voice['meanfun'] == 0.0000 )|(voice['minfun'] == 0.0000 )|(voice['maxfun'] == 0.0
000)
     |(voice['meandom'] == 0.0000 )|(voice['mindom']== 0.0000 )|(voice['maxdom']== 0.000
0 )
     |(voice['dfrange']== 0.0000 )|(voice['modindx']== 0.0000 )|(voice['label']== 0.0000
)].head(4)
```

Out[45]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	mode	centroid
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893369	0.491918	0.0	0.059781
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892193	0.513724	0.0	0.066009
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.846389	0.478905	0.0	0.077316
73	0.200830	0.053066	0.210059	0.185332	0.236198	0.050866	1.840901	6.006801	0.907683	0.386818	0.0	0.200830

In [46]:

```
voice.drop(voice[(voice['meanfreq'] == 0.0000 ) | (voice['sd'] == 0.0000 ) |(voice['medi
an'] == 0.0000)
     |(voice['Q25'] == 0.0000 )|(voice['Q75'] == 0.0000 )|(voice['IQR'] == 0.0000 )
     |(voice['skew'] == 0.0000 )|(voice['kurt'] == 0.0000 )|(voice['sp.ent'] == 0.0000 )
     |(voice['sfm'] == 0.0000)|(voice['mode'] == 0.0000 )|(voice['centroid'] == 0.0000 )
     |(voice['meanfun'] == 0.0000 )|(voice['minfun'] == 0.0000 )|(voice['maxfun'] == 0.0
000)
     |(voice['meandom'] == 0.0000 )|(voice['mindom']== 0.0000 )|(voice['maxdom']== 0.000
0 )
     |(voice['dfrange']== 0.0000 )|(voice['modindx']== 0.0000 )|(voice['label']== 0.0000
)].index , inplace = True)
```

Then, by showing the class's unique values, we made sure that there were only two labels, male and female.

In [47]:

```
voice['label'].unique()
```

Out[47]:

```
array(['male', 'female'], dtype=object)
```

And as a result of the deletion of a group record, there was a slight difference between the number of female records and the number of male records.

In [48]:

```
voice['label'].value_counts()
```

Out[48]:

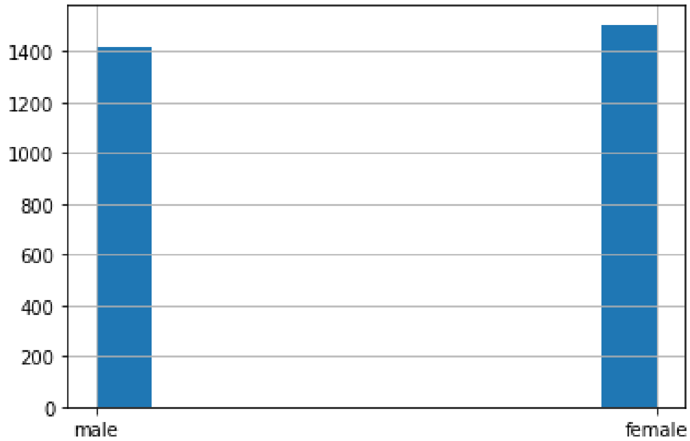
```
female    1507
male      1419
Name: label, dtype: int64
```

In [49]:

```
voice['label'].hist()
```

Out[49]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47b1ec1470>
```



Also, we gave values for class labels where females are '0' and males are '1' by mapping.

In [50]:

```
mapdec={'male':1,'female':0}
voice['labelnew']=voice['label'].map(mapdec)
```

In [51]:

```
assert len(voice['label'].unique()) == 2
```

In [52]:

```
voice.to_csv('voice-clean.csv', index=False)
```

Second step, is the pre-processing phase, at this stage, we showed a correlation of features between each other and based on the results, like figure below, there were approximately five features with a high correlation of more than 90%,

Preprocessing

Drop Highly Correlated Features

Drop Highly Correlated Features

In [53]:

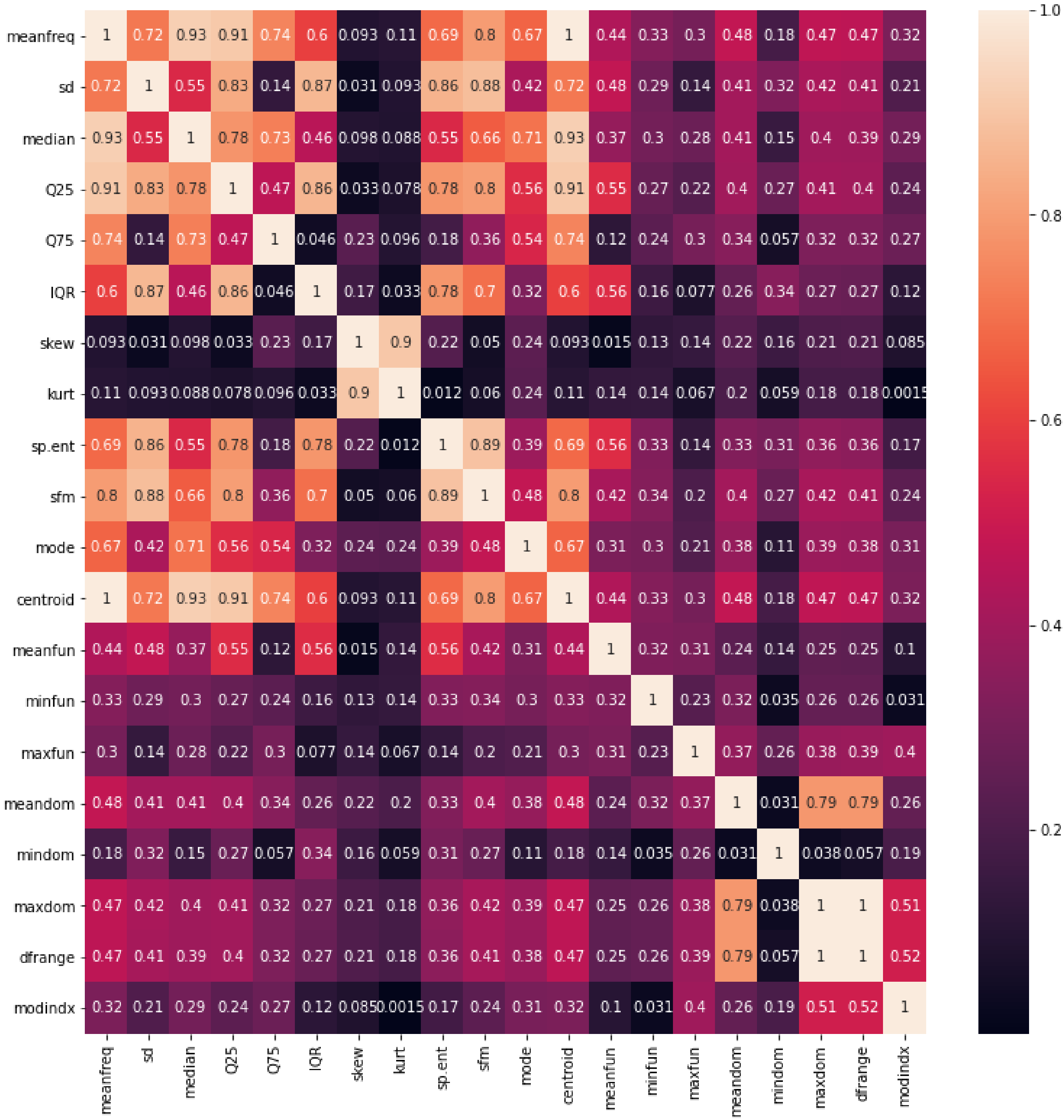
```
clnvoice= pd.read_csv('voice-clean.csv')
```

In [54]:

```
column=clnvoice.columns[:-1:]
cor_matrix = clnvoice[column].corr().abs()
fig=plt.figure(figsize=(13,13))
sns.heatmap(cor_matrix,annot=True)
```

Out[54]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47b3107ba8>
```



According to the above figure, we have many features that have a strong correlation with other features, which means that the values of these properties are approximately equal, which means that they do not affect the results and may reduce accuracy sometimes, so we often delete few of them.

In [55]:

```
upper_tri = abs(cor_matrix.where(np.triu(np.ones(cor_matrix.shape),k=1).astype(np.bool))
```

```
)
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.900000)]
print(20-len(to_drop))
```

15

After deleting a group of features based on their correlation, the sum of what remains of our features is 15 out of 20

In [56]:

```
voice1 = clnvoice.drop(clnvoice[to_drop], axis=1)
```

In [57]:

```
voice1.drop('label', axis=1,inplace=True)
```

In [58]:

```
voice1.head(3)
```

Out[58]:

	meanfreq	sd	Q75	IQR	skew	sp.ent	sfm	mode	meanfun	minfun	maxfun	meandom
0	0.151228	0.072111	0.207955	0.111374	1.232831	0.963322	0.727232	0.083878	0.088965	0.017798	0.250000	0.201497
1	0.135120	0.079146	0.206045	0.127325	1.101174	0.971955	0.783568	0.104261	0.106398	0.016931	0.266667	0.712812
2	0.132786	0.079557	0.209592	0.141634	1.932562	0.963181	0.738307	0.112555	0.110132	0.017112	0.253968	0.298222

Standard Scaler

After that, we divided the data into training and testing data by using (train-test-split) and then applied a standard scale on them

In [59]:

```
all_inputs = voice1[voice1.columns.drop('labelnew')]
all_labels = voice1['labelnew'].values
```

In [74]:

```
(training_inputs,
 testing_inputs,
 training_classes,
 testing_classes) = train_test_split(all_inputs, all_labels, test_size=0.30, random_stat
e=111)
```

In [75]:

```
sc =StandardScaler()
training_inputs = sc.fit_transform(training_inputs)
testing_inputs = sc.transform(testing_inputs)
```

5. Results:

classification

5.1 DecisionTree

The decision tree has many parameters each one of them has its effect on the result so we need to choose the best choice for every parameter. At first, we started with criterion, what is the criterion? It is the function to measure the quality of a split, and have two choices,first gini where it use for the Gini impurity, and entropy for the information gain.

However, at entropy criterion, we have a decision tree score of 0.972

In [79]:

```
from sklearn.tree import DecisionTreeClassifier

# Create the classifier
decision_tree_classifier = DecisionTreeClassifier( criterion='entropy',class_weight='bala
nced',random_state=40)
# Train the classifier on the training set
decision_tree_classifier.fit(training_inputs, training_classes)

# Validate the classifier on the testing set using classification accuracy
print('decision_tree_classifier.score =',decision_tree_classifier.score(testing_inputs, t
esting_classes))
```

```
decision_tree_classifier.score = 0.9726651480637813
```

In the other side, at gini criterion, we have a decision tree score of 0.963

In [80]:

```
from sklearn.tree import DecisionTreeClassifier

# Create the classifier
decision_tree_classifier = DecisionTreeClassifier( criterion='gini',class_weight='balance
d',random_state=40)
# Train the classifier on the training set
decision_tree_classifier.fit(training_inputs, training_classes)

# Validate the classifier on the testing set using classification accuracy
print('decision_tree_classifier.score =',decision_tree_classifier.score(testing_inputs, t
esting_classes))
```

```
decision_tree_classifier.score = 0.9635535307517085
```

In the conclusion, entropy criterion is better than gini criterion because it gives a higher score or accuracy.

In [34]:

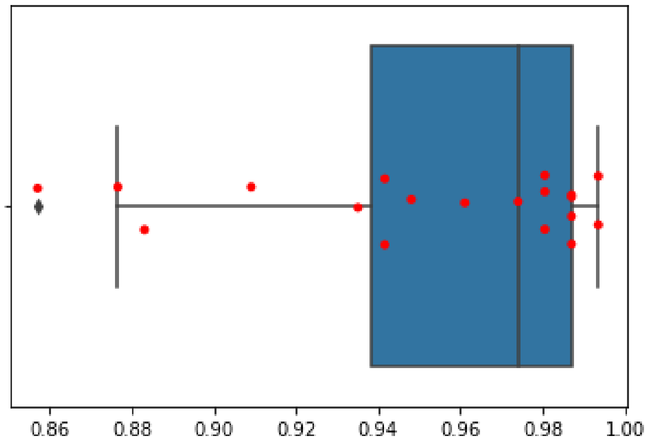
```
dt_scores = cross_val_score(decision_tree_classifier, all_inputs, all_labels, cv=19)

sns.boxplot(dt_scores)
sns.stripplot(dt_scores, jitter=True, color='r')
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the
following variable as a keyword arg: x. From version 0.12, the only valid positional argu
ment will be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  FutureWarning
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the
following variable as a keyword arg: x. From version 0.12, the only valid positional argu
ment will be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  FutureWarning
```

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47b25df2e8>
```



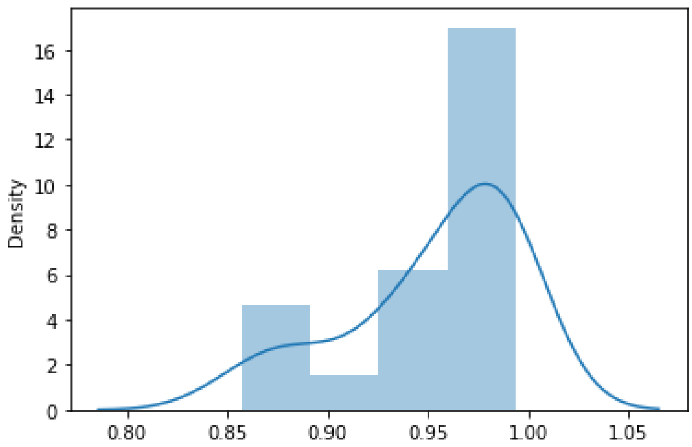
In [35]:

```
sns.distplot(dt_scores)
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
tplot` is a deprecated function and will be removed in a future version. Please adapt you
r code to use either `displot` (a figure-level function with similar flexibility) or `his
tplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47b22dacf8>
```



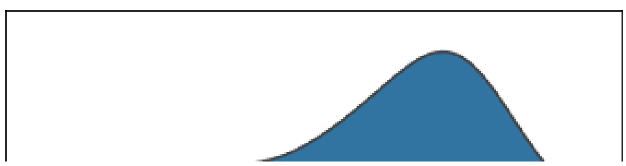
In [36]:

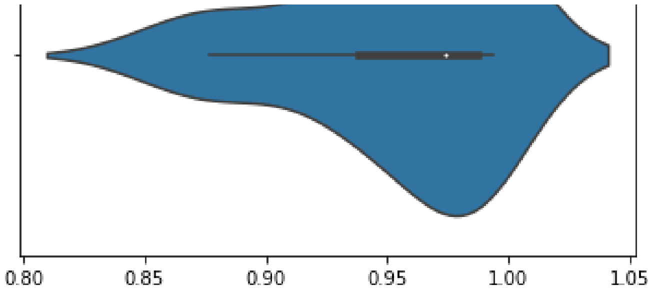
```
sns.violinplot(dt_scores)
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the
following variable as a keyword arg: x. From version 0.12, the only valid positional argu
ment will be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  FutureWarning
```

Out[36]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f47b22560f0>
```





5.2 RandomForest

Random forest is optimized for the decision tree so we applied what we did in the decision tree to the random forest. However, at entropy criterion, we have a random forest score of 0.984.

In [78]:

```
from sklearn.ensemble import RandomForestClassifier

random_forest_classifier = RandomForestClassifier(bootstrap=True ,class_weight='balanced'
,
                                                 criterion='entropy',random_state=43 )

random_forest_classifier.fit(training_inputs,training_classes)
random_forest_classifier.score(testing_inputs,testing_classes)
```

Out[78]:

```
0.9840546697038725
```

In the other hand, at gini criterion, we have a random forest score of 0.980.

In [82]:

```
from sklearn.ensemble import RandomForestClassifier

random_forest_classifier = RandomForestClassifier(bootstrap=True, class_weight='balanced'
,
                                                 criterion='gini',random_state=40)

random_forest_classifier.fit(training_inputs,training_classes)
random_forest_classifier.score(testing_inputs,testing_classes)
```

Out[82]:

```
0.9806378132118451
```

Finally, as we saw there is no difference between gini and entropy criterion or it was so low.

5.3 Logistic Regression Model

Optimizers

In this algorithm we used grid search to find the best option for each parameter in order to get the best accuracy. So we applied grid search on optimizer to choose best optimizer from this list

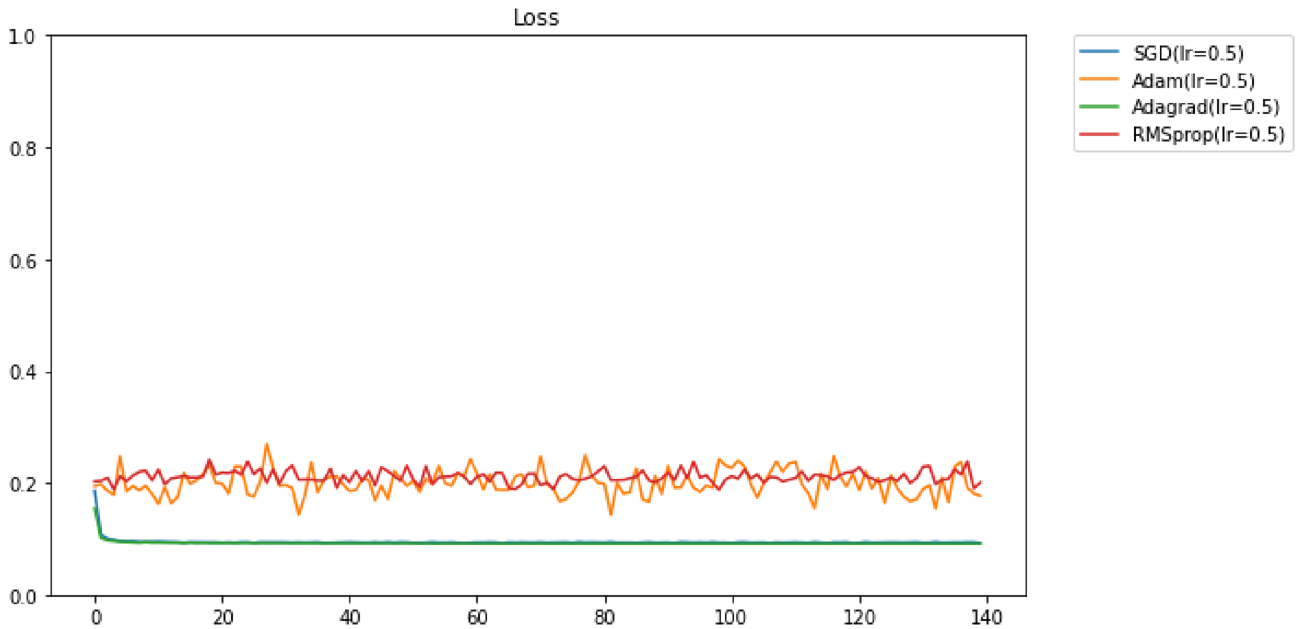
[SGD,Adam,Adagrad,RMSprop] ,and the best two were [SGD,Adagrad]

In [ ]:

```
dflist = []

optimizers = ['SGD(lr=0.5)',
              'Adam(lr=0.5)',
              'Adagrad(lr=0.5)',
              'RMSprop(lr=0.5)',]
for opt_name in optimizers:

    K.clear_session()

    model = Sequential()
    model.add(Dense(1, input_shape=(15,), activation='sigmoid'))
    model.compile(loss='binary_crossentropy',
                  optimizer=eval(opt_name),
                  metrics=['accuracy'])
    h = model.fit(training_inputs, training_classes, batch_size=20, epochs=140, verbose=
0)

    dflist.append(pd.DataFrame(h.history, index=h.epoch))

historydf = pd.concat(dflist, axis=1)
metrics_reported = dflist[0].columns
idx = pd.MultiIndex.from_product([optimizers, metrics_reported],
                                 names=['optimizers', 'metric'])
historydf.columns = idx
```

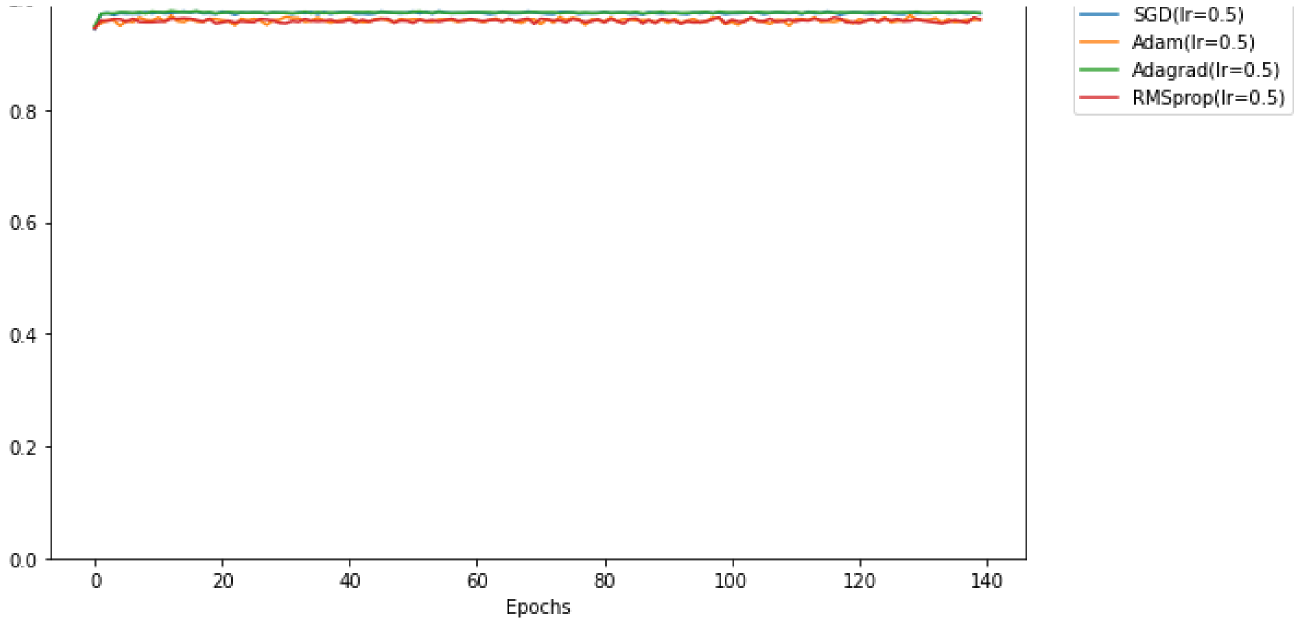
In [ ]:

```
mpl.rc('figure', figsize=(10, 10))

ax = plt.subplot(211)
historydf.xs('loss', axis=1, level='metric').plot(ylim=(0,1), ax=ax)
plt.title("Loss")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

ax = plt.subplot(212)
historydf.xs('accuracy', axis=1, level='metric').plot(ylim=(0,1), ax=ax)
plt.title("Accuracy")
plt.xlabel("Epochs")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.tight_layout()
```



Accuracy

1.0



Learning Rates

The next grid search was on learning rate, the figure below show us the best rate, and the best rates from the list [0.05, 0.01, 0.1, 0.5, 0.8, 0.2] were 0.8 and 0.5.

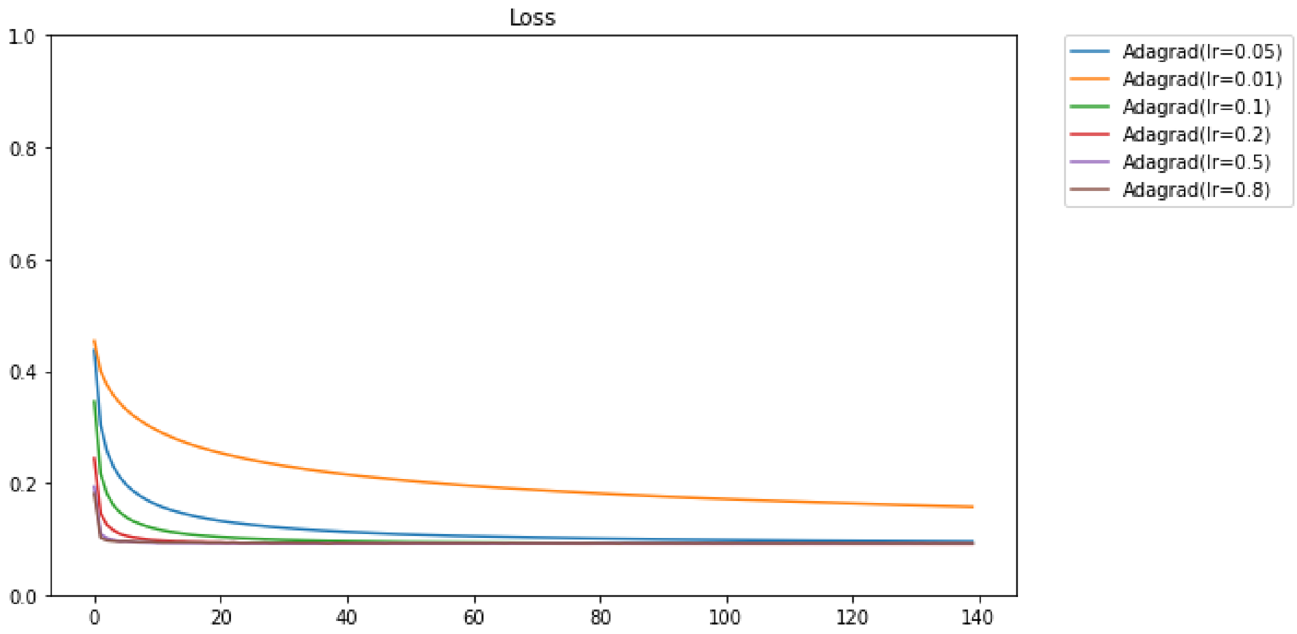
In [ ]:

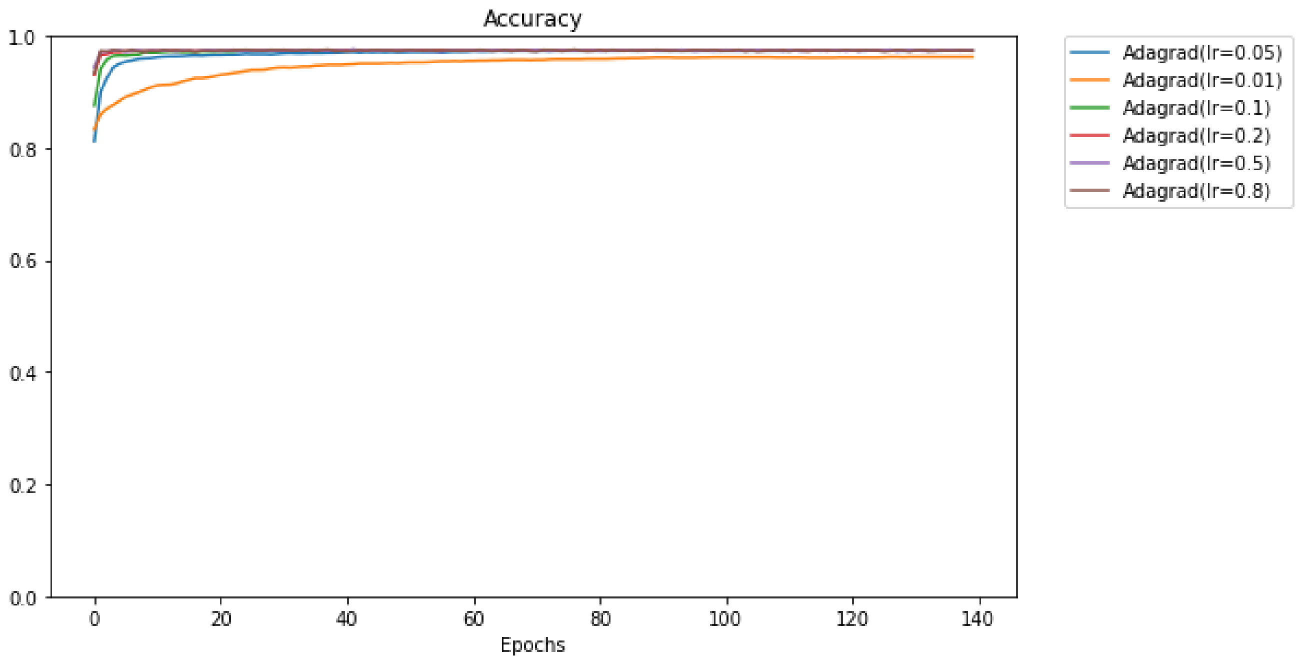
```
dflist = []

optimizers = ['Adagrad(lr=0.05)',
              'Adagrad(lr=0.01)',
              'Adagrad(lr=0.1)',
              'Adagrad(lr=0.2)',
              'Adagrad(lr=0.5)',
              'Adagrad(lr=0.8)']

for opt_name in optimizers:

    K.clear_session()

    model = Sequential()
    model.add(Dense(1, input_shape=(15,), activation='sigmoid'))
    model.compile(loss='binary_crossentropy',
                  optimizer=eval(opt_name),
                  metrics=['accuracy'])
    h = model.fit(training_inputs, training_classes, batch_size=30, epochs=140, verbose=
0)

    dflist.append(pd.DataFrame(h.history, index=h.epoch))

historydf = pd.concat(dflist, axis=1)
metrics_reported = dflist[0].columns
idx = pd.MultiIndex.from_product([optimizers, metrics_reported],
                                 names=['optimizers', 'metric'])
historydf.columns = idx
```

In [ ]:

```
mpl.rc('figure', figsize=(10, 10))

ax = plt.subplot(211)
historydf.xs('loss', axis=1, level='metric').plot(ylim=(0,1), ax=ax)
plt.title("Loss")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

ax = plt.subplot(212)
```

```
historydf.xs('accuracy', axis=1, level='metric').plot(ylim=(0,1), ax=ax)
plt.title("Accuracy")
plt.xlabel("Epochs")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.tight_layout()
```





Batch Sizes

Then we have the batch size, whose options are as follows [2,3,7,10,11,15,20,25,30] and according to the results below, it was the best batch size from the list 20.

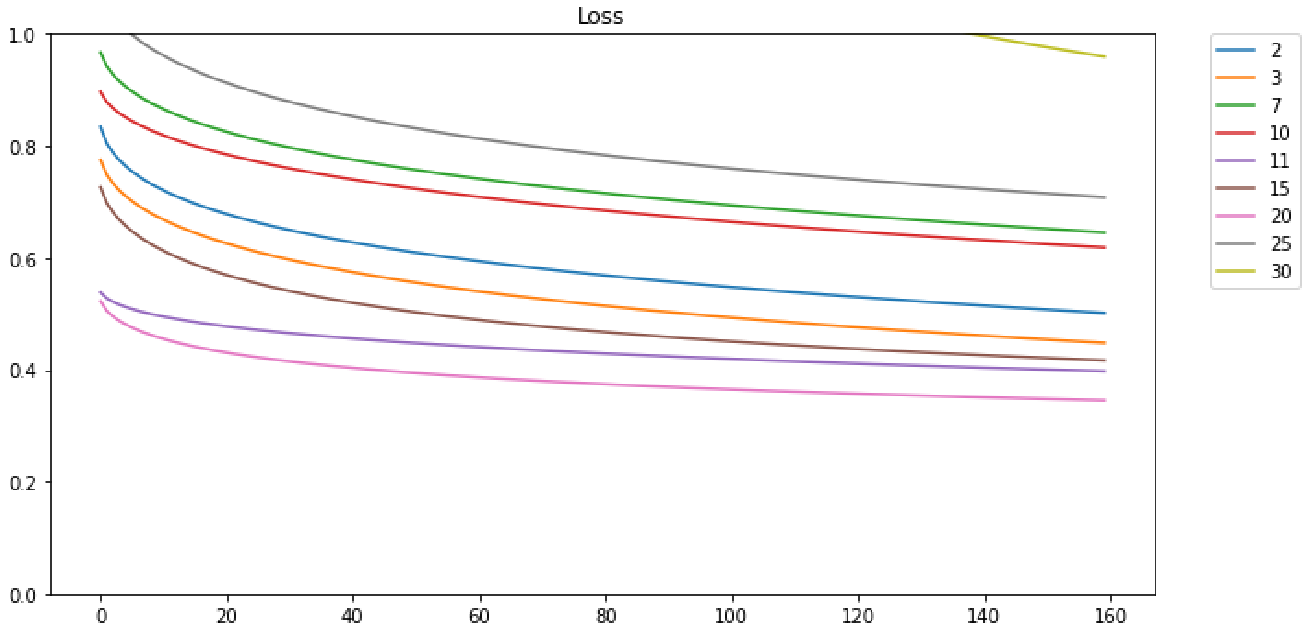
In [ ]:

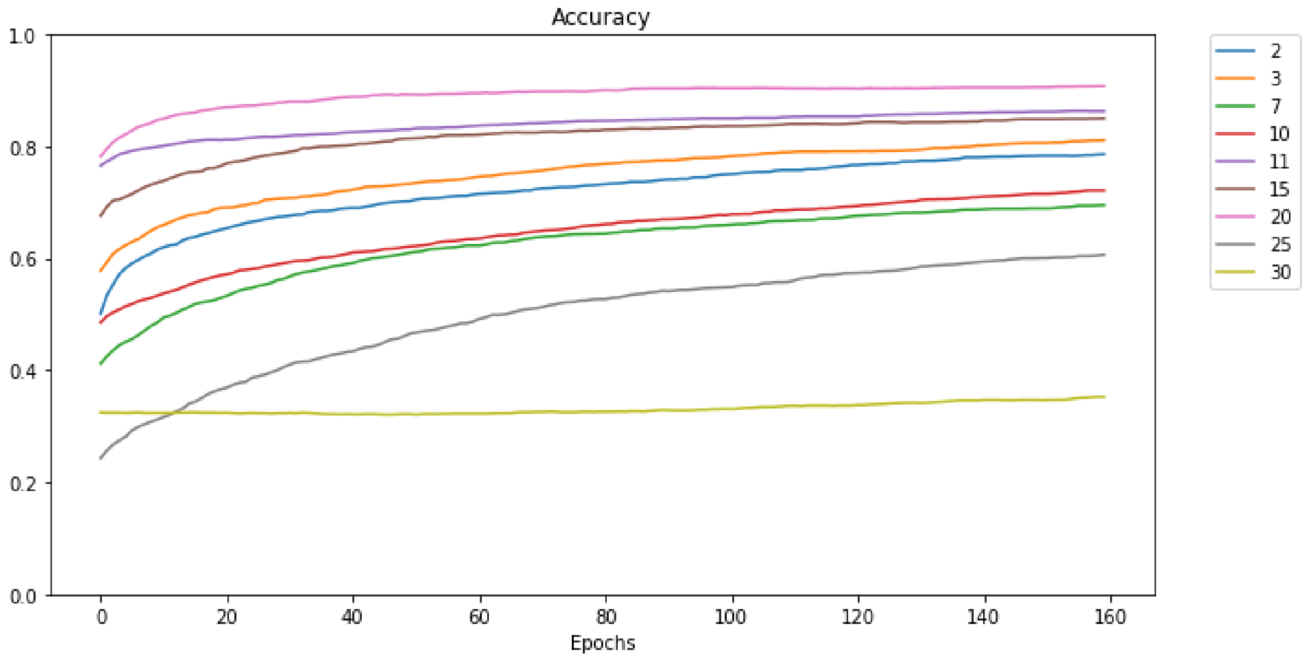
```
dflist = []

batch_sizes = [2,3,7,10,11,15,20,25,30]

for batch_size in batch_sizes:
    K.clear_session()

    model = Sequential()
    model.add(Dense(1, input_shape=(15,), activation='sigmoid'))
    model.compile(loss='binary_crossentropy',
```

```
                    optimizer='Adagrad',
                    metrics=['accuracy'])
    h = model.fit(training_inputs, training_classes, batch_size=batch_size, verbose=0,ep
ochs=160)

    dflist.append(pd.DataFrame(h.history, index=h.epoch))

historydf = pd.concat(dflist, axis=1)
metrics_reported = dflist[0].columns
idx = pd.MultiIndex.from_product([batch_sizes, metrics_reported],
                                 names=['batch_size', 'metric'])
historydf.columns = idx
```

In [ ]:

```
ax = plt.subplot(211)
historydf.xs('loss', axis=1, level='metric').plot(ylim=(0,1), ax=ax)
plt.title("Loss")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

ax = plt.subplot(212)
historydf.xs('accuracy', axis=1, level='metric').plot(ylim=(0,1), ax=ax)
plt.title("Accuracy")
plt.xlabel("Epochs")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.tight_layout()
```





Initialization

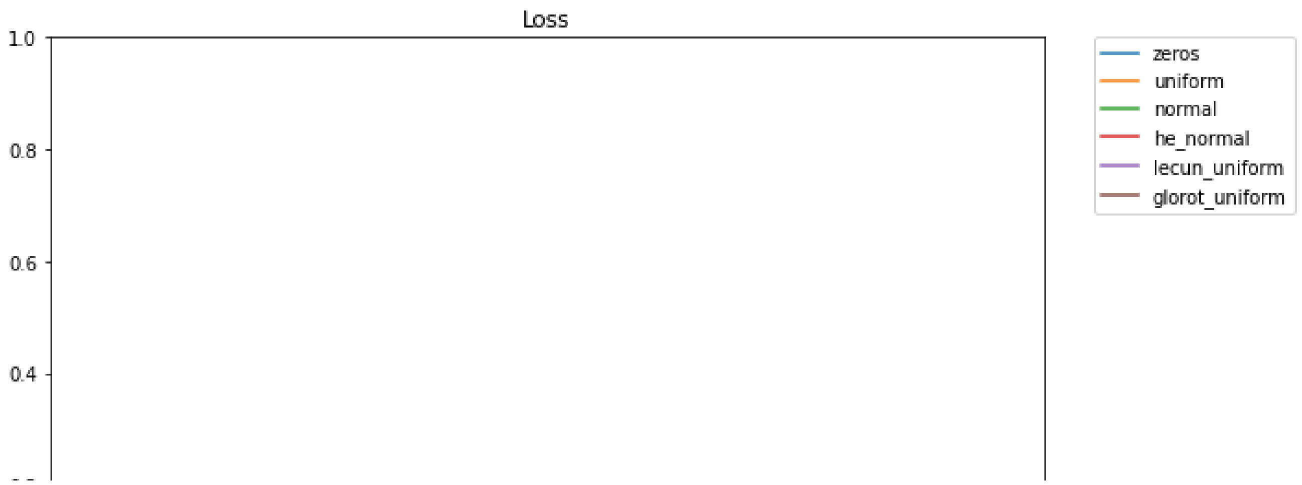
Finally, it was the last grid search for the initialization which the options were as follows [zeros, uniform, normal, he_normal, lecun_uniform, glorot_uniform] and according to the results below, the configuration menu was all good.

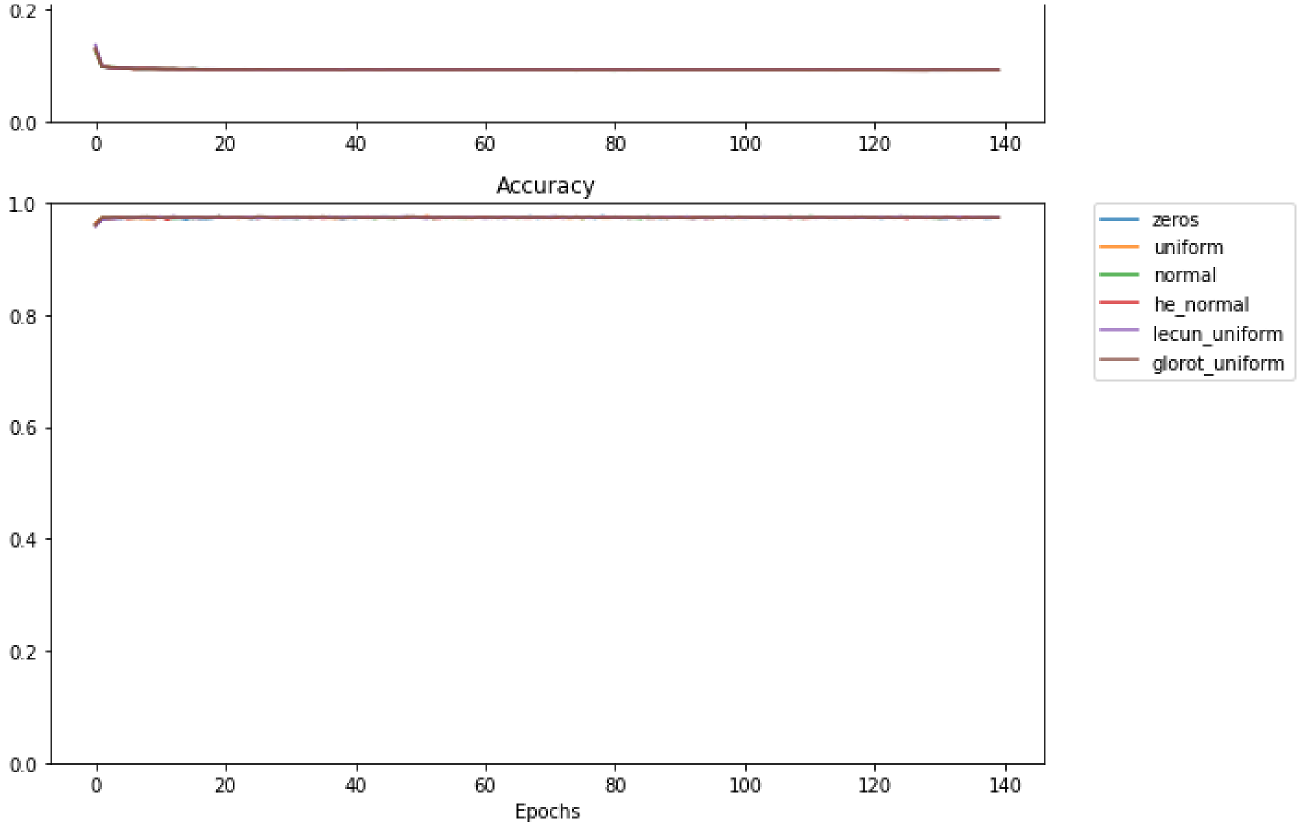
In [ ]:

```
dflist = []
initializers = ['zeros', 'uniform', 'normal',
                'he_normal', 'lecun_uniform','glorot_uniform']

for init in initializers:

    K.clear_session()

    model = Sequential()
    model.add(Dense(1, input_shape=(15,),
                    kernel_initializer=init,
                    activation='sigmoid'))

    model.compile(loss='binary_crossentropy',
                  optimizer=Adagrad(lr=0.5),
                  metrics=['accuracy'])

    h = model.fit(training_inputs, training_classes, batch_size=7, epochs=140, verbose=0
)

    dflist.append(pd.DataFrame(h.history, index=h.epoch))

historydf = pd.concat(dflist, axis=1)
metrics_reported = dflist[0].columns
idx = pd.MultiIndex.from_product([initializers, metrics_reported],
                                 names=['initializers', 'metric'])

historydf.columns = idx
```

In [ ]:

```
ax = plt.subplot(211)
historydf.xs('loss', axis=1, level='metric').plot(ylim=(0,1), ax=ax)
plt.title("Loss")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

ax = plt.subplot(212)
historydf.xs('accuracy', axis=1, level='metric').plot(ylim=(0,1), ax=ax)
plt.title("Accuracy")
plt.xlabel("Epochs")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.tight_layout()
```





Logistic Regression Model

The grid search is useful to choose the best options for parameters like what we've done before, and after we've got the best options, we've used the Logistic Regression Model. Figure below to view the results of the model when entering data

---

In [ ]:

```
del model
model = Sequential()
model.add(Dense(1,kernel_initializer='lecun_uniform' , input_shape=(15,), activation='sig
moid'))
model.add(Dense(1,kernel_initializer='lecun_uniform' ,  activation='sigmoid'))
model.compile(Adagrad(lr=0.8), 'binary_crossentropy', metrics=['accuracy'])
model.fit(training_inputs, training_classes,batch_size=20, epochs=160, verbose=0,shuffle
=False )
model.evaluate(testing_inputs,testing_classes)
```

```
30/30 [==============================] - 0s 1ms/step - loss: 0.0843 - accuracy: 0.9789
```

Out[ ]:

```
[0.08433771133422852, 0.9789473414421082]
```

According to the previous figure, the accuracy was 0.9789, and what could also be important as a result of the high accuracy we achieved is that the loss was very low and the value was 0.08.

5.4 Confusion Matrix

A confusion matrix is a technique used to summarize the results of a classification algorithm. If you have an unequal number of observations in each class, or if you have more than two classes in your dataset, the accuracy of classification alone can be misleading. It will give you a better understanding of what your

classification model is getting right and what kinds of mistakes it makes by computing a confusion matrix. Based on the previous definition, we chose a group of algorithms to choose the best algorithm and use a confusion matrix to see more details on the results. Seven different algorithms were used to compare accuracy and loss, and the algorithms were labeled as follows: 'LR', Logistic Regression LDA ', Linear Discriminant Analysis 'LRCV', Logistic Regression CV 'KNN', K Neighbors Classifier 'CART', Decision Tree Classifier 'NB', Gaussian NB 'SVM', Support victor machine

---

In [ ]:

```
models = []
models.append(('LR', LogisticRegression(random_state=42,solver='liblinear')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('LRCV', LogisticRegressionCV(solver='liblinear' )))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier(criterion='entropy', max_depth=4, max_feat
ures=12,
                                             random_state=42, splitter='best')))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(random_state=42)))
```

In [ ]:

```
results = []
names = []
seed=4
for name, model in models:
    kfold = KFold(n_splits=8, random_state=seed, shuffle=True)
    cv_results = cross_val_score(model, training_inputs, training_classes, cv=kfold, sco
ring='accuracy')
    results.append(cv_results)
    names.append(name)
    print(f"{name}, {cv_results.mean()}, {cv_results.std()}))")
```

```
LR, 0.9724729241877257, 0.011117946746168676))
LDA, 0.9684115523465704, 0.010170963600708155))
LRCV, 0.9724729241877257, 0.010820919502059366))
KNN, 0.9711191335740073, 0.006990944668244427))
CART, 0.970216606498195, 0.010009509482318947))
NB, 0.9237364620938628, 0.023426619769706503))
SVM, 0.9796931407942238, 0.005690216702580535))
```

As we can see from the result figure, the support vector machine gives us the best result with higher accuracy and lower loss, so the confusion matrix has been applied on it.

In [ ]:

```
cls = SVC(class_weight='balanced',random_state=111)
cls.fit(training_inputs,training_classes)
label_pred = cls.predict(testing_inputs)
print('-------------------------------------------------------')

print("------- Accuracy --------\n")
print(accuracy_score(testing_classes,label_pred))
print('-------------------------------------------------------')

print("------- Confusion Matrix --------\n")

print(confusion_matrix(testing_classes,label_pred))

print('-------------------------------------------------------')


print("------- Classifcation Report-------- \n")
```

```
print(classification_report(testing_classes,label_pred))

print('-------------------------------------------------------')
```

```
-------------------------------------------------------
------- Accuracy --------

0.9810526315789474
-------------------------------------------------------
------- Confusion Matrix --------

[[448   9]
 [  9 484]]
-------------------------------------------------------
------- Classifcation Report--------

              precision    recall  f1-score   support

           0       0.98      0.98      0.98       457
           1       0.98      0.98      0.98       493

    accuracy                           0.98       950
   macro avg       0.98      0.98      0.98       950
weighted avg       0.98      0.98      0.98       950

-------------------------------------------------------
```

From the confusion matrix, we can see that there are nine male records classified as female from 457 records, and nine female records classified as male from 493 records. And as expected from the support vector machine algorithm, it has produced great results for the test data as the accuracy is 98 and the loss is estimated to be 0.01.

5.5 neuralnetwork algorethm

A neural network is a series of algorithms that seek to recognize underlying relationships in a data set through a process that mimics the way the human brain operates. Neural networks can adapt to changing inputs; therefore, the network generates the best possible result without the need to redesign the output criteria. Perceptron and Multilayer Perceptron are some of the neural network algorithms, so we've expected high accuracy, and as known, neural network algorithms are powerful and always give amazing results.

In [83]:

```
(training_inputs,
 testing_inputs,
 training_classes,
 testing_classes) = train_test_split(all_inputs, all_labels, test_size=0.30, random_stat
e=42)
```

In [84]:

```
sc =StandardScaler()
training_inputs = sc.fit_transform(training_inputs)
testing_inputs = sc.transform(testing_inputs)
```

5.5.1 Perceptron

Starting with the Perceptron algorithm and as is expected of it to give promising results, the accuracy of the test data was equivalent to 0.971

In [86]:

```
perceptron= Perceptron(shuffle=True,random_state=25,class_weight='balanced')
perceptron.fit(training_inputs,training_classes)
perceptron.score(training_inputs,training_classes)
```

Out[86]:

```
0.97216796875
```

In [87]:

```
y=perceptron.predict(testing_inputs)
accuracy_score(testing_classes,y)
```

Out[87]:

```
0.9715261958997722
```

5.5.2 Multilayer Perceptron

Multilayer Perceptron Algorithm It was developed from the Perceptron algorithm so it was expected that its results would be better than the Perceptron results, where testing data accuracy was 0.981, and because its results were wonderful, we used a confusion matrix to see the results in more detail.

In [ ]:

```
mlp= MLPClassifier(hidden_layer_sizes=25, activation='relu', solver='sgd',
                   batch_size='auto', learning_rate='adaptive', learning_rate_init=0.005
, power_t='invscaling',
                   shuffle=True, random_state=43, tol=0.0001)
mlp.fit(training_inputs,training_classes)
print(mlp.score(training_inputs,training_classes))
y=mlp.predict(testing_inputs)

print(accuracy_score(testing_classes,y))
```

```
0.98014440433213
0.9810526315789474
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the op
timization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

In [ ]:

```
cls = MLPClassifier(hidden_layer_sizes=25, activation='relu', solver='sgd',
                   batch_size='auto', learning_rate='adaptive', learning_rate_init=0.005
, power_t='invscaling',
                   shuffle=True, random_state=43, tol=0.0001)
cls.fit(training_inputs,training_classes)
label_pred = cls.predict(testing_inputs)
print('--------------------------------------------------------')

print("------- Accuracy --------\n")
print(accuracy_score(testing_classes,label_pred))
print('--------------------------------------------------------')

print("------- Confusion Matrix --------\n")

print(confusion_matrix(testing_classes,label_pred))

print('--------------------------------------------------------')
```

```
print("------- Classifcation Report-------- \n")


print(classification_report(testing_classes,label_pred))
```

```
--------------------------------------------------------
------- Accuracy --------

0.9810526315789474
--------------------------------------------------------
------- Confusion Matrix --------

[[447  10]
 [  8 485]]
--------------------------------------------------------
------- Classifcation Report--------

              precision    recall  f1-score   support

           0       0.98      0.98      0.98       457
           1       0.98      0.98      0.98       493

    accuracy                           0.98       950
   macro avg       0.98      0.98      0.98       950
weighted avg       0.98      0.98      0.98       950
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:5
71: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the op
timization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

Reproducibility

In [ ]:

```
!pip install watermark
```

```
Collecting watermark
  Downloading watermark-2.1.0-py2.py3-none-any.whl (5.7 kB)
Requirement already satisfied: ipython in c:\users\global\anaconda3\lib\site-packages (fr
om watermark) (7.12.0)
Requirement already satisfied: importlib-metadata<3.0; python_version < "3.8" in c:\users
\global\anaconda3\lib\site-packages (from watermark) (1.5.0)
Requirement already satisfied: jedi>=0.10 in c:\users\global\anaconda3\lib\site-packages
(from ipython->watermark) (0.14.1)
Requirement already satisfied: pickleshare in c:\users\global\anaconda3\lib\site-packages
(from ipython->watermark) (0.7.5)
Requirement already satisfied: colorama; sys_platform == "win32" in c:\users\global\anaco
nda3\lib\site-packages (from ipython->watermark) (0.4.3)
Requirement already satisfied: pygments in c:\users\global\anaconda3\lib\site-packages (f
rom ipython->watermark) (2.5.2)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in c:\users\g
lobal\anaconda3\lib\site-packages (from ipython->watermark) (3.0.3)
Requirement already satisfied: traitlets>=4.2 in c:\users\global\anaconda3\lib\site-packa
ges (from ipython->watermark) (4.3.3)
Requirement already satisfied: backcall in c:\users\global\anaconda3\lib\site-packages (f
rom ipython->watermark) (0.1.0)
Requirement already satisfied: setuptools>=18.5 in c:\users\global\anaconda3\lib\site-pac
kages (from ipython->watermark) (45.2.0.post20200210)
Requirement already satisfied: decorator in c:\users\global\anaconda3\lib\site-packages (
from ipython->watermark) (4.4.1)
Requirement already satisfied: zipp>=0.5 in c:\users\global\anaconda3\lib\site-packages (
from importlib-metadata<3.0; python_version < "3.8"->watermark) (2.2.0)
Requirement already satisfied: parso>=0.5.0 in c:\users\global\anaconda3\lib\site-package
s (from jedi>=0.10->ipython->watermark) (0.5.2)
Requirement already satisfied: wcwidth in c:\users\global\anaconda3\lib\site-packages (fr
om prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->watermark) (0.1.8)
Requirement already satisfied: six in c:\users\global\anaconda3\lib\site-packages (from t
raitlets>=4.2->ipython->watermark) (1.14.0)
```

```
Requirement already satisfied: ipython-genutils in c:\users\global\anaconda3\lib\site-pac
kages (from traitlets>=4.2->ipython->watermark) (0.2.0)
Installing collected packages: watermark
Successfully installed watermark-2.1.0
```

In [ ]:

```
%load_ext watermark
```

In [ ]:

```
%watermark -a 'Anwar_Abbass' -nmv --packages numpy,pandas,sklearn,matplotlib,seaborn
```

```
Author: Anwar_Abbass

Python implementation: CPython
Python version       : 3.7.6
IPython version      : 7.12.0

numpy     : 1.18.1
pandas    : 1.0.1
sklearn   : 0.22.1
matplotlib: 3.1.3
seaborn   : 0.10.0

Compiler    : MSC v.1916 64 bit (AMD64)
OS          : Windows
Release     : 10
Machine     : AMD64
Processor   : Intel64 Family 6 Model 69 Stepping 1, GenuineIntel
CPU cores   : 4
Architecture: 64bit
```

6. Conclusion

In this work, we proposed to apply a set of algorithms on datasets based on gender as a class after tremendous studies to make voice recognition clearer and more understanding. The results we achieved varying according to the differences in the working mechanism of the algorithms. All of them were unexpectedly very good. In conclusion, we can choose the best algorithms that give higher accuracy, starting with the Multilayer Perception Algorithm with (98.1) accuracy then the Support Vector Machine algorithm with (98.1) accuracy and the Random Forest Algorithm with (98.4) accuracy. However, we can have higher accuracy if we have a larger dataset with a better feature, larger data give algorithm more record to train, and a good feature makes it easier for algorithms to separate labels by features. The challenge for us is to find large data with good features to build a conventional system with good algorithms. Human nature is passionate about talking to machines and computers. Because of this very reason, voice classification systems and their improvement will not die soon and we hope to find other powerful algorithms.

7. References

1. **Sheikh, H., Who is speaking? Male or female. 2013: GRIN Verlag.**
2. **Jiao, D., et al., Age estimation in foreign-accented speech by non-native speakers of English. Speech Communication, 2019. 106: p. 118-126.**
3. **Adi, K., et al., Identifying the developmental phase of Plasmodium falciparum in malaria-infected red blood cells using adaptive color segmentation and back propagation neural network. Int J Appl Eng Res, 2016. 11: p. 8754-9.**
4. **Wu, K. and D.G. Childers, Gender recognition from speech. Part I: Coarse analysis. The journal of the Acoustical society of America, 1991. 90(4): p. 1828-1840.**
5. **Childers, D., K. Wu, and D. Hicks. Factors in voice quality: Acoustic features related to gender. in ICASSP'87. IEEE International Conference on Acoustics, Speech, and Signal Processing. 1987. IEEE.**

6. Sedaaghi, M., A comparative study of gender and age classification in speech signals. Iranian Journal of Electrical and Electronic Engineering, 2009. 5(1): p. 1-12.
7. Lin, X. and S. Simske. Phoneme-less hierarchical accent classification. in Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004. 2004. IEEE.
8. Xiao, Z., et al., Hierarchical classification of emotional speech. IEEE Transactions on Multimedia, 2007. 37.
9. Burkhardt, F., et al. A database of German emotional speech. in Ninth European Conference on Speech Communication and Technology. 2005.
10. House, A.S. and E.P. Neuburg, Toward automatic identification of the language of an utterance. I. Preliminary methodological considerations. The Journal of the Acoustical Society of America, 1977. 62(3): p. 708-713.
11. Muthusamy, Y.K. and R.A. Cole. Automatic segmentation and identification of ten languages using telephone speech. in Second International Conference on Spoken Language Processing. 1992.
12. Hansen, J.H. and L.M. Arslan. Foreign accent classification using source generator based prosodic features. in 1995 International Conference on Acoustics, Speech, and Signal Processing. 1995. IEEE.
13. Bocklet, T., et al. Age and gender recognition for telephone applications based on gmm supervectors and support vector machines. in 2008 IEEE International Conference on Acoustics, Speech and Signal Processing. 2008. IEEE.
14. Dobry, G., et al., Supervector dimension reduction for efficient speaker age estimation based on the acoustic speech signal. IEEE Transactions on Audio, Speech, and Language Processing, 2011. 19(7): p. 1975-1985.
15. Přibil, J., A. Přibilová, and J. Matoušek, GMM-based speaker age and gender classification in Czech and Slovak. Journal of Electrical Engineering, 2017. 68(1): p. 3-12.
16. Breiman, L., Random forests. Machine learning, 2001. 45(1): p. 5-32.
17. Liaw, A., Documentation for R package randomForest. PDF). Retrieved, 2013. 15: p. 191.
18. Ho, Tin Kam (1995). “Random Decision Forest”. Proceedings of the 3rdInternational Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. 19.. Ho, Tin Kam (1998). “The Random Subspace Method for Constructing Decision Forests” IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (8): 832– 844. doi:10.1109/34.709601
19. Amit, Yali; Geman, Donald (1997). “Shape quantization and recognitionwith randomized trees”.Neural Computation 9 (7): 1545–1588. doi:10.1162/neco.1997.9.7.1545.
20. Kleinberg, Eugene (1996). “An Overtraining-Resistant Stochastic Modeling Method for Pattern Recognition”. Annals of Statistics 24 (6): 2319–2349. doi:10.1214/aos/1032181157. MR 1425956.
21. Adi, K., et al., Identifying the developmental phase of Plasmodium falciparum in malaria-infected red blood cells using adaptive color segmentation and back propagation neural network. Int J Appl Eng Res, 2016. 11: p. 8754-9.
22. LI Jing (1 April 2015). "SEEM 3430 Tutorial: Decision Tables" (PDF). p. 23. Retrieved 11 November 2017.
23. "Creating a Decision Table in Business Rules". Oracle Help Center. 6 August 2017. Retrieved 11 November 2017.
24. Ross, Ronald G. (2005). "Decision Tables, Part 2 ~ The Route to Completeness". Business Rules Journal. 6 (8). Retrieved 11 November 2017.
25. Snow, Paul (19 July 2012). "Decision Tables". DTRules: A Java Based Decision Table Rules Engine. Retrieved 11 November 2017.
26. LI Jing 2015, p. 24-25.

In [ ]: