Minor Project

Live Audio Gender Recognition using Machine Learning

Submitted in partial fulfillment of the requirments for the award of
Degree of Bachelor of Technology
(Batch: 2014 - 2018)

By:
Arpit Gogia
2K14/MC/15

Under the supervision of
Dr. C.P. Singh
(Assistant Professor)
(Department of Applied Mathematics)

Delhi Technological University
Shahbad, Daulatpur, Bawana, Delhi - 110042

Declaration

I hereby certify that the work which is presented in the Minor Project entitled Live Audio based Gender Recognition using Machine Learning in fulfillment of the requirement for the award of the Degree of Bachelor of Technology is an authentic record of my own, carried out during a period from January 2017 to May 2017, under the supervision of Dr. C.P. Singh(Assistant Professor, Department of Applied Mathematics). The matter presented in this report has not been submitted by me for the award of any other degree of this or any other Institute/University.

I acknowledge and understand that plagiarism is wrong. I understand that my research must be accurately referenced. I have followed the rules and conventions concerning referencing, citation and the use of quotations. This assignment is my own work, or my groups own unique group assignment. I acknowledge that copying someone elses assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism.

Arpit Gogia
2K14/MC/15

Supervisor Certificate

This is to certify that **ARPIT GOGIA (2K14/MC/015)** a bonafide student of Bachelor of Technology in Mathematics and Computing Engineering of Delhi Technological University, New Delhi of 2014 - 2018 batch has completed his minor project entitled Live Gender Recognition under the supervision of **Dr. C.P. Singh (Assistant Professor, Department of Applied Mathematics)**. It is further certified that the work done in this dissertation is a result of the candidate's own efforts. I wish him all success in his life.

Dr. C.P. Singh
Assistant Professor
Department of Applied Mathematics
Delhi Technological University
Bawana Road, Delhi 110042

Date: _____

Acknowledgement

"The successful completion of any task would be incomplete without accomplishing the people who made it all possible and whose constant guidance and encouragement secured us the success".

I am grateful to Dr. Sangeeta Kansal (HoD, Department of Applied Mathematics) , Delhi Technological University, New Delhi and all other faculty members of our department, for their astute guidance, constant encouragement and sincere support for this project work.

I owe a debt of gratitude to my guide and mentor, Dr. C.P. Singh (Assistant Professor, Department of Applied Mathematics) for incorporating in me the idea of a creative Minor Project, helping me in undertaking this project and also for being there whenever I needed his assistance.

I also place on record, my sense of gratitude to one and all, who directly or indirectly have lent their helping hand in this venture. I feel proud and privileged in expressing my deep sense of gratitude to all those who have helped me in presenting this project.

Last but never the least, I thank my parents for always being with me, in every sense.

Abstract

Computers have long been an essential part of our lives. The past few years have seen the rise of computers as extremely powerful computational tools. Machine Learning and Artificial Intelligence domains have flourished in recent times. Computers have gained the power to understand and learn from what we give them to as input. Learning from human generated audio, if achieved to a high accuracy, can be a big advantage of this increased computational power.

My motive, through this project, is to recognise the gender of subjects using various machine learning techniques on live recorded audio samples. Accomplishing this task would mean a step forward towards improving human-computer interaction, a vital aspect of a robotic system.

Contents

1 Introduction

1.1 Basics of Machine Learning

Machine Learning is a field of computer science that is responsible for giving computers the ability to learn without being explicitly programmed or given instruction to do so. Evolving from pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study of algorithms that can interpret and learn from already available data. Such algorithms can then be used on large scale to execute data driven decisions. Tom M. Mitchell provided a widely quoted, more formal definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

1.2 Basics of Deep Learning

Deep Learning or Hierarchical Learning is a branch of machine learning based on a set of algorithms that model high level abstractions in data by using a deep graph with multiple processing layers, composed of multiple linear and nonlinear transformations. Deep learning is part of a broader family of machine learning methods based on learning representations of data.

An observation or a feature, in machine learning terms, can be represented in many ways like a vector of color values (pixel) of an image, or a vector of frequencies of a music tune. Some features are better at simplify the learning computation.

Various deep learning architectures such as deep neural networks, convolutional deep neural networks, deep belief networks and recurrent neural networks have been applied to fields like computer vision, automatic speech recognition, natural language processing, audio recognition and bioinformatics where they have been shown to produce state-of-the-art results on various tasks.

Deep Learning algorithms are based on distributed representations of data. The underlying assumption behind distributed assumptions is that observed features originate due to organisation of factors in layers. Varying number of layers and layer sizes provide various levels of abstractions.

Many deep learning algorithms are applied to unsupervised learning tasks. This is an important benefit because unlabeled data are usually more abundant than labeled data. Examples of deep structures that can be trained in an unsupervised manner are neural history compressors and deep belief networks.

1.3 Audio and Signal Processing

Audio signal processing or audio processing is the intentional alteration of audio signals often through an audio effect or effects unit. As audio signals may be electronically represented in either digital or analog format, signal processing may occur in either domain. Analog processors operate directly on the electrical signal, while digital processors operate mathematically on the digital representation of that signal.

2 Dataset Analysis

2.1 About the dataset

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

2.2 Dataset features

The dataset has the following acoustic features:

1. **meanfreq** : *mean frequency (in kHz)*
2. **sd** : *standard deviation of frequency*
3. **median** : *median frequency (in kHz)*
4. **Q25** : *first quartile (in kHz)*
5. **Q75** : *third quartile (in kHz)*
6. **iqr** : *interquartile range(in kHz)*
7. **skew** : *skewness*
8. **kurt** : *kurtosis*

9. **sp.ent** : *spectral entropy*
10. **sfm** : *spectral flatness*
11. **mode** : *mode frequency*
12. **centroid** : *frequency centroid*
13. **peakf** : *peak frequency i.e. frequency with highest energy*
14. **meanfun** : *average of fundamental frequency measured across acoustic signal*
15. **minfun** : *minimum of fundamental frequency measured across acoustic signal*
16. **maxfun** : *maximum of fundamental frequency measured across acoustic signal*
17. **meandom** : *average of dominant frequency measured across acoustic signal*
18. **mindom** : *minimum of dominant frequency measured across acoustic signal*
19. **maxdom** : *maximum of dominant frequency measured across acoustic signal*
20. **dfrage** : *range of dominant frequency measured across acoustic signal*
21. **modindx** : *modulation index. Calculated as the accumulated absolute difference between adjacent measurements of fundamental frequencies divided by the frequency range*
22. **label** : *male or female*

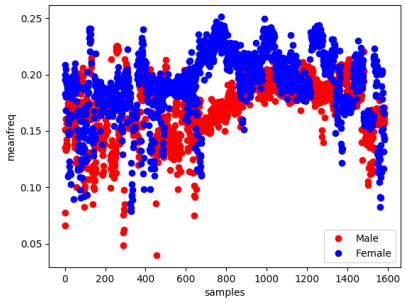
2.3 Feature Variation

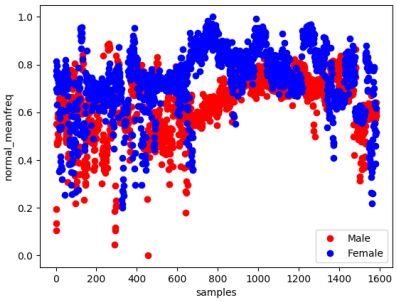
In this section we will observe how the features vary numerically across the category. Scatter plots are shown for the original and normalised data. Normalisation is carried out in two steps:

$$x = log(x + 1)$$

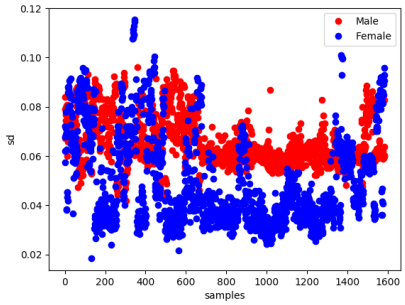
$$x = \frac{x - \bar{x}}{max(x)}$$

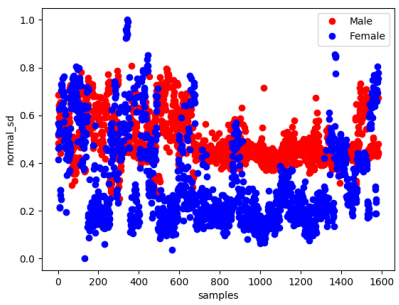
meanfreq



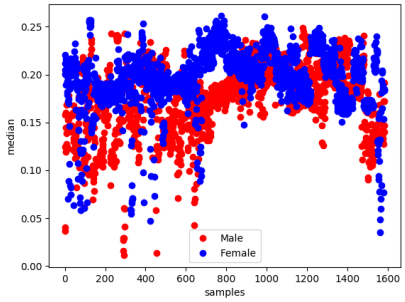


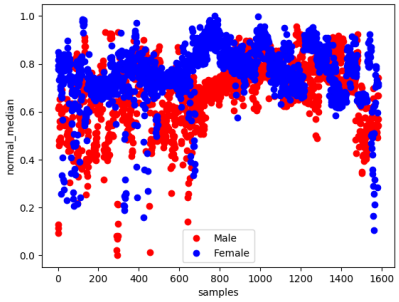
sd



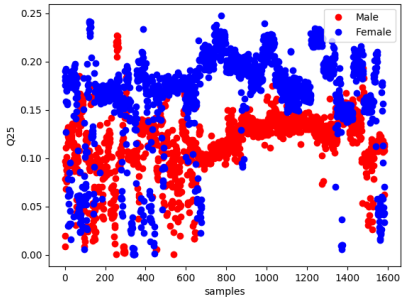


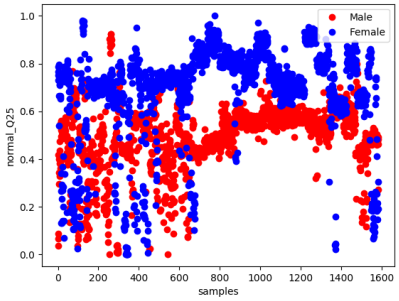
median



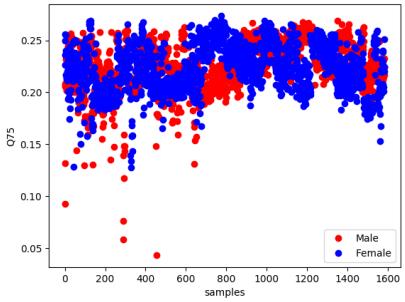


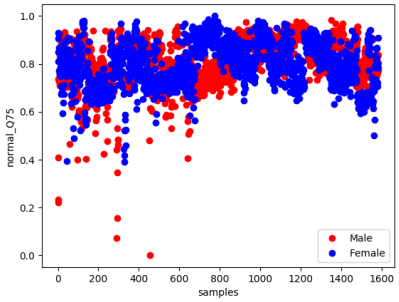
Q25



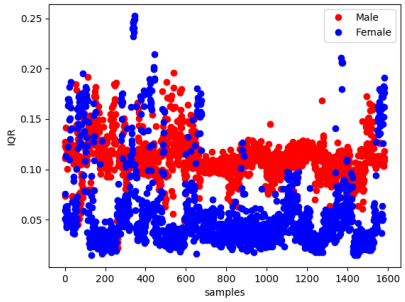


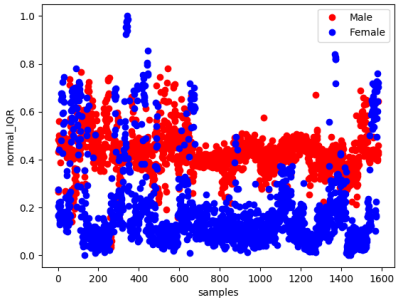
Q75



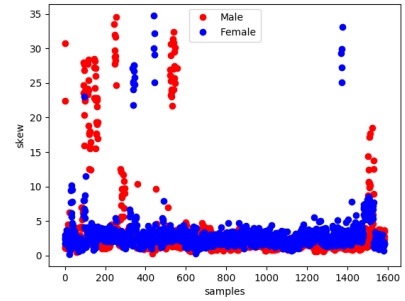


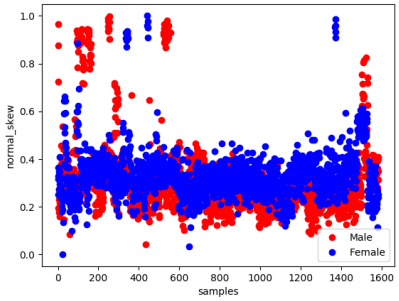
IQR



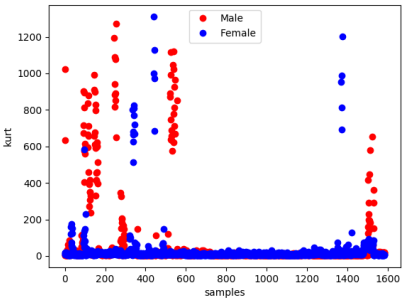


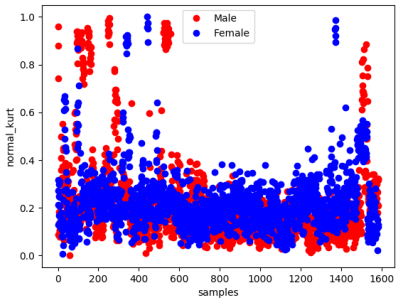
skew



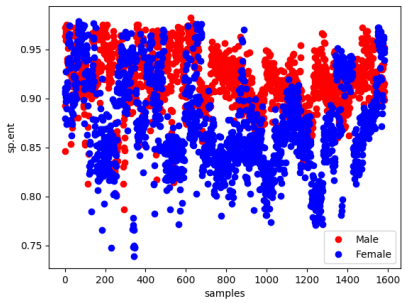


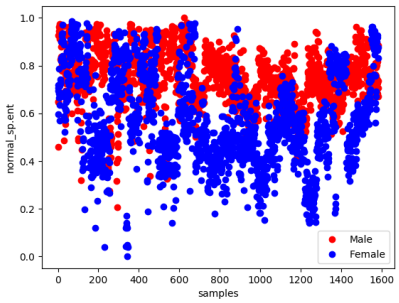
kurt



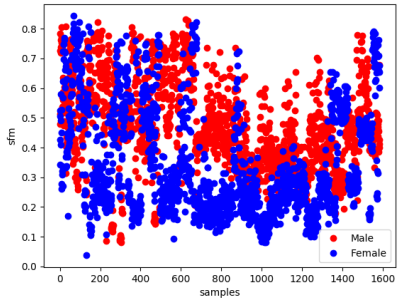


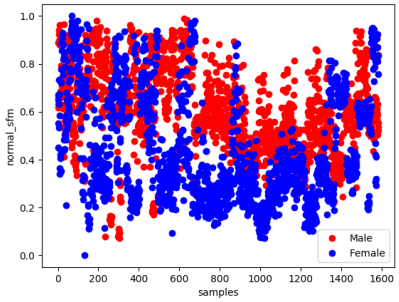
sp.ent



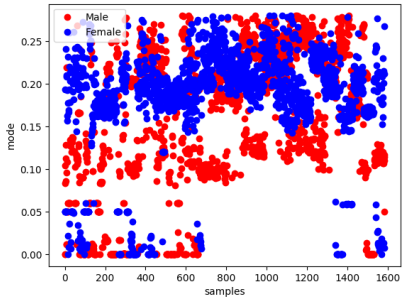


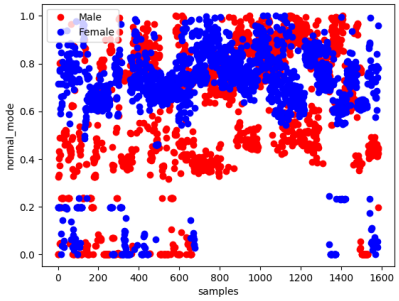
sfm



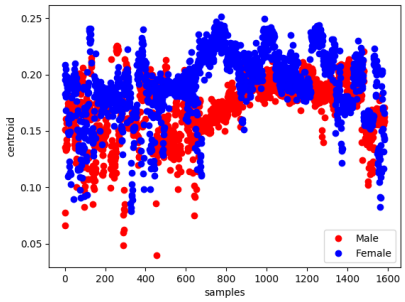


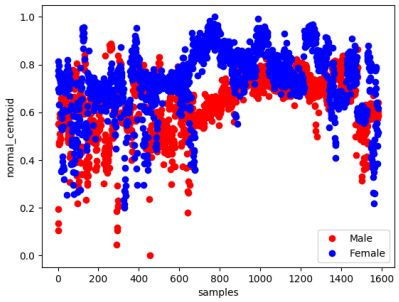
mode



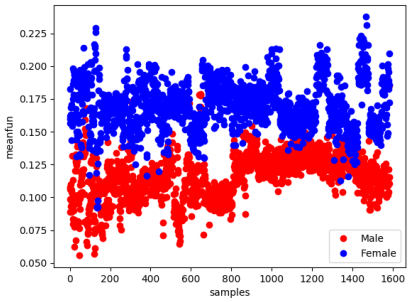


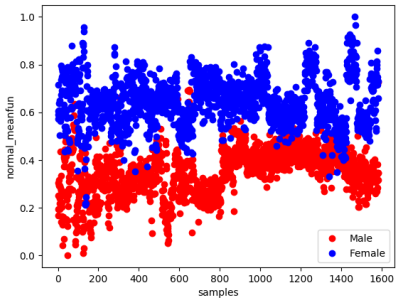
centroid



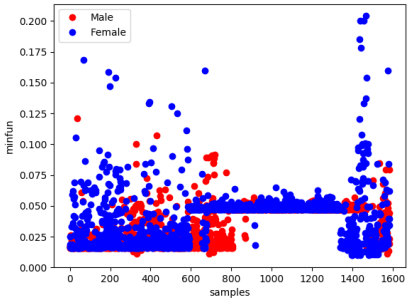


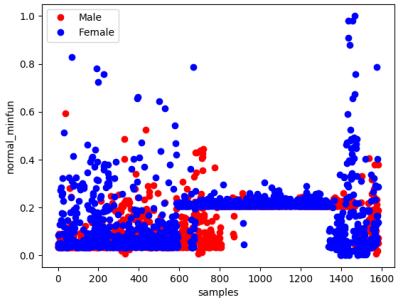
meanfun



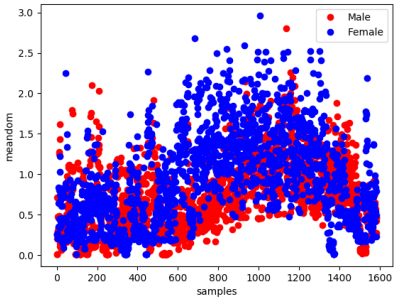


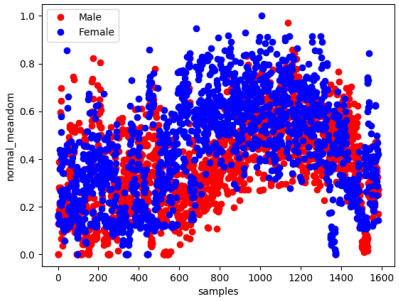
minfun



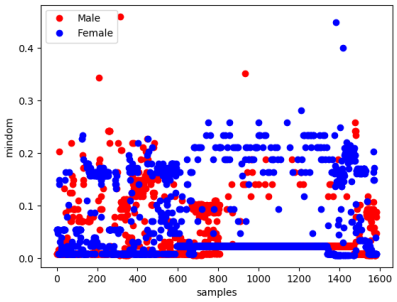


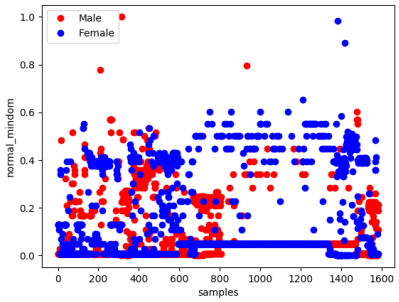
meandom



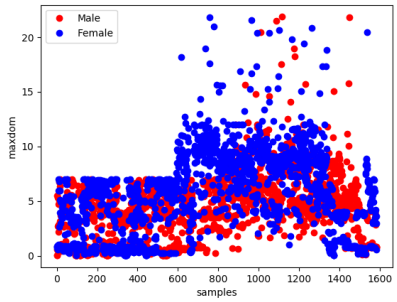


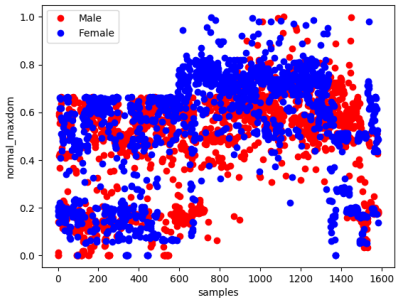
mindom



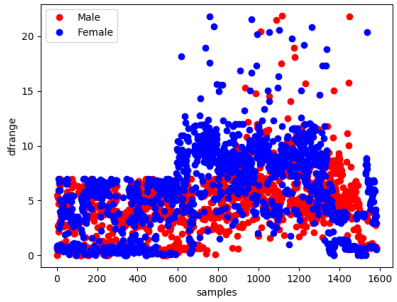


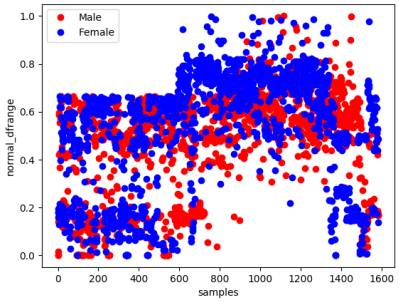
maxdom



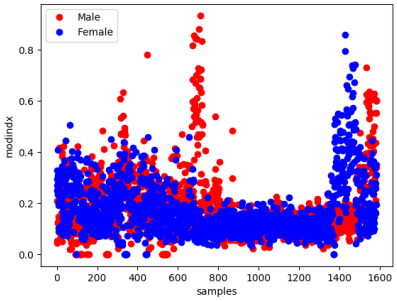


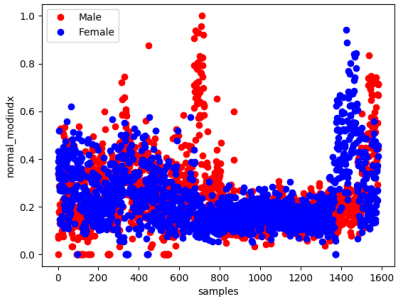
dfrange





modindx





As we can see from the graphs none of the features individually can visually account for a clearly defined segmentation between the two genders. Mean fundamental frequency comes very close to classifying the samples into the two genders.

3 Algorithms and Techniques

3.1 ADA Boost Classifier

Boosting is an approach to machine learning based on the idea of creating a highly accurate prediction rule by combining many relatively weak and inaccurate rules. This is done by building a model from the training data, then creating a second model that attempts to correct the errors from the first model. Models are added until the training set is predicted perfectly or a maximum number of models are added. AdaBoost was the first really successful boosting algorithm developed for binary classification. It is the best starting point for understanding boosting.

AdaBoost in general can be used to boost the performance of any machine learning algorithm. It is best used with weak learners, a classifier which is only slightly correlated with the true classification. These are models that achieve accuracy just above random chance on a classification problem.

The most suited and therefore the most common algorithm used with AdaBoost are decision trees with one level. Because these trees are so short and contain only one decision for classification, they are often called decision stumps. Only a binary(two-class) classification

problems are supported, so each decision stump make sone decision on one input variable and outputs a +1.0 or -1.0 value for the first or second class value. The misclassification rate is then calculated as follows:

$$error = \frac{correct - N}{N}$$

This is then modified to use the weighting of the training instances:

$$error = sum(w(i) * terror(i))/sum(w)$$

which is the weighted sum of the misclassification rate, where w is the weight for training instance i and terror is the prediction error for training instance i which is 1 if misclassified and 0 if correctly classified.
For example if we had 3 training instances with the weights 0.01, 0.5 and 0.2. The predicted values were -1, -1 and -1, and the actual output variables in the instances were -1, 1 and -1, then the terrors would be 0, 1 and 0. The misclassification rate would be calculated as:

$$error = \frac{0.01*0+0.5*1+0.2*0}{0.01+0.5+0.2}$$

A stage value is calculated for the trained model which provides a weighting for any predictions that the model makes. The stage value for a trained model is calculated as follows:

$$stage = ln(\frac{1-error}{error})$$

where stage is the stage value used to weight predictions from the model and error is the misclassification error for the model. The effect of the stage weight is that more accurate models have more weight or contribution to the final prediction.

The training weights are updated giving more weight to incorrectly predicted instances, and less weight to correctly predicted instances.

For example, the weight of one training instance (w) is updated using:

$$w = w.e^{stage.terror}$$

where w is the weight for a specific training instance and stage is the misclassification rate for the weak classifier and terror is the error the weak classifier made predicting the output variable for the training instance, evaluated as:

$$terror = 0 if (y == p), otherwise 1$$

where y is the output variable for the training instance and p is the prediction from the weak learner.

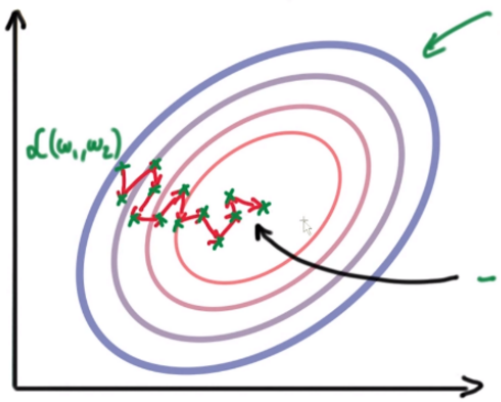
This has the effect of not changing the weight if the training instance was classified correctly and making the weight slightly larger if the weak learner misclassified the instance.

A few things that need to be kept in mind while using an ensemble AdaBoost classifier:

- **Quality Data:** Because the ensemble method continues to attemp to correct misclassifications in the training data, you need to be careful that the training data is of high-quality.
- **Outliers:** Outliers will force the ensemble down the rabbit hole of working hard to correct for cases that are unrealistic. These could be removed from the training dataset.
- **Noisy Data:** Noisy data, specifically noise in the output variable can be problematic. If possible, attempt to isolate and clean these from your training dataset.

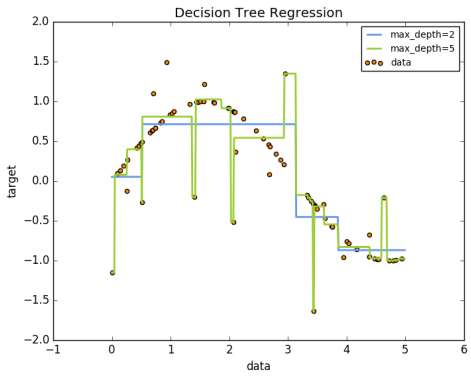
3.2 Stochastic Gradient Descent Classifier

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning. SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing. Given that the data is sparse, the classifiers in this module easily scale to problems with more than 10^5 training examples and more than 10^5 features. The SGD classifier gets its importance from its error minimization algorithm known as Stochastic Gradient Descent. Stochastic gradient descent (often shortened in SGD), also known as incremental gradient descent, is a stochastic approximation of the gradient descent optimization method for minimizing an objective function that is written as a sum of differentiable functions. In other words, SGD tries to find minima or maxima by iteration. Gradient Descent is a method to reduce loss/error by moving along the gradient of the loss function. The primary difference between stochastic gradient descent and gradient descent is the fact that in stochastic gradient descent **only one** sample from the dataset is used to update the weights in each iteration as opposed to multiple samples being used in normal or mini-batch gradient descent. Hence, stochastic gradient descent is faster and converges well before mini-batch gradient descent does.



3.3 Decision Tree

Decision Trees are a non-paramteric supervised learning method used for oclassisifcation and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from data features.



Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree. Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.
- Able to handle multi-output problems. Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.

- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.

- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.

- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.

- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

3.4 Random Forest Classifier

Random forests or random decision forests[1][2] are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$ bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples. For $b = 1, ..., B$:

- Sample, with replacement, B training examples from X, Y; call these Xb, Yb.

- Train a decision or regression tree fb on Xb, Yb.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

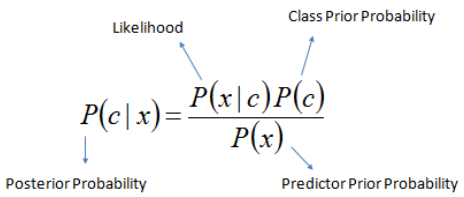
$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

or by taking the majority vote in the case of decision trees. This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

The number of samples/trees, B, is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal number of trees B can be found using cross-validation, or by observing the out-of-bag error: the mean prediction error on each training sample x, using only the trees that did not have x in their bootstrap sample. The training and test error tend to level off after some number of trees have been fit.
The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated. An analysis of how bagging and random subspace projection contribute to accuracy gains under different conditions is given by Ho. Typically, for a classification problem with p features, $\sqrt{p}$ (rounded down) features are used in each split. For regression problems the inventors recommend p/3 (rounded down) with a minimum node size of 5 as the default.

3.5 Naive Bayes Classifier

The Bayesian Classification represents a supervised learning method as well as a statistical method for classification. Assumes an underlying probabilistic model and it allows us to capture uncertainty about the model in a principled way by determining probabilities of the outcomes. It can solve diagnostic and predictive problems. This Classification is named after Thomas Bayes (1702-1761), who proposed the Bayes Theorem. Bayesian classification provides practical learning algorithms and prior knowledge and observed data can be combined. Bayesian Classification provides a useful perspective for understanding and evaluating many learning algorithms. It calculates explicit probabilities for hypothesis and it is robust to noise in input data.
Nave Bayes Algorithm is based on the Bayesian theorem It is particularly suited when the dimensionality of the inputs is high. Parameter estimation for naive Bayes models uses the method of maximum likelihood. In spite over-simplified assumptions, it often performs better in many complex real world situations Advantage: Requires a small amount of training data to estimate the parameters.



$$P(c \mid \mathbf{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

4 Languages and Frameworks

4.1 Python

Python is a high-level, general purpose, interpreted, dynamic programming language which focuses primarily on increased readability of code. Its syntax is closer to spoken english and allows programmers to use comparatively fewer lines of code, as opposed to the more popular languages like C++ or Java. Python is highly platform independent and interpreters are available for all popular operating systems. Common tools like py2exe or pyInstaller allow python programs to be packaged into operating system level executables allowing execution of the program even without a Python interpreter installed.
All the code for modeling, training, predicting and analysis has been written in Python 2.7.

4.2 Sounddevice

A simple Python module that provides bindings for the PortAudio library. This module helps in recording and playing WAV files. I picked this library because it returns a NumPy array of amplitudes after recording the sound.

4.3 NumPy

NumPy is an extension to the Python programming language, adding support for large, multidimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open source and has many contributors.

4.4 SciPy

SciPy is an open source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, signal and image processing, ODE solvers and other tasks common in science and engineering. SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy. There is an expanding set of scientific computing libraries that are being added to the NumPy stack every day. This NumPy stack has similar users to other applications such as MATLAB, GNU Octave, and Scilab. The NumPy stack is also sometimes referred to as the SciPy stack.

4.5 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely

resemble that of MATLAB. SciPy makes use of matplotlib. All the graphs for analysis have been plotted using matplotlib.

4.6 R and RPy2

R is an open source programming language and software environment for statistical computing and graphics that is supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis. Polls, surveys of data miners, and studies of scholarly literature databases show that R's popularity has increased substantially in recent years.
R is a GNU package. The source code for the R software environment is written primarily in C, Fortran, and R. R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems. While R has a command line interface, there are several graphical front-ends available.
R as a language was not explicitly used in this project, however, the dataset I obtained was built after processing audio files using R packages specializing in signal processing. The specific package used is called **seewave**. To process the incoming live audio, I have used the python bridge module to R named **RPy2**. It maps R objects and functions to the python interface for easy use.

4.7 WAV Audio format

Waveform Audio File Format (WAVE, or more commonly known as WAV due to its filename extension) (rarely, Audio for Windows) is a Microsoft and IBM audio file format standard for storing an audio bitstream on PCs. It is an application of the Resource Interchange File Format (RIFF) bitstream format method for storing data in "chunks", and thus is also close to the 8SVX and the AIFF format used on Amiga and Macintosh computers, respectively. It is the main format used on Windows systems for raw and typically uncompressed audio. The usual bitstream encoding is the linear pulse-code modulation (LPCM) format. The format takes into account some differences of the Intel CPU such as little-endian byte order. The RIFF format acts as a "wrapper" for various audio coding formats. Though a WAV file can contain compressed audio, the most common WAV audio format is uncompressed audio in the linear pulse code modulation (LPCM) format. LPCM is also the standard audio coding format for audio CDs, which store two-channel LPCM audio sampled 44,100 times per second with 16 bits per sample. Since LPCM is uncompressed and retains all of the samples of an audio track, professional users or audio experts may use the WAV format with LPCM audio for maximum audio quality. WAV files can also be edited and manipulated with relative ease using software.

5 Code and Implementation

The main python scripts are **record.py** and **prediction.py**.

5.1 record.py

This script handles audio recording for gender prediction. It also calls the prediction function to predict the gender of the recorded audio.

```
import wave
from scipy.io import wavfile
import sounddevice as sd
import numpy as np
from rpy2.robjects.conversion import py2ro, ri2py
from rpy2.robjects.packages import importr
from rpy2.robjects import r
from rpy2.robjects import numpy2ri
from prediction import Prediction
```

In the above code, I import the necessary modules **rpy2** for getting the features from the recorded audio using R packages; and **sounddevice** for recording the audio.

```
numpy2ri.activate()
seewave = importr("seewave")
```

These three lines of code import the required R packages through rpy2.

```
def get_audio():
    duration = 3
    sample_rate = 44100
    print "Recording..."
    recording = sd.rec(int(duration * sample_rate), samplerate=sample_rate,
                       channels=2, blocking=True)
    print "Recorded output.wav for", str(duration), "seconds @", str(sample_rate), "kHz"
    wavfile.write('./test.wav', rate=44100, data=recording)
```

This function records audio from the user's microphone. The duration of the recorded audio sample is fixed at 3 seconds. The sample rate is also fixed at a standard 44100 kHz.

```
def vectorize_audio():
    get_audio()
    audio = tuner.readWave('./test.wav')
    spec_object = seewave.spec(audio, 44100, plot=False)
    f = seewave.specprop(spec_object)
    r_features = common.convert_robj(f)
    fundamental_frequencies = seewave.fund(audio, 44100, plot=False)
    fundamental_frequencies = np.array(
        common.convert_robj(fundamental_frequencies).iloc[:, 0])
    fundamental_frequencies = fundamental_frequencies * 1000
    dominant_frequencies = seewave.dfreq(audio, 44100, plot=False)
    dominant_frequencies = np.array(
        common.convert_robj(dominant_frequencies).iloc[:, 0])
```

```
dominant_frequencies = dominant_frequencies * 1000
features = []
features.append(r_features['mean'][0])
features.append(r_features['sd'][0])
features.append(r_features['median'][0])
features.append(r_features['Q25'][0])
features.append(r_features['Q75'][0])
features.append(r_features['IQR'][0])
features.append(r_features['skewness'][0])
features.append(r_features['kurtosis'][0])
features.append(r_features['sh'][0])
features.append(r_features['sfm'][0])
features.append(r_features['mode'][0])
features.append(r_features['cent'][0])
features.append(np.mean(fundamental_frequencies))
features.append(np.min(fundamental_frequencies))
features.append(np.max(fundamental_frequencies))
features.append(np.mean(dominant_frequencies))
features.append(np.min(dominant_frequencies))
features.append(np.max(dominant_frequencies))
features.append(np.max(dominant_frequencies) -
                np.min(dominant_frequencies))
features.append(0)
```

This function is a crucial part of the script. It calls **get_audio()** to record the audio and save it in **test.wav**. The next lines call the required R package functions to calculate all the required features.

```
prediction = Prediction()
```

The prediction object is created and the classifiers are trained. To get the results from the classifiers, the **predict()** function is called.

5.2 prediction.py

This script defines the prediction class which is responsible for training the classifiers using the Kaggle dataset.

```
def __init__(self):
    cout("Reading data")
    data = read_csv('voice.csv')
    data_x = data.iloc[:, :data.shape[1] - 1]
    data_x = data_x[['meanfun', 'IQR']]
    data_y = data.iloc[:, data.shape[1] - 1:data.shape[1]]
    data_x = data_x.apply(lambda x: np.log(x + 1))
    scaler = MinMaxScaler(feature_range=(0, 1))

    data_x = scaler.fit_transform(data_x)

    train_x, test_x, train_y, test_y = train_test_split(
```

```
    data_x, data_y, test_size=0.33, random_state=42)

for i in range(len(train_y)):
    if train_y.iloc[i].get_value('label') == 'female':
        train_y.iloc[i].set_value('label', int(0))
    else:
        train_y.iloc[i].set_value('label', int(1))
for i in range(len(test_y)):
    if test_y.iloc[i].get_value('label') == 'female':
        test_y.iloc[i].set_value('label', int(0))
    else:
        test_y.iloc[i].set_value('label', int(1))

train_x = np.array(train_x)
train_y = np.array(train_y, dtype='int')
test_x = np.array(test_x)
test_y = np.array(test_y, dtype='int')
train_y = train_y.flatten(order='C')
cout("Training SGD Classifier")
clf = SGDClassifier()
clf.fit(train_x, train_y)
predictions = clf.predict(test_x)
cout("Testing accuracy:" + str(accuracy_score(test_y, predictions)))
cout("Training Decision Tree")
classifier = DecisionTreeClassifier(
    random_state=42, min_samples_split=50)
classifier = classifier.fit(train_x, train_y)
predictions = classifier.predict(test_x)
cout("Testing accuracy:" + str(classifier.score(test_x, test_y)))
cout("Training AdaBoostClassifier")
ada_clf = AdaBoostClassifier()
ada_clf.fit(train_x, train_y)
predictions = ada_clf.predict(test_x)
cout("Testing accuracy:" + str(accuracy_score(test_y, predictions)))
cout("Training Random Forest Classifier")
rf_clf = RandomForestClassifier()
rf_clf.fit(train_x, train_y)
predictions = rf_clf.predict(test_x)
cout("Testing accuracy:" + str(accuracy_score(test_y, predictions)))
cout("Training Gaussian Naive Bayes")
gnb_clf = GaussianNB()
gnb_clf.fit(train_x, train_y)
predictions = gnb_clf.predict(test_x)
cout("Testing accuracy:" + str(accuracy_score(test_y, predictions)))
cout("Training Support Vector Classifier")
sv_clf = SVC()
sv_clf.fit(train_x, train_y)
predictions = sv_clf.predict(test_x)
cout("Testing accuracy:" + str(accuracy_score(test_y, predictions)))

self.sgd_classifier = deepcopy(clf)
del clf
self.dt_classifier = deepcopy(classifier)
del classifier
```

```
        self.ada_clf = deepcopy(ada_clf)
        del ada_clf
        self.rf_clf = deepcopy(rf_clf)
        del rf_clf
        self.gnb_clf = deepcopy(gnb_clf)
        del gnb_clf
        self.sv_clf = deepcopy(sv_clf)
        del sv_clf
```

This is the constructor of the Prediction object. It trains the classifiers and attaches them to the objects for further use. Before the classification, the features are normalised and reduced to the range **(0, 1)**.
After doing that, the 6 classifiers are trained on the data. The classifiers can have various hyper parameters that need to be tuned. Grid Search has been used to find the most optimal classifier by trying many permutations of the various settings.

```
    def predict(self, audio_vector):
        audio_vector = audio_vector / 1000
        needed_audio_vector = np.array(
            [audio_vector[5], audio_vector[12]]).astype('float')
        results = []
        results.append(self.sgd_classifier.predict(needed_audio_vector)[0])
        results.append(self.dt_classifier.predict(needed_audio_vector)[0])
        results.append(self.ada_clf.predict(needed_audio_vector)[0])
        results.append(self.rf_clf.predict(needed_audio_vector)[0])
        results.append(self.gnb_clf.predict(needed_audio_vector)[0])
        results.append(self.sv_clf.predict(needed_audio_vector)[0])
        was_it = raw_input("Were 3 or more predictions correct? (y/n)")
        from collections import Counter
        majority = Counter(results).most_common(n=1)[0][0]
        if majority == 0:
            majority = ',"female"'
            opp = ',"male"'
        else:
            majority = ',"male"'
            opp = ',"female"'
        if was_it == 'y':
            fd = open('voice.csv', 'a')
            fd.write('\n' + ','.join(map(str, audio_vector)) + majority)
            fd.close()
        else:
            fd = open('voice.csv', 'a')
            fd.write('\n' + ','.join(map(str, audio_vector)) + opp)
            fd.close()
```

The pre-trained classifiers are used to predict the gender of the recorded audio. The prediction and audio features are then appended to the csv file to be stored for future training.

6 Observations

Classifier	Testing Accuracy
SGD	0.961759
Decision Tree	0.9550669
AdaBoost	0.9665391
Random Forest	0.9665391
Naive Bayes	0.97036328
Support Vector	0.9684512

7 Applications and Future Scope

There exist a number of applications which can benefit from automatic facial gender and emotion recognition. Voice emotion and gender recognition provide essential context while answering a question. Being able to recognize voice expressions can assist voice recognition algorithms. Furthermore, body language is an important part of human-human communication. Developing methods for a computer to automatically recognize human expressions, could enhance the quality of human-computer interaction and enable the construction of more natural adaptive interfaces and environments. Recognition of expressions in voice is useful for adapting interactive feedback in a tutoring system based on the students level of interest, or for monitoring pilots and drivers alertness state. Automatic recognition could also be used on video recording of group interactions, to trace and document changes in the expressions of the participants, or to retrieve pieces of a video based upon a particular expression of a subject. Yet another application is found in the development of psychological emotion theories by facilitating the experimental data collection of facial expressions associated with particular emotions. The recognition of gender and emotion in voice would further facilitate Personal Assistants such as Siri and Google Now to interact in a better way with their user keeping in mind, their expressions at that state and using the correct pronouns to identify the user.

8 Appendix

record.py

```
import wave
from scipy.io import wavfile
import sounddevice as sd
import numpy as np
from rpy2.robjects.conversion import py2ro, ri2py
from rpy2.robjects.packages import importr
from rpy2.robjects import r
from rpy2.robjects import numpy2ri
from prediction import Prediction

numpy2ri.activate()
seewave = importr("seewave")
tuner = importr("tuneR")


def get_audio():
    duration = 3
    sample_rate = 44100
    print "Recording..."
    recording = sd.rec(int(duration * sample_rate), samplerate=sample_rate,
                       channels=2, blocking=True)
    print "Recorded output.wav for", str(duration), "seconds @", str(sample_rate), "kHz"
    wavfile.write('./test.wav', rate=44100, data=recording)
    return recording


def vectorize_audio():
    get_audio()
    audio = tuner.readWave('./test.wav')
    spec_object = seewave.spec(audio, 44100, plot=False)
    f = seewave.specprop(spec_object)
    r_features = common.convert_robj(f)
    fundamental_frequencies = seewave.fund(audio, 44100, plot=False)
    fundamental_frequencies = np.array(
        common.convert_robj(fundamental_frequencies).iloc[:, 0])
    fundamental_frequencies = fundamental_frequencies * 1000
    dominant_frequencies = seewave.dfreq(audio, 44100, plot=False)
    dominant_frequencies = np.array(
        common.convert_robj(dominant_frequencies).iloc[:, 0])
    dominant_frequencies = dominant_frequencies * 1000
    features = []
    features.append(r_features['mean'][0])
    features.append(r_features['sd'][0])
    features.append(r_features['median'][0])
    features.append(r_features['Q25'][0])
    features.append(r_features['Q75'][0])
    features.append(r_features['IQR'][0])
    features.append(r_features['skewness'][0])
    features.append(r_features['kurtosis'][0])
```

```
    features.append(r_features['sh'][0])
    features.append(r_features['sfm'][0])
    features.append(r_features['mode'][0])
    features.append(r_features['cent'][0])
    features.append(np.mean(fundamental_frequencies))
    features.append(np.min(fundamental_frequencies))
    features.append(np.max(fundamental_frequencies))
    features.append(np.mean(dominant_frequencies))
    features.append(np.min(dominant_frequencies))
    features.append(np.max(dominant_frequencies))
    features.append(np.max(dominant_frequencies) -
                    np.min(dominant_frequencies))
    features.append(0)
    return np.array(features)

prediction = Prediction()
prediction.predict(vectorize_audio())
```

prediction.py

```
import numpy as np
from datetime import datetime
from copy import deepcopy

from pandas import read_csv, to_numeric
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC


def cout(message):
    message = str(datetime.isoformat(datetime.now())) + \
        " @ " + __file__ + ": " + message
    print(message)


class Prediction:

    def __init__(self):
        cout("Reading data")
        data = read_csv('voice.csv')
        data_x = data.iloc[:, :data.shape[1] - 1]
        data_x = data_x[['meanfun', 'IQR']]
        data_y = data.iloc[:, data.shape[1] - 1:data.shape[1]]
        data_x = data_x.apply(lambda x: np.log(x + 1))
        scaler = MinMaxScaler(feature_range=(0, 1))

        data_x = scaler.fit_transform(data_x)

        train_x, test_x, train_y, test_y = train_test_split(
            data_x, data_y, test_size=0.33, random_state=42)

        for i in range(len(train_y)):
            if train_y.iloc[i].get_value('label') == 'female':
                train_y.iloc[i].set_value('label', int(0))
            else:
                train_y.iloc[i].set_value('label', int(1))
        for i in range(len(test_y)):
            if test_y.iloc[i].get_value('label') == 'female':
                test_y.iloc[i].set_value('label', int(0))
            else:
                test_y.iloc[i].set_value('label', int(1))

        train_x = np.array(train_x)
        train_y = np.array(train_y, dtype='int')
        test_x = np.array(test_x)
        test_y = np.array(test_y, dtype='int')
```

```
        train_y = train_y.flatten(order='C')
        cout("Training SGD Classifier")
        clf = SGDClassifier()
        clf.fit(train_x, train_y)
        predictions = clf.predict(test_x)
        cout("Testing accuracy:" + str(accuracy_score(test_y, predictions)))
        cout("Training Decision Tree")
        classifier = DecisionTreeClassifier(
            random_state=42, min_samples_split=50)
        classifier = classifier.fit(train_x, train_y)
        predictions = classifier.predict(test_x)
        cout("Testing accuracy:" + str(classifier.score(test_x, test_y)))
        cout("Training AdaBoostClassifier")
        ada_clf = AdaBoostClassifier()
        ada_clf.fit(train_x, train_y)
        predictions = ada_clf.predict(test_x)
        cout("Testing accuracy:" + str(accuracy_score(test_y, predictions)))
        cout("Training Random Forest Classifier")
        rf_clf = RandomForestClassifier()
        rf_clf.fit(train_x, train_y)
        predictions = rf_clf.predict(test_x)
        cout("Testing accuracy:" + str(accuracy_score(test_y, predictions)))
        cout("Training Gaussian Naive Bayes")
        gnb_clf = GaussianNB()
        gnb_clf.fit(train_x, train_y)
        predictions = gnb_clf.predict(test_x)
        cout("Testing accuracy:" + str(accuracy_score(test_y, predictions)))
        cout("Training Support Vector Classifier")
        sv_clf = SVC()
        sv_clf.fit(train_x, train_y)
        predictions = sv_clf.predict(test_x)
        cout("Testing accuracy:" + str(accuracy_score(test_y, predictions)))

        self.sgd_classifier = deepcopy(clf)
        del clf
        self.dt_classifier = deepcopy(classifier)
        del classifier
        self.ada_clf = deepcopy(ada_clf)
        del ada_clf
        self.rf_clf = deepcopy(rf_clf)
        del rf_clf
        self.gnb_clf = deepcopy(gnb_clf)
        del gnb_clf
        self.sv_clf = deepcopy(sv_clf)
        del sv_clf

    def predict(self, audio_vector):
        audio_vector = audio_vector / 1000
        needed_audio_vector = np.array(
            [audio_vector[5], audio_vector[12]]).astype('float')
        results = []
        results.append(self.sgd_classifier.predict(needed_audio_vector)[0])
        results.append(self.dt_classifier.predict(needed_audio_vector)[0])
        results.append(self.ada_clf.predict(needed_audio_vector)[0])
```

```
        results.append(self.rf_clf.predict(needed_audio_vector)[0])
        results.append(self.gnb_clf.predict(needed_audio_vector)[0])
        results.append(self.sv_clf.predict(needed_audio_vector)[0])
        was_it = raw_input("Were 3 or more predictions correct? (y/n)")
        from collections import Counter
        majority = Counter(results).most_common(n=1)[0][0]
        if majority == 0:
            majority = ',"female"'
            opp = ',"male"'
        else:
            majority = ',"male"'
            opp = ',"female"'
        if was_it == 'y':
            fd = open('voice.csv', 'a')
            fd.write('\n' + ','.join(map(str, audio_vector)) + majority)
            fd.close()
        else:
            fd = open('voice.csv', 'a')
            fd.write('\n' + ','.join(map(str, audio_vector)) + opp)
            fd.close()
```

graphs.py

```
import matplotlib.pyplot as plt
import csv
from pandas import read_csv
from sklearn.preprocessing import MinMaxScaler
import numpy as np

data = read_csv('voice.csv')
data_x = data.iloc[:, :data.shape[1] - 1]
data_y = data.iloc[:, data.shape[1] - 1:data.shape[1]]
data_x = data_x.apply(lambda x: np.log(x + 1))
scaler = MinMaxScaler(feature_range=(0, 1))

data_x = scaler.fit_transform(data_x)
cols = data_x.columns

for col in cols:
    values = data_x[col]
    plt.clf()
    male = []
    female = []
    for x in range(1, 1585):
        male.append(values[x])
    plt.plot(male, 'ro', label='Male')
    for x in range(1585, 3168):
        female.append(values[x])
    plt.plot(female, 'bo', label='Female')
    plt.legend()
    plt.ylabel(col)
    plt.xlabel('samples')
    plt.show()

data_x = data_x.apply(lambda x: np.log(x + 1))
scaler = MinMaxScaler(feature_range=(0, 1))
data_x = scaler.fit_transform(data_x)
for i in xrange(len(cols)):
    values = data_x[:, i]
    col = cols[i]
    plt.clf()
    male = []
    female = []
    for x in range(0, 1585):
        male.append(values[x])
    plt.plot(male, 'ro', label='Male')
    for x in range(1585, 3168):
        female.append(values[x])
    plt.plot(female, 'bo', label='Female')
    plt.legend()
    plt.ylabel('normal_' + col)
    plt.xlabel('samples')
    plt.show()
```

9 References

1. Explaining AdaBoost - http://rob.schapire.net/papers/explaining-adaboost.pdf
2. Boosting and AdaBoost for Machine Learning - http://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/
3. Gender Recognition by Voice - https://www.kaggle.com/primaryobjects/voicegender
4. StackOverflow
5. StackExchange Math
6. Wikipedia
7. SciKit Documentation