PROJECT TITLE : MALE-FEMALE CLASSIFICATION OF AUDIO DATA (TIMIT DATASET)

BACKGROUND:

Audio Database:

TIMIT is a corpus of phonemically and lexically transcribed speech of American English speakers of different sexes and dialects. Each transcribed element has been delineated in time. TIMIT was designed to further acoustic-phonetic knowledge and automatic speech recognition systems. It was commissioned by DARPA and corpus design was a joint effort between the Massachusetts Institute of Technology, SRI International, and Texas Instruments (TI). The speech was recorded at TI, transcribed at MIT, and verified and prepared for publishing by the National Institute of Standards and Technology (NIST).[1] There is also a telephone bandwidth version called NTIMIT (Network TIMIT).

Folder Structure of Data:

```
    convert/ - directory containing ESPRIT SAM software for converting
           TIMIT speech files into a SAM compatible format.

      /readme.doc - this file

     sphere/ - directory containing a new version (1.5) of the NIST SPeech
           HEader REsources (SPHERE) software;  SPHERE is a set of
           "C" library routines and programs for manipulating the NIST
           header structure prepended to the TIMIT waveform files.

      timit/ - directory containing the TIMIT corpus as well as TIMIT-
           related documentation.
```

Each of these directories contains a "readme.doc" file which may be consulted for more detailed information.

The Dataset consists of NIST .wav files. We had to convert it to RIFF .wav files.
The Code for doing so will be provided.

Each file is a 10 second snippet.

OBJECTIVES FOR PRELIMINARY PROCESSING:

1. First Goal is to create a csv file (our train and test data for model building) containing all the features from these audio files.
2. Also, the male female tagging is not provided explicitly. We need to grab them from the folder names.
3. We created a small script to grab all the wav files which were there in their own individual folders and renamed them using their parent directory and gathered them in one folder. Code will be shown.
4. Applying Librosa package, we extract features from this audio database and create our csv files.

DESIGN

DATA LOADING:
We use the librosa.load() method to load the audio files.
One of its parameters is res_type (resampling type).
We used the ‘kaiser_fast’ setting. This was done for the purpose of speed.

FEATURE EXTRACTION:
The LibROSA 0.5.1 version was used for feature extraction.
LibROSA is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems.

We used the **librosa.feature.mfcc** function to find out the **Mel-Frequency Cepstral Coefficients (MFCCs)** of the wave files.

Mel-frequency cepstrum

In sound processing, the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression.

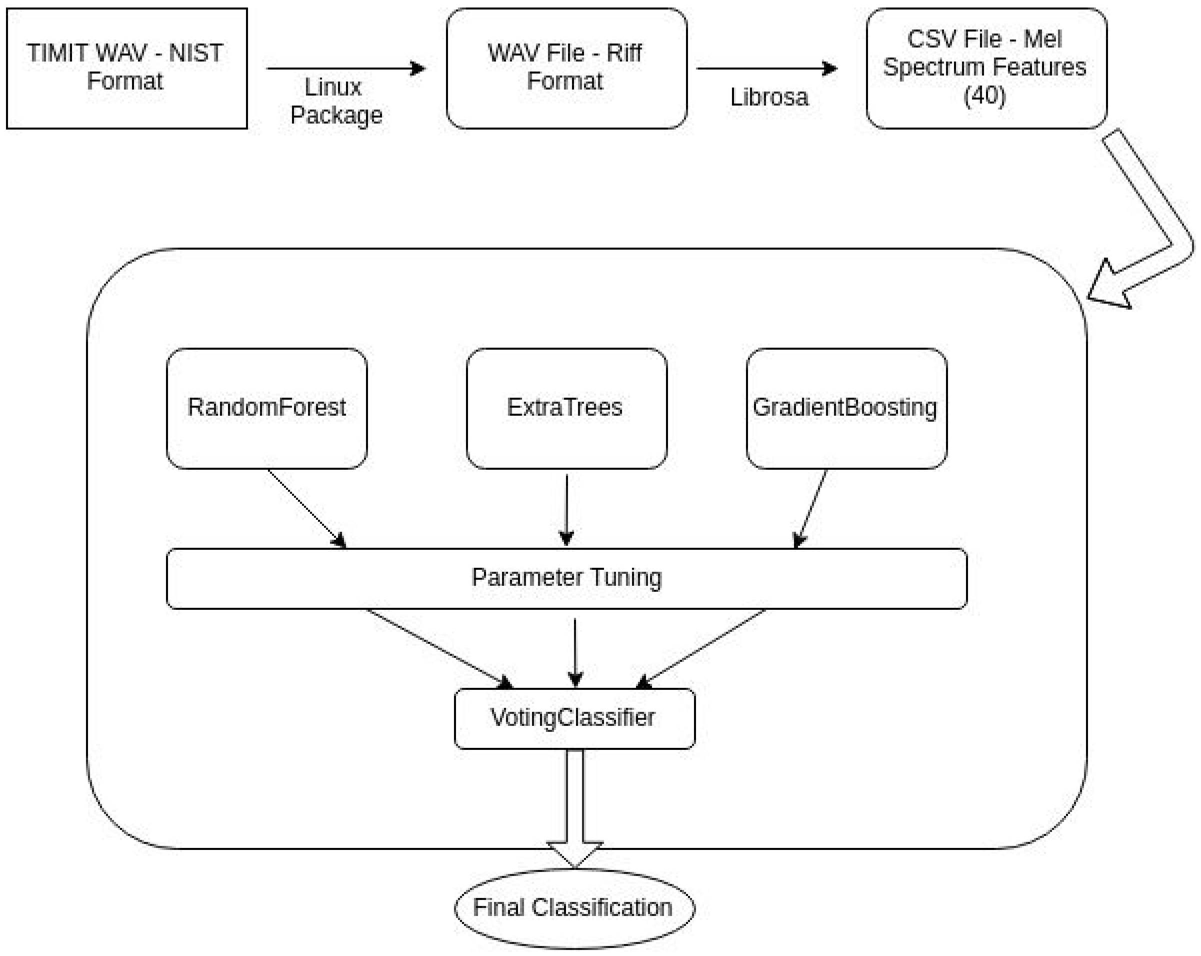
MFCCs are commonly derived as follows:

- Take the Fourier transform of (a windowed excerpt of) a signal.
- Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows.
- Take the logs of the powers at each of the mel frequencies.
- Take the discrete cosine transform of the list of mel log powers, as if it were a signal.
- The MFCCs are the amplitudes of the resulting spectrum.

There can be variations on this process, for example: differences in the shape or spacing of the windows used to map the scale, or addition of dynamics features such as "delta" and "delta-delta" (first- and second-order frame-to-frame difference) coefficients.

The European Telecommunications Standards Institute in the early 2000s defined a standardised MFCC algorithm to be used in mobile phones.

For our audio-files we took 40 MFCC's for each file as the features for our dataset
PIPELINE:



IMPLEMENTATION

ALGORITHMS

Random Forests

In random forests (see RandomForestClassifier and RandomForestRegressor classes), each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best split among a random subset of the features. As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model.

In contrast to the original publication, the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.

Extremely Randomized Trees

In extremely randomized trees (see ExtraTreesClassifier and ExtraTreesRegressor classes), randomness goes one step further in the way splits are computed. As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule. This usually allows to reduce the variance of the model a bit more, at the expense of a slightly greater increase in bias:

Parameters

The main parameters to adjust when using these methods is n_estimators and max_features. The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute. In addition, note that results will stop getting significantly better beyond a critical number of trees. The latter is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, but also the greater the increase in bias. Empirical good default values are max_features=n_features for regression problems, and max_features=sqrt(n_features) for classification tasks (where n_features is the number of features in the data). Good results are often achieved when setting max_depth=None in combination with min_samples_split=2 (i.e., when fully developing the trees). Bear in mind though that these values are usually not optimal, and might result in models that consume a lot of RAM. The best parameter values should always be cross-validated. In addition, note that in random forests, bootstrap samples are used by default (bootstrap=True) while the default strategy for extra-trees is to use the whole dataset (bootstrap=False). When using bootstrap sampling the generalization accuracy can be estimated on the left out or out-of-bag samples. This can be enabled by setting oob_score=True.

Gradient Tree Boosting

Gradient Tree Boosting or Gradient Boosted Regression Trees (GBRT) is a generalization of boosting to arbitrary differentiable loss functions. GBRT is an accurate and effective off-the-shelf procedure that can be used for both regression and classification problems. Gradient Tree Boosting models are used in a variety of areas including Web search ranking and ecology.

The advantages of GBRT are:

- Natural handling of data of mixed type (= heterogeneous features)
- Predictive power
- Robustness to outliers in output space (via robust loss functions)

The disadvantages of GBRT are:

- Scalability, due to the sequential nature of boosting it can hardly be parallelized.
- The module sklearn.ensemble provides methods for both classification and regression via gradient boosted regression trees.

Voting Classifier

The idea behind the VotingClassifier is to combine conceptually different machine learning classifiers and use a majority vote or the average predicted probabilities (soft vote) to predict the class labels. Such a classifier can be useful for a set of equally well performing model in order to balance out their individual weaknesses.

RESULTS & INTERPRETATION

Refer to the attached Jupyter Notebook, containing the entire code with results and visualizations.

Performance on unscaled Data - Without Parameter Tuning

Random Forest - Test Data

	precision	recall	f1-score	support
0	0.99	0.94	0.96	560
1	0.97	0.99	0.98	1120
avg / total	0.98	0.97	0.97	1680

ExtraTrees Classifier - Test Data

	precision	recall	f1-score	support
0	0.98	0.95	0.96	560
1	0.97	0.99	0.98	1120
avg / total	0.97	0.97	0.97	1680

XGBoost Classifier - Test Data

	precision	recall	f1-score	support
0	0.98	0.95	0.97	560
1	0.97	0.99	0.98	1120

avg / total	0.98	0.98	0.98	1680

Performance on Scaled data - Without Parameter Tuning

Random Forest - Test Data

	precision	recall	f1-score	support
0	0.98	0.93	0.96	560
1	0.97	0.99	0.98	1120
avg / total	0.97	0.97	0.97	1680

ExtraTrees Classifier - Test Data

	precision	recall	f1-score	support
0	0.98	0.94	0.96	560
1	0.97	0.99	0.98	1120
avg / total	0.98	0.98	0.98	1680

XGBoost Classifier - Test Data

	precision	recall	f1-score	support
0	0.98	0.95	0.97	560
1	0.97	0.99	0.98	1120
avg / total	0.98	0.98	0.98	1680

Performance of Unscaled Data - Parameter Tuned

Random Forest Model - Test Performance

RandomForestClassifier(bootstrap=True, class_weight='balanced', criterion='entropy', max_depth=5, max_features=15, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=80, n_jobs=1, oob_score=False, random_state=45, verbose=0, warm_start=False)

	precision	**recall**	**f1-score**	**support**
0	0.98	0.94	0.96	560
1	0.97	0.99	0.98	1120
avg / total	0.97	0.97	0.97	1680

XGBoost Classifier - Test Performance

GradientBoostingClassifier(criterion='friedman_mse', init=None, learning_rate=0.2, loss='exponential', max_depth=3, max_features=3, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=150, presort='auto', random_state=45, subsample=1.0, verbose=0, warm_start=False)

	precision	**recall**	**f1-score**	**support**
0	0.98	0.95	0.97	560
1	0.97	0.99	0.98	1120
avg / total	0.98	0.98	0.98	1680

ExtraTrees Classifier - Test Performance

ExtraTreesClassifier(bootstrap=False, class_weight='balanced', criterion='entropy', max_depth=5, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=60, n_jobs=1, oob_score=False, random_state=45, verbose=0, warm_start=False)

	precision	**recall**	**f1-score**	**support**
0	0.97	0.96	0.96	560
1	0.98	0.99	0.98	1120
avg / total	0.98	0.98	0.98	1680

Performance of Scaled Data - Parameter Tuned

Random Forest Model - Test Performance

RandomForestClassifier(bootstrap=True, class_weight='balanced', criterion='entropy', max_depth=5, max_features=15, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=80, n_jobs=1, oob_score=False, random_state=45, verbose=0, warm_start=False)

	precision	**recall**	**f1-score**	**support**
0	0.98	0.94	0.96	560
1	0.97	0.99	0.98	1120
avg / total	0.97	0.97	0.97	1680

XGBoost Classifier - Test Performance

GradientBoostingClassifier(criterion='friedman_mse', init=None, learning_rate=0.2, loss='exponential', max_depth=3, max_features=3, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=150, presort='auto', random_state=45, subsample=1.0, verbose=0, warm_start=False)

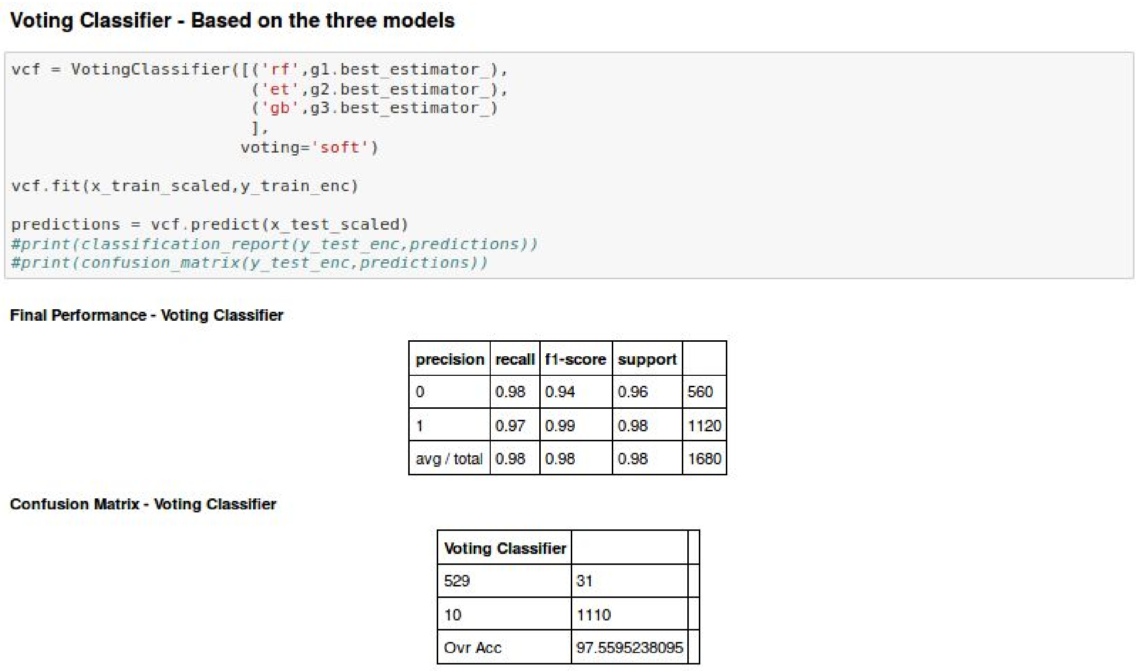
	precision	**recall**	**f1-score**	**support**
0	0.98	0.94	0.96	560
1	0.97	0.99	0.98	1120
avg / total	0.98	0.97	0.97	1680

ExtraTrees Classifier - Test Performance

ExtraTreesClassifier(bootstrap=False, class_weight='balanced', criterion='entropy', max_depth=5, max_features=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=60, n_jobs=1, oob_score=False, random_state=45, verbose=0, warm_start=False)

	precision	**recall**	**f1-score**	**support**
0	0.98	0.98	0.98	1360
1	0.99	0.99	0.99	3260
avg / total	0.99	0.99	0.99	4620

<u>Performance - Snapshot Table</u>



Voting Classifier - Based on the three models

```
vcf = VotingClassifier([('rf',g1.best_estimator_),
                        ('et',g2.best_estimator_),
                        ('gb',g3.best_estimator_)
                        ],
                       voting='soft')

vcf.fit(x_train_scaled,y_train_enc)

predictions = vcf.predict(x_test_scaled)
#print(classification_report(y_test_enc,predictions))
#print(confusion_matrix(y_test_enc,predictions))
```

Final Performance - Voting Classifier

precision	recall	f1-score	support	
0	0.98	0.94	0.96	560
1	0.97	0.99	0.98	1120
avg / total	0.98	0.98	0.98	1680

Confusion Matrix - Voting Classifier

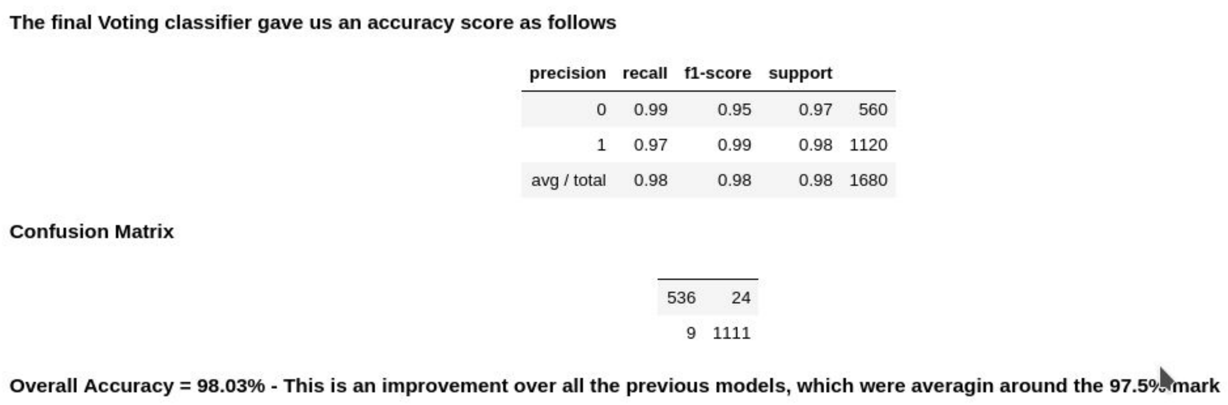
Voting Classifier		
529	31	
10	1110	
Ovr Acc	97.5595238095	

Final Model Improvement Steps

Based on the three individual Grid Search Models, we took the feature importances of each of the columns. Top 20 columns, from each of the three models, and took the total unique number of columns. Using these important columns, we re-ran the best models from each of the previous grid Searches, on the set of 24 predictors on unscaled Data

Final List of Important Columns

['feature_2', 'feature_26', 'feature_33', 'feature_30', 'feature_38', 'feature_31', 'feature_15', 'feature_34', 'feature_27', 'feature_9', 'feature_37', 'feature_20', 'feature_4', 'feature_8', 'feature_12', 'feature_28', 'feature_29', 'feature_25', 'feature_36', 'feature_39', 'feature_32', 'feature_23', 'feature_35', 'feature_0']



The final Voting classifier gave us an accuracy score as follows

precision	recall	f1-score	support	
0	0.99	0.95	0.97	560
1	0.97	0.99	0.98	1120
avg / total	0.98	0.98	0.98	1680

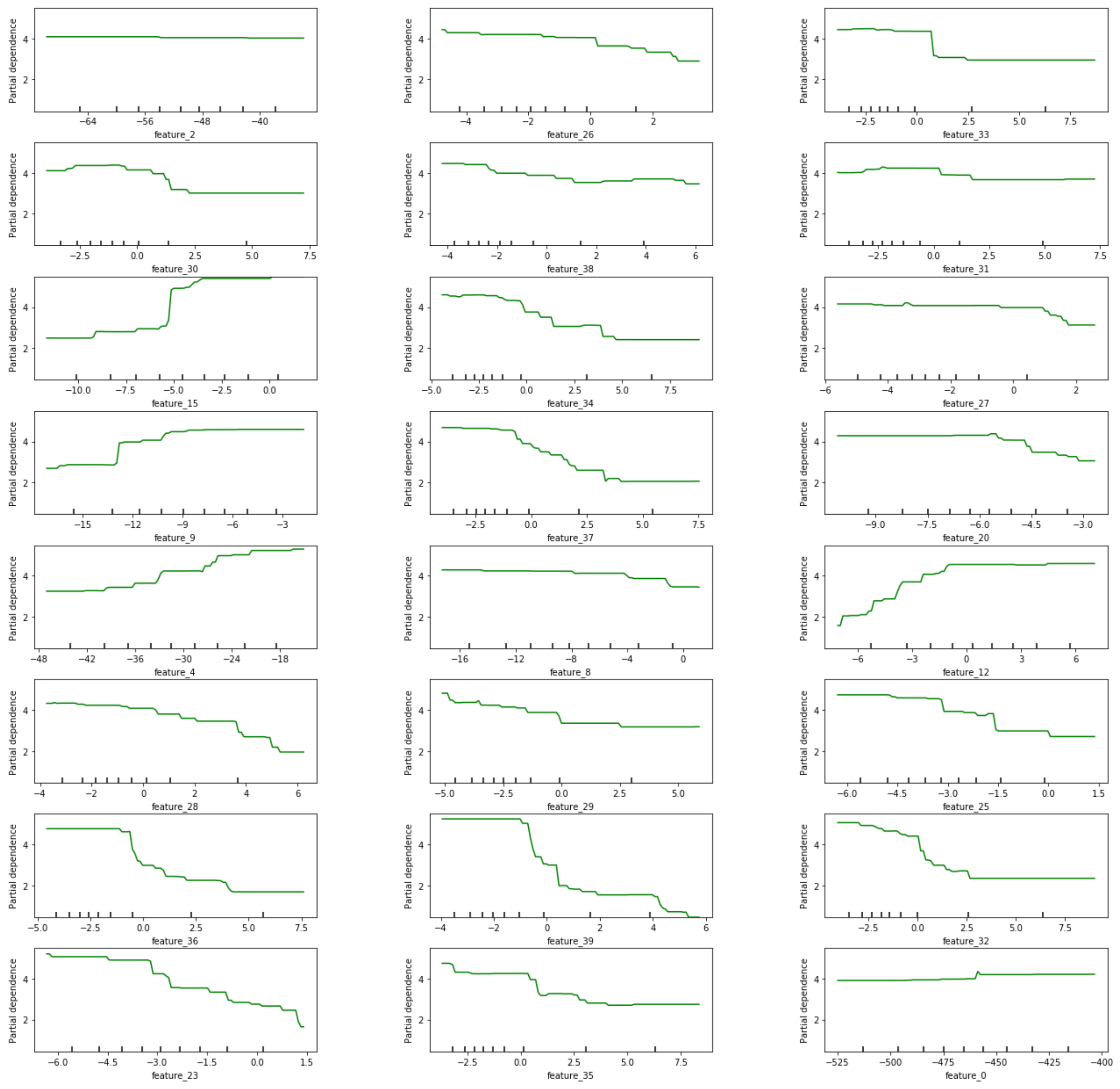
Confusion Matrix

536	24
9	1111

Overall Accuracy = 98.03% - This is an improvement over all the previous models, which were averagin around the 97.5% mark

In the section below, we plotted the partial dependency of all the important features for the target (here male/female) for the GradientBoost Model

Partial dependence plots show the dependence between the target function and a set of ‘target’ features, marginalizing over the values of all other features (the complement features). Due to the limits of human perception the size of the target feature set must be small (usually, one or two) thus the target features are usually chosen among the most important features.


Partial dependence
feature_2
feature_26
feature_33
feature_30
feature_38
feature_31
feature_15
feature_34
feature_27
feature_9
feature_37
feature_20
feature_4
feature_8
feature_12
feature_28
feature_29
feature_25
feature_36
feature_39
feature_32
feature_23
feature_35
feature_0