502 Final Project

Gender Recognition by Voice

Fangzheng Sun

Huimin Ren

Sahil Shahani

Shanhao Li

Yun Yue

Worcester Polytechnic Institute

Abstract

Herein we apply most supervised classification models to the voice gender dataset and identify the voice as male or female. Our work uses most common classification models such as Logistic Regression, K-Nearest-Neighbor (KNN), Decision Tree, Random Forest, Boosting and Support Vector Machines (SVM). Our work involves applying pre-processing techniques to the data, splitting the data into training and testing data, splitting training data in validation and hyper-parameter data, applying principle component analysis (PCA) for dimension reduction, using bootstrap to generate different data sets, and applying cross validation techniques, finding the best tuning parameters for different models. Finally, we apply our best model to the testing data and get accuracy which validates our results.

1. Introduction

In the context of content-based multimedia, voice-based gender identification is an important task. In this paper, we are going to test several classification algorithms mentioned above on the voice dataset provided by *kaggle.com* [1]. 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. 20 acoustic properties of each voice are measured and included within the first 20 columns of the CSV file, whereas the genders of samples are placed at the last column. Figure 1-4 demonstrate data features in some visualized ways.

The purpose of our work is to figure out whether the voice belongs to male or female by using the right number of principle components of the 20 predictors that explain more than 95% of the variance. Our process includes data split, PCA analysis and cross validation for each model, model selection and final test for our optimal algorithm. Our implementation includes Logistic Regression, KNN, Decision Tree, Random Forest, Boosting and SVM. To compare the performances of each model, we outputted optimal parameters (if necessary), testing error and confusion matrix for each algorithm to determine whether these methods could generate accurate, balanced voice-based gender identification models. The final result indicates that KNN when k=1 is the best model for our dataset and the test error is 0.0101.

In the rest of this chapter, we will provide more details about our usage of data split, PCA and cross validation.

The remainder of our report is organized as follows. Data and pre-processing are discussed in Section 2. Section 3 describes the models. Section 4 discusses the results and best model. Finally, Section 5 concludes the report.

2. Data & Pre-processing

2.1 Data Description

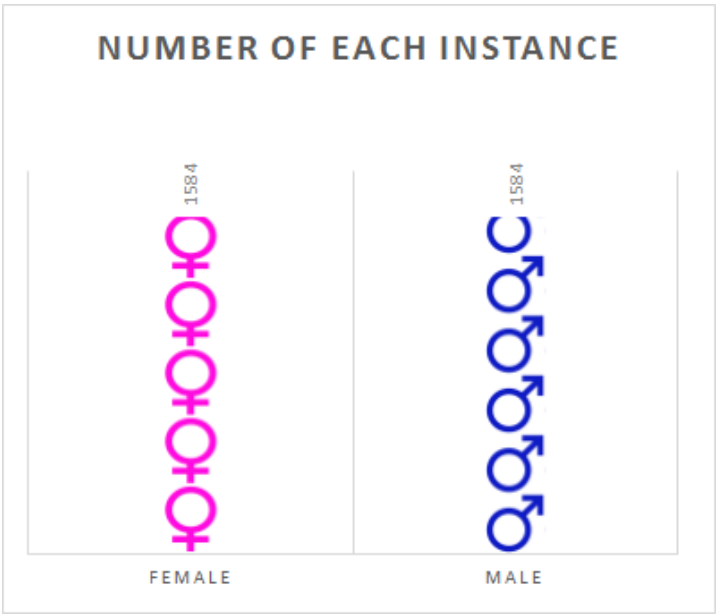

Figure 1: Number of Each Instance

By Fig 1, we can see that the number of female and male samples are the same, indicating that there is no unbalanced problem in our dataset.

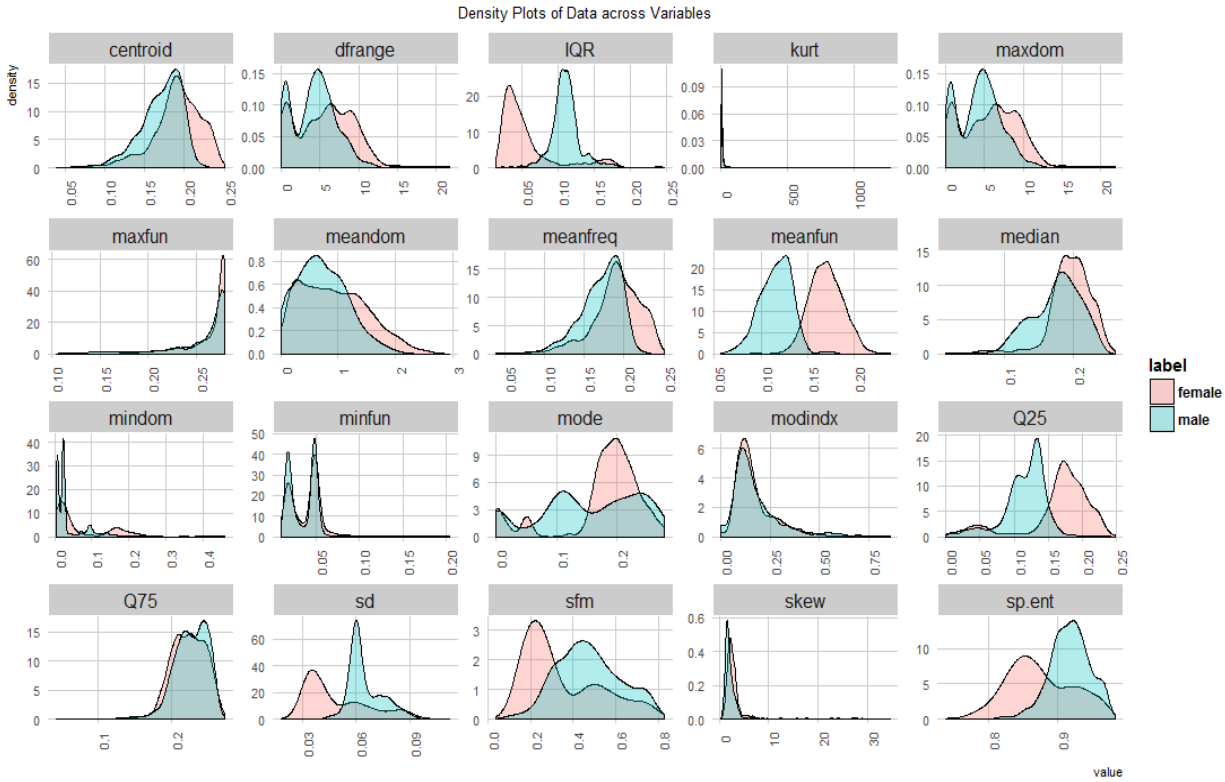

Figure 2: How does each numerical variable vary across the labels

Fig 2. shows that for there is some overlap between the values for the data that is labelled as two gender for almost all features, especially the variable of centroid, meanfreq, minfun, modindx and Q75. In addition, the distribution of variables like IQR, meanfun, mode, Q25, sd are scattered which will be shown as important features in Fig 3.

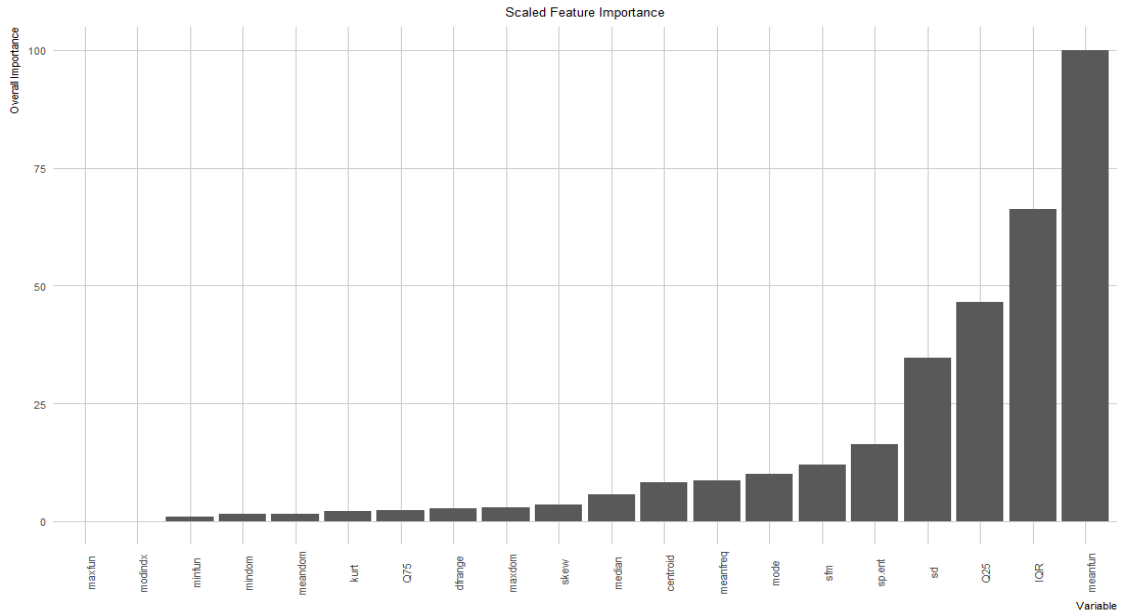

Figure 3: Feature Importance

The feature importance shown in Fig 3. assigns score to features based on usefulness in building the decision trees. we can see from the plot above that meanfun, IQR, Q25 are the three most significant attributes in the dataset.

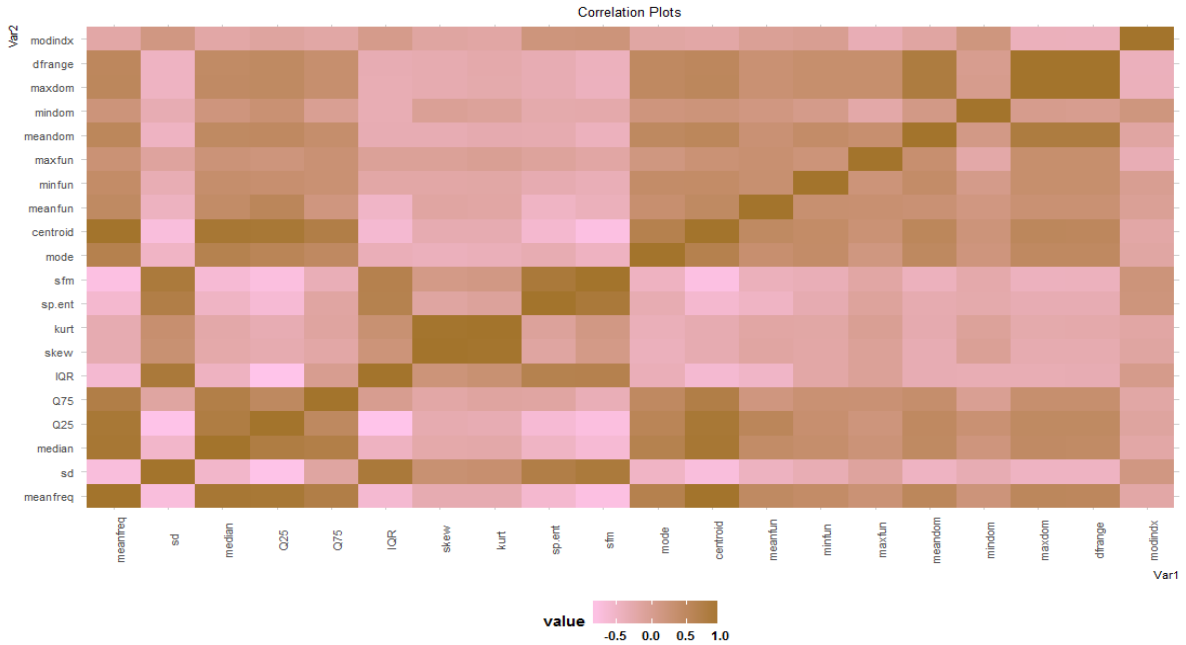

Figure 4: Correlation Plot

Some of the variables have high correlation which is more than 0.7 (Fig 4), such as meanfreq, centroid, Q25, sd, median, sfm, maxdom, dfrange and skew. Due to the high correlation among variables, we used PCA to reduce the dimensions.

2.2 Data Split

We first split whole dataset into two parts: randomly splitting 75% as training data and 25% remaining as testing data, where the training set is used for testing different algorithms and the testing set, fixed (therefore setting seed in the first split) for the final testing of the most optimal method, remained unused until that step. This process was implemented in a separate file to ensure the testing set unchangeable during other implementations.

Bootstrap can be used to quantify the uncertainty associated with a given estimator or statistical learning method [2]. Here we set up a function called *BootStrap* to implement this process to increase randomness and decrease haphazardness. In each method, we would apply bootstrap to the training set at the beginning.

DataSplit is a function we designed to split the training data randomly again, where 75% of the training data goes to validation set and other 25% goes to Hyper-Parameter set. In this process, we did not set seed so that every call of this function outputs different splitting. *DataSplit* were used in each method iteratively to build optimal models and generate average testing errors.

2.3 PCA

We perform PCA using the *prcomp()* function. We set a function *PCA* to do PCA for the given training time and will call this function in each method. *BootStrap* will be called in *PCA* thus there is no need to call it again in the implementation of the classification algorithms. *PCA* will return several analysis results including least number of principle components “t” that explains more than 95% variance and the columns of principle components (treated as predictors in each method).

Figure 5: PCA Visualization

Based on Fig 5, there is an overlap between the 95% confidence regions of the male and female clusters.

2.4 Cross Validation

Firstly, in each method, we applied Bootstrap to the training set as our first step of cross validation. Then we implemented the next step of cross validation for each algorithm by iterating with t, t-1 and t+1 principle components where t is the least number of optimal components that explains more than 95% variance of the training set. By calling train function in *caret* package, we got 3 pairs of prediction classes and true classes standing for t, t-1 and t+1, each of which is a combination of the testing results regarding to 5 different random data splits. By comparing the average error of 5 totally different splits, we finished one more step of cross validation.

Besides, in the function of KNN, Boosting and SVM, we implemented another step of cross validation by choosing the optimal parameters. (k in KNN and number of trees in boosting, and type of kernel in SVM because there is no default value of parameters in package *e1071*)

3. Classification Models

3.1 Logistic Regression

Logistic regression is a classification model to analyze the dataset in which there are one or more predictors that determine an outcome. Different with other classification models, the outcome of logistic regression is measured with a dichotomous variable and the output is the possibility of a data point's label. Compared with other classification algorithms, it is more robust: the independent variables do not have to be normally distributed, or have equal variance in each group.

In our model, we did not use cross validation to seek out the optimal parameter because the parameter for logistic regression is threshold boundary rate. In real world, whatever a male voice is mistakenly predicted as a female voice or a female voice is mistakenly predicted as a male is equally hurt. So, the parameter of threshold rate should be set as 0.5 artificially.

3.2 KNN

KNN is an instance-based learning, where the function is only approximated locally and all computations are deferred until classification. It calculates the biggest number of the label of a data point's K-Nearest Neighbor and assigns the point's label as the biggest number one.

KNN is effective when using a large-size of data and it is robust to outliers. In our KNN model, we did cross validation by using train function in the caret package to get the value of optimal k. Then we set the value as parameter of KNN to iterate three different dimensions of PCA.

3.3 Decision Tree

Here we used classification tree in our model. we predicted that each observation belongs to the *most commonly occurring class* of training observations in the region to which it belongs.

Some people believe that decision tree is more closely mirror human decision-making than do the other regression and classification approaches [2]. Besides this point of view, our decision tree has another advantage: the package *tree* helps us to determine the optimal number of terminal nodes for us. Thus, compared with Boosting or SVM, we saved time for one step of cross validation (select the optimal parameter) in decision tree implementation.

3.4 Random Forest

Random forest is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees. It is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier, and this brings Random Forest great popularity [4].

Similar with *tree* for decision tree, the package *randomForest* gives us the optimal parameters, saving time of one layer of cross validation. What's more, random forest will lead to a substantial reduction in variance over a single tree by forcing each split to consider only a subset of the predictors. Therefore, on average $(p-m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance [2].

3.5 Boosting

Boosting is computationally cheaper than random forest as it uses a new dataset every time which is smaller than the previous one. Secondly, it grows serially, unlike random forest which works on parallel trees.

In our code, we used cross validation, to find these parameters (number of trees, depth, and shrinkage parameter). We call functions from *caret* package (*trainControl, expand.grid)* which are given various ranges for number of trees, depth and minimum shrinkage 0.001. The shrinkage parameter is very small to avoid overfitting, because each tree ultimately depends on original data (y). We reduce this dependency by choosing a very small value.

3.6 SVM

In SVM, we solve an optimization problem i.e. to maximize the minimum distance of support vectors from the separator. We have a budget which allows the data to be misclassified i.e. on wrong side of the margin or hyperplane/separator. We have a constraint that prohibits on the number of misclassifications. This is called normalizing the betas or budget constraint.

In our code, the first thing that clicks mind here now is using SVM with all kernels (linear, polynomial, radial, sigmoid). we use cross validation by calling the *tune* function from *e1071* package to identify the penalty (lambda), for the misclassifications. Tune function performs 10-fold cross validation (by default) to identify the best lambda. This lambda is identified by the ***cost*** parameter of tune function. We use various kernels such as radial, linear, polynomial

and sigmoid to identify how data is separated.

4. Results

4.1 Comparison of All Models

In our R implementation, each algorithm has a separate function which will output the optimal parameters (if necessary), optimal principle component number, testing error regarding to Hyper-Parameter set, and together with confusion matrix.

Comparing the testing error of each model, as shown in the Table 1, we chose KNN and random forest as our best models which has lowest testing error and are quite balanced (no much difference between false positive and false negative in confusion matrix). In the following chapter, we would check KNN and random forest by using the testing set, which had been fixed and unused by then.

Optimal parameters, principle component dimensions, and testing error of each method.	Confusion matrix of each method.

Table 1. Results of All Methods

4.2 Best Models

As we got that KNN and random forest generated the lowest testing error with Hyper-Parameter set, we took a further step to check their performances on testing set.

Below is the result for KNN. The test error is 0.0101, while consistent with testing error in our model implementation, clearly KNN performs well. (balanced and with low testing error)

```
> knn.train.x = pca.voice$x[train_index, 1:knn$best.pca]
> newlabel <- ifelse(voice.train$label == "male", 2,1)
> train.knn = cbind(data.frame(knn.train.x),newlabel)
>
> knn.test.x = pca.voice$x[test_index, 1:knn$best.pca]
> newlabel2 <- ifelse(voice.test$label == "male",2,1)
> test.knn = cbind(data.frame(knn.test.x),newlabel2)
>
> voice.pre.knn <- knn(train.knn, test.knn, train.knn$newlabel, k=knn$k)
> error.knn <- mean(voice.pre.knn != test.knn$newlabel)
> error.knn
[1] 0.01010101
> # Confusion Matrix
> voice.pre.knn = ifelse(voice.pre.knn == 2, "male", "female")
> test.knn$newlabel = ifelse(test.knn$newlabel == 2, "male", "female")
> confusion_matrix.knn <- table(voice.pre.knn,test.knn$newlabel)
> confusion_matrix.knn

voice.pre.knn female male
       female    392    2
       male        6  392
```

The table below shows the result of random forest. Although it generated higher testing error than KNN did, we can still say that random forest performs quite well for our final testing.

```
> # # RandomForest
> # #train dataset
> rf.train.x = pca.voice$x[train_index, 1:rf$best.pca]
> train.rf = cbind(data.frame(rf.train.x),voice.train$label)
> colnames(train.rf)[rf$best.pca+1] <- "label"
>
> # test dataset
> rf.test.x = pca.voice$x[test_index, 1:rf$best.pca]
> test.rf = cbind(data.frame(rf.test.x),voice.test$label)
> colnames(test.rf)[rf$best.pca+1] <- "label"
>
> # model
> fit.rf <- randomForest(label~.,data = train.rf,
+                        mtryin = 4,importance = TRUE)
> pre.rf <- predict(fit.rf, test.rf)
> error.rf <- mean(pre.rf != test.rf$label)
> error.rf
[1] 0.0239899
>
> confusion_matrix.rf <- table(pre.rf,test.rf$label)
> confusion_matrix.rf

pre.rf   female male
  female    385    6
  male       13  388
```

5. Conclusions and Limitations

5.1 Conclusions

Each method had a satisfied error rate which was about or below 0.10, so generally their performances were not bad. Among all models, KNN and random forest generated the most optimal results for our training set, which gave testing error less than 0.01 (0.0057 for KNN and 0.0077 for random forest) and showed balanced predictions in the confusion matrix. As we tested the two models with testing set, KNN and random forest gave testing error 0.0101 and 0.0240.

Generally, KNN gave better predictions than the 5 other algorithms, together with stable testing errors. Based on the comparison, we would like to suggest to use KNN as the classification models in the similar voice-based gender identification issues.

5.2 Limitations

For some algorithms, it is very time-consuming to find all optimal parameters by multiple layers of cross validation. Thus, we applied some “magic values” to the models that we got from previous training to ensure both accuracy and efficiency.

Since the voice dataset is not likely a Gaussian-distributed database, we failed to implement LDA and QDA models. These two methods could be potentially powerful in some other classification issues.

In the real world, there will be some boundary points in the big database. In the voice-based dataset, for instance, some neutral-gender like voice could be hard for both human beings and our models to identify. Further researches could be down on this to help improve the performance of our models.

Reference

[1] https://www.kaggle.com/primaryobjects/voicegender

[2] James Gareth, Daniela Witten, and Trevor Hastie. "An Introduction to Statistical Learning: With Applications in R." (2014).

[3] https://www.medcalc.org/manual/logistic_regression.php

[4] home.etf.rs/~vm/os/dmsw/Random%20Forest.pptx