IIT Bombay

CS419M Project
Introduction to Machine Learning

Gender Recognition using Voice

30th April, 2022

Yash Sanjeev	180070068
Garaga V V S Krishna Vamsi	180070020
Devak Sinha	180070017
Harshit Shrivastava	18D070011
Parin Senta	190070042

Github Repository Link

Contents

1 Problem Statement

The problem comprises of predicting gender based on voice of the target. We are provided with the relevant features pertaining to the voice clips which will serve as the input vector for various classification algorithms.

2 Dataset Procurement, Preprocessing & Augmentation

The dataset was available at various sources. We used Kaggle Voice Dataset as our source, primarily due to the popularity and reliability of Kaggle. The dataset has 3168 training samples, each of which has 21 features with the last one being the gender label we wish to predict. The dataset has 50% male samples and 50% female samples. The 21 features are generated by acoustic analysis in R. The acoustic properties are:

Feature	**Description**
meanfreq	mean frequency (in kHz)
sd	standard deviation of frequency
sd	standard deviation of frequency
Q25	first quantile (in kHz)
Q75	third quantile (in kHz)
IQR	interquantile range (in kHz)
skew	skewness
kurt	kurtosis
sp.ent	spectral entropy
sfm	spectral flatness
mode	mode frequency
centroid	frequency centroid
peakf	peak frequency (frequency with highest energy)
meanfun	average of fundamental frequency measured across acoustic signal
minfun	minimum fundamental frequency measured across acoustic signal
maxfun	maximum fundamental frequency measured across acoustic signal
meandom	average of dominant frequency measured across acoustic signal
mindom	minimum of dominant frequency measured across acoustic signal
maxdom	maximum of dominant frequency measured across acoustic signal
dfrange	range of dominant frequency measured across acoustic signal
modindx	modulation index
label	male or female

We have split the data into training and testing sub-datasets, with 30% of the total samples used for testing.

We normalised the data to zero mean and unit variance for better performance in models like SVM and Neural Networks. We also performed data augmentation on the training data by adding artificial noise to create more training samples and then compared the accuracy of the models with and without augmentation.

```
import numpy as np
noisy_phi_train = phi_train + np.random.normal(
        scale=0.008,
        size = phi_train.shape
    )

# creating the augmented training data
phi_train_new = np.concatenate([phi_train, noisy_phi_train], axis = 0)
y_train_new = np.concatenate([y_train, y_train])
```

3 Classification Models and Performances

We are given a set of input vectors with 20 features, and we have to predict a binary output signifying the gender. Since this is a binary classification problem, we have various models that can be employed.

3.1 Logistic Regression

Since the classification is binary, we start with one of the most popular and simple linear classification algorithm, Logistic Regression (LR). After normalisation of the data, this model gives an impressive accuracy of $\sim$**97%** thereby showing that the data is linearly separable.

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=1000)
model.fit(phi_train, y_train)
print(f'Score without aug = {model.score(phi_test, y_test)}')
```

```
Score without aug = 0.9737118822292324
```

Augmentation does not have a significant change in its accuracy as linear predictors are rarely overfitting and more training data does not make them better.

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=1000)
model.fit(phi_train_new, y_train_new)
print(f'Score with aug = {model.score(phi_test, y_test)}')
```

```
Score with aug = 0.9726603575184016
```

3.2 Support Vector Machine

Moving on to non-linear classifiers, a popular one is the Support Vector Machine (with the RBF kernel). In this case, it gives an accuracy of $\sim$**98%** on the normalised data.

```
from sklearn.svm import SVC
model = SVC()
model.fit(phi_train, y_train)
print(f'Score without aug = {model.score(phi_test, y_test)}')
```

```
Score without aug = 0.9831756046267087
```

Data Augmentation does not affect its accuracy a lot, primarily due to the fact that the number of training samples were more than sufficient to prevent overfitting of this particular model.

```
from sklearn.svm import SVC
model = SVC()
model.fit(phi_train_new, y_train_new)
print(f'Score with aug = {model.score(phi_test, y_test)}')
```

```
Score with aug = 0.9842271293375394
```

3.3 Neural Network

Neural Networks (NN) are one of the most well-known and researched models in Machine Learning. Observing how we obtained great results with linear classifiers, we decided to use a rather shallow neural network with only one hidden layer. We obtained the best results of $\sim$**98.5%** on the normalised data.

```
from sklearn.neural_network import MLPClassifier
np.random.seed(42)
model = MLPClassifier(max_iter=1000)
model.fit(phi_train, y_train)
print(f'Score without aug = {model.score(phi_test, y_test)}')
```

```
Score without aug = 0.9852786540483701
```

We observed absolutely no change in the accuracy of the model after data augmentation, which suggests that the scale of noise added was well within the classification boundary of the neural network and thus had no effect on it. This was further verified by observing that the accuracy of the neural network decreased by increasing the scale of the noise by 10-fold.

```
from sklearn.neural_network import MLPClassifier
np.random.seed(42)
model = MLPClassifier(max_iter=1000)
model.fit(phi_train_new, y_train_new)
print(f'Score with aug = {model.score(phi_test, y_test)}')
```

```
Score with aug = 0.9852786540483701
```

3.4 K-Nearest Neighbour Classifier

KNN is one of the simplest classification algorithm, where we just remember the entire training data and choose the output for the test data using the nearest points in the training data. It achieved $\sim$**98%** accuracy on the normalised data.

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(phi_train, y_train)
print(f'Score without aug = {model.score(phi_test, y_test)}')
```

```
Score without aug = 0.9779179810725552
```

Since the algorithm is very simple and remembers the entire training set, we suspected that adding more noisy training data derived from the original would not increase the accuracy of the algorithm, and confirming our expectation, the algorithm indeed achieved only $\sim$**97.5%** on the augmented data.

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(phi_train_new, y_train_new)
print(f'Score with aug = {model.score(phi_test, y_test)}')
```

```
Score with aug = 0.9747634069400631
```

3.5 Gradient Boosting Classifier

Gradient Boosting CLassifier (GBC) is an ensemble of weak prediction models (hence boosting), which in this case are random forests. It achieved an accuracy of $\sim$**98%** on the data.

```
from sklearn.ensemble import GradientBoostingClassifier
np.random.seed(42)
model = GradientBoostingClassifier()
model.fit(phi_train, y_train)
model.score(phi_test, y_test)
```

```
Score without aug = 0.9789695057833859
```

Augmentation would not have a significant effect unless the model was overfitting, and in our case the random forests were not overfitting. Even then, we observed a minuscule improvement in the performance,

```
from sklearn.ensemble import GradientBoostingClassifier
np.random.seed(42)
model = GradientBoostingClassifier()
model.fit(phi_train_new, y_train_new)
model.score(phi_test, y_test)
```

```
Score with aug = 0.9800210304942166
```

4 Summary of Accuracies Achieved

Model	Accuracy before Augmentation (%)	Accuracy after Augmentation (%)
SVM	98.32	98.42
LR	97.37	97.27
NN	98.53	98.53
KNN	97.79	97.48
GBC	97.90	98.00

5 Analysis and Conclusions

- It is observed that even a simple neural network with just one hidden layer performs the best among all the models in both settings, which shows us its propensity to capture non-linearities in the feature space effectively. Further, its performance remain invariant to the data augmentation.
- The SVM (with an RBF kernel) and the Random Forest classifiers also perform well as expected. They both show only a small fluctuation in performance as a result of data augmentation (positive and negative respectively).
- The simpler models of Logistic Regression and K Neighbors perform the worst (though still almost as good as the others), probably due to their inability to capture non-linear and space-nonuniform relationships. Both of their performances decrease slightly on augmentation.
- We can thus safely conclude that data augmentation is, at the very least, not a very effective means of improvement for the task at hand. This is also somewhat to be expected since we have enough training data as to what is required by the simple model architectures we are working with. Data Augmentation is more suited to, say, vision tasks where the architectures are complex and data is scarce.