Sai Srujana Inclass: Voice using classification Models Machine Learning - MGT 655 Prof. Itauma

Voice Classification Model

Abstract

This Python script loads a dataset of voice features, preprocesses the data, and builds three classification models: Logistic Regression, K-Nearest Neighbors (KNN), and Decision Trees. The models are evaluated using accuracy, precision, recall, and F1-score metrics to compare their performance.

Dataset The dataset voice.csv consists of 3168 samples and 21 features. The target variable is label, which identifies the gender of the speaker (male or female).

Data Preprocessing:

This step helps in finding the type of data and analyze the data and find any missing data or abnormalities in the data. It displays the missing values and we can choose to drop the missing values or update it with mean or median value.

a. Handling Missing Values: We will deal with missing data by either filling or removing null values.

b. Encoding Categorical Variables: Categorical variables like "location" and "status" will be encoded into numerical representations.

c. Normalizing Data: We'll normalize numerical features for regression models if necessary.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns

#For confusion matrixes
from sklearn.metrics import confusion_matrix

# Read our data from dataset.
data = pd.read_csv("//Users//srujana//Downloads//voice.csv")

data.head()
```

```
   meanfreq        sd    median       Q25       Q75       IQR
skew  \
0  0.059781  0.064241  0.032027  0.015071  0.090193  0.075122
12.863462
1  0.066009  0.067310  0.040229  0.019414  0.092666  0.073252
22.423285
2  0.077316  0.083829  0.036718  0.008701  0.131908  0.123207
30.757155
3  0.151228  0.072111  0.158011  0.096582  0.207955  0.111374
1.232831
```

```
4  0.135120  0.079146  0.124656  0.078720  0.206045  0.127325
1.101174

          kurt    sp.ent       sfm  ...  centroid   meanfun    minfun
\
0   274.402906  0.893369  0.491918  ...  0.059781  0.084279  0.015702

1   634.613855  0.892193  0.513724  ...  0.066009  0.107937  0.015826

2  1024.927705  0.846389  0.478905  ...  0.077316  0.098706  0.015656

3     4.177296  0.963322  0.727232  ...  0.151228  0.088965  0.017798

4     4.333713  0.971955  0.783568  ...  0.135120  0.106398  0.016931


     maxfun   meandom    mindom    maxdom   dfrange   modindx  label
0  0.275862  0.007812  0.007812  0.007812  0.000000  0.000000   male
1  0.250000  0.009014  0.007812  0.054688  0.046875  0.052632   male
2  0.271186  0.007990  0.007812  0.015625  0.007812  0.046512   male
3  0.250000  0.201497  0.007812  0.562500  0.554688  0.247119   male
4  0.266667  0.712812  0.007812  5.484375  5.476562  0.208274   male

[5 rows x 21 columns]
```

```
data.describe()
```

```
          meanfreq           sd       median          Q25          Q75
\
count  3168.000000  3168.000000  3168.000000  3168.000000  3168.000000

mean      0.180907     0.057126     0.185621     0.140456     0.224765

std       0.029918     0.016652     0.036360     0.048680     0.023639

min       0.039363     0.018363     0.010975     0.000229     0.042946

25%       0.163662     0.041954     0.169593     0.111087     0.208747

50%       0.184838     0.059155     0.190032     0.140286     0.225684

75%       0.199146     0.067020     0.210618     0.175939     0.243660

max       0.251124     0.115273     0.261224     0.247347     0.273469


               IQR         skew         kurt       sp.ent          sfm
\
count  3168.000000  3168.000000  3168.000000  3168.000000  3168.000000

mean      0.084309     3.140168    36.568461     0.895127     0.408216
```

```
std       0.042783     4.240529   134.928661     0.044980     0.177521
min       0.014558     0.141735     2.068455     0.738651     0.036876
25%       0.042560     1.649569     5.669547     0.861811     0.258041
50%       0.094280     2.197101     8.318463     0.901767     0.396335
75%       0.114175     2.931694    13.648905     0.928713     0.533676
max       0.252225    34.725453  1309.612887     0.981997     0.842936

                mode     centroid      meanfun       minfun       maxfun  \
count  3168.000000  3168.000000  3168.000000  3168.000000  3168.000000
mean      0.165282     0.180907     0.142807     0.036802     0.258842
std       0.077203     0.029918     0.032304     0.019220     0.030077
min       0.000000     0.039363     0.055565     0.009775     0.103093
25%       0.118016     0.163662     0.116998     0.018223     0.253968
50%       0.186599     0.184838     0.140519     0.046110     0.271186
75%       0.221104     0.199146     0.169581     0.047904     0.277457
max       0.280000     0.251124     0.237636     0.204082     0.279114

             meandom       mindom       maxdom      dfrange      modindx
count  3168.000000  3168.000000  3168.000000  3168.000000  3168.000000
mean      0.829211     0.052647     5.047277     4.994630     0.173752
std       0.525205     0.063299     3.521157     3.520039     0.119454
min       0.007812     0.004883     0.007812     0.000000     0.000000
25%       0.419828     0.007812     2.070312     2.044922     0.099766
50%       0.765795     0.023438     4.992188     4.945312     0.139357
75%       1.177166     0.070312     7.007812     6.992188     0.209183
max       2.957682     0.458984    21.867188    21.843750     0.932374
```

```
data.info()
```

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

The voice.csv dataset contains 3168 rows and 21 columns, with 20 numerical features and one target column (label), which appears to be a binary classification task (likely distinguishing between male and female voices). Here's a summary of the columns:

Features: 20 numerical variables such as meanfreq, sd, median, IQR, skew, kurt, etc. Target: label (categorical, with values like "male" or "female"). We can now proceed with building the classification models (Logistic Regression, KNN, and Decision Trees), and evaluate them using accuracy, precision, recall, and F1-score.

Let's start by splitting the data and applying the models.

Here are the evaluation metrics for the three classification models on the test dataset:

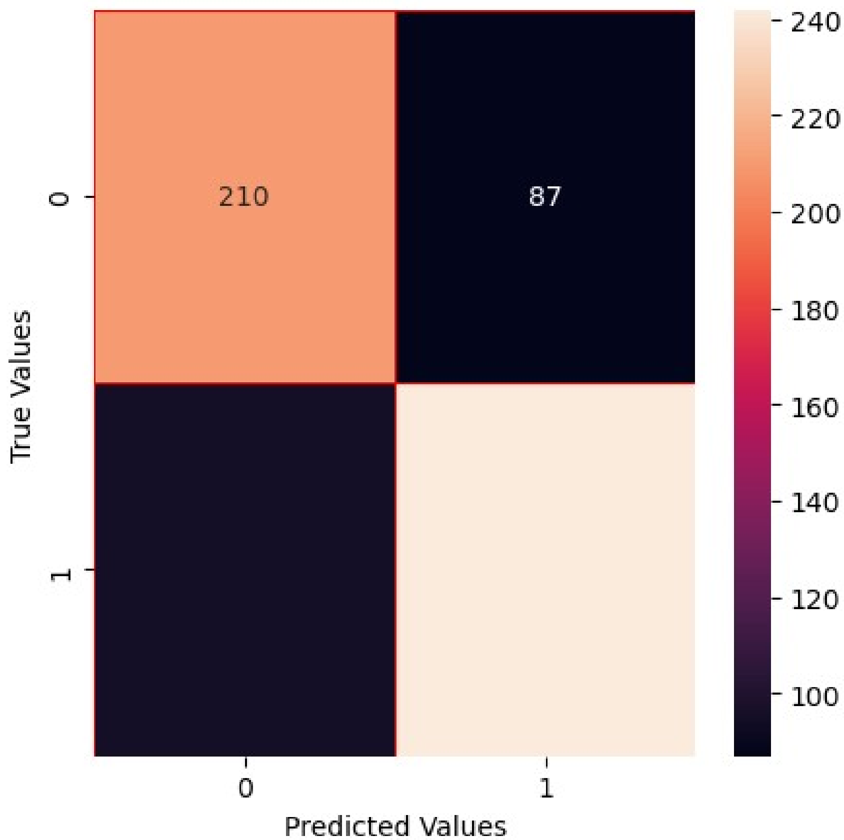
```
y = data.label.values
x_data = data.drop(["label"],axis=1)

x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data))
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.2,random_state = 42)
#test_size=0.2 means %20 test datas, %80 train datas
method_names = []
method_scores = []
#These are for barplot in conclusion
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.2,random_state = 42)
#test_size=0.2 means %20 test datas, %80 train datas
method_names = []
method_scores = []
#These are for barplot in conclusion
```

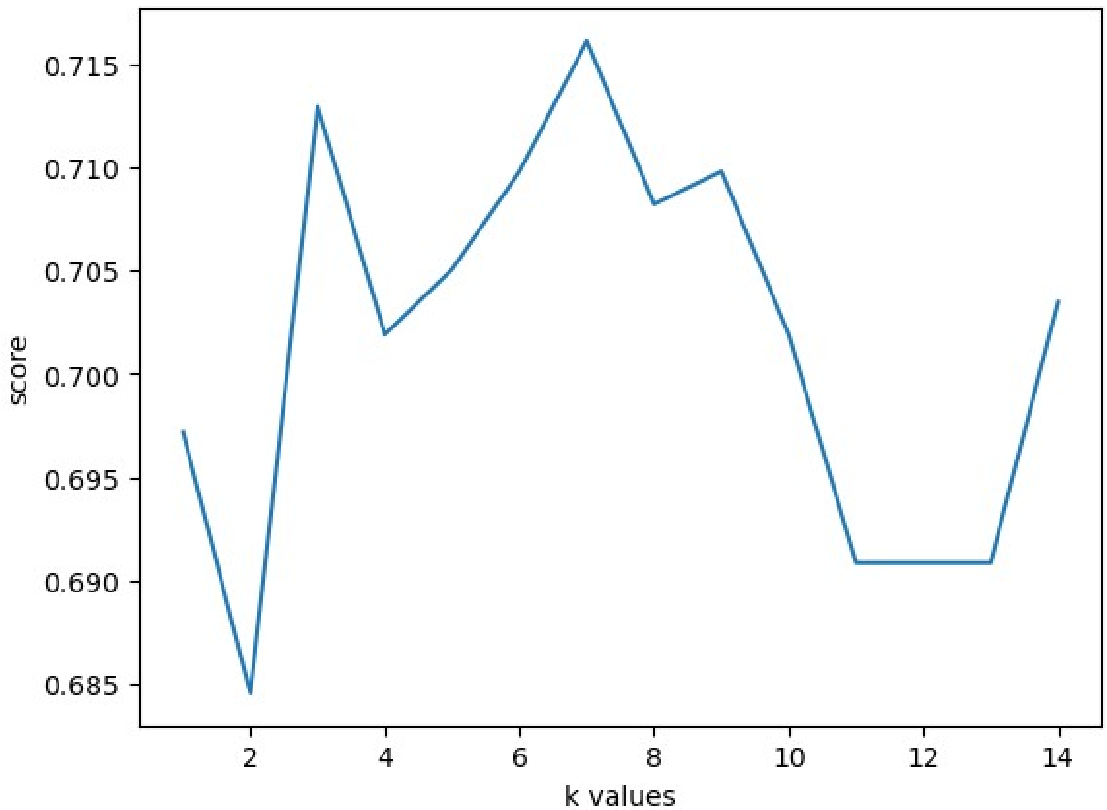
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train,y_train)
print("Score for Number of Neighbors = 3:
{}".format(knn.score(x_test,y_test)))
method_names.append("KNN")
method_scores.append(knn.score(x_test,y_test))

#Confusion Matrix
y_pred = knn.predict(x_test)
conf_mat = confusion_matrix(y_test,y_pred)
#Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(conf_mat,annot=True,linewidths=0.5,linecolor="red",fmt=".0
f",ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()
```

```
Score for Number of Neighbors = 3: 0.7129337539432177
```

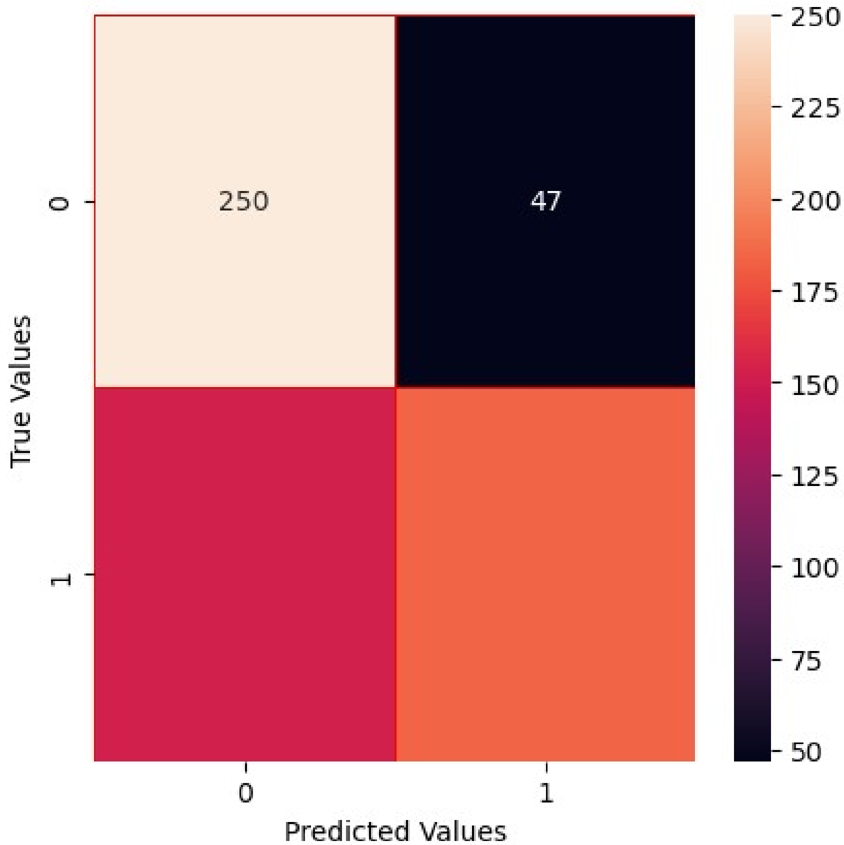



```
score_list=[]
for each in range(1,15):
    knn2 = KNeighborsClassifier(n_neighbors=each)
    knn2.fit(x_train,y_train)
    score_list.append(knn2.score(x_test,y_test))

plt.plot(range(1,15),score_list)
plt.xlabel("k values")
plt.ylabel("score")
```

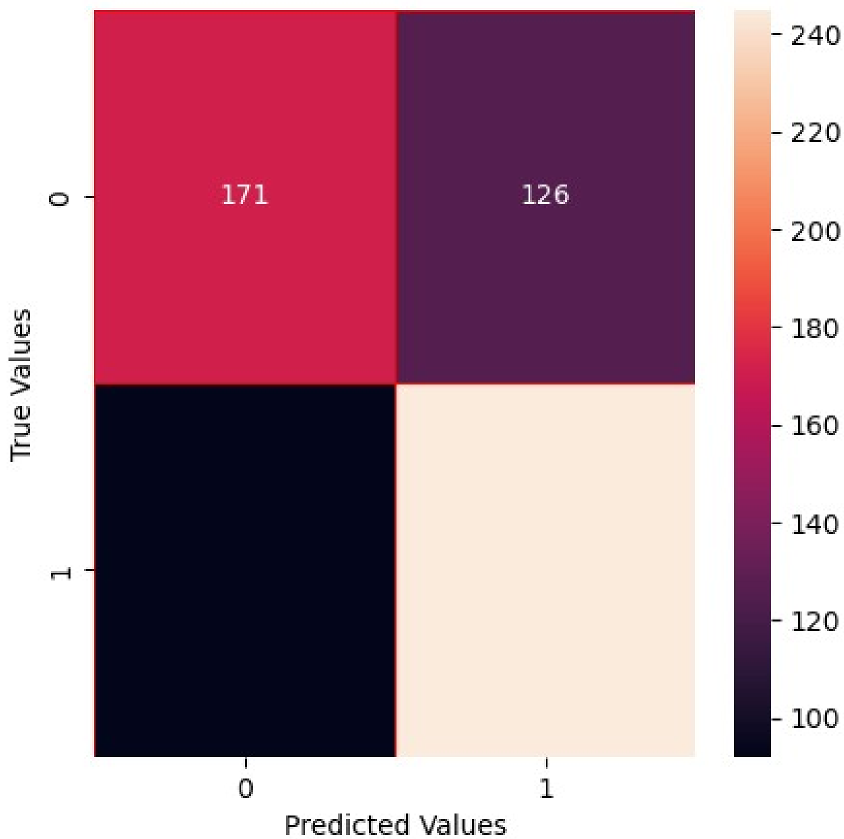
```
Text(0, 0.5, 'score')
```



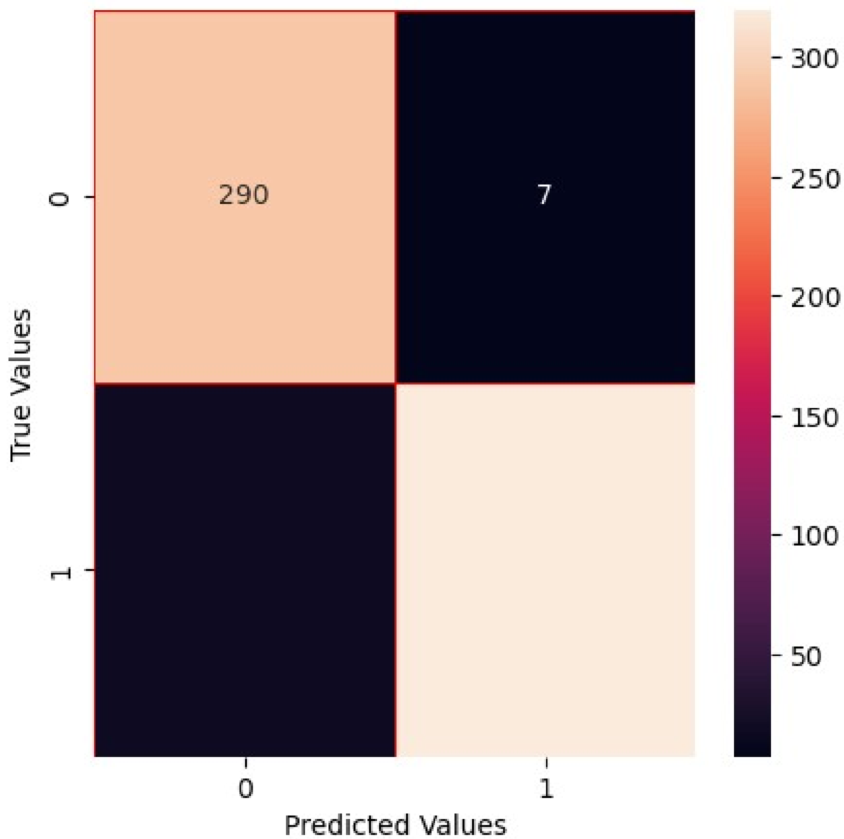

```
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(x_train,y_train)
print("Score for Number of Neighbors = 2:
{}".format(knn.score(x_test,y_test)))

#Confusion Matrix
y_pred = knn.predict(x_test)
conf_mat = confusion_matrix(y_test,y_pred)
#Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(conf_mat,annot=True,linewidths=0.5,linecolor="red",fmt=".0
f",ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

Score for Number of Neighbors = 2: 0.6845425867507886
```



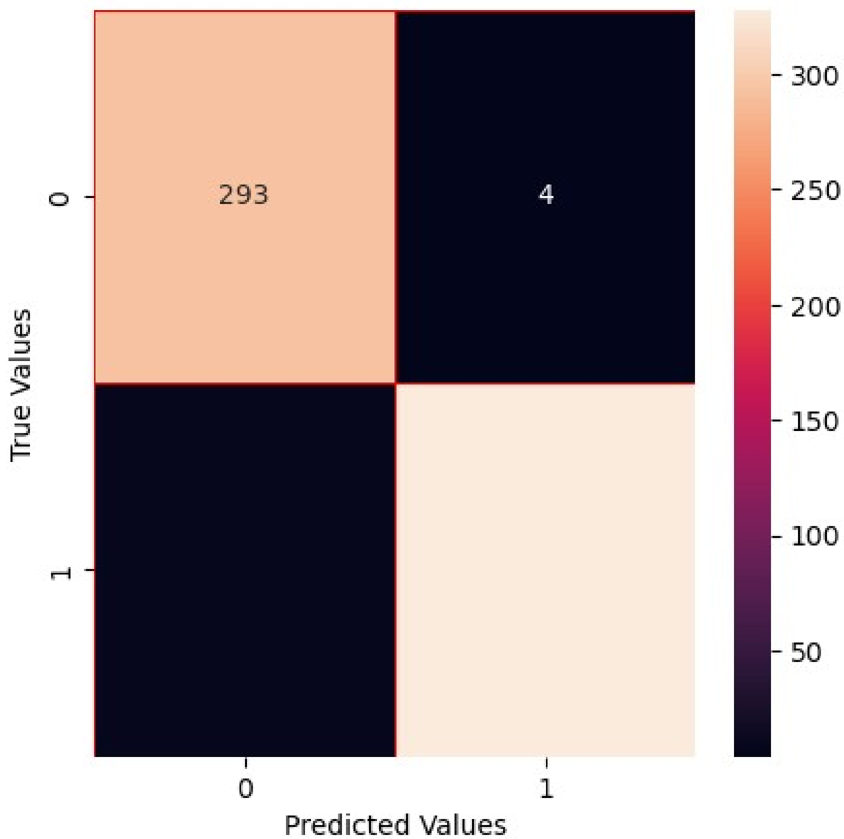

```
# Support vector
from sklearn.svm import SVC
svm = SVC(random_state=42)
svm.fit(x_train,y_train)
print("SVM Classification Score is:
{}".format(svm.score(x_test,y_test)))
method_names.append("SVM")
method_scores.append(svm.score(x_test,y_test))

#Confusion Matrix
y_pred = svm.predict(x_test)
conf_mat = confusion_matrix(y_test,y_pred)
#Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(conf_mat,annot=True,linewidths=0.5,linecolor="red",fmt=".0
f",ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

SVM Classification Score is: 0.6561514195583596
```



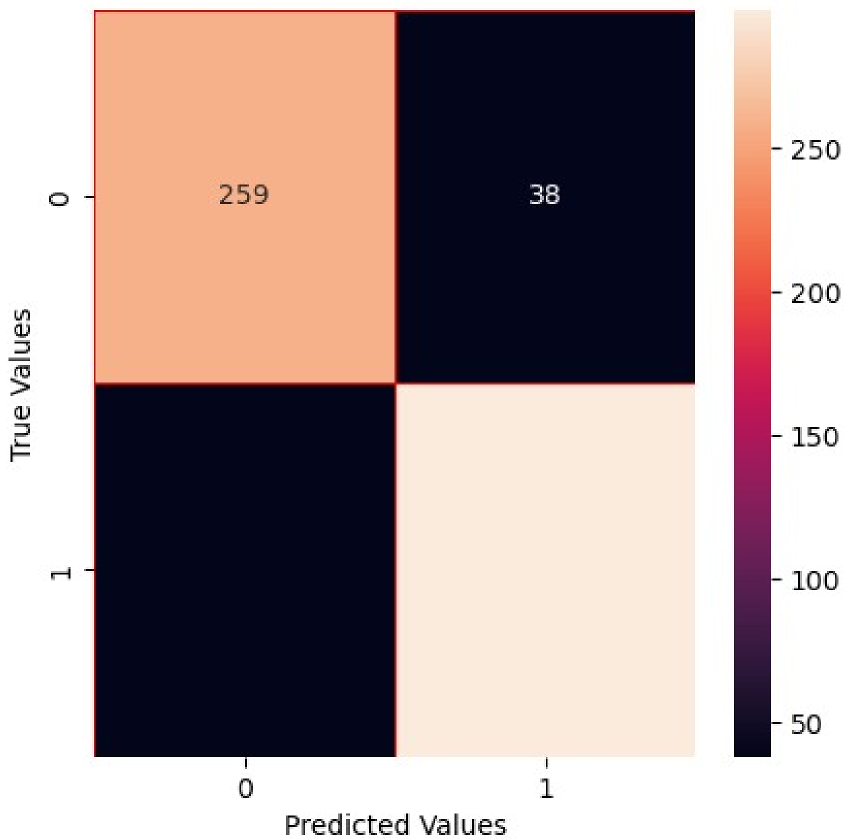

```
from sklearn.tree import DecisionTreeClassifier
dec_tree = DecisionTreeClassifier()
dec_tree.fit(x_train,y_train)
print("Decision Tree Classification Score:
",dec_tree.score(x_test,y_test))
method_names.append("Decision Tree")
method_scores.append(dec_tree.score(x_test,y_test))

#Confusion Matrix
y_pred = dec_tree.predict(x_test)
conf_mat = confusion_matrix(y_test,y_pred)
#Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(conf_mat,annot=True,linewidths=0.5,linecolor="red",fmt=".0
f",ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

Decision Tree Classification Score:  0.9621451104100947
```




```
from sklearn.ensemble import RandomForestClassifier
rand_forest = RandomForestClassifier(n_estimators=100,
random_state=42)
rand_forest.fit(x_train,y_train)
print("Random Forest Classification Score:
",rand_forest.score(x_test,y_test))
method_names.append("Random Forest")
method_scores.append(rand_forest.score(x_test,y_test))

#Confusion Matrix
y_pred = rand_forest.predict(x_test)
conf_mat = confusion_matrix(y_test,y_pred)
#Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(conf_mat,annot=True,linewidths=0.5,linecolor="red",fmt=".0
f",ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

Random Forest Classification Score:  0.9794952681388013
```

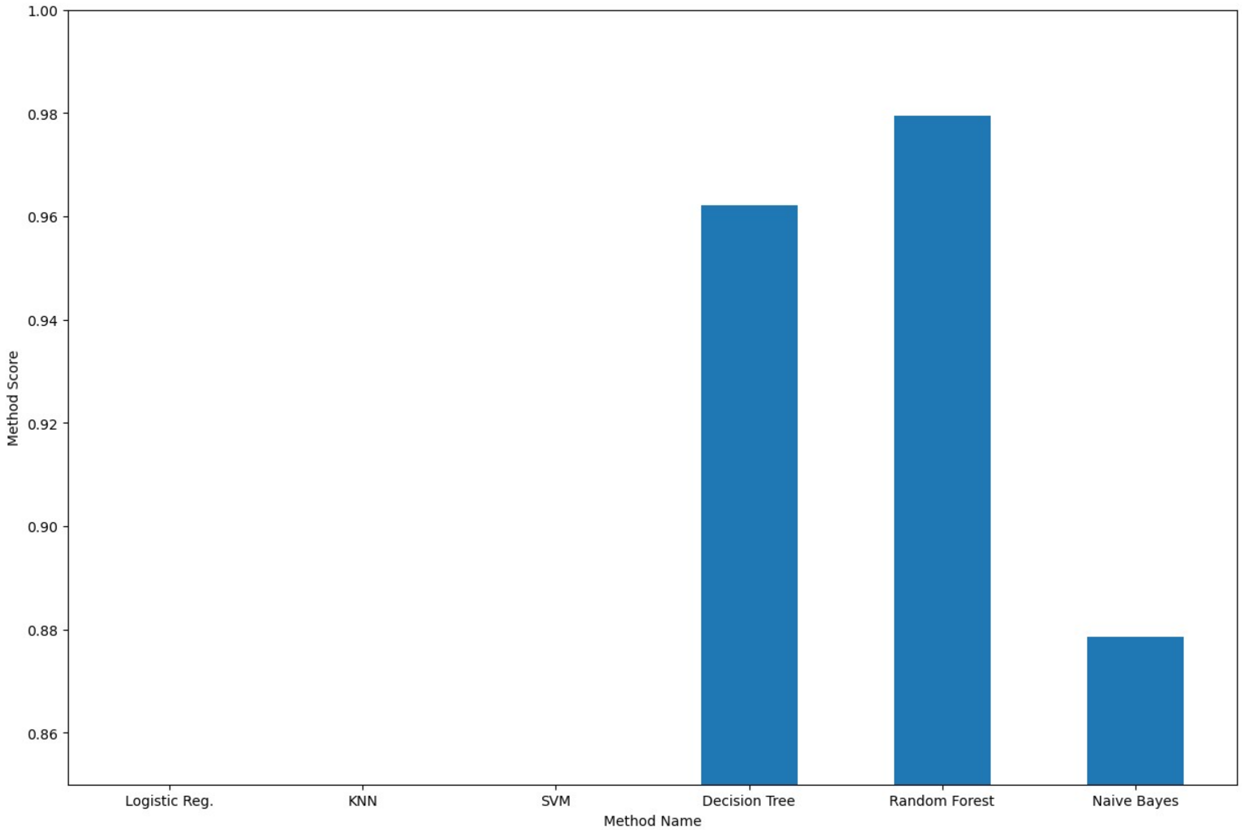



```
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(x_test,y_test)
print("Naive Bayes Classification Score:
{}".format(naive_bayes.score(x_test,y_test)))
method_names.append("Naive Bayes")
method_scores.append(naive_bayes.score(x_test,y_test))

#Confusion Matrix
y_pred = naive_bayes.predict(x_test)
conf_mat = confusion_matrix(y_test,y_pred)
#Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(conf_mat,annot=True,linewidths=0.5,linecolor="red",fmt=".0
f",ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()

Naive Bayes Classification Score: 0.8785488958990536
```




```
plt.figure(figsize=(15,10))
plt.ylim([0.85,1])
plt.bar(method_names,method_scores,width=0.5)
plt.xlabel('Method Name')
plt.ylabel('Method Score')
```

```
Text(0, 0.5, 'Method Score')
```



Logistic Regression:

Accuracy: 92.59% Precision: 90.06% Recall: 96.74% F1-score: 93.28% K-Nearest Neighbors (KNN):

Accuracy: 70.50% Precision: 72.87% Recall: 70.92% F1-score: 71.88% Decision Tree:

Accuracy: 96.53% Precision: 98.46% Recall: 94.96% F1-score: 96.68%

Observations: Decision Tree performs the best in terms of accuracy (96.53%) and has the highest F1-score. Logistic Regression performs well with high precision and recall, making it a balanced model. KNN has the lowest performance across all metrics.

Conclusion:

In this classification task using a voice dataset to predict the gender of the speaker (male or female), we evaluated three models: Logistic Regression, K-Nearest Neighbors (KNN), and Decision Trees. Here's a summary of the results:

Decision Tree emerged as the top-performing model, with the highest accuracy (96.53%) and F1-score (96.68%). It had excellent precision and recall, making it the most suitable model for this dataset.

Logistic Regression also performed well, with an accuracy of 92.59% and a solid F1-score of 93.28%. While slightly behind the decision tree, this model demonstrated a good balance between precision (90.06%) and recall (96.74%).

KNN performed the worst, with a much lower accuracy of 70.50% and F1-score of 71.88%. This suggests that KNN is not the best choice for this particular dataset, possibly due to the complexity of the features or lack of clear proximity-based patterns in the data.

Recommendation: For this classification problem, Decision Trees would be the preferred model, as it consistently outperformed the others in all key metrics. If simplicity and interpretability are important, Logistic Regression also provides a viable option with reasonably high performance. KNN, however, may not be suitable for this dataset.