bee_classification

April 23, 2023

0.1 Classificação de Áudio com Python

Com o intuito de aprofundar meus conhecimentos em machine learning, procurei realizar algo diferente do normalmente eu vejo entre projetos de machine learning, com isso achei o tema de classificação de áudio, por ser formado em engenharia elétrica, me interessei pelo tema, e aqui estamos.

Depois de um processo de aprendizagem sobre alguns temas sobre processamento de áudio, consegui criar este notebook, no início ia abordar modelos baseados em redes neurais também, porém quando fiz os modelos de machine learning teve ótimos resultados, logo invés de construir redes neurais, utilizei métodos ensemble.

Quando tive que escolher a base de dados que futuramente iria trabalhar, busquei referencias entre meus conhecidos, então uma certa bióloga que ama abelhas e adora falar delas, me influenciou muito na minha decisão rsrs, nesse projeto estou utilizando uma base de dados do Kaggle que contém vários áudios de abelhas, divididos entre "abelha" e "não abelha" https://www.kaggle.com/datasets/andrewlca/bee-audio-object-detection.

As principais bibliotecas usadas nesse projeto foram `librosa` e `scikit learning`, sendo `librosa` uma poderosa biblioteca para processamento de áudios e `scikit learning` favorita quando o assunto é machine learning.

No preprocessamento utilizei a técnica `Coeficientes Cepstral de Frequência Mel (MFCC)`, isto é, uma escala de transformação não linear onde transforma a faixa de frequência do áudio em uma faixa de valor diferente, para que quando reproduzir o áudio soar idêntico para uma pessoa, ou seja, independente da distância que foi gravado o áudio, depois dessa transformação, o áudio vai ser percebido de igual distância para os humanos.

Os modelos de machine learning foram `Regressão Logística`, `Gaussian Naive Bayes`, `K Vizinhos mais Próximos`, `Floresta Randômica`, `Máquina de Vetores de Suporte (SVM)`, também utilizei dois métodos ensemble, que faz o uso de vários modelos para construir um modelo mais robusto, que são `Voting Classifier` e `Stacking Classifier`.

Os resultados superaram minha expectativa, de forma que nem utilizei redes neurais ou otimizei os modelos criados, todos estão com parâmetros padrões da biblioteca, e nas métricas que usei para avaliar os modelos, todos ficarão acima de 98%, seja a acurácia, a precisão, o recall ou o F1 score.

[ ]:
```
import os
import warnings
warnings.filterwarnings("ignore")
import numpy as np
```

```
import pandas as pd
import librosa as la
import librosa.display as ld
from IPython.display import Audio
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, StackingClassifier,␣
 ↪VotingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
from sklearn.model_selection import train_test_split
```

[ ]:
```
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Zero Zero" --iversions
```

```
Author: Zero Zero

matplotlib: 3.7.1
sklearn   : 1.2.2
numpy     : 1.23.5
librosa   : 0.10.0.post2
seaborn   : 0.12.2
pandas    : 2.0.0
```

[ ]:
```
paths = [os.path.join(name) for name in os.listdir("./data")]
paths_bee = []
paths_no_bee = []

for path in paths:
    if "No_Bee" in path:
        paths_no_bee.append(path)
    else:
        paths_bee.append(path)

len(paths_no_bee), len(paths_bee)
```

[ ]:
```
(1423, 1417)
```

[ ]:
```
rnd = np.random.randint(0, len(paths_bee))

data, sr = la.load(f"./data/{paths_bee[rnd]}", sr=44100, res_type="kaiser_best")

print("Canais ", len(data.shape))
print("# total de amostras ", data.shape[0])
print("Arquivo: ", paths_bee[rnd])
print("Taxa de amostragem ", sr)
print("Duração ", la.get_duration(y=data, sr=sr))

Audio(data=data, rate=sr)

# plt.title("SOM ABELHA", size=16)
# ld.waveshow(y=data, sr=sr)
# plt.show();
```

```
Canais  1
# total de amostras  441000
Arquivo:  Bee_1414.wav
Taxa de amostragem  44100
Duração  10.0
```

[ ]:
```
<IPython.lib.display.Audio object>
```

[ ]:
```
# rnd = np.random.randint(0, len(paths_bee))

# data, sr = la.load(f"./data/{paths_bee[rnd]}", sr=44100,␣
 ↪res_type="kaiser_best")

# stft = la.stft(data)
# stft_db = la.amplitude_to_db(np.abs(stft))
# ld.specshow(stft_db, x_axis="time", y_axis="log", cmap="Spectral")
# plt.show();
```

[ ]:
```
# rnd = np.random.randint(0, len(paths_bee))

# data, sr = la.load(f"./data/{paths_bee[rnd]}", sr=44100,␣
 ↪res_type="kaiser_best")

# mfccs = la.feature.mfcc(y=data, sr=sr, n_mfcc=40)
# mfccs_db = la.amplitude_to_db(np.abs(mfccs))
# ld.specshow(mfccs_db, x_axis="time", y_axis="log", cmap="Spectral")
# plt.show()
```

[ ]:
```
rnd = np.random.randint(0, len(paths_no_bee))

data, sr = la.load(f"./data/{paths_no_bee[rnd]}", sr=44100,␣
 ↪res_type="kaiser_best")
```

```
print("Canais ", len(data.shape))
print("# total de amostras ", data.shape[0])
print("Arquivo: ", paths_no_bee[rnd])
print("Taxa de amostragem ", sr)
print("Duração ", la.get_duration(y=data, sr=sr))

Audio(data=data, rate=sr)

# plt.title("SOM SEM ABELHA", size=16)
# ld.waveshow(y=data, sr=sr)
# plt.show();
```

```
Canais  1
# total de amostras  441000
Arquivo:  No_Bee_635.wav
Taxa de amostragem  44100
Duração  10.0
```

[ ]:
```
<IPython.lib.display.Audio object>
```

[ ]:
```
# rnd = np.random.randint(0, len(paths_no_bee))

# data, sr = la.load(f"./data/{paths_no_bee[rnd]}", sr=44100,␣
 ↪res_type="kaiser_best")

# stft = la.stft(data)
# stft_db = la.amplitude_to_db(np.abs(stft))

# plt.show(ld.specshow(stft_db, x_axis="time", y_axis="log", cmap="Spectral"));
```

[ ]:
```
# rnd = np.random.randint(0, len(paths_no_bee))

# data, sr = la.load(f"./data/{paths_no_bee[rnd]}", sr=44100,␣
 ↪res_type="kaiser_best")

# mfccs = la.feature.mfcc(y=data, sr=sr, n_mfcc=40)
# mfccs_db = la.amplitude_to_db(np.abs(mfccs))
# ld.specshow(mfccs_db, x_axis="time", y_axis="log", cmap="Spectral")
# plt.show()
```

[ ]:
```
def features_extrator(file_name):
    data, sample_rate = la.load(f"./data/{file_name}", sr=44100,␣
 ↪res_type="kaiser_best")
    mfccs_features = la.feature.mfcc(y=data, sr= sample_rate, n_mfcc=40)
    mfccs_features_scaled = np.mean(mfccs_features.T, axis=0)
    return mfccs_features_scaled
```

[ ]:
```
extracted_features_bee = []
extracted_features_nobee = []

for path in tqdm(paths_bee):
    try:
        data = features_extrator(path)
        extracted_features_bee.append(data)
    except:
        print(path)
        continue

for path in tqdm(paths_no_bee):
    try:
        data = features_extrator(path)
        extracted_features_nobee.append(data)
    except:
        print(path)
        continue
```

```
 67%|       | 954/1417 [01:08<00:50,  9.17it/s]
```

```
Bee_538.wav
```

```
 89%|      | 1268/1417 [01:49<00:16,  8.95it/s]
```

```
Bee_207.wav
```

```
100%|      | 1417/1417 [02:08<00:00, 11.03it/s]
 64%|       | 914/1423 [01:45<01:01,  8.23it/s]
```

```
No_Bee_203.wav
```

```
100%|      | 1423/1423 [02:47<00:00,  8.49it/s]
```

[ ]:
```
len(extracted_features_bee), len(extracted_features_nobee)
```

[ ]:
```
(1415, 1422)
```

[ ]:
```
df = pd.DataFrame({"feature": extracted_features_bee, "class": 1})
df2 = pd.DataFrame({"feature": extracted_features_nobee, "class": 0})
data = pd.concat([df, df2])
```

[ ]:
```
X = np.array(data["feature"].tolist())
y = np.array(data["class"].tolist())
```

[ ]:
```
X.shape, y.shape
```

[ ]:
```
((2837, 40), (2837,))
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3,␣
↪random_state=0)
```

[ ]:
```
clf_lr = LogisticRegression()
clf_lr.fit(X_train, y_train)
predict = clf_lr.predict(X_test)

print(accuracy_score(y_test, predict))
print(classification_report(y_test, predict))
confusion_matrix(y_test, predict)
```

```
0.994131455399061
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       415
           1       0.99      1.00      0.99       437

    accuracy                           0.99       852
   macro avg       0.99      0.99      0.99       852
weighted avg       0.99      0.99      0.99       852
```

[ ]:
```
array([[412,   3],
       [  2, 435]])
```

[ ]:
```
clf_nb = GaussianNB()
clf_nb.fit(X_train, y_train)
predict = clf_nb.predict(X_test)

print(accuracy_score(y_test, predict))
print(classification_report(y_test, predict))
confusion_matrix(y_test, predict)
```

```
0.9800469483568075
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       415
           1       0.96      1.00      0.98       437

    accuracy                           0.98       852
   macro avg       0.98      0.98      0.98       852
weighted avg       0.98      0.98      0.98       852
```

[ ]:
```
array([[398,  17],
       [  0, 437]])
```

[ ]:
```
clf_knn = KNeighborsClassifier()
clf_knn.fit(X_train, y_train)
predict = clf_knn.predict(X_test)
```

```
print(accuracy_score(y_test, predict))
print(classification_report(y_test, predict))
confusion_matrix(y_test, predict)
```

```
0.9953051643192489
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       415
           1       1.00      1.00      1.00       437

    accuracy                           1.00       852
   macro avg       1.00      1.00      1.00       852
weighted avg       1.00      1.00      1.00       852
```

[ ]:
```
array([[413,   2],
       [  2, 435]])
```

[ ]:
```
clf_rf = RandomForestClassifier()
clf_rf.fit(X_train, y_train)
predict = clf_rf.predict(X_test)

print(accuracy_score(y_test, predict))
print(classification_report(y_test, predict))
confusion_matrix(y_test, predict)
```

```
0.9976525821596244
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       415
           1       1.00      1.00      1.00       437

    accuracy                           1.00       852
   macro avg       1.00      1.00      1.00       852
weighted avg       1.00      1.00      1.00       852
```

[ ]:
```
array([[414,   1],
       [  1, 436]])
```

[ ]:
```
clf_svm = SVC()
clf_svm.fit(X_train, y_train)
predict = clf_svm.predict(X_test)

print(accuracy_score(y_test, predict))
print(classification_report(y_test, predict))
confusion_matrix(y_test, predict)
```

```
0.9800469483568075
```

```
                precision    recall  f1-score   support

           0       0.97      0.99      0.98       415
           1       0.99      0.97      0.98       437

    accuracy                           0.98       852
   macro avg       0.98      0.98      0.98       852
weighted avg       0.98      0.98      0.98       852
```

[ ]:
```
array([[410,   5],
       [ 12, 425]])
```

[ ]:
```
estimators = [("lr", LogisticRegression()),
              ("svm", SVC()),
              ("rf", RandomForestClassifier())]
```

[ ]:
```
clf_st = StackingClassifier(estimators=estimators)
clf_st.fit(X_train, y_train)
predict = clf_st.predict(X_test)

print(accuracy_score(y_test, predict))
print(classification_report(y_test, predict))
confusion_matrix(y_test, predict)
```

```
0.9964788732394366
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       415
           1       1.00      1.00      1.00       437

    accuracy                           1.00       852
   macro avg       1.00      1.00      1.00       852
weighted avg       1.00      1.00      1.00       852
```

[ ]:
```
array([[414,   1],
       [  2, 435]])
```

[ ]:
```
clf_vt = VotingClassifier(estimators=estimators)
clf_vt.fit(X_train, y_train)
predict = clf_vt.predict(X_test)

print(accuracy_score(y_test, predict))
print(classification_report(y_test, predict))
confusion_matrix(y_test, predict)
```

```
0.9964788732394366
              precision    recall  f1-score   support
```

```
              0       1.00      1.00      1.00       415
              1       1.00      1.00      1.00       437

       accuracy                           1.00       852
      macro avg       1.00      1.00      1.00       852
   weighted avg       1.00      1.00      1.00       852
```

[ ]:
```
array([[414,   1],
       [  2, 435]])
```