speaker-identification-train

May 5, 2021

[1]:
```
# -*- coding: utf-8 -*-

import os
import sys
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from features import FeatureExtractor
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
import pickle


# %%---------------------------------------------------------------------------
#
#                                Load Data From Disk
#
# -----------------------------------------------------------------------------

#data_dir = 'data/train' # directory where the data files are stored
data_dir = 'data' # directory where the data files are stored

output_dir = 'training_output' # directory where the classifier(s) are stored

if not os.path.exists(output_dir):
        os.mkdir(output_dir)

# the filenames should be in the form 'speaker-data-subject-1.csv', e.g.␣
↪'speaker-data-Erik-1.csv'.

#class_names = ["EDM", "Classical", "Metal"] # the set of classes, i.e. speakers
class_names = [] # the set of classes, i.e. speakers
```

```
data = np.zeros((0,8002)) #8002 = 1 (timestamp) + 8000 (for 8kHz audio data) +␣
↪1 (label)

for filename in os.listdir(data_dir):
        if filename.endswith(".csv") and filename.startswith("speaker-data"):
                filename_components = filename.split("-") # split by the '-'␣
↪character
                speaker = filename_components[2]
                print("Loading data for {}.".format(speaker))
                if speaker not in class_names:
                        class_names.append(speaker)
                speaker_label = class_names.index(speaker)
                sys.stdout.flush()
                data_file = os.path.join(data_dir, filename)
                data_for_current_speaker = np.genfromtxt(data_file,␣
↪delimiter=',')
                print("Loaded {} raw labelled audio data samples.".
↪format(len(data_for_current_speaker)))
                sys.stdout.flush()
                data = np.append(data, data_for_current_speaker, axis=0)

print("Found data for {} speakers : {}".format(len(class_names), ", ".
↪join(class_names)))
```

```
Loading data for Classical.
Loaded 3600 raw labelled audio data samples.
Loading data for EDM.
Loaded 3600 raw labelled audio data samples.
Loading data for Metal.
Loaded 3600 raw labelled audio data samples.
Found data for 3 speakers : Classical, EDM, Metal
```

[2]:
```
#
#                               Extract Features & Labels
#
# -----------------------------------------------------------------------------

# Update this depending on how you compute your features
n_features = 1095

print("Extracting features and labels for {} audio windows...".format(data.
↪shape[0]))
sys.stdout.flush()

X = np.zeros((0,n_features))
y = np.zeros(0,)
```

```
# change debug to True to show print statements we've included:
feature_extractor = FeatureExtractor(debug=False)
nr_total_windows = 0
nr_bad_windows = 0
nr_windows_with_zeros = 0

for i,window_with_timestamp_and_label in enumerate(data):
        window = window_with_timestamp_and_label[1:-1]
        label = data[i,-1]
        nr_total_windows += 1
        try:
                x = feature_extractor.extract_features(window)
                if (len(x) != X.shape[1]):
                        print("Received feature vector of length {}. Expected␣
 ↪feature vector of length {}.".format(len(x), X.shape[1]))
                X = np.append(X, np.reshape(x, (1,-1)), axis=0)
                y = np.append(y, label)
        except:
                nr_bad_windows += 1
                if np.all((window == 0)):
                        nr_windows_with_zeros += 1

print("{} windows found".format(nr_total_windows))
print("{} bad windows found, with {} windows with only zeros".
 ↪format(nr_bad_windows, nr_windows_with_zeros))

print("Finished feature extraction over {} windows".format(len(X)))
print("Unique labels found: {}".format(set(y)))
sys.stdout.flush()
```

```
Extracting features and labels for 10800 audio windows…
c:\Users\zacha\Classify-Genre-Final\features.py:174: ComplexWarning: Casting
complex values to real discards the imaginary part
  fft = np.mean(np.fft.rfft(window, axis=0).astype(float))
10800 windows found
0 bad windows found, with 0 windows with only zeros
Finished feature extraction over 10800 windows
Unique labels found: {0.0, 1.0, 2.0}
```

[3]:

```
#
#                                 Train & Evaluate Classifier
#
# -----------------------------------------------------------------------------

n = len(y)
n_classes = len(class_names)
```

```python
print("\n")
print("---------------------- Decision Tree -------------------------")

total_accuracy = 0.0
total_precision = [0.0, 0.0, 0.0]
total_recall = [0.0, 0.0, 0.0]

cv = KFold(n_splits=10, shuffle=True, random_state=None)
for i, (train_index, test_index) in enumerate(cv.split(X)):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        tree = DecisionTreeClassifier(criterion="entropy", max_depth=3)
        print("Fold {} : Training decision tree classifier over {} points...".format(i, len(y_train)))
        sys.stdout.flush()
        tree.fit(X_train, y_train)
        print("Evaluating classifier over {} points...".format(len(y_test)))

        # predict the labels on the test data
        y_pred = tree.predict(X_test)

        # show the comparison between the predicted and ground-truth labels
        conf = confusion_matrix(y_test, y_pred, labels=[0,1,2])

        accuracy = np.sum(np.diag(conf)) / float(np.sum(conf))
        precision = np.nan_to_num(np.diag(conf) / np.sum(conf, axis=1).astype(float))
        recall = np.nan_to_num(np.diag(conf) / np.sum(conf, axis=0).astype(float))

        total_accuracy += accuracy
        total_precision += precision
        total_recall += recall

print("The average accuracy is {}".format(total_accuracy/10.0))
print("The average precision is {}".format(total_precision/10.0))
print("The average recall is {}".format(total_recall/10.0))

print("Training decision tree classifier on entire dataset...")
tree.fit(X, y)

print("\n")
print("---------------------- Random Forest Classifier -------------------------")
total_accuracy = 0.0
total_precision = [0.0, 0.0, 0.0]
total_recall = [0.0, 0.0, 0.0]
```

```
for i, (train_index, test_index) in enumerate(cv.split(X)):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        print("Fold {} : Training Random Forest classifier over {} points...".
 ↪format(i, len(y_train)))
        sys.stdout.flush()
        clf = RandomForestClassifier(n_estimators=100)
        clf.fit(X_train, y_train)

        print("Evaluating classifier over {} points...".format(len(y_test)))
        # predict the labels on the test data
        y_pred = clf.predict(X_test)

        # show the comparison between the predicted and ground-truth labels
        conf = confusion_matrix(y_test, y_pred, labels=[0,1,2])

        accuracy = np.sum(np.diag(conf)) / float(np.sum(conf))
        precision = np.nan_to_num(np.diag(conf) / np.sum(conf, axis=1).
 ↪astype(float))
        recall = np.nan_to_num(np.diag(conf) / np.sum(conf, axis=0).
 ↪astype(float))

        total_accuracy += accuracy
        total_precision += precision
        total_recall += recall

print("The average accuracy is {}".format(total_accuracy/10.0))
print("The average precision is {}".format(total_precision/10.0))
print("The average recall is {}".format(total_recall/10.0))

# TODO: (optional) train other classifiers and print the average metrics using␣
 ↪10-fold cross-validation

print("\n")
print("---------------------- Naive Bayes Classifier -------------------------")
total_accuracy = 0.0
total_precision = [0.0, 0.0, 0.0]
total_recall = [0.0, 0.0, 0.0]

for i, (train_index, test_index) in enumerate(cv.split(X)):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        print("Fold {} : Training Naive Bayes classifier over {} points...".
 ↪format(i, len(y_train)))
        sys.stdout.flush()
        nb = GaussianNB()
```

```
        nb.fit(X_train, y_train)

        print("Evaluating classifier over {} points...".format(len(y_test)))
        # predict the labels on the test data
        y_pred = nb.predict(X_test)

        # show the comparison between the predicted and ground-truth labels
        conf = confusion_matrix(y_test, y_pred, labels=[0,1,2])

        accuracy = np.sum(np.diag(conf)) / float(np.sum(conf))
        precision = np.nan_to_num(np.diag(conf) / np.sum(conf, axis=1).
↪astype(float))
        recall = np.nan_to_num(np.diag(conf) / np.sum(conf, axis=0).
↪astype(float))

        total_accuracy += accuracy
        total_precision += precision
        total_recall += recall

print("The average accuracy is {}".format(total_accuracy/10.0))
print("The average precision is {}".format(total_precision/10.0))
print("The average recall is {}".format(total_recall/10.0))

print("\n")
print("---------------------- SVM Classifier ------------------------")
total_accuracy = 0.0
total_precision = [0.0, 0.0, 0.0]
total_recall = [0.0, 0.0, 0.0]

for i, (train_index, test_index) in enumerate(cv.split(X)):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        print("Fold {} : Training SVM classifier over {} points...".format(i,␣
↪len(y_train)))
        sys.stdout.flush()
        clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
        clf.fit(X_train, y_train)

        print("Evaluating classifier over {} points...".format(len(y_test)))
        # predict the labels on the test data
        y_pred = clf.predict(X_test)

        # show the comparison between the predicted and ground-truth labels
        conf = confusion_matrix(y_test, y_pred, labels=[0,1,2])

        accuracy = np.sum(np.diag(conf)) / float(np.sum(conf))
```

```
        precision = np.nan_to_num(np.diag(conf) / np.sum(conf, axis=1).
 ↪astype(float))
        recall = np.nan_to_num(np.diag(conf) / np.sum(conf, axis=0).
 ↪astype(float))

        total_accuracy += accuracy
        total_precision += precision
        total_recall += recall

print("The average accuracy is {}".format(total_accuracy/10.0))
print("The average precision is {}".format(total_precision/10.0))
print("The average recall is {}".format(total_recall/10.0))


# Set this to the best model you found, trained on all the data:
best_classifier = RandomForestClassifier(n_estimators=100)
best_classifier.fit(X,y)

classifier_filename='classifier.pickle'
print("Saving best classifier to {}...".format(os.path.join(output_dir,␣
 ↪classifier_filename)))
with open(os.path.join(output_dir, classifier_filename), 'wb') as f: # 'wb'␣
 ↪stands for 'write bytes'
        pickle.dump(best_classifier, f)
```

```
---------------------- Decision Tree -------------------------
Fold 0 : Training decision tree classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 1 : Training decision tree classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 2 : Training decision tree classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 3 : Training decision tree classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 4 : Training decision tree classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 5 : Training decision tree classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 6 : Training decision tree classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 7 : Training decision tree classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 8 : Training decision tree classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 9 : Training decision tree classifier over 9720 points…
```

```
Evaluating classifier over 1080 points…
The average accuracy is 0.8061111111111111
The average precision is [0.49216355 0.93588626 0.99008133]
The average recall is [0.9721522  0.94051798 0.66058526]
Training decision tree classifier on entire dataset…


---------------------- Random Forest Classifier -------------------------
Fold 0 : Training Random Forest classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 1 : Training Random Forest classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 2 : Training Random Forest classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 3 : Training Random Forest classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 4 : Training Random Forest classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 5 : Training Random Forest classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 6 : Training Random Forest classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 7 : Training Random Forest classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 8 : Training Random Forest classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 9 : Training Random Forest classifier over 9720 points…
Evaluating classifier over 1080 points…
The average accuracy is 0.9355555555555556
The average precision is [0.89982164 0.99269601 0.91398268]
The average recall is [0.91390569 0.96540885 0.92560054]


---------------------- Naive Bayes Classifier -------------------------
Fold 0 : Training Naive Bayes classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 1 : Training Naive Bayes classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 2 : Training Naive Bayes classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 3 : Training Naive Bayes classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 4 : Training Naive Bayes classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 5 : Training Naive Bayes classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 6 : Training Naive Bayes classifier over 9720 points…
Evaluating classifier over 1080 points…
```

```
Fold 7 : Training Naive Bayes classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 8 : Training Naive Bayes classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 9 : Training Naive Bayes classifier over 9720 points…
Evaluating classifier over 1080 points…
The average accuracy is 0.695462962962963
The average precision is [0.56300816 0.9270376  0.5973069 ]
The average recall is [0.63217627 0.83239024 0.60140841]


---------------------- SVM Classifier -------------------------
Fold 0 : Training SVM classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 1 : Training SVM classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 2 : Training SVM classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 3 : Training SVM classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 4 : Training SVM classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 5 : Training SVM classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 6 : Training SVM classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 7 : Training SVM classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 8 : Training SVM classifier over 9720 points…
Evaluating classifier over 1080 points…
Fold 9 : Training SVM classifier over 9720 points…
Evaluating classifier over 1080 points…
The average accuracy is 0.9103703703703703
The average precision is [0.82357004 0.98629322 0.92167092]
The average recall is [0.90811261 0.98000501 0.84839045]
Saving best classifier to training_output\classifier.pickle…
```

[4]:
```
#
#                                Load Data From Disk
#
# ----------------------------------------------------------------------------

data_dir = 'test' # directory where the data files are stored

output_dir = 'testing_output' # directory where the classifier(s) are stored

if not os.path.exists(output_dir):
```

```
        os.mkdir(output_dir)

# the filenames should be in the form 'speaker-data-subject-1.csv', e.g.␣
↪'speaker-data-Erik-1.csv'.

class_names = [] # the set of classes, i.e. speakers

data = np.zeros((0,8002)) #8002 = 1 (timestamp) + 8000 (for 8kHz audio data) +␣
↪1 (label)

for filename in os.listdir(data_dir):
        if filename.endswith(".csv") and filename.startswith("speaker-data"):
                filename_components = filename.split("-") # split by the '-'␣
↪character
                speaker = filename_components[2]
                print("Loading data for {}.".format(speaker))
                if speaker not in class_names:
                        class_names.append(speaker)
                speaker_label = class_names.index(speaker)
                sys.stdout.flush()
                data_file = os.path.join(data_dir, filename)
                data_for_current_speaker = np.genfromtxt(data_file,␣
↪delimiter=',')
                print("Loaded {} raw labelled audio data samples.".
↪format(len(data_for_current_speaker)))
                sys.stdout.flush()
                data = np.append(data, data_for_current_speaker, axis=0)

print("Found data for {} speakers : {}".format(len(class_names), ", ".
↪join(class_names)))
```

```
Loading data for Classical.
Loaded 1180 raw labelled audio data samples.
Loading data for EDM.
Loaded 1348 raw labelled audio data samples.
Loading data for Metal.
Loaded 1741 raw labelled audio data samples.
Found data for 3 speakers : Classical, EDM, Metal
```

[5]:
```
#
#                                Extract Features & Labels
#
# -----------------------------------------------------------------------------
# Update this depending on how you compute your features
n_features = 1095
```

```
print("Extracting features and labels for {} audio windows...".format(data.
 ↪shape[0]))
sys.stdout.flush()

X = np.zeros((0,n_features))
y = np.zeros(0,)

nr_total_windows = 0
nr_bad_windows = 0
nr_windows_with_zeros = 0

for i,window_with_timestamp_and_label in enumerate(data):
    window = window_with_timestamp_and_label[1:-1]
    label = data[i,-1]
    nr_total_windows += 1
    try:
        x = feature_extractor.extract_features(window)
        if (len(x) != X.shape[1]):
            print("Received feature vector of length {}. Expected feature␣
 ↪vector of length {}.".format(len(x), X.shape[1]))
        X = np.append(X, np.reshape(x, (1,-1)), axis=0)
        y = np.append(y, label)
    except:
        nr_bad_windows += 1
        if np.all((window == 0)):
            nr_windows_with_zeros += 1

print("{} windows found".format(nr_total_windows))
print("{} bad windows found, with {} windows with only zeros".
 ↪format(nr_bad_windows, nr_windows_with_zeros))

print("Finished feature extraction over {} windows".format(len(X)))
print("Unique labels found: {}".format(set(y)))
sys.stdout.flush()
```

```
Extracting features and labels for 4269 audio windows…
c:\Users\zacha\Classify-Genre-Final\features.py:174: ComplexWarning: Casting
complex values to real discards the imaginary part
  fft = np.mean(np.fft.rfft(window, axis=0).astype(float))
4269 windows found
177 bad windows found, with 177 windows with only zeros
Finished feature extraction over 4092 windows
Unique labels found: {0.0, 1.0, 2.0}
```

[7]:
```
#
#                                   Predict on new data
#
```

```python
# ------------------------------------------------------------------------------
total_accuracy = 0.0
total_precision = [0.0, 0.0, 0.0]
total_recall = [0.0, 0.0, 0.0]

y_pred = best_classifier.predict(X)

# show the comparison between the predicted and ground-truth labels
conf = confusion_matrix(y, y_pred, labels=[0,1,2])
print(conf)

accuracy = np.sum(np.diag(conf)) / float(np.sum(conf))
precision = np.nan_to_num(np.diag(conf) / np.sum(conf, axis=1).astype(float))
recall = np.nan_to_num(np.diag(conf) / np.sum(conf, axis=0).astype(float))

print("The accuracy is {}".format(accuracy))
print("The precision is {}".format(precision))
print("The recall is {}".format(recall))
```

```
[[ 986  125   64]
 [  13 1163    4]
 [ 867  191  679]]
The accuracy is 0.6911045943304008
The precision is [0.83914894 0.98559322 0.39090386]
The recall is [0.528403   0.78634212 0.90896921]
```