Racovita Andra-Georgiana 232

Surgical mask detection

Documentation

I. Description of the Machine Learning approach

I have transformed the audio files in MFCCs, Mel-frequency cepstral coefficients, to help me classify the data. After I tried different approaches to find the best classification, such as SVM, Linear SVM, Logistic Regression, Random Forest Classification, Logistic Regression, the models that gave me the best scores on the validation data were SVM and Logistic Regression.

II. The steps of my implementation

1) First, I read from train.txt, test.txt, validation.txt, the name of every audio file and their label, 0 means without mask and 1 with mask, and saved the names (wavTrainNameFiles, wavTestNameFiles, wavValidationNameFiles) and the labels (yTrain, yTest, yValidation) in different vectors.

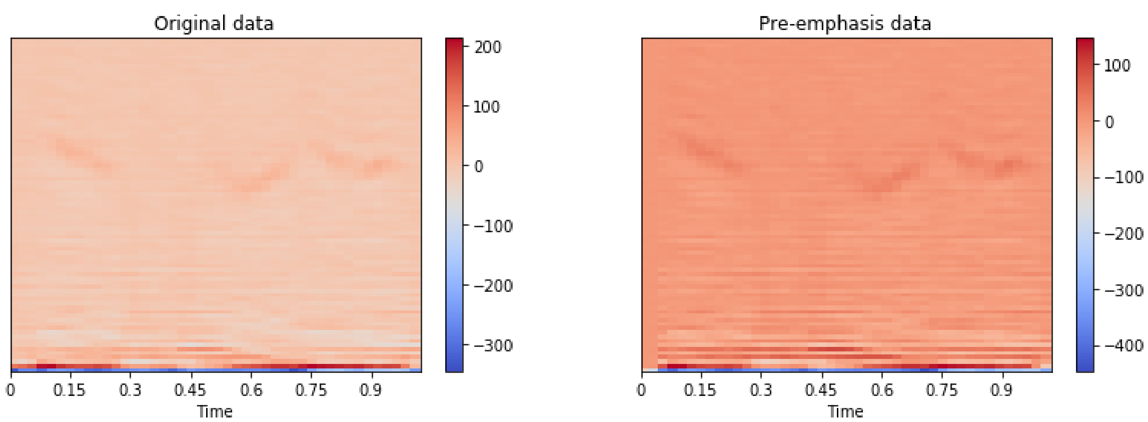
2) Second, I read the audio files for Train, Test and Validation and saved them in TrainData, TestData and ValidationData. Moreover, I did some preprocessing.

a. I used librosa with the default parameters to load an audio file as a floating point time series.

b. I pre-emphasized an audio signal with a first-order autoregressive filter:

$$y[n] \text{ -> } y[n] - coef * y[n-1]$$

to boost the high frequency component.



c. “Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound, for example, in audio compression."[1] From librosa I used librosa.feature.mfcc to get the Mel-frequency cepstral coefficients of every audio file. The parameters are:

- I keep the default sampling rate of data from returned by the load function, 22050.
- n_mfcc = 80, the number of mfccs to return.

d. For better data, I normalized the MFCCS using sklearn.preprocessing.scale with the following parameters:

- axis = 0, independently standardizing each feature.
- with_mean = True, centering the data.
- with_std = True, scale the data to unit variance.

e. Finally, appending the flattened mfcc_feat to TrainData, TestData or ValidationData.

```
[5]
# Read files

import librosa as lb
import sklearn

# Train files
TrainData = []
for i in range(len(wavTrainNameFiles)):
    data, rate = lb.load(wavTrainNameFiles[i])
    data = lb.effects.preemphasis(data)
    mfcc_feat = lb.feature.mfcc(data, rate, n_mfcc = 80)
    mfcc_feat = sklearn.preprocessing.scale(mfcc_feat, axis = 0, with_mean = True)
    TrainData.append(mfcc_feat.flatten())

# Test files
TestData = []
for i in range(len(wavTestNameFiles)):
    data, rate = lb.load(wavTestNameFiles[i])
    data = lb.effects.preemphasis(data)
    mfcc_feat = lb.feature.mfcc(data, rate, n_mfcc = 80)
    mfcc_feat = sklearn.preprocessing.scale(mfcc_feat, axis=0, with_mean = True)
    TestData.append(mfcc_feat.flatten())

# Validation files
ValidationData = []
for i in range(len(wavValidationNameFiles)):
    data, rate = lb.load(wavValidationNameFiles[i])
    data = lb.effects.preemphasis(data)
    mfcc_feat = lb.feature.mfcc(data, rate, n_mfcc = 80)
    mfcc_feat = sklearn.preprocessing.scale(mfcc_feat, axis=0, with_mean = True)
    ValidationData.append(mfcc_feat.flatten())
```

[1] https://en.wikipedia.org/wiki/Mel-frequency_cepstrum - Wikipedia last edited on 21 December 2019, at 11:54 (UTC)

3) I prepared the data for classification, changing the variabile names for better understanding.

```
[15]
# Train
X_train = TrainData
y_train = yTrain

# Validation
X_test = ValidationData
y_test = yValidation

# Test
X_test_final = TestData
```

4) To prevent overfitting I split the X_test and y_test in 2 parts.

```
[14]
# To prevent overfitting, I will do predictions only on 50% of the validation data
from sklearn.model_selection import train_test_split
X_test1, X_test2, y_test1, y_test2 = train_test_split(X_test, y_test, test_size = 1/2)
```

5) To choose the best model I used GridSearchCV[2].

a. For **LogisticRegression** for X_test2:

I used the following parameters:

```
tuned_parameters = {'penalty': ['l2'],'C':[0.001, 0.01, 1, 5, 10, 25]}
```

Grid scores on development set:

Score	Penalty	C
0.634 (+/-0.019)	l2	0.001
0.649 (+/-0.022)	l2	0.01
0.621 (+/-0.030)	l2	1
0.624 (+/-0.025)	l2	5
0.625 (+/-0.030)	l2	10
0.623 (+/-0.025)	l2	25

Best parameters set found on development set: {'C': 0.01, 'penalty': 'l2'}.

[2] https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html

```
Detailed classification report:


The model is trained on the full development set.

The scores are computed on the full evaluation set.


[[149  94]
 [ 93 164]]

              precision    recall  f1-score   support

           0       0.62      0.61      0.61       243
           1       0.64      0.64      0.64       257

    accuracy                           0.63       500
   macro avg       0.63      0.63      0.63       500
weighted avg       0.63      0.63      0.63       500


0.6256806970337625
```

b. For **SVM** for X_Test:

I used the following parameters:

```
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-1, 1e-2, 1e-3, 1e-4, 1e-5],
                     'C': [1, 100, 500, 1000, 2000]}]
```

Grid scores on development set:

```
0.648 (+/-0.016) for {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}

0.644 (+/-0.013) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}

0.601 (+/-0.021) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}

0.659 (+/-0.016) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}

0.667 (+/-0.020) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}

0.637 (+/-0.017) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}

0.602 (+/-0.021) for {'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'}

0.659 (+/-0.016) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}

0.649 (+/-0.016) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}

0.644 (+/-0.011) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}

0.635 (+/-0.019) for {'C': 100, 'gamma': 1e-05, 'kernel': 'rbf'}
```

```
0.659 (+/-0.016) for {'C': 500, 'gamma': 0.01, 'kernel': 'rbf'}

0.651 (+/-0.019) for {'C': 500, 'gamma': 0.001, 'kernel': 'rbf'}

0.645 (+/-0.007) for {'C': 500, 'gamma': 0.0001, 'kernel': 'rbf'}

0.639 (+/-0.011) for {'C': 500, 'gamma': 1e-05, 'kernel': 'rbf'}

0.659 (+/-0.016) for {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}

0.651 (+/-0.019) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}

0.640 (+/-0.008) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}

0.635 (+/-0.011) for {'C': 1000, 'gamma': 1e-05, 'kernel': 'rbf'}

0.659 (+/-0.016) for {'C': 2000, 'gamma': 0.01, 'kernel': 'rbf'}

0.651 (+/-0.019) for {'C': 2000, 'gamma': 0.001, 'kernel': 'rbf'}

0.632 (+/-0.011) for {'C': 2000, 'gamma': 0.0001, 'kernel': 'rbf'}

0.633 (+/-0.004) for {'C': 2000, 'gamma': 1e-05, 'kernel': 'rbf'}
```

Best parameters set found on development set: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}

```
Detailed classification report:


The model is trained on the full development set.

The scores are computed on the full evaluation set.


[[309 163]
 [175 353]]

              precision    recall  f1-score   support

           0       0.64      0.65      0.65       472
           1       0.68      0.67      0.68       528

    accuracy                           0.66      1000
   macro avg       0.66      0.66      0.66      1000
weighted avg       0.66      0.66      0.66      1000


0.6612691395989494
```

6) Fitting the models from the Grid Search

a. For **LogisticRegression**:

```
from sklearn.linear_model import LogisticRegression
regr = LogisticRegression(C=0.001, penalty='l2')
regr.fit(X_train, y_train)
```

```
LogisticRegression(C=0.001, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

- penalty = ’l2’. As an optimization problem, binary class ℓ2 penalized logistic regression minimizes the following cost function[3]:

$$\min_{w,c} \frac{1}{2}w^T w + C\sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

- C = 0.001
- Solver = ‘lbfgs’. “L-BFGS uses an estimate of the inverse Hessian matrix to steer its search through variable space, but where BFGS stores a dense n x n approximation to the inverse Hessian (n being the number of variables in the problem), L-BFGS stores only a few vectors that represent the approximation implicitly.”[4]

I predicted the labels for both, X_test1 and X_test2, and did the accuracy score.

```
[36]
pred1_30 = regr.predict(X_test2)
pred1_70 = regr.predict(X_test1)
pred2 = regr.predict(X_test_final)
```

```
[37]
from sklearn.metrics import accuracy_score
accs = accuracy_score(y_test2, pred1_30)
print(accs)
print()
accs = accuracy_score(y_test1, pred1_70)
print(accs)
```

```
0.664

0.6
```

[3] https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

[4] https://en.wikipedia.org/wiki/Limited-memory_BFGS- Wikipedia last edited on 16 May 2020, at 08:01 (UTC).

Cross Validation:

```
[39]
from sklearn.model_selection import cross_val_score
scores = cross_val_score(regr, X_test2, y_test2, cv=5)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
print()
scores = cross_val_score(regr, X_test1, y_test1, cv=5)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

Accuracy: 0.60 (+/- 0.04)

Accuracy: 0.56 (+/- 0.09)
```

Classification report:

```
[38]
from sklearn.metrics import classification_report
print(classification_report(y_test2, pred1_30))
print()
print(classification_report(y_test1, pred1_70))

precision    recall  f1-score   support

           0       0.64      0.65      0.65       237
           1       0.68      0.68      0.68       263

    accuracy                           0.66       500
   macro avg       0.66      0.66      0.66       500
weighted avg       0.66      0.66      0.66       500


              precision    recall  f1-score   support

           0       0.58      0.54      0.56       235
           1       0.61      0.66      0.64       265

    accuracy                           0.60       500
   macro avg       0.60      0.60      0.60       500
weighted avg       0.60      0.60      0.60       500
```

Confusion matrix[5]:

a) X_test1

```
Confusion matrix X_test1, without normalization
[[146  91]
 [ 87 176]]
Normalized confusion matrix X_test1
[[0.62 0.38]
 [0.33 0.67]]
```

b) X_test2

[5] https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

```
Confusion matrix X_test2, without normalization
[[134 101]
 [ 89 176]]
Normalized confusion matrix X_test2
[[0.57 0.43]
 [0.34 0.66]]
```

b. For **SVM**:

First, I tried the model with the parameters that GridSearchCV gave me as the best.

I. C=10, gamma=0.001, kernel='rbf

- C=10, "C is the penalty parameter of the error term. It controls the trade off between smooth decision boundary and classifying the training points correctly."[6]
- gamma=0.001
- kernel='rbf, the type of hyperplane used to separate the data

Fitting the model and making predictions:

```
[14]
# The model with the best score on validation
from sklearn import svm

regr = svm.SVC(C=10, gamma=0.001)
regr.fit(X_train, y_train)

SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)

[15]
pred_test1 = regr.predict(X_test2)
pred_test2 = regr.predict(X_test1)
pred2 = regr.predict(X_test_final)
```

Accuracy score: 1. X_test1: 0.678
2. X_test2: 0.682

Confusion matrix:

1. X_test1:

```
Confusion matrix X_test1, without normalization
[[158  81]
 [ 78 183]]
Normalized confusion matrix X_test1
[[0.66 0.34]
 [0.3  0.7 ]]
```

[6] https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769

2. X_test2:

```
Confusion matrix X_test2, without normalization
[[156  77]
 [ 84 183]]
Normalized confusion matrix X_test2
[[0.67 0.33]
 [0.31 0.69]]
```

Cross Validation:

3. X_Test1: Accuracy: 0.55 (+/- 0.10)
4. X_Test2: Accuracy: 0.56 (+/- 0.07)

Classification report:

```python
from sklearn.metrics import classification_report
print(classification_report(y_test2, pred_test1))
print()
print(classification_report(y_test1, pred_test2))
```

```
precision    recall  f1-score   support

           0       0.65      0.67      0.66       233
           1       0.70      0.69      0.69       267

    accuracy                           0.68       500
   macro avg       0.68      0.68      0.68       500
weighted avg       0.68      0.68      0.68       500

              precision    recall  f1-score   support

           0       0.67      0.66      0.67       239
           1       0.69      0.70      0.70       261

    accuracy                           0.68       500
   macro avg       0.68      0.68      0.68       500
weighted avg       0.68      0.68      0.68       500
```

Second, the model that gave the best score on test data.

I used a default SVC.

- C=1
- gamma='scale', calculated after this formula 1 / (n_features * X.var())
- kernel='rbf'

Fitting the model and making predictions:

```
# Model that got the best scor on leaderboard
from sklearn import svm

regr = svm.SVC()
regr.fit(X_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
pred_test1 = regr.predict(X_test2)
pred_test2 = regr.predict(X_test1)
pred2 = regr.predict(X_test_final)
```

Accuracy score: 1. X_test1: 0.612
2. X_test2: 0.634

Confusion matrix:

1. X_test1:

```
Confusion matrix X_test1, without normalization
[[135 102]
 [ 81 182]]
Normalized confusion matrix X_test1
[[0.57 0.43]
 [0.31 0.69]]
```

2. X_test2:

```
Confusion matrix X_test2, without normalization
[[125 110]
 [ 84 181]]
Normalized confusion matrix X_test2
[[0.53 0.47]
 [0.32 0.68]]
```

Cross Validation:

1. X_Test1: Accuracy: 0.53 (+/- 0.01)
2. X_Test2: Accuracy: 0.53 (+/- 0.03)

Classification report:

```
[20]
from sklearn.metrics import classification_report
print(classification_report(y_test2, pred_test1))
print()
print(classification_report(y_test1, pred_test2))
```

```
precision    recall  f1-score   support

           0       0.60      0.53      0.56       235
           1       0.62      0.68      0.65       265

    accuracy                           0.61       500
   macro avg       0.61      0.61      0.61       500
weighted avg       0.61      0.61      0.61       500


              precision    recall  f1-score   support

           0       0.62      0.57      0.60       237
           1       0.64      0.69      0.67       263

    accuracy                           0.63       500
   macro avg       0.63      0.63      0.63       500
weighted avg       0.63      0.63      0.63       500
```