Feature_Selection

October 23, 2022

1 Assignment 2 - Sound Recognition-Feature Selection, Taking domain-specific no window as example

1.1 1. Pre-processing: Load All Recordings, Median Filter, and FFT

[1]:
```
import librosa
import cv2
import glob
import numpy as np
import scipy
import joblib
import matplotlib.pyplot as plt
import scipy.io.wavfile as wavfile

from scipy.fftpack import fft
from scipy import signal


from numpy.lib.stride_tricks import sliding_window_view
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, accuracy_score
from sklearn import preprocessing

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import ConfusionMatrixDisplay
```

2 2. Domain Specific Models To Select Features

[2]:
```
def load_data(classes):
    class Audios():
        f= None
        fs= None
        tag= None
```

```
    tags=[]
    data=[]

    for i_class in classes:
        for each_file in glob.glob("Dataset/"+i_class+"/*.wav"):
            audio=Audios()
            f, fs=librosa.load(each_file, sr=None, mono=True, offset=0.0,␣
 ↪duration=None) # load file
            f = signal.medfilt(f, kernel_size=3) # median filter with zero␣
 ↪padding
            audio.f= f
            audio.fs= fs
            audio.tag=i_class
            tags.append(audio.tag)
            data.append(audio)
    return data, tags
```

[3]:
```
classes = ["Alarm", "Silence", "Music", "Microwave", "Clean", "Blender"]
data, tags = load_data(classes)
```

2.1 2. Model Building: Domain Model with Windows

2.2 I. Training and Save Model

[4]:
```
def cutting_into_windows(sample, sample_fs):
    """Cutting sample into windows, every is 6s"""
    window_size = int(sample_fs * 2)# 6s instances/window
    window_overlap = window_size // 2
    windows = []
    i = 0
    win_count = 0
    while ((i + window_size) < len(sample)) and (win_count < 20):
        windows.append(sample[i: i + window_size])
        i = i + window_size - window_overlap
        win_count = win_count + 1
    return np.array(windows)
```

[6]:
```
def get_mfcc_window_features(FFT_SIZE, num_freq_bins, num_time_bins, data):
    """Obtain all data's features"""
    # Use mfcc
    features=[]
    for sample in data:
        windows = cutting_into_windows(sample.f, sample.fs)
        sample_features = []

        for window in windows:
            # If only use mfcc
```

```
            mfccs=librosa.feature.mfcc(y=window,sr=sample.fs,n_mfcc=20)
            mfccs=np.mean(mfccs.T, axis=0)
            sample_features.append(mfccs.reshape((-1, )))

            pass
        features.append(np.array(sample_features).mean(axis=0))
    features = np.array(features)
    return features
```

[7]:
```
def get_window_features(FFT_SIZE, num_freq_bins, num_time_bins, data):
    """Obtain all data's features"""

    features=[]
    for sample in data:
        windows = cutting_into_windows(sample.f, sample.fs)
        sample_features = []

        for window in windows:
            # If use other, can self select

            f,t,pxx = signal.spectrogram(window, nperseg=FFT_SIZE, fs=sample.
↪fs, noverlap=int(FFT_SIZE/4))
            # max_index = np.argmax(pxx, axis=0)
            # min_index = np.argmin(pxx, axis=0)
            # var = np.var(pxx,axis=0)
            max_per_window = np.max(pxx, axis=0)
            min_per_window = np.min(pxx, axis=0)
            # sum_per_window = np.sum(pxx, axis=0)
            # mean_per_window = np.mean(pxx, axis=0)
            # std_per_window = np.std(pxx, axis=0)
            median_per_window = np.median(pxx, axis=0)
            quan_per_window_1 = np.quantile(pxx, 0.25, axis=0)
            quan_per_window_3 = np.quantile(pxx, 0.75, axis=0)

            sample_features.append([max_per_window.mean(), min_per_window.
↪mean(), median_per_window.mean(), quan_per_window_1.mean(),␣
↪quan_per_window_3.mean()])
                #  max_index.mean(), min_index.mean()])#, var.mean(),␣
↪max_per_window.mean(), min_per_window.mean(), sum_per_window.mean(),\
                 #mean_per_window.mean(), std_per_window.mean(),␣
↪median_per_window.mean(), quan_per_window_1.mean(), quan_per_window_3.
↪mean()])
            pass
        features.append(np.array(sample_features).mean(axis=0))
    features = np.array(features)
    return features
```

[8]:
```
FFT_SIZE=1024
num_freq_bins=20
num_time_bins=20


# domain_window_features = get_mfcc_window_features(FFT_SIZE, num_freq_bins,␣
 ↪num_time_bins, data)
domain_window_features = get_window_features(FFT_SIZE, num_freq_bins,␣
 ↪num_time_bins, data)


scaler = preprocessing.StandardScaler()
domain_window_features = scaler.fit_transform(domain_window_features)


xtrain, xtest, ytrain, ytest = train_test_split(domain_window_features, tags,␣
 ↪test_size=0.2, random_state=100)
clf = RandomForestClassifier(random_state=100)
# clf = SVC()
clf.fit(xtrain, ytrain)
ypred = clf.predict(xtrain)

scores = cross_val_score(clf, xtrain, ytrain, cv=10)
print('Average Cross Validation Score from Training:', scores.mean(), sep='\n',␣
 ↪end='\n\n\n')

#testing the model
ypred = clf.predict(xtest)

cm = confusion_matrix(ytest, ypred)
cr = classification_report(ytest, ypred)

# print('Confusion Matrix:', cm, sep='\n', end='\n\n\n')
# print('Test Statistics:', cr, sep='\n', end='\n\n\n')

print('Testing Accuracy:', accuracy_score(ytest, ypred))
# names = ["Alarm", "Blender", "Clean", "Microwave", "Music", "Silence"]
# cm_display = ConfusionMatrixDisplay(cm, display_labels=names)
# cm_display.plot()
# plt.savefig("Pictures/feature_selection.png")
# plt.show()

joblib.dump(clf, "Model/feature_selection_model.joblib")
```

```
Average Cross Validation Score from Training:
```

```
0.9888888888888889


Testing Accuracy: 1.0
```

[8]:
```
['Model/feature_selection_model.joblib']
```

[ ]:

[ ]: