audio_classification

September 29, 2020

0.1 Imports

[1]:

```
import os
import matplotlib.pyplot as plt
import seaborn as sn
import pandas as pd
import numpy as np
import librosa
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils
from keras.models import Sequential
from keras import layers
from sklearn.metrics import confusion_matrix, precision_score, recall_score,␣
 ↪f1_score
```

0.2 Helper Methods

[2]:

```
def get_files(extension, path=os.getcwd()):
    """returns files in folder with given extension and path"""
    onlyfiles = [f for f in os.listdir(path) if os.path.isfile(os.path.
 ↪join(path, f))]
    required_ones = [f for f in onlyfiles if f.lower().endswith("." + extension.
 ↪lower())]
    return required_ones

def get_audio_features(filepath, n_mfcc=20):
    y, sr = librosa.load(filepath, sr=None)
    features = librosa.feature.mfcc(y=y, n_mfcc=n_mfcc)
    # THEY ARE GETTING THE MEAN, but somtimes they transpose the array before␣
 ↪that IDK WHY
    # BUT MAYBE IT MAKE SENSE TO TAKE THE MEAN OF THE SAME FEATURE OF SAME␣
 ↪SAMPLE?
    meaned_features = np.mean(features.T,axis=0)
    #pirnt(meaned_features.shape)
    return meaned_features
```

0.2.1 Preparing data

[3]:
```
## Data paths
train_data_path = "data\\train\\"
test_data_path = "data\\test\\"
target_file_path = "targets.csv"
```

[4]:
```
# Read the data
target_df = pd.read_csv(target_file_path)
target_df.head()
```

[4]:
```
            file_name           target
0  1-101404-A-34.wav      can_opening
1  1-103999-A-30.wav  door_wood_knock
2  1-104089-A-22.wav         clapping
3  1-105224-A-22.wav         clapping
4   1-110389-A-0.wav              dog
```

[5]:
```
# get the files paths
files_in_train = get_files('wav', path=train_data_path)
files_in_test = get_files('wav', path=test_data_path)
# filter the data
train_df = target_df[target_df['file_name'].isin(files_in_train)]
test_df = target_df[target_df['file_name'].isin(files_in_test)]
```

[6]:
```
#GET THE FILE PATH ON THE DATAFRAMe
train_df['file_path'] = train_data_path + train_df['file_name']
test_df['file_path'] = test_data_path + test_df['file_name']
```

```
<ipython-input-6-45fee7f49d38>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['file_path'] = train_data_path + train_df['file_name']
<ipython-input-6-45fee7f49d38>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['file_path'] = test_data_path + test_df['file_name']
```

[7]:
```
train_df.head()
```

[7]:
```
            file_name           target                         file_path
0  1-101404-A-34.wav      can_opening  data\train\1-101404-A-34.wav
1  1-103999-A-30.wav  door_wood_knock  data\train\1-103999-A-30.wav
2  1-104089-A-22.wav         clapping  data\train\1-104089-A-22.wav
3  1-105224-A-22.wav         clapping  data\train\1-105224-A-22.wav
4   1-110389-A-0.wav              dog   data\train\1-110389-A-0.wav
```

[8]:
```
# GET THE FEATURES ON DATA FRAME
train_df['features']= train_df['file_path'].apply(lambda x:␣
 ↪get_audio_features(x,n_mfcc=50))
test_df['features']= test_df['file_path'].apply(lambda x:␣
 ↪get_audio_features(x,n_mfcc=50))
# # SAVE IT? CHECKPOINT
train_df.to_csv('train_feauters.csv')
test_df.to_csv('test_feauters.csv')
train_df.head()
```

```
<ipython-input-8-5cf3a435bd59>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train_df['features']= train_df['file_path'].apply(lambda x:
get_audio_features(x,n_mfcc=50))
<ipython-input-8-5cf3a435bd59>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['features']= test_df['file_path'].apply(lambda x:
get_audio_features(x,n_mfcc=50))
```

[8]:
```
            file_name           target                         file_path  \
0  1-101404-A-34.wav      can_opening  data\train\1-101404-A-34.wav
1  1-103999-A-30.wav  door_wood_knock  data\train\1-103999-A-30.wav
2  1-104089-A-22.wav         clapping  data\train\1-104089-A-22.wav
3  1-105224-A-22.wav         clapping  data\train\1-105224-A-22.wav
4   1-110389-A-0.wav              dog   data\train\1-110389-A-0.wav

                                            features
0  [-496.36215, 10.321696, 13.323994, 16.677523, …
1  [-649.1613, 39.419155, -2.5168386, 3.928489, 6…
2  [-88.55881, 11.2822, -68.96386, -25.667105, -2…
3  [-10.424058, 49.623047, -96.72966, -13.436092,…
4  [-611.451, 8.705131, -5.6718044, -1.5926154, 0…
```

[9]:

```
# PREPARING DATA TO TRAIN
## Conveting to list of lists
train_x = np.array(train_df.features.tolist())
train_y = np.array(train_df.target.tolist())
########################
test_x = np.array(test_df.features.tolist())
test_y = np.array(test_df.target.tolist())

# CONVERTIN CATEGORIES TO NUMBERS
lb = LabelEncoder()
train_y_numbers = lb.fit_transform(train_y)

# MAKE DICTIONRY FOR LATER USE AND TESTING
labeling_dict_number_to_label = {train_y_numbers[i]:train_y[i] for i in␣
 ↪range(0,len(train_y_numbers))}
labeling_dict_label_to_number = {train_y[i]:train_y_numbers[i] for i in␣
 ↪range(0,len(train_y_numbers))}
print(labeling_dict_label_to_number)
test_y_numbers = [labeling_dict_label_to_number[x] for x in test_y]

#THEN MAKE ONE HOT ENCODING [1,0,0,0,0....]
train_y = np_utils.to_categorical(train_y_numbers)
test_y = np_utils.to_categorical(test_y_numbers)
print(train_x.shape)
print(train_y.shape)
```

```
{'can_opening': 3, 'door_wood_knock': 20, 'clapping': 9, 'dog': 18, 'fireworks':
23, 'mouse_click': 33, 'train': 45, 'wind': 49, 'footsteps': 24, 'frog': 25,
'water_drops': 48, 'brushing_teeth': 2, 'helicopter': 28, 'drinking_sipping':
21, 'rain': 36, 'laughing': 32, 'insects': 30, 'vacuum_cleaner': 46, 'chainsaw':
6, 'toilet_flush': 44, 'washing_machine': 47, 'car_horn': 4, 'thunderstorm': 43,
'pig': 34, 'rooster': 37, 'snoring': 42, 'breathing': 1, 'coughing': 12,
'siren': 40, 'cat': 5, 'clock_alarm': 10, 'airplane': 0, 'chirping_birds': 7,
'crow': 16, 'sea_waves': 38, 'crackling_fire': 14, 'sneezing': 41,
'church_bells': 8, 'clock_tick': 11, 'sheep': 39, 'pouring_water': 35, 'engine':
22, 'door_wood_creaks': 19, 'cow': 13, 'crickets': 15, 'crying_baby': 17,
'keyboard_typing': 31, 'hen': 29, 'hand_saw': 27, 'glass_breaking': 26}
(700, 50)
(700, 50)
```

[10]:

```
# Preparing the nueral netowrk
model = Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(train_x.shape[1],)))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(50, activation='softmax'))
model.
 ↪compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

[11]:

```
# Start Training
model.fit(train_x, train_y, batch_size=32, epochs=100, verbose=1)
```

```
Epoch 1/100
22/22 [==============================] - 0s 2ms/step - loss: 10.3432 - accuracy:
0.0457
Epoch 2/100
22/22 [==============================] - 0s 3ms/step - loss: 3.9453 - accuracy:
0.0871
Epoch 3/100
22/22 [==============================] - 0s 2ms/step - loss: 3.4171 - accuracy:
0.1286
Epoch 4/100
22/22 [==============================] - 0s 2ms/step - loss: 3.1970 - accuracy:
0.1971
Epoch 5/100
22/22 [==============================] - 0s 2ms/step - loss: 3.0641 - accuracy:
0.2071
Epoch 6/100
22/22 [==============================] - 0s 2ms/step - loss: 2.8830 - accuracy:
0.2614
Epoch 7/100
22/22 [==============================] - 0s 2ms/step - loss: 2.7383 - accuracy:
0.3071
Epoch 8/100
22/22 [==============================] - 0s 2ms/step - loss: 2.5253 - accuracy:
0.3343
Epoch 9/100
22/22 [==============================] - 0s 2ms/step - loss: 2.3490 - accuracy:
0.3914
Epoch 10/100
22/22 [==============================] - 0s 2ms/step - loss: 2.2253 - accuracy:
0.4286
Epoch 11/100
22/22 [==============================] - 0s 2ms/step - loss: 2.0829 - accuracy:
0.4614
Epoch 12/100
22/22 [==============================] - 0s 2ms/step - loss: 1.9194 - accuracy:
0.5029
Epoch 13/100
22/22 [==============================] - 0s 2ms/step - loss: 1.8291 - accuracy:
0.5143
Epoch 14/100
22/22 [==============================] - 0s 2ms/step - loss: 1.7729 - accuracy:
```

```
0.5143
Epoch 15/100
22/22 [==============================] - 0s 2ms/step - loss: 1.6211 - accuracy:
0.5614
Epoch 16/100
22/22 [==============================] - 0s 2ms/step - loss: 1.4559 - accuracy:
0.5986
Epoch 17/100
22/22 [==============================] - 0s 2ms/step - loss: 1.4320 - accuracy:
0.5943
Epoch 18/100
22/22 [==============================] - 0s 2ms/step - loss: 1.4526 - accuracy:
0.6300
Epoch 19/100
22/22 [==============================] - 0s 2ms/step - loss: 1.2450 - accuracy:
0.6500
Epoch 20/100
22/22 [==============================] - 0s 2ms/step - loss: 1.1870 - accuracy:
0.6829
Epoch 21/100
22/22 [==============================] - 0s 2ms/step - loss: 1.2067 - accuracy:
0.6571
Epoch 22/100
22/22 [==============================] - 0s 2ms/step - loss: 1.1179 - accuracy:
0.6900
Epoch 23/100
22/22 [==============================] - 0s 2ms/step - loss: 1.0116 - accuracy:
0.7086
Epoch 24/100
22/22 [==============================] - 0s 2ms/step - loss: 0.9473 - accuracy:
0.7229
Epoch 25/100
22/22 [==============================] - 0s 2ms/step - loss: 0.8255 - accuracy:
0.7757
Epoch 26/100
22/22 [==============================] - 0s 2ms/step - loss: 0.8158 - accuracy:
0.7814
Epoch 27/100
22/22 [==============================] - 0s 2ms/step - loss: 0.7757 - accuracy:
0.7900
Epoch 28/100
22/22 [==============================] - 0s 3ms/step - loss: 0.7894 - accuracy:
0.7886
Epoch 29/100
22/22 [==============================] - 0s 2ms/step - loss: 0.7280 - accuracy:
0.7986
Epoch 30/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6812 - accuracy:
```

```
0.8029
Epoch 31/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6890 - accuracy:
0.8014
Epoch 32/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6722 - accuracy:
0.8100
Epoch 33/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6034 - accuracy:
0.8300
Epoch 34/100
22/22 [==============================] - 0s 3ms/step - loss: 0.5662 - accuracy:
0.8386
Epoch 35/100
22/22 [==============================] - 0s 3ms/step - loss: 0.5935 - accuracy:
0.8286
Epoch 36/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5626 - accuracy:
0.8271
Epoch 37/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5750 - accuracy:
0.8414
Epoch 38/100
22/22 [==============================] - 0s 3ms/step - loss: 0.7517 - accuracy:
0.8114
Epoch 39/100
22/22 [==============================] - 0s 4ms/step - loss: 0.6685 - accuracy:
0.8029
Epoch 40/100
22/22 [==============================] - 0s 3ms/step - loss: 0.5375 - accuracy:
0.8457
Epoch 41/100
22/22 [==============================] - 0s 3ms/step - loss: 0.5033 - accuracy:
0.8543
Epoch 42/100
22/22 [==============================] - 0s 3ms/step - loss: 0.4525 - accuracy:
0.8714
Epoch 43/100
22/22 [==============================] - 0s 3ms/step - loss: 0.3948 - accuracy:
0.8857
Epoch 44/100
22/22 [==============================] - 0s 3ms/step - loss: 0.4313 - accuracy:
0.8771
Epoch 45/100
22/22 [==============================] - 0s 3ms/step - loss: 0.3837 - accuracy:
0.8929
Epoch 46/100
22/22 [==============================] - 0s 3ms/step - loss: 0.3811 - accuracy:
```

```
0.8914
Epoch 47/100
22/22 [==============================] - 0s 3ms/step - loss: 0.3524 - accuracy:
0.8943
Epoch 48/100
22/22 [==============================] - 0s 3ms/step - loss: 0.3086 - accuracy:
0.9100
Epoch 49/100
22/22 [==============================] - 0s 3ms/step - loss: 0.2893 - accuracy:
0.9200
Epoch 50/100
22/22 [==============================] - 0s 3ms/step - loss: 0.2810 - accuracy:
0.9243
Epoch 51/100
22/22 [==============================] - 0s 3ms/step - loss: 0.2740 - accuracy:
0.9243
Epoch 52/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2797 - accuracy:
0.9214
Epoch 53/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3124 - accuracy:
0.9100
Epoch 54/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3213 - accuracy:
0.9129
Epoch 55/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2705 - accuracy:
0.9300
Epoch 56/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2499 - accuracy:
0.9257
Epoch 57/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2428 - accuracy:
0.9357
Epoch 58/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2126 - accuracy:
0.9443
Epoch 59/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1995 - accuracy:
0.9443
Epoch 60/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1828 - accuracy:
0.9486
Epoch 61/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2294 - accuracy:
0.9343
Epoch 62/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1984 - accuracy:
```

```
0.9429
Epoch 63/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2058 - accuracy:
0.9414
Epoch 64/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2027 - accuracy:
0.9343
Epoch 65/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1772 - accuracy:
0.9471
Epoch 66/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1581 - accuracy:
0.9543
Epoch 67/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1617 - accuracy:
0.9543
Epoch 68/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1476 - accuracy:
0.9557
Epoch 69/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1533 - accuracy:
0.9543
Epoch 70/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1545 - accuracy:
0.9571
Epoch 71/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1598 - accuracy:
0.9514
Epoch 72/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1576 - accuracy:
0.9500
Epoch 73/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1468 - accuracy:
0.9600
Epoch 74/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1295 - accuracy:
0.9657
Epoch 75/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1825 - accuracy:
0.9486
Epoch 76/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1276 - accuracy:
0.9629
Epoch 77/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1533 - accuracy:
0.9557
Epoch 78/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1550 - accuracy:
```

```
0.9557
Epoch 79/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1357 - accuracy:
0.9614
Epoch 80/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1079 - accuracy:
0.9686
Epoch 81/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1029 - accuracy:
0.9743
Epoch 82/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1039 - accuracy:
0.9657
Epoch 83/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1117 - accuracy:
0.9657
Epoch 84/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1312 - accuracy:
0.9614
Epoch 85/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1185 - accuracy:
0.9700
Epoch 86/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1158 - accuracy:
0.9629
Epoch 87/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1176 - accuracy:
0.9714
Epoch 88/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1456 - accuracy:
0.9529
Epoch 89/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1997 - accuracy:
0.9443
Epoch 90/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3334 - accuracy:
0.8886
Epoch 91/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6485 - accuracy:
0.8214
Epoch 92/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5950 - accuracy:
0.8557
Epoch 93/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4853 - accuracy:
0.8929
Epoch 94/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3155 - accuracy:
```

```
0.9229
Epoch 95/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2394 - accuracy:
0.9286
Epoch 96/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1383 - accuracy:
0.9686
Epoch 97/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0935 - accuracy:
0.9714
Epoch 98/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1099 - accuracy:
0.9700
Epoch 99/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1142 - accuracy:
0.9671
Epoch 100/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1186 - accuracy:
0.9657
```

```
[11]: <tensorflow.python.keras.callbacks.History at 0x24a0c9a85b0>
```

0.3 Testing the data

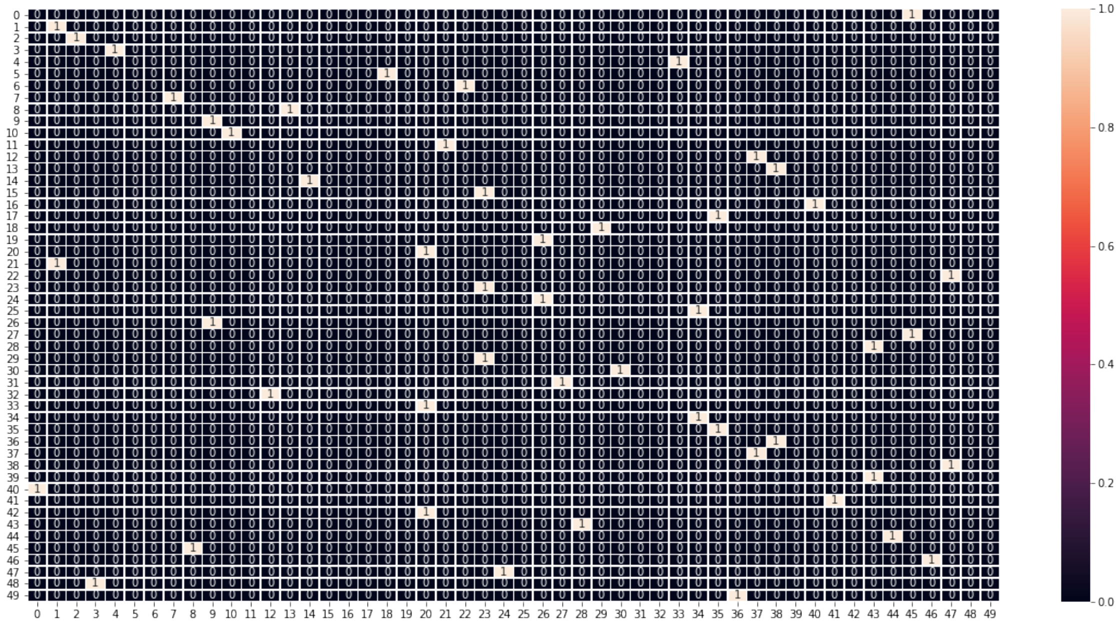
```
[12]: #predicting
pred_test_y = model.predict(test_x)
pred_test_y = pred_test_y.argmax(1)
print('prediction :', pred_test_y)
test_y_decoded = test_y.argmax(1)
print('actual result :',test_y_decoded)
# PREPARING TESTING DATA
score = model.evaluate(test_x, test_y,verbose=1)
print(score)
```

```
prediction : [34 26 34 29 18 12 13 45 14 30 35 35  9 36 23 22  2 47 38 27 41 43
38 23
  1  0 44 10 46 21 40 43 20  1  7 23 28 26 33  8 37 37 47  9 20  4 24 20
 45  3]
actual result : [25 24 34 18  5 32  8  0 14 30 35 17 26 49 23  6  2 22 36 31 41
28 13 15
  1 40 44 10 46 11 16 39 20 21  7 29 43 19  4 45 37 12 38  9 33  3 47 42
 27 48]
2/2 [==============================] - 0s 997us/step - loss: 6.2390 - accuracy:
0.3000
[6.239011287689209, 0.30000001192092896]
```

[13]:
```python
#confusion matrix
df_cm = confusion_matrix(test_y_decoded, pred_test_y)
print(df_cm)
fig, ax = plt.subplots(figsize=(20,10))
#The confusion matrix, which is a breakdown of predictions into a table showing correct predictions
#and the types of incorrect predictions made.Ideally, you will only see numbers in the diagonal,
#which means that all your predictions were correct!
sn.heatmap(df_cm, annot=True, annot_kws={"size": 10},ax=ax, linewidths=.5) # font size
```

```
[[0 0 0 … 0 0 0]
 [0 1 0 … 0 0 0]
 [0 0 1 … 0 0 0]
 …
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]]
```

[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x24a14003c40>
```



[14]:
```python
# other factors
p_s =precision_score(test_y_decoded, pred_test_y,average='micro')
r_s = recall_score(test_y_decoded, pred_test_y,average='micro')
f_s =f1_score(test_y_decoded,pred_test_y,average='micro')
print('precision :', p_s)
print('recall score :', r_s)
print('f1 score :', r_s)
```

```
precision : 0.3
recall score : 0.3
f1 score : 0.3
```

0.4 Production Testing

[15]:

```
# one sample testing
def test(file_path):
    features = np.array([get_audio_features(file_path,n_mfcc=50).tolist()])
    #return features
    prediction = model.predict(features)
    pred_index = prediction.argmax(1)[0]
    result = labeling_dict_number_to_label[pred_index]
    return result
```

[16]:

```
file_path = 'data\\test\\1-46272-A-12.wav'
result = test(file_path)
print(result)
```

```
crackling_fire
```