CIS4930 Assignment 3

May 1, 2023

[116]:
```
import os
import librosa
import random
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix,␣
↪roc_curve, auc, classification_report
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier # multi-layer perceptron model
```

0.0.1 Step 1: Split the dataset into training and test sets

[80]:
```
pathHappy = "data_folder/happy/"
pathFear = "data_folder/fear/"
pathAngry = "data_folder/angry/"
pathSad = "data_folder/sad/"

filesHappy = librosa.util.find_files(pathHappy)
filesFear = librosa.util.find_files(pathFear)
filesAngry = librosa.util.find_files(pathAngry)
filesSad = librosa.util.find_files(pathSad)
```

[81]:
```
filesHappy = np.asarray(filesHappy)
filesFear = np.asarray(filesFear)
filesAngry = np.asarray(filesAngry)
filesSad = np.asarray(filesSad)
```

[82]:
```
targets_h = np.full(100, 1) # array of 1s for happy target
```

```
train_h, test_h, target_h,  t_h= train_test_split(filesHappy, targets_h,␣
 ↪test_size = 0.3, shuffle=True, random_state=42)

targets_f = np.full(100, 2) # array of 2s for fear target

train_f, test_f, target_f, t_f = train_test_split(filesFear, targets_f,␣
 ↪test_size = 0.3, shuffle=True, random_state=42)

targets_a = np.full(100, 3) # array of 3s for angry target

train_a, test_a, target_a, t_a = train_test_split(filesAngry, targets_a,␣
 ↪test_size = 0.3, shuffle=True, random_state=42)

targets_s = np.full(100, 4) # array of 4s for sad target

train_s, test_s, target_s, t_s = train_test_split(filesSad, targets_s,␣
 ↪test_size = 0.3, shuffle=True, random_state=42)

np.info(train_s)
```

```
class:  ndarray
shape:  (70,)
strides:  (372,)
itemsize:  372
aligned:  True
contiguous:  True
fortran:  True
data pointer: 0x1cdf9030400
byteorder:  little
byteswap:  False
type: <U93
```

[83]:
```
train = np.append(train_h, train_f)
train = np.append(train, train_a)
train = np.append(train, train_s)

np.info(train)
```

```
class:  ndarray
shape:  (280,)
strides:  (388,)
itemsize:  388
aligned:  True
contiguous:  True
fortran:  True
data pointer: 0x1cdf02fac10
byteorder:  little
byteswap:  False
```

```
type: <U97
```

[84]:
```
target = np.append(target_h, target_f)
target = np.append(target, target_a)
target = np.append(target, target_s)

np.info(target)
```

```
class:  ndarray
shape:  (280,)
strides:  (4,)
itemsize:  4
aligned:  True
contiguous:  True
fortran:  True
data pointer: 0x1cdf224bd70
byteorder:  little
byteswap:  False
type: int32
```

[85]:
```
test = np.append(test_h, test_f)
test = np.append(test, test_a)
test = np.append(test, test_s)

np.info(test)
```

```
class:  ndarray
shape:  (120,)
strides:  (388,)
itemsize:  388
aligned:  True
contiguous:  True
fortran:  True
data pointer: 0x1cde7e3d190
byteorder:  little
byteswap:  False
type: <U97
```

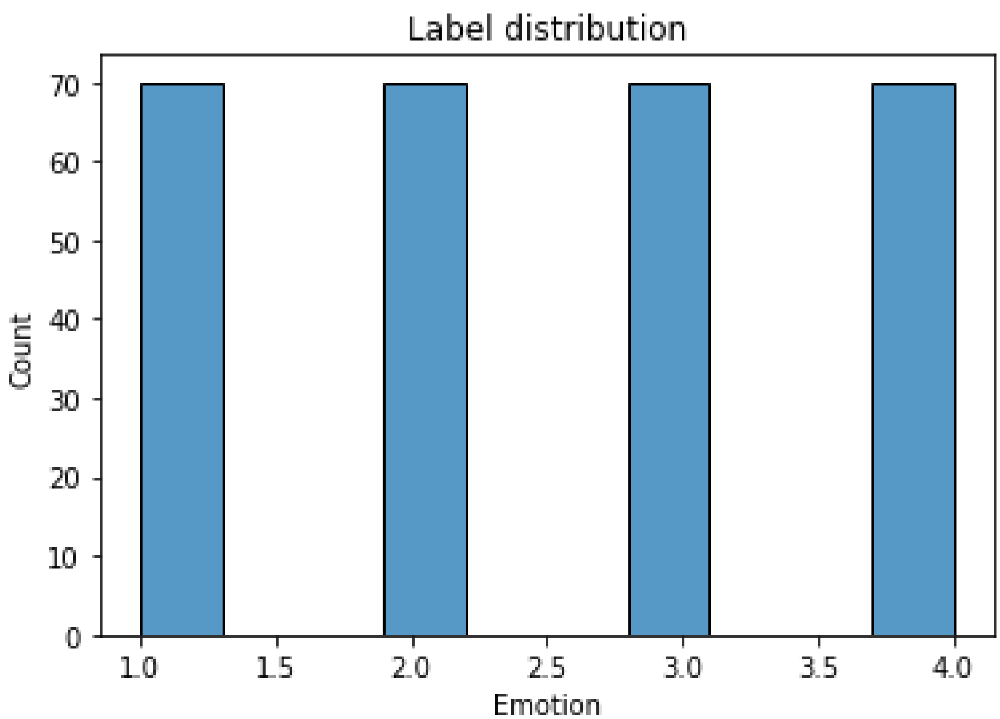
[86]:
```
t = np.append(t_h, t_f)
t = np.append(t, t_a)
t = np.append(t, t_s)

np.info(t)
```

```
class:  ndarray
shape:  (120,)
strides:  (4,)
itemsize:  4
aligned:  True
```

```
contiguous:  True
fortran:  True
data pointer: 0x1cde295e880
byteorder:  little
byteswap:  False
type: int32
```

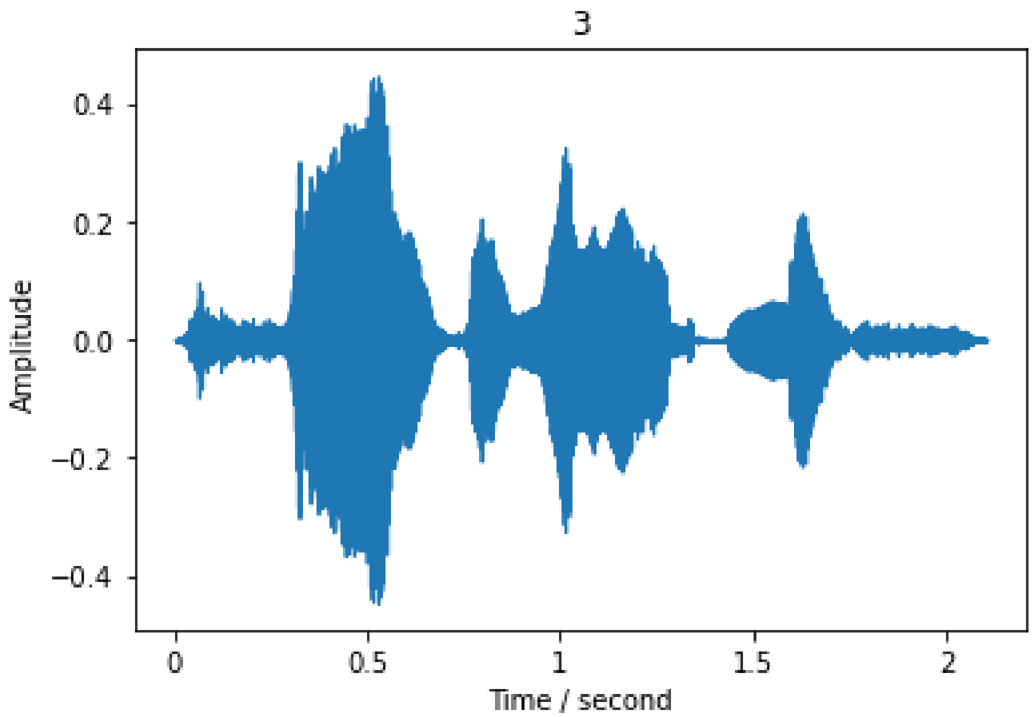
0.0.2 Step 2. Exploratory Data Analysis

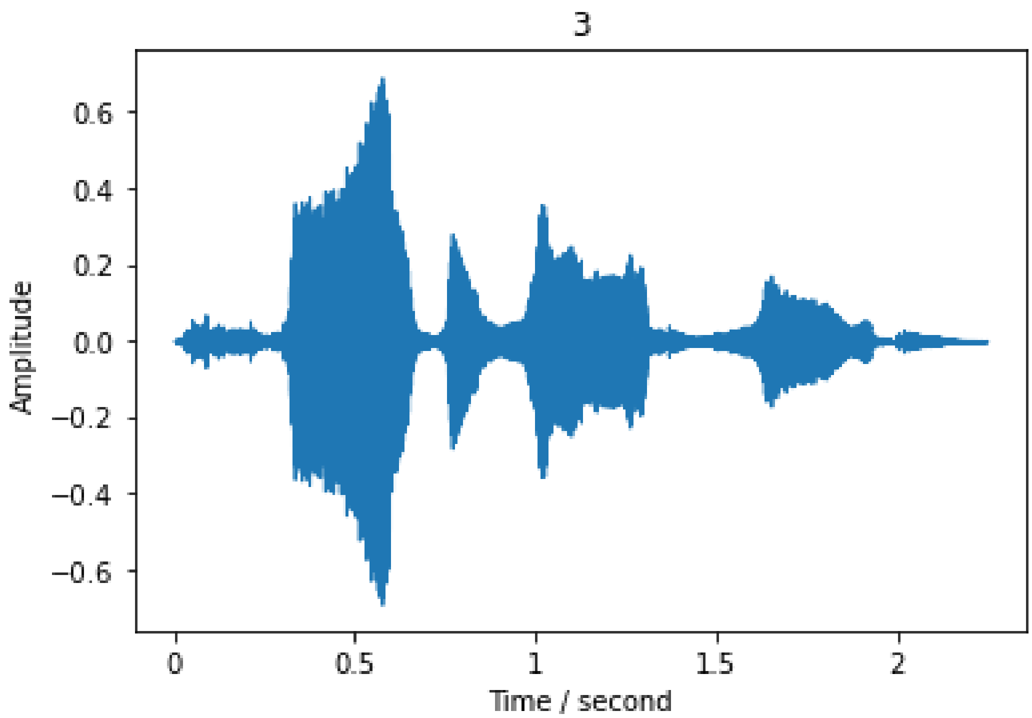
[93]:
```
sns.histplot(data=target, palette='bright')
plt.title("Label distribution")
plt.xlabel("Emotion")
plt.show()
```

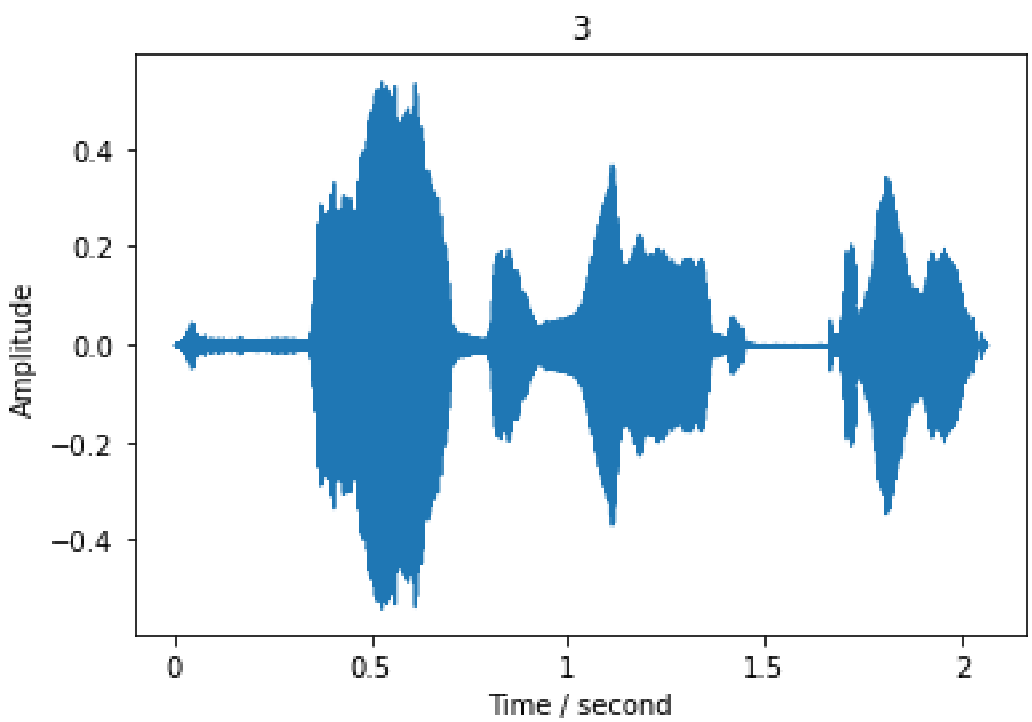


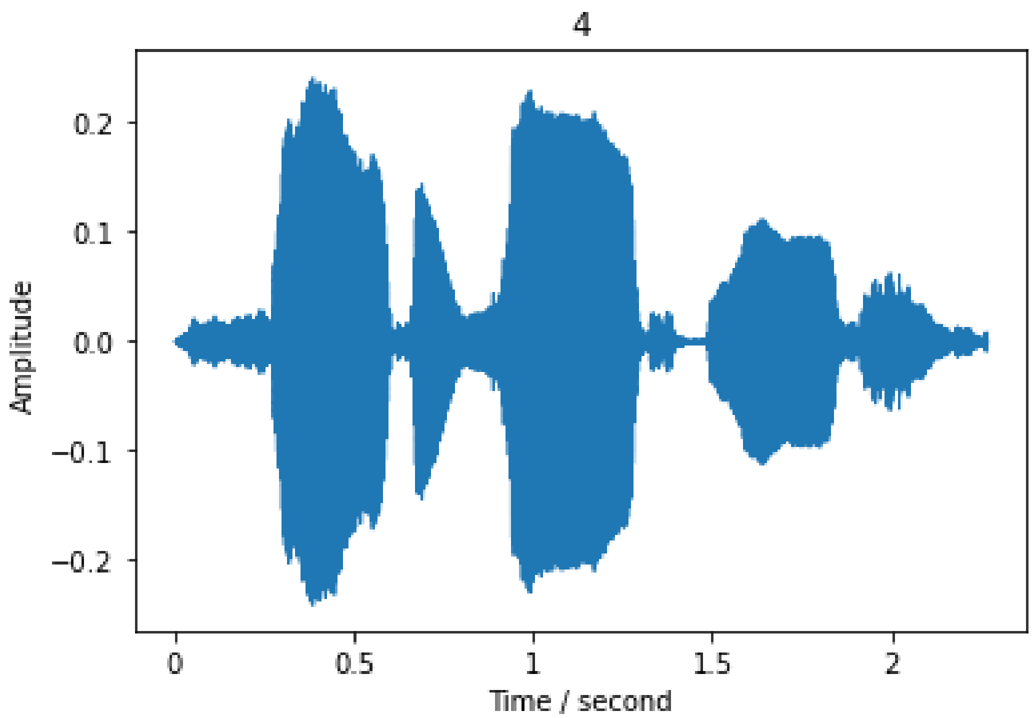
[88]:
```
# sample of 10 files in time domain
rand_nums = random.sample(range(280), 10)
train_rand = np.array([train[i] for i in rand_nums])
t_rand = np.array([target[i] for i in rand_nums])

for i in range(len(train_rand)):
    signal, sample_rate = librosa.load(train_rand[i], sr = 16000,mono = True)
    librosa.display.waveshow(y=signal, sr=sample_rate)
    plt.xlabel('Time / second')
```

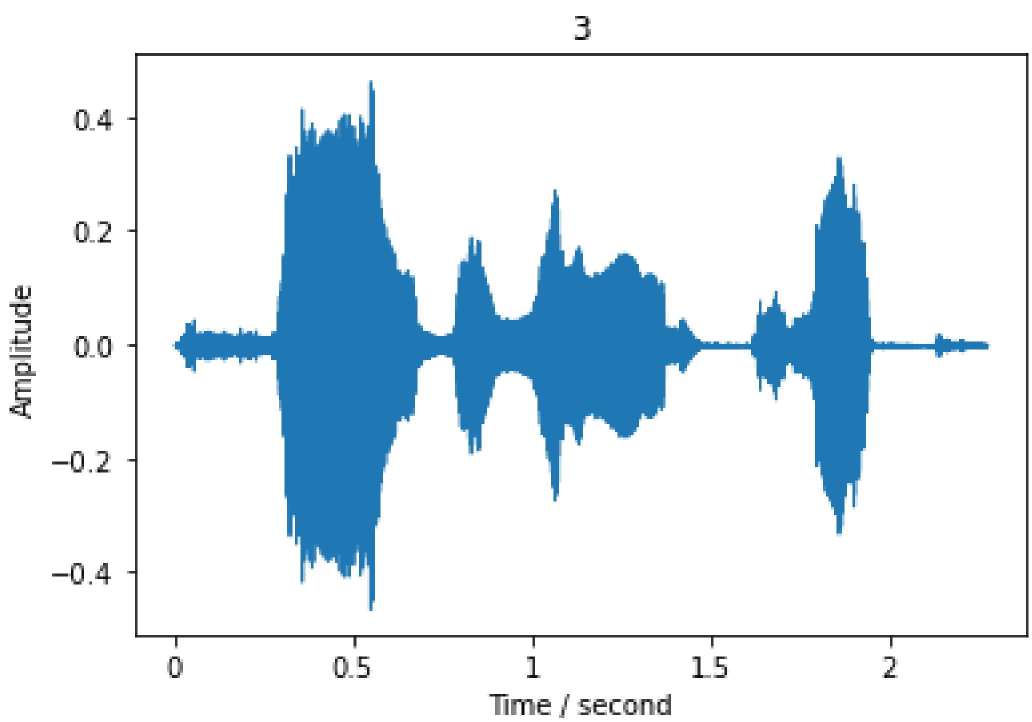
```
    plt.ylabel('Amplitude')
    plt.title(t_rand[i])
    plt.show()
```

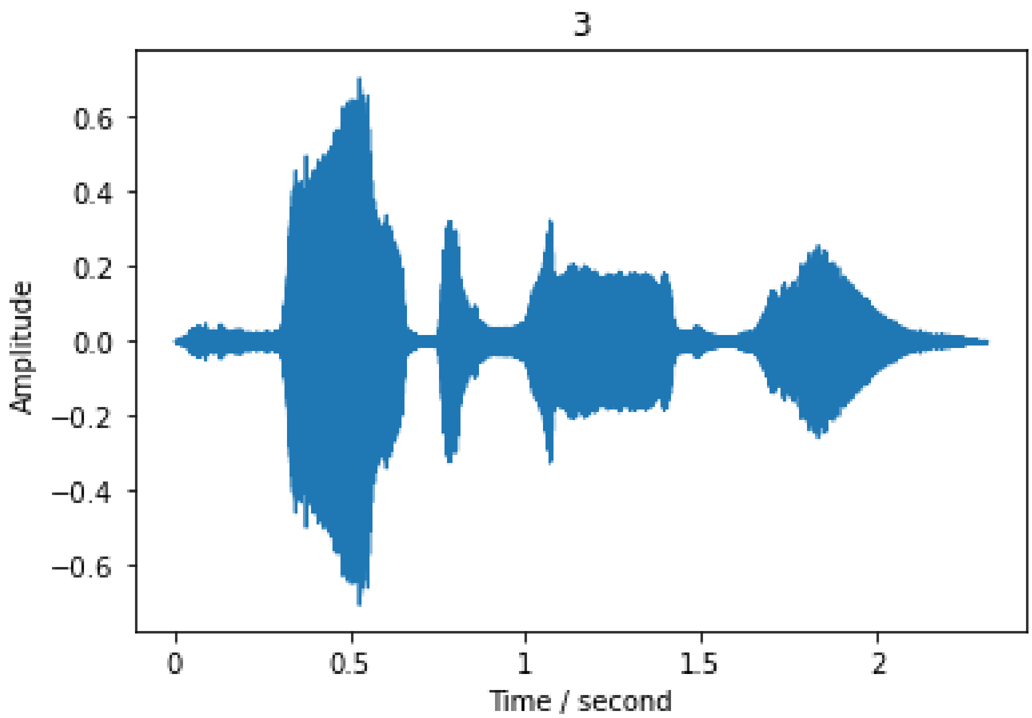


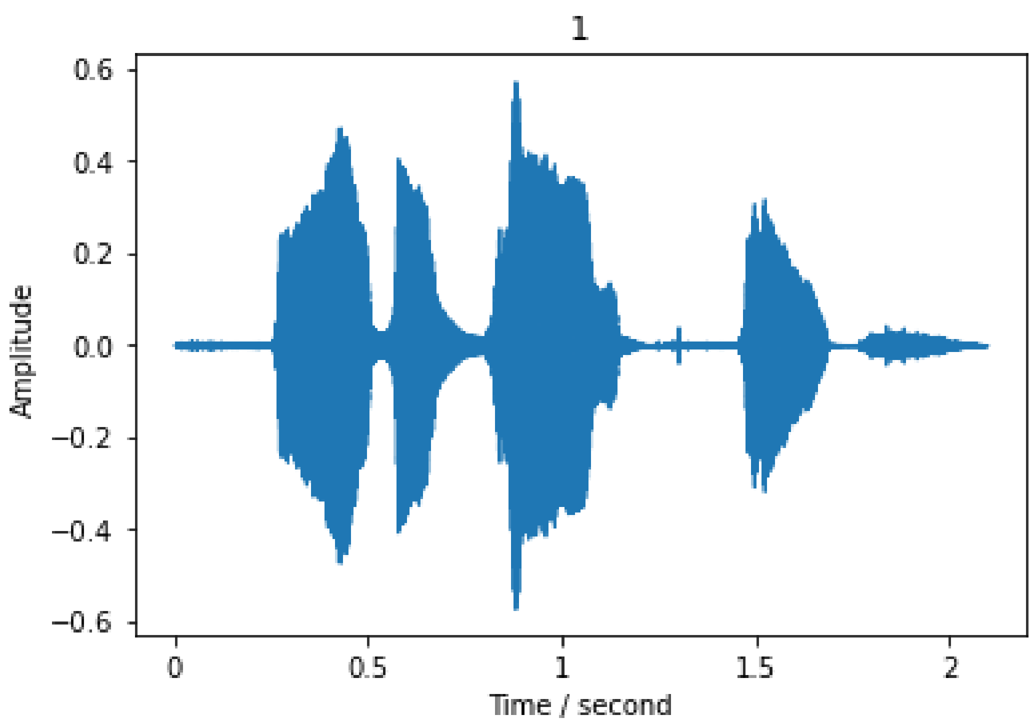

3
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
Amplitude
0
0.5
1
1.5
2
Time / second

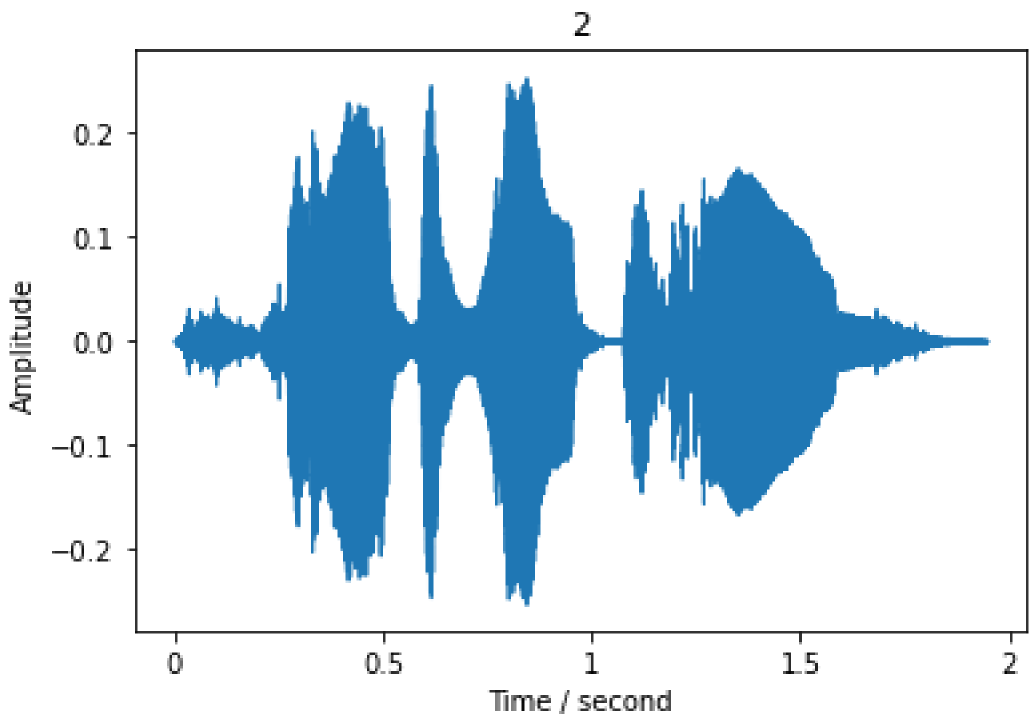

3
0.4
0.2
0.0
−0.2
−0.4
Amplitude
0
0.5
1
1.5
2
Time / second

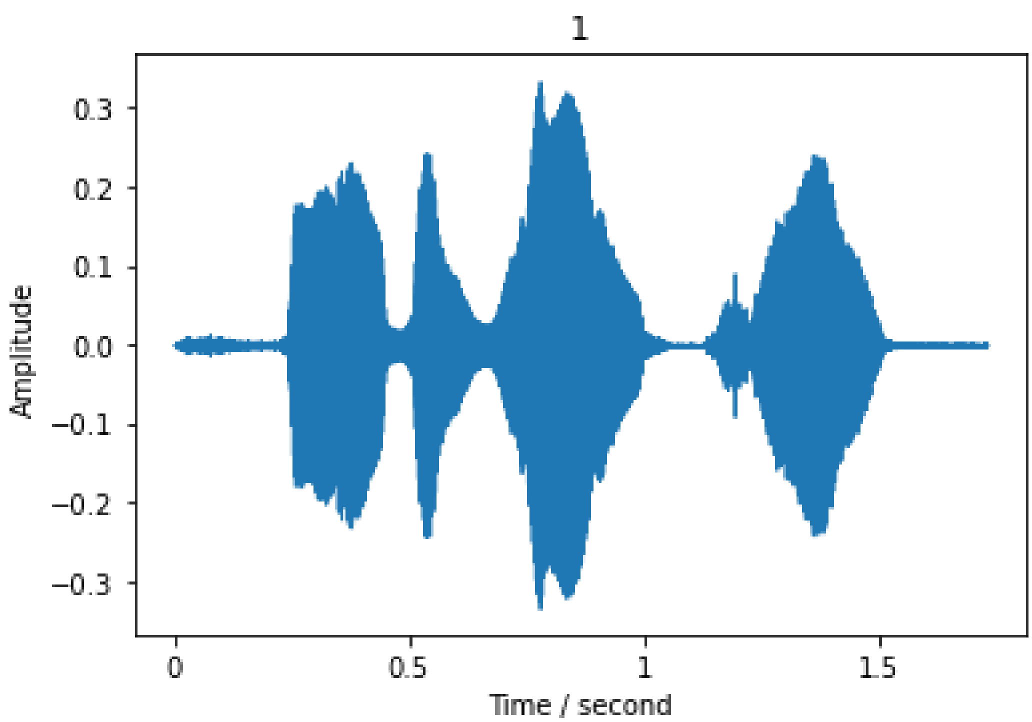

4
0.2
0.1
0.0
−0.1
−0.2
Amplitude
0
0.5
1
1.5
2
Time / second

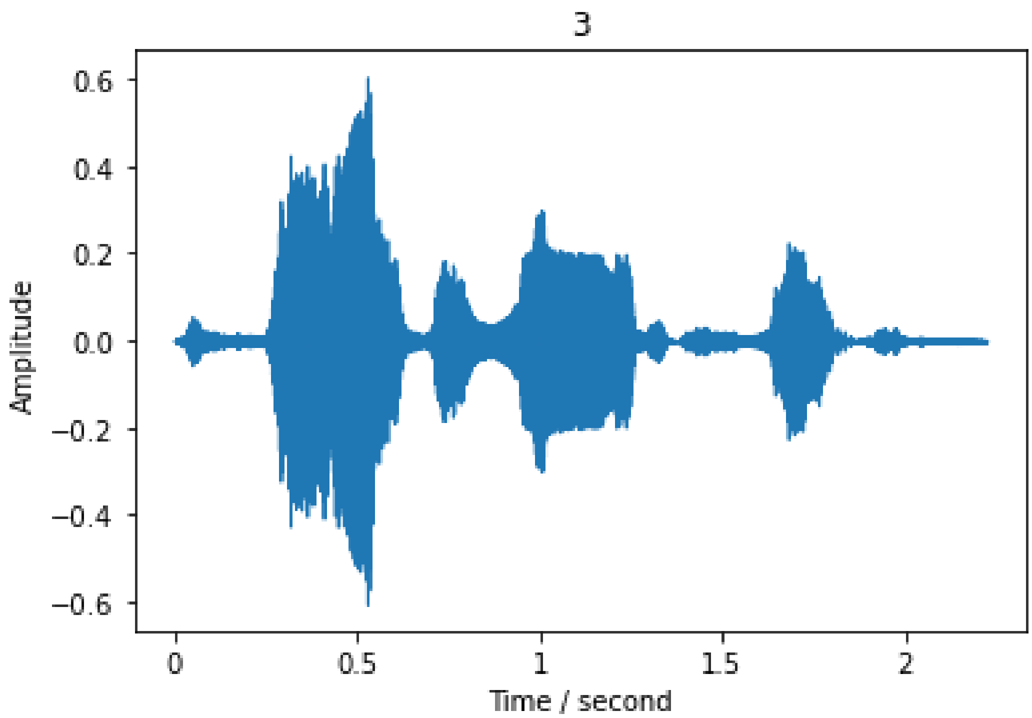

3
0.4
0.2
0.0
−0.2
−0.4
Amplitude
0
0.5
1
1.5
2
Time / second


3
Amplitude
Time / second
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
0.5
1
1.5
2

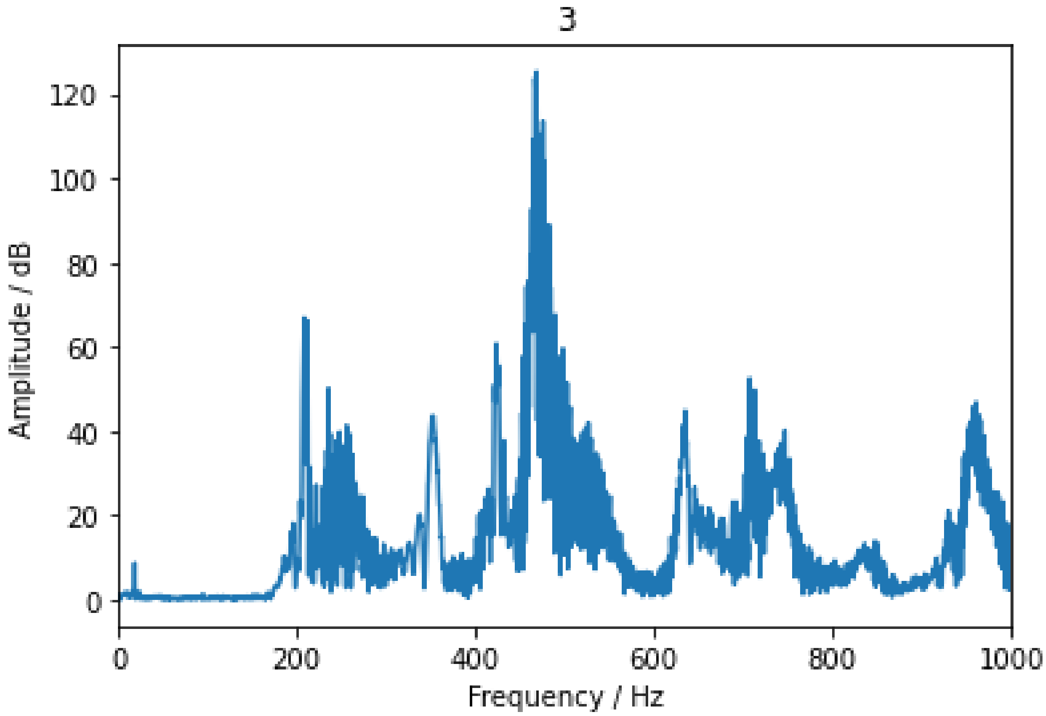

1
Amplitude
Time / second
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
0.5
1
1.5
2

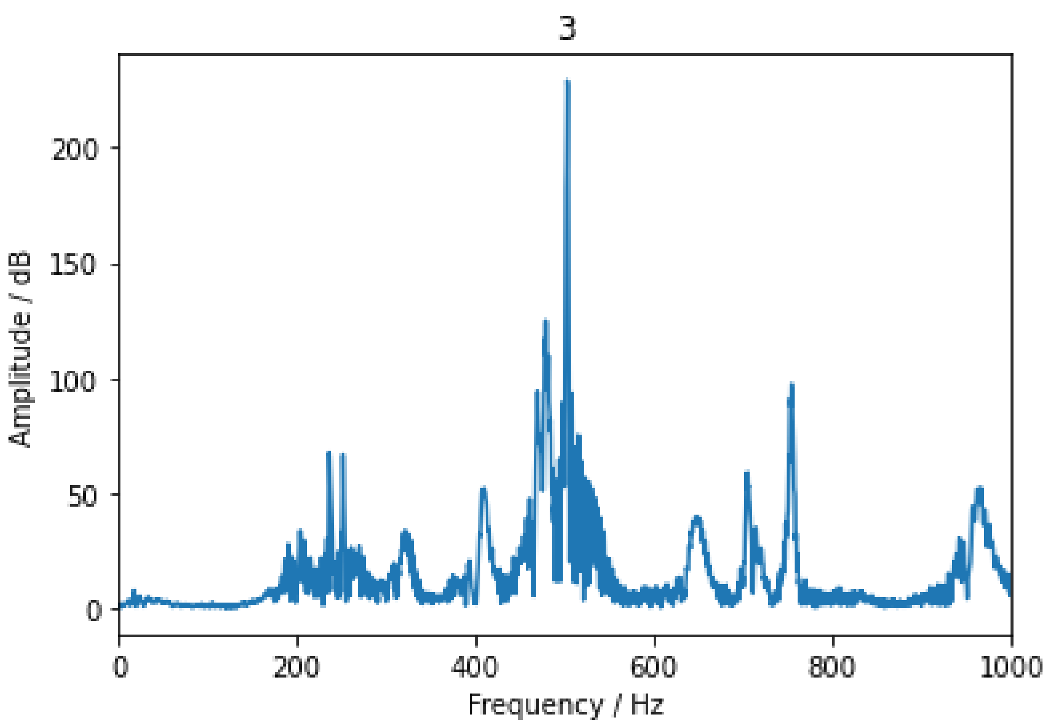

2
Amplitude
0.2
0.1
0.0
−0.1
−0.2
0
0.5
1
1.5
2
Time / second

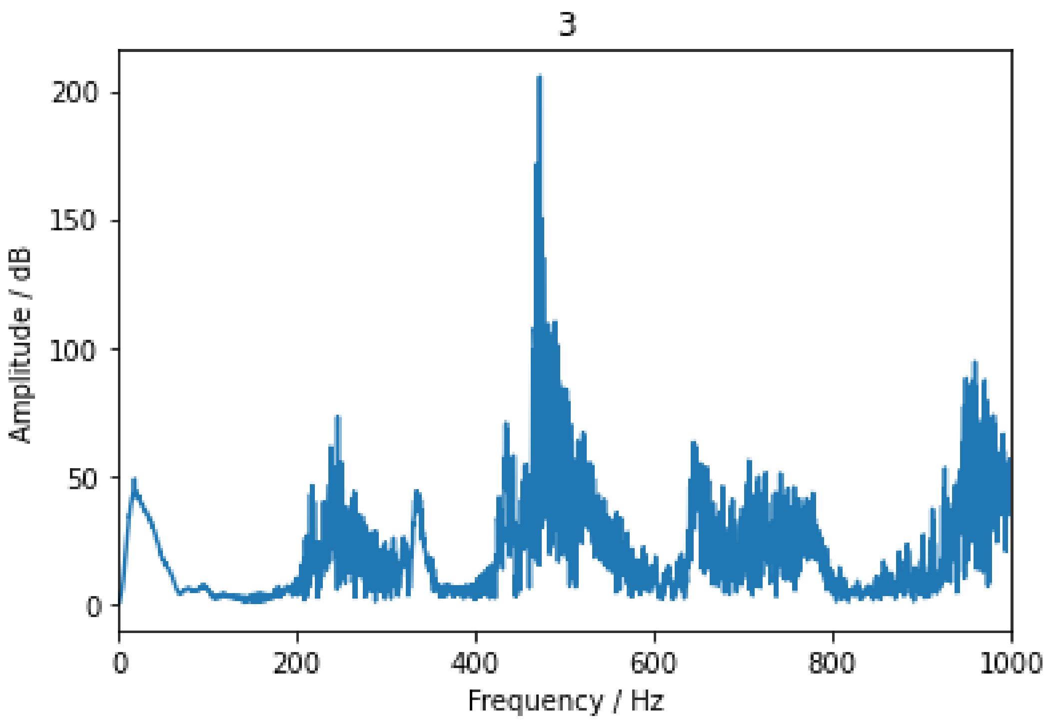

1
Amplitude
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
0.5
1
1.5
Time / second

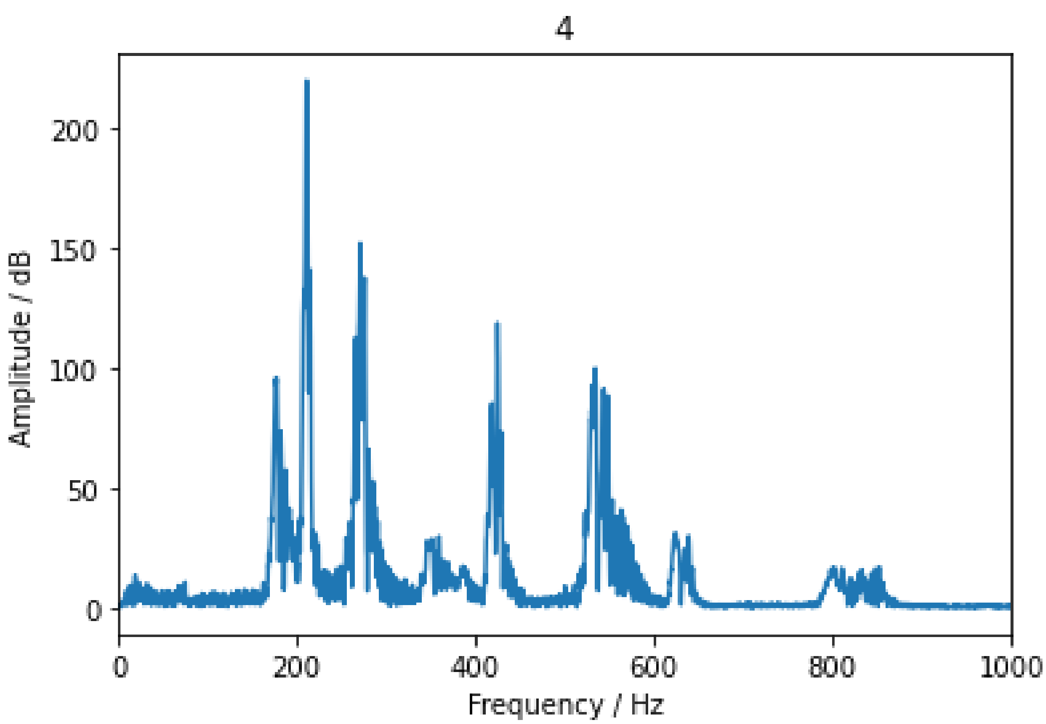


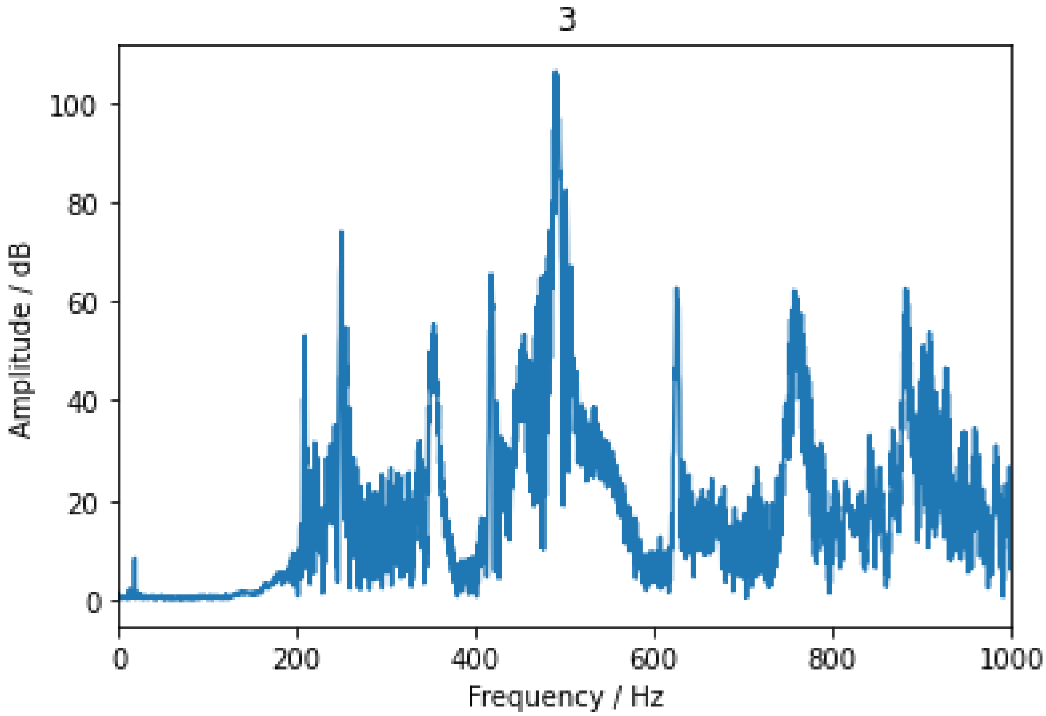
[89]:
```
# sample of 10 files in the frequency domain (some code copied from Coding␣
↪Materials)
for i in range(len(train_rand)):
    signal, sample_rate = librosa.load(train_rand[i], sr = 16000,mono = True)
    k = np.arange(len(signal))
    T = len(signal)/sample_rate
    freq = k/T

    DATA_0 = np.fft.fft(signal)
    abs_DATA_0 = abs(DATA_0)
    plt.figure(2)
    plt.plot(freq, abs_DATA_0)
    plt.title(t_rand[i])
    plt.xlabel("Frequency / Hz")
    plt.ylabel("Amplitude / dB")
    plt.xlim([0, 1000])
    plt.show()
```

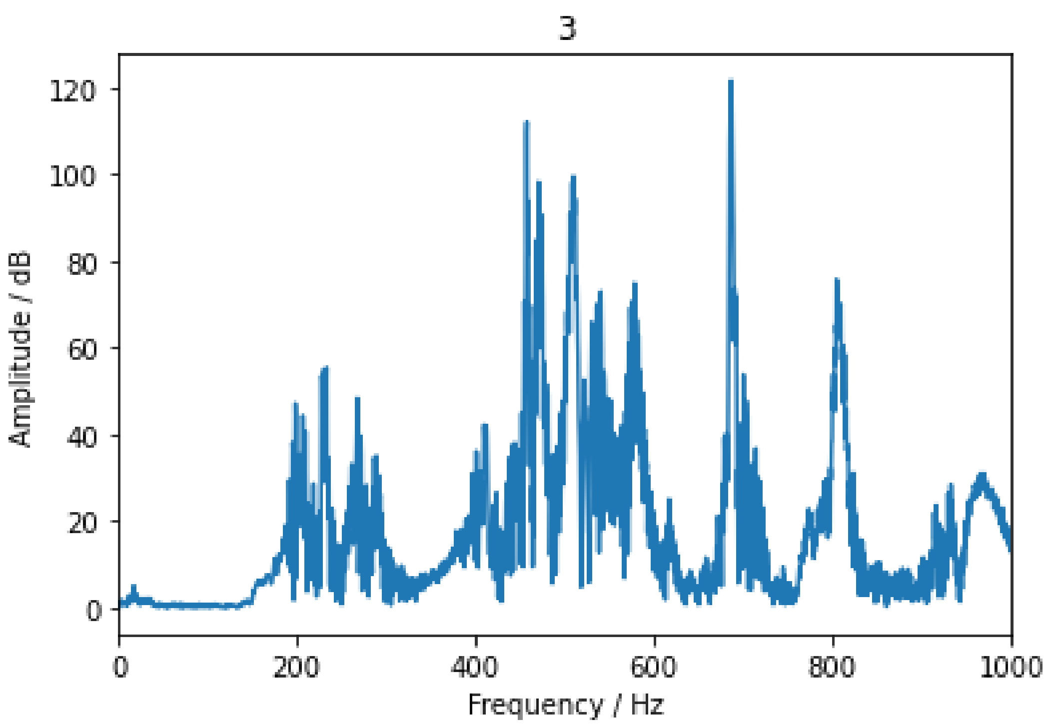

3
Amplitude / dB
Frequency / Hz
0
20
40
60
80
100
120
0
200
400
600
800
1000

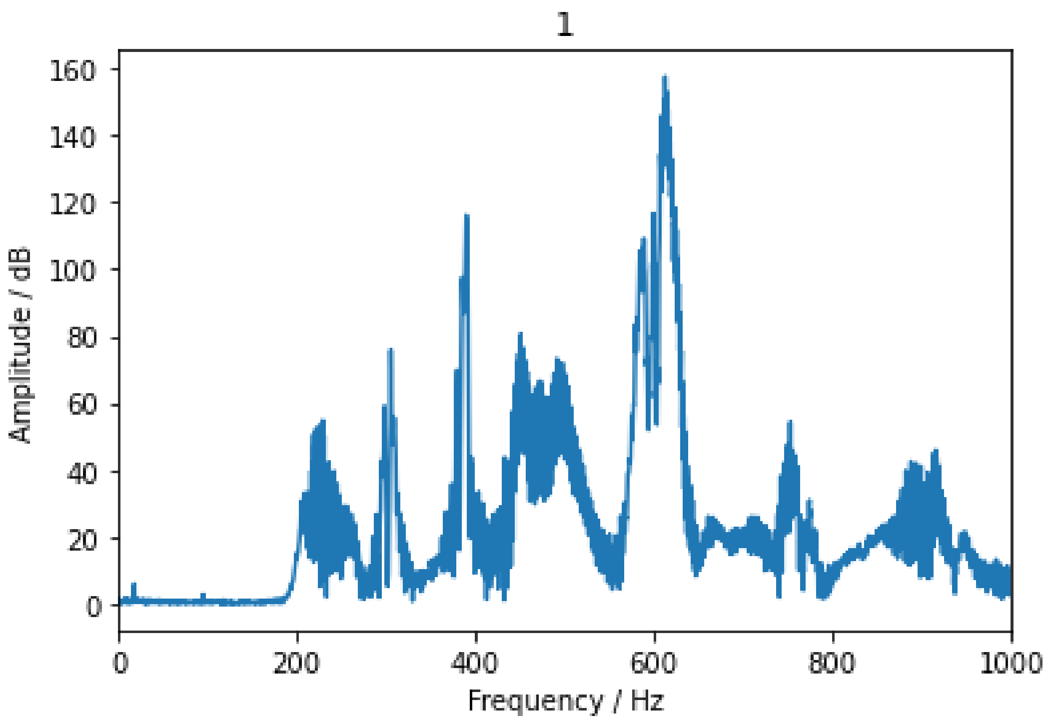

3
Amplitude / dB
Frequency / Hz
0
50
100
150
200
0
200
400
600
800
1000

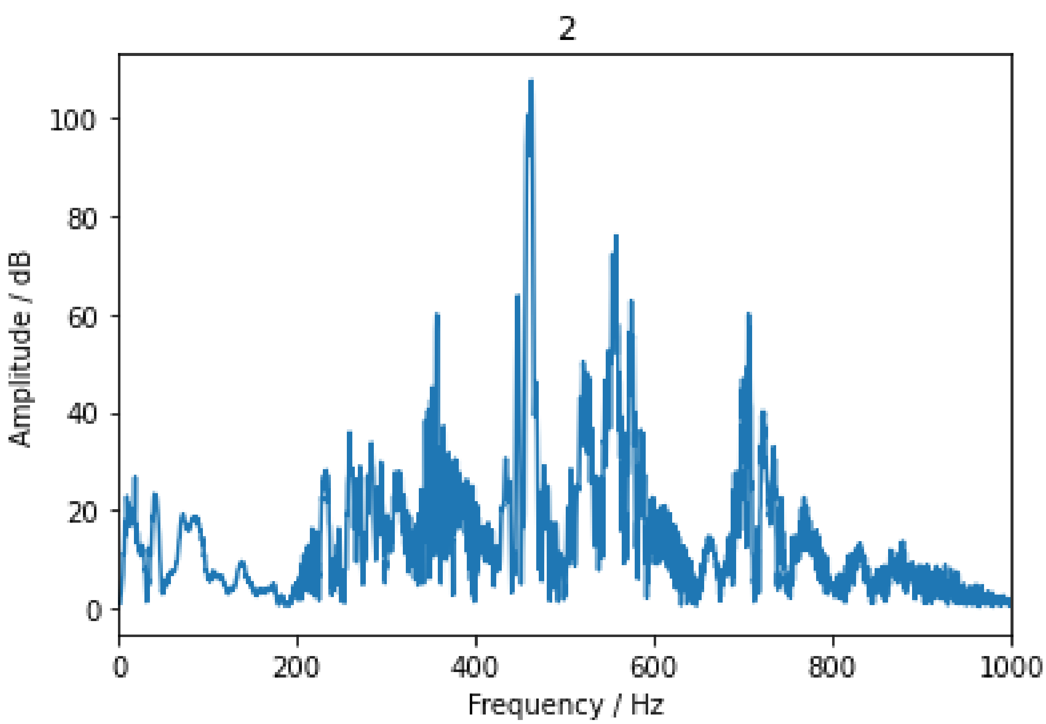

3
Amplitude / dB
Frequency / Hz
0
50
100
150
200
0
200
400
600
800
1000

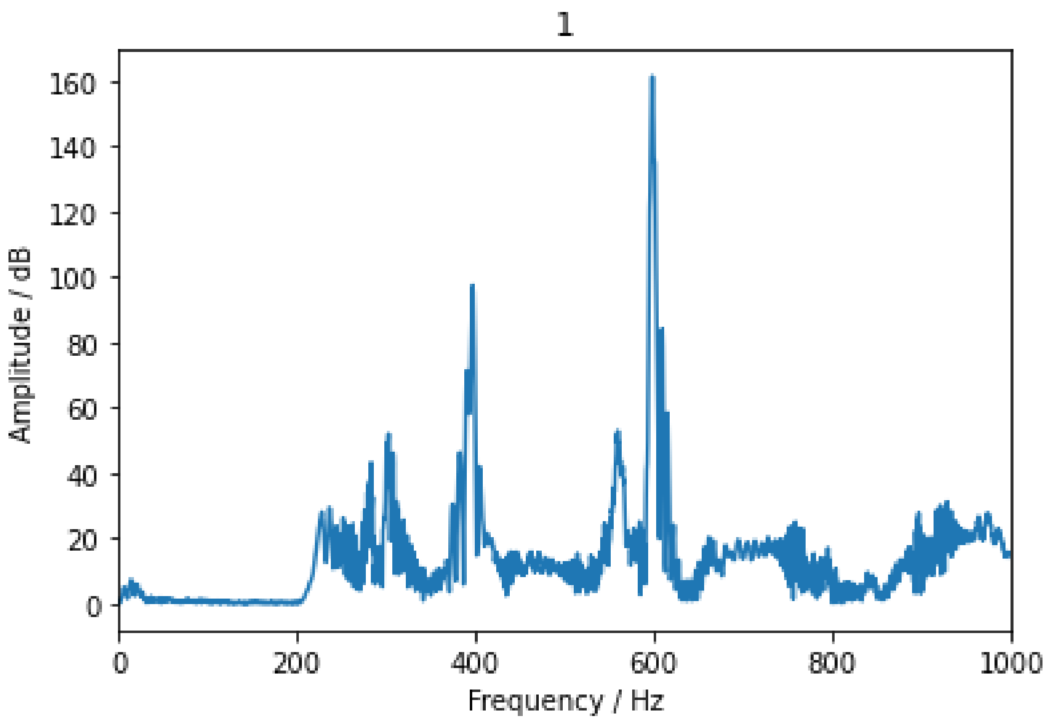

4
Amplitude / dB
Frequency / Hz
0
50
100
150
200
0
200
400
600
800
1000

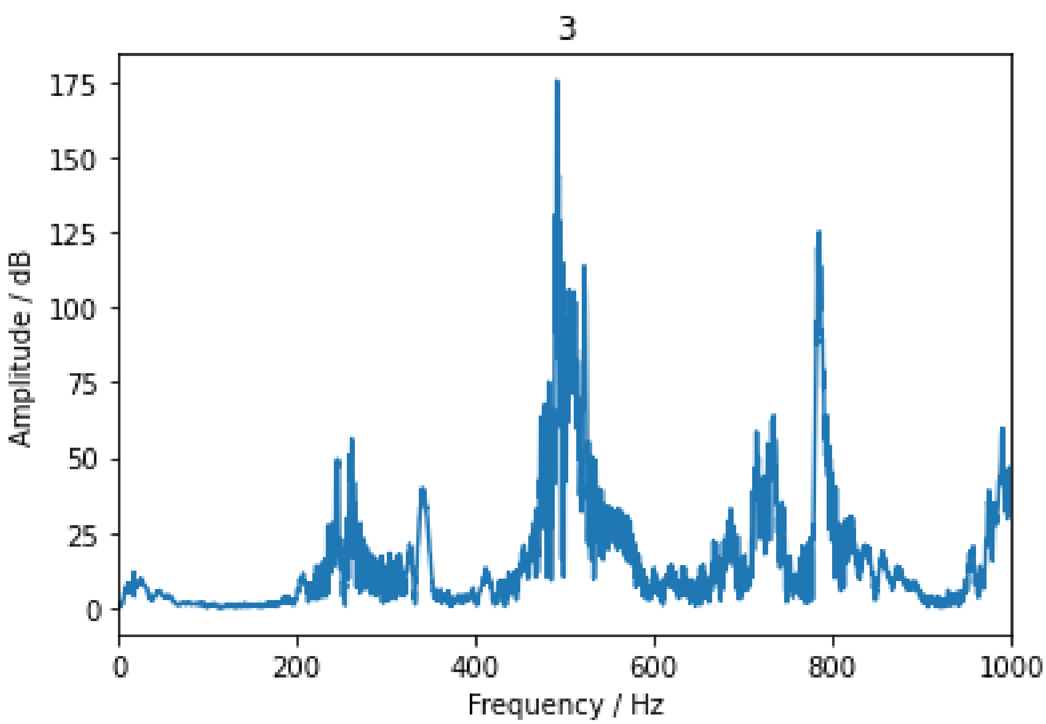

3
Amplitude / dB
Frequency / Hz
0
20
40
60
80
100
0
200
400
600
800
1000


3
Amplitude / dB
Frequency / Hz
0
20
40
60
80
100
120
0
200
400
600
800
1000


1
Amplitude / dB
160
140
120
100
80
60
40
20
0
0
200
400
600
800
1000
Frequency / Hz

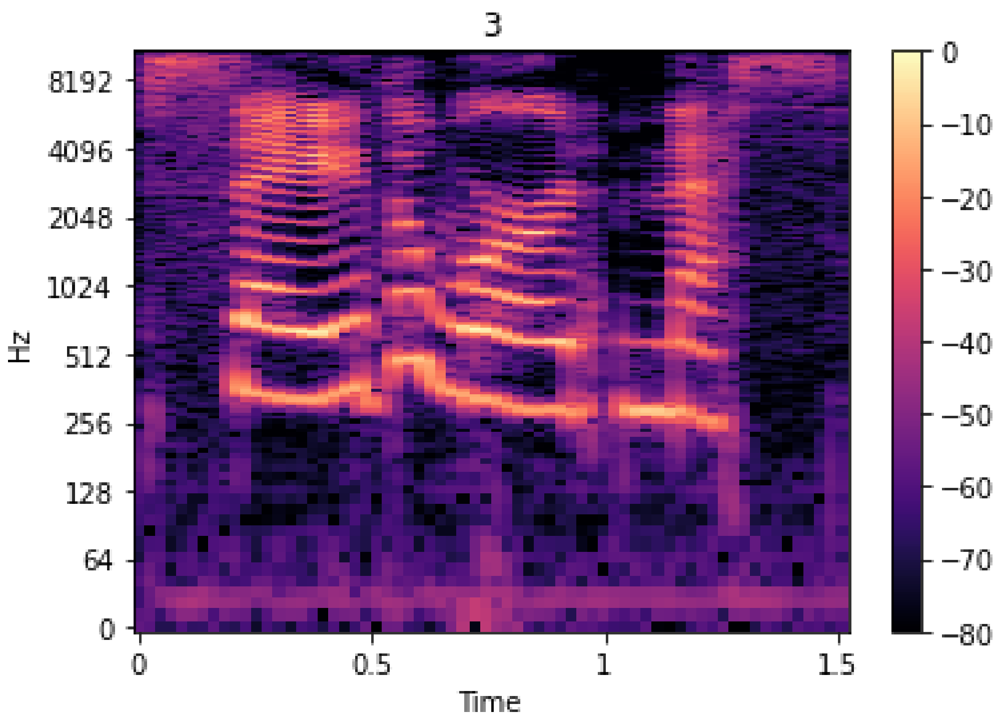

2
Amplitude / dB
100
80
60
40
20
0
0
200
400
600
800
1000
Frequency / Hz

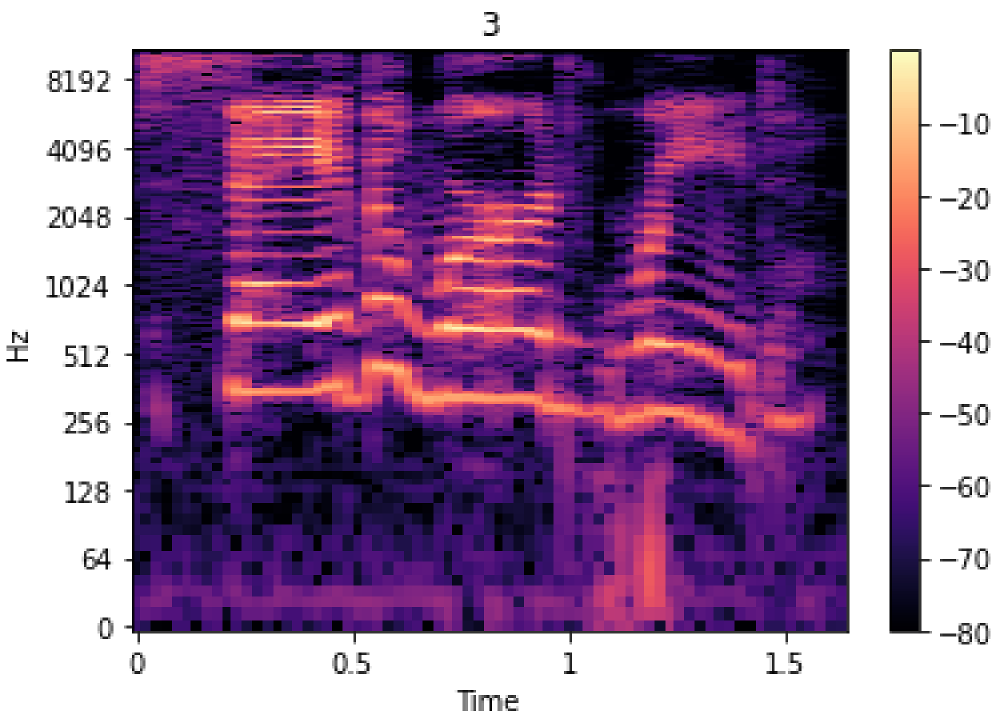

1
Amplitude / dB
0
20
40
60
80
100
120
140
160
0
200
400
600
800
1000
Frequency / Hz

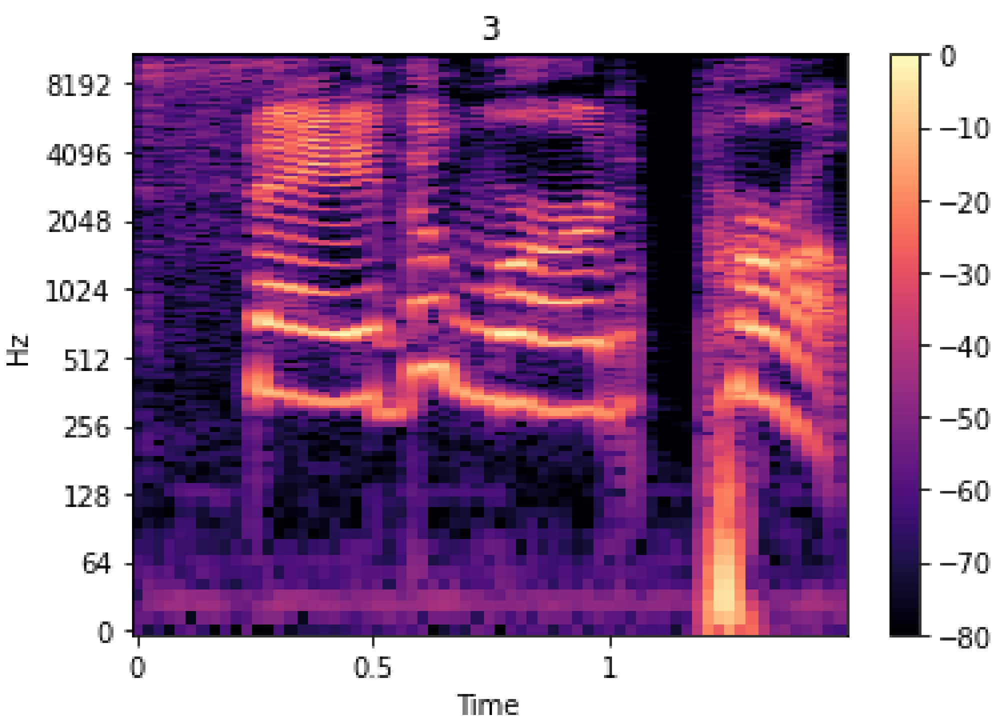

3
Amplitude / dB
0
25
50
75
100
125
150
175
0
200
400
600
800
1000
Frequency / Hz

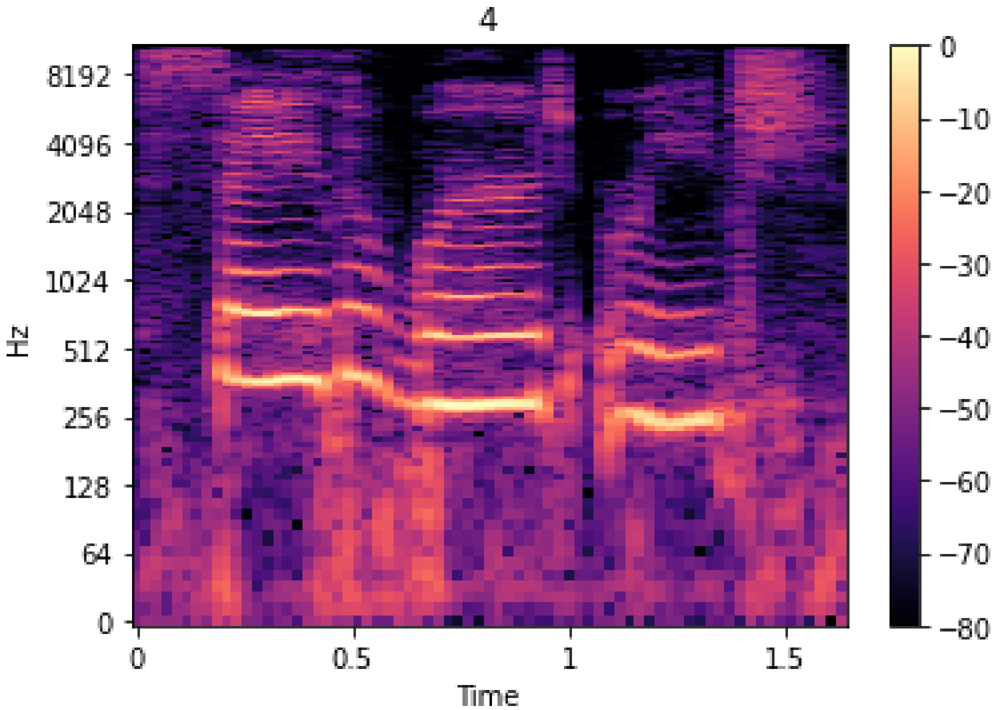
[90]:

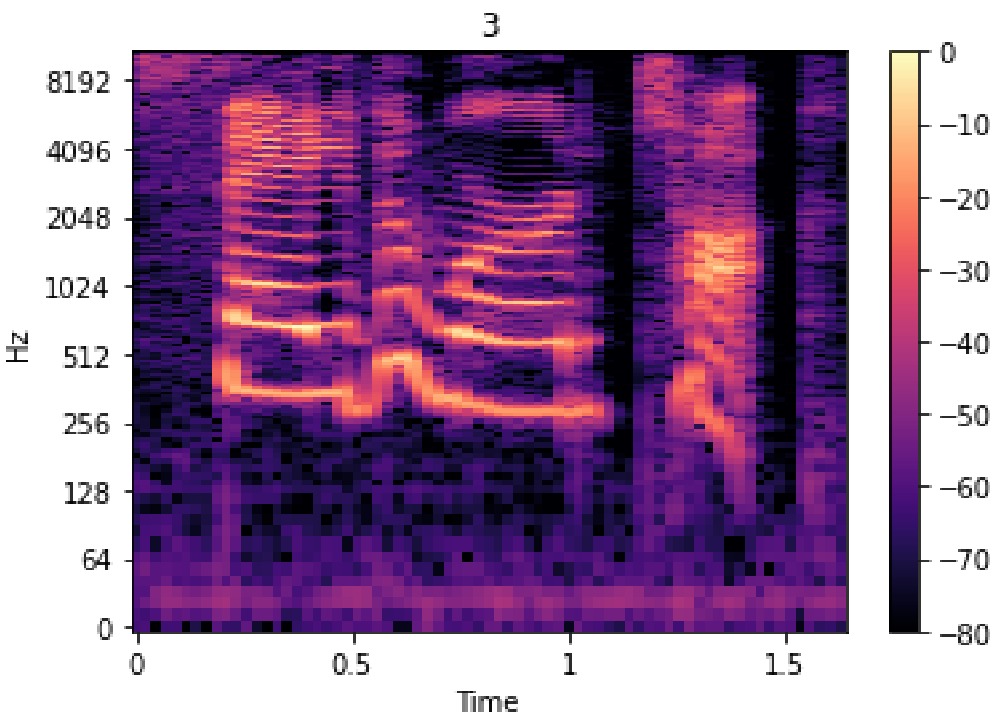
```python
# sample of 10 files in the time-frequency domain (some code copied from Coding␣
↪Materials)
for i in range(len(train_rand)):
    signal, sample_rate = librosa.load(train_rand[i], sr = 16000,mono = True)
    D = librosa.stft(signal)  # STFT of y
    S_db = librosa.amplitude_to_db(np.abs(D), ref=np.max)
    plt.figure(3)
    librosa.display.specshow(S_db, x_axis='time', y_axis='log')
    plt.title(t_rand[i])
    plt.colorbar()
    plt.show()
```

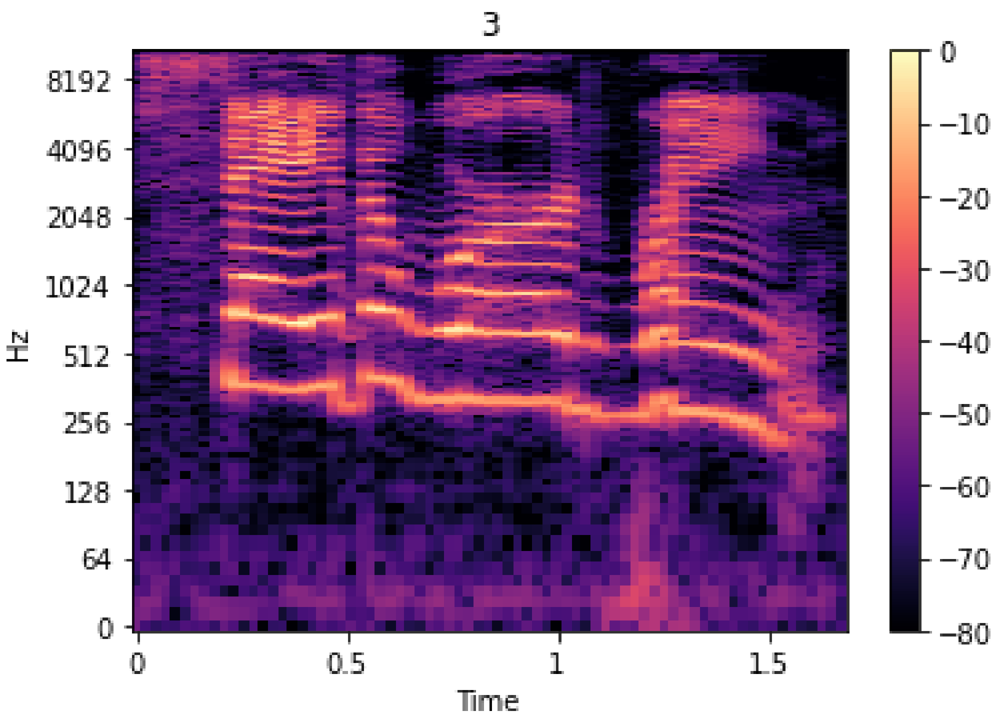


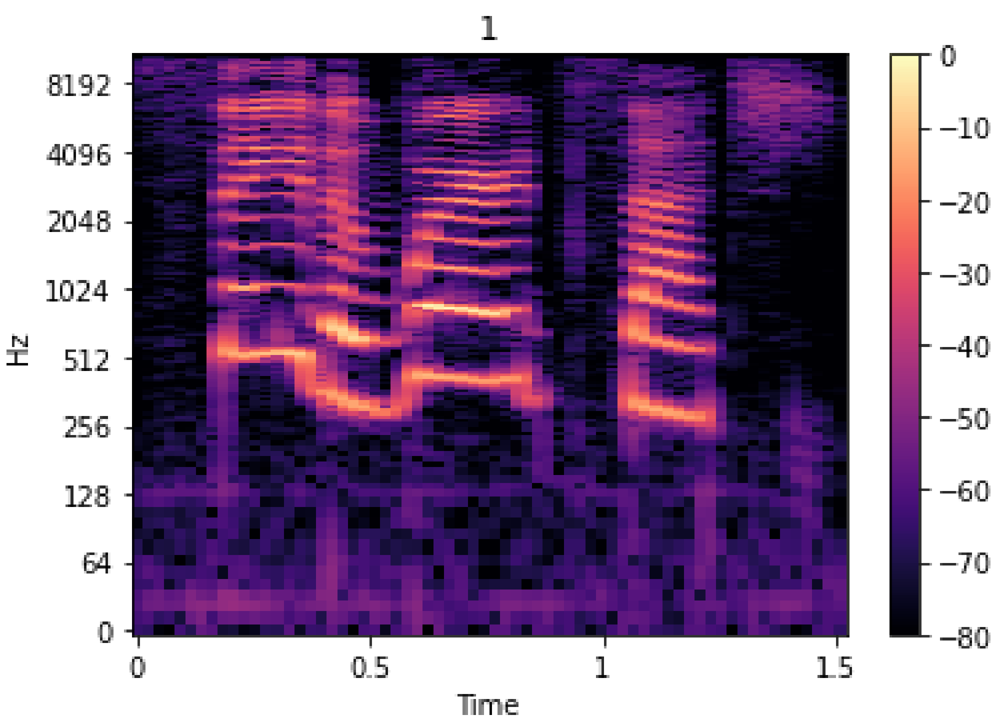

3
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
1.5
Time
−10
−20
−30
−40
−50
−60
−70
−80

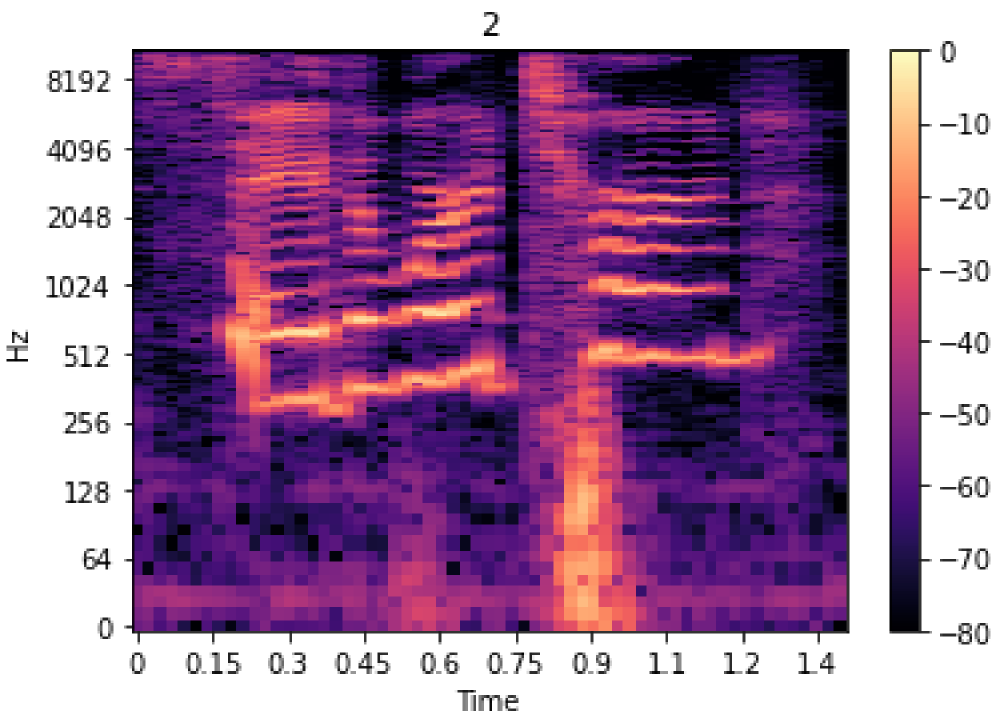

3
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
Time
0
−10
−20
−30
−40
−50
−60
−70
−80

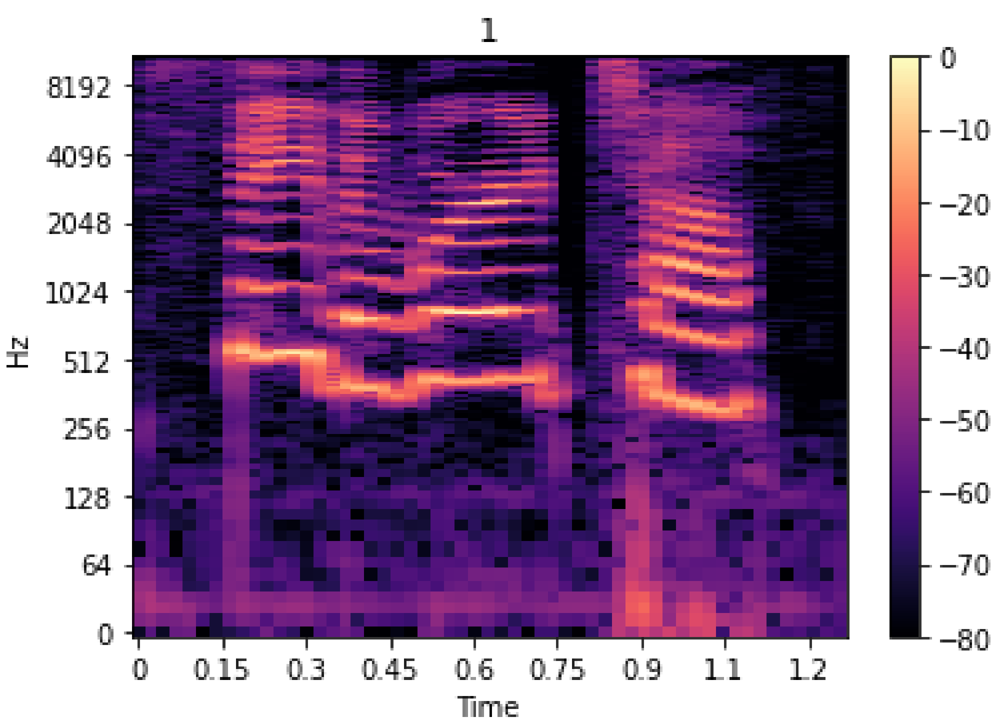

4
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
1.5
Time
0
−10
−20
−30
−40
−50
−60
−70
−80

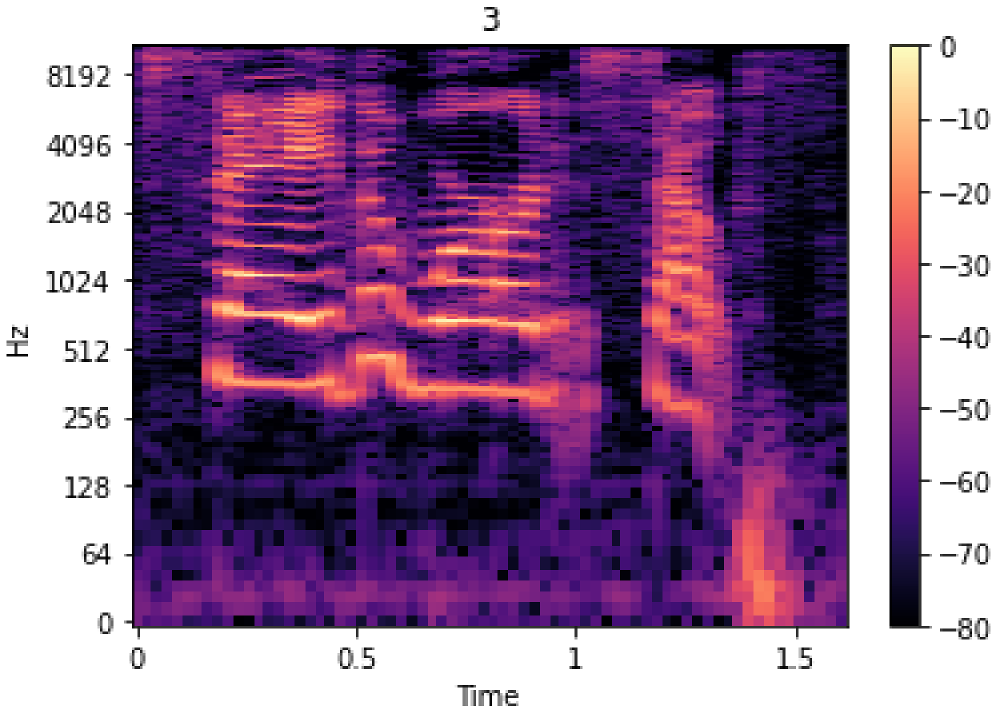

3
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
1.5
Time
0
−10
−20
−30
−40
−50
−60
−70
−80


3
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
1.5
Time
0
−10
−20
−30
−40
−50
−60
−70
−80


1
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.5
1
1.5
Time
0
−10
−20
−30
−40
−50
−60
−70
−80


2
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.15
0.3
0.45
0.6
0.75
0.9
1.1
1.2
1.4
Time
0
−10
−20
−30
−40
−50
−60
−70
−80


1
8192
4096
2048
1024
512
256
128
64
0
Hz
0
0.15
0.3
0.45
0.6
0.75
0.9
1.1
1.2
Time
0
−10
−20
−30
−40
−50
−60
−70
−80



0.0.3 Step 3 + 4. Acoustic Feature Extraction and Feature Post-Processing

[125]:
```python
train_final = []
# going to use mel, chroma, mfcc, and zero-crossing rate
for X in train:
    result = np.array([])
    signal, sample_rate = librosa.load(X, sr = 16000,mono = True)

    mel = np.mean(librosa.feature.melspectrogram(y=signal, sr=sample_rate,
 ↪n_mels=12).T, axis=0)
    result = np.hstack((result, mel))

    stft = np.abs(librosa.stft(signal))
    chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).
 ↪T,axis=0)
    result = np.hstack((result, chroma))

    mfcc = np.mean(librosa.feature.mfcc(y=signal, sr=sample_rate, n_mfcc=40).T,
 ↪axis=0)
    result = np.hstack((result, mfcc))

    zcr = np.mean(librosa.feature.zero_crossing_rate(y=signal).T, axis=0)
    result = np.hstack((result, zcr))
```

```
    train_final.append(result)

train_final = np.array(train_final)
```

[126]:
```
#Same for test set
test_final = []
# going to use mel, chroma, mfcc, and zero-crossing rate
for X in test:
    result = np.array([])
    signal, sample_rate = librosa.load(X, sr = 16000,mono = True)

    mel = np.mean(librosa.feature.melspectrogram(y=signal, sr=sample_rate,␣
 ↪n_mels=12).T, axis=0)
    result = np.hstack((result, mel))

    stft = np.abs(librosa.stft(signal))
    chroma = np.mean(librosa.feature.chroma_stft(S=stft, sr=sample_rate).
 ↪T,axis=0)
    result = np.hstack((result, chroma))

    mfcc = np.mean(librosa.feature.mfcc(y=signal, sr=sample_rate, n_mfcc=40).T,␣
 ↪axis=0)
    result = np.hstack((result, mfcc))

    zcr = np.mean(librosa.feature.zero_crossing_rate(y=signal).T, axis=0)
    result = np.hstack((result, zcr))

    test_final.append(result)

test_final = np.array(test_final)
test_final[:5]
```

[126]:
```
array([[ 2.78944135e+00,  1.08825836e+01,  7.50644064e+00,
         4.32202637e-01,  4.96461809e-01,  7.82982945e-01,
         1.01789522e+00,  7.35965133e-01,  2.70323604e-01,
         1.84393838e-01,  2.24445969e-01,  3.10021476e-03,
         4.48365122e-01,  3.80865276e-01,  4.84215349e-01,
         5.33626258e-01,  3.89551610e-01,  3.56823772e-01,
         3.89460534e-01,  3.82027835e-01,  4.61981893e-01,
         5.09998560e-01,  5.12017787e-01,  4.61066157e-01,
        -3.11361664e+02,  2.50672646e+01, -4.85495710e+00,
         1.70774899e+01, -1.87905521e+01,  7.78478909e+00,
        -2.57263489e+01, -1.17969990e+00, -1.16400042e+01,
        -3.37210584e+00,  7.46299458e+00, -9.65892220e+00,
         7.87922525e+00, -5.83500338e+00,  3.52104807e+00,
         6.09738922e+00, -5.50199413e+00,  1.12446797e+00,
```

```
 -7.41369438e+00,  2.75130749e-01,  6.27318192e+00,
  1.16764174e+01,  4.25448942e+00,  4.02352428e+00,
  8.67032051e-01,  8.37811565e+00,  1.01635923e+01,
  8.60661316e+00,  3.83869529e+00,  5.13594770e+00,
  2.21762300e+00,  7.25418234e+00,  7.44681358e-01,
 -3.95436358e+00, -7.36813545e+00,  1.74653018e+00,
 -5.45726642e-02, -6.90226221e+00, -6.96888638e+00,
  3.47546721e+00,  1.98598710e-01],
[ 1.65515089e+00,  7.66044188e+00,  3.79416418e+00,
  4.99445975e-01,  3.46056819e-01,  5.96384048e-01,
  1.60152197e+00,  1.30369580e+00,  3.03460687e-01,
  2.15496764e-01,  3.91554117e-01,  5.63468002e-02,
  4.41658795e-01,  4.25027430e-01,  5.58049023e-01,
  4.51253355e-01,  4.27997649e-01,  4.47885841e-01,
  4.41898584e-01,  4.87369567e-01,  3.90537977e-01,
  4.95090246e-01,  4.75138575e-01,  3.87781769e-01,
 -2.99568176e+02,  1.78370726e+00, -4.99913073e+00,
  1.99108067e+01, -1.81189766e+01, -1.11572854e-01,
 -2.79010220e+01,  1.10832968e+01, -1.52225819e+01,
 -2.48857880e+00,  6.89424467e+00, -4.61101627e+00,
  6.50692272e+00, -6.72597742e+00,  1.37654853e+00,
  2.91989136e+00, -2.16592860e+00,  4.78574276e+00,
 -7.04761076e+00,  6.57415316e-02,  1.12547183e+00,
  1.36249199e+01,  6.07105875e+00,  4.79144669e+00,
 -3.11455154e+00,  8.13684177e+00,  1.13685188e+01,
  7.81255388e+00, -1.24262297e+00,  1.17965177e-01,
  5.08546710e-01,  3.32952237e+00,  1.38774484e-01,
  1.93644571e+00, -6.90308762e+00, -1.14420092e+00,
 -5.64524174e+00, -2.75184751e+00, -4.83487511e+00,
  2.56790376e+00,  3.00240385e-01],
[ 1.00555825e+00,  3.06099486e+00,  1.34361959e+00,
  1.50587901e-01,  6.00629635e-02,  1.02310129e-01,
  2.46379614e-01,  1.56211928e-01,  9.88141820e-02,
  5.31216934e-02,  3.40581648e-02,  8.15953652e-04,
  5.32615781e-01,  4.58756596e-01,  4.43288177e-01,
  4.53076184e-01,  3.53131115e-01,  3.12055767e-01,
  4.26560670e-01,  5.33697963e-01,  4.12981272e-01,
  4.45122331e-01,  4.34200048e-01,  4.35449123e-01,
 -3.33361755e+02,  3.03308964e+01, -2.32181492e+01,
  2.64116306e+01, -2.18025818e+01, -7.27690554e+00,
 -3.06044216e+01, -2.94929910e+00, -5.12519741e+00,
 -1.54732876e+01,  1.02284241e+01, -8.37602329e+00,
  8.07937145e+00, -4.58481693e+00, -4.18523455e+00,
  1.02065792e+01, -6.27694464e+00,  7.10070801e+00,
 -5.10299015e+00,  6.00077915e+00,  4.69414949e+00,
  1.23806019e+01,  5.03416920e+00,  6.91114569e+00,
  3.56492639e+00,  7.13018274e+00, -3.81400794e-01,
```

```
 -4.06779528e+00, -1.47803175e+00,  1.02162476e+01,
  7.33367634e+00,  1.67338121e+00, -2.07164049e+00,
  5.98239899e+00, -1.96504998e+00, -5.78470182e+00,
 -6.99425983e+00,  5.73050261e+00,  8.53379667e-01,
  1.73338187e+00,  1.78179060e-01],
[ 2.07529426e+00,  9.32762527e+00,  7.82351351e+00,
  4.83738363e-01,  3.04884493e-01,  7.66575098e-01,
  1.70551419e+00,  1.04967344e+00,  5.00820041e-01,
  2.66805768e-01,  3.92901182e-01,  1.08738542e-02,
  4.43905234e-01,  4.38201725e-01,  4.92714018e-01,
  5.13586640e-01,  3.77176046e-01,  3.64430040e-01,
  3.60108137e-01,  4.30033624e-01,  3.82262886e-01,
  4.71397609e-01,  5.51218867e-01,  4.94253159e-01,
 -2.97813629e+02,  1.67689075e+01, -1.25956230e+01,
  1.85121841e+01, -2.71667004e+01,  8.89603257e-01,
 -2.67165527e+01,  5.11802077e-01, -8.98195267e+00,
 -4.81542921e+00,  6.20172691e+00, -9.24335194e+00,
  6.04521942e+00, -9.31447887e+00, -3.36215401e+00,
  5.45173883e+00, -8.53123760e+00,  5.71556664e+00,
 -8.81052399e+00,  1.71462607e+00,  5.54342794e+00,
  9.80182838e+00,  3.59147143e+00,  9.25455630e-01,
  1.74968982e+00,  1.39496279e+01,  1.46064672e+01,
  7.63466072e+00, -2.28827333e+00,  3.77166772e+00,
  5.14326906e+00,  5.69856644e+00, -4.36772728e+00,
 -2.08989930e+00, -4.68058157e+00, -1.00114107e+00,
 -5.72277784e+00, -5.02071953e+00, -3.45618725e+00,
  5.08010721e+00,  2.39818135e-01],
[ 3.72918725e+00,  1.26064482e+01,  8.97039795e+00,
  1.72111654e+00,  6.25567555e-01,  1.33679271e+00,
  1.69793391e+00,  6.09546781e-01,  3.06072414e-01,
  1.19133599e-01,  1.39582410e-01,  4.12690965e-03,
  3.98368627e-01,  4.90425766e-01,  4.86752570e-01,
  3.39914858e-01,  3.06609273e-01,  3.31718892e-01,
  4.81638491e-01,  4.46581423e-01,  4.41755861e-01,
  4.57142174e-01,  4.63110864e-01,  4.84478503e-01,
 -2.70139771e+02,  2.72193832e+01, -8.81241989e+00,
  1.84456234e+01, -1.67785549e+01, -3.67823148e+00,
 -2.25727501e+01,  5.24397802e+00, -9.16108322e+00,
 -1.02447958e+01,  6.47690821e+00, -9.75319290e+00,
  6.57380819e+00, -1.08549700e+01, -1.95615575e-01,
  3.43057537e+00, -5.99858761e+00,  7.69930458e+00,
 -5.15514183e+00, -3.21374774e+00, -4.58371115e+00,
  1.00067387e+01,  1.28023739e+01,  1.55847530e+01,
  3.94800711e+00,  5.48131752e+00,  4.79337645e+00,
  8.59108543e+00,  2.31970310e+00, -1.72788954e+00,
 -3.29890800e+00,  6.95505285e+00,  5.27746916e+00,
  3.62640762e+00, -3.17840672e+00, -4.17687833e-01,
```

```
-2.98221946e+00, -1.08079448e-01, -6.92885876e+00,
-5.08901787e+00,  2.21525065e-01]])
```

0.0.4 Step 5: Audio emotion Recognition Model

[129]:
```
nbc = GaussianNB()
rfc = RandomForestClassifier()
svm = SVC(probability=True)
knn = KNeighborsClassifier(n_neighbors = 7)

model_params = {
    'alpha': 0.01,
    'batch_size': 256,
    'epsilon': 1e-08,
    'hidden_layer_sizes': (300,),
    'learning_rate': 'adaptive',
    'max_iter': 500,
}

mlp = MLPClassifier(**model_params)
```

[130]:
```
nbc.fit(train_final, target)
rfc.fit(train_final, target)
svm.fit(train_final, target)
knn.fit(train_final, target)
mlp.fit(train_final, target)
```

[130]:
```
MLPClassifier(alpha=0.01, batch_size=256, hidden_layer_sizes=(300,),
              learning_rate='adaptive', max_iter=500)
```

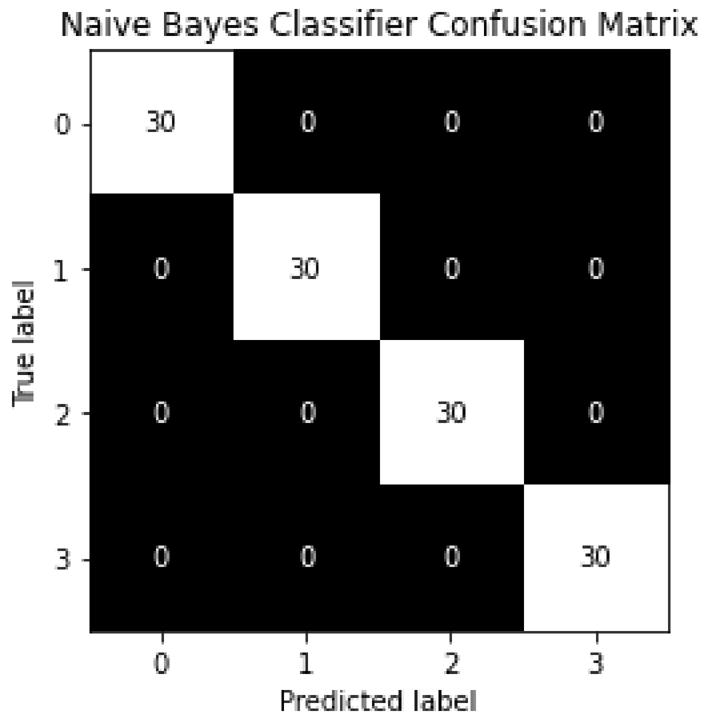
[131]:
```
y_nbc_predicted = nbc.predict(test_final)
y_nbc_pred_proba = nbc.predict_proba(test_final)

y_rfc_predicted = rfc.predict(test_final)
y_rfc_pred_proba = rfc.predict_proba(test_final)

y_svc_predicted = svm.predict(test_final)
y_svc_pred_proba = svm.predict_proba(test_final)

y_knn_predicted = knn.predict(test_final)
y_knn_pred_proba = knn.predict_proba(test_final)

y_mlp_predicted = mlp.predict(test_final)
y_mlp_pred_proba = mlp.predict_proba(test_final)
```

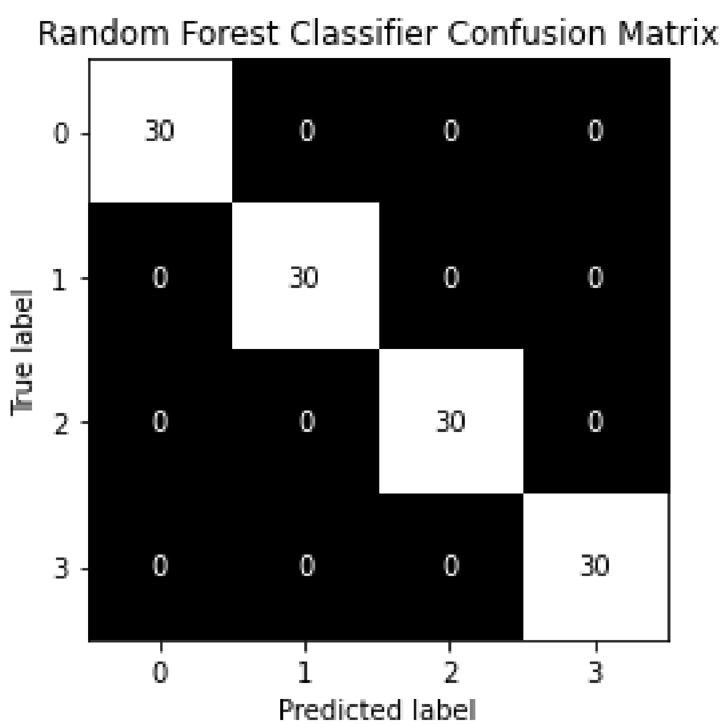
[135]:
```
#Copied from class coding materials with a few adjustments
```

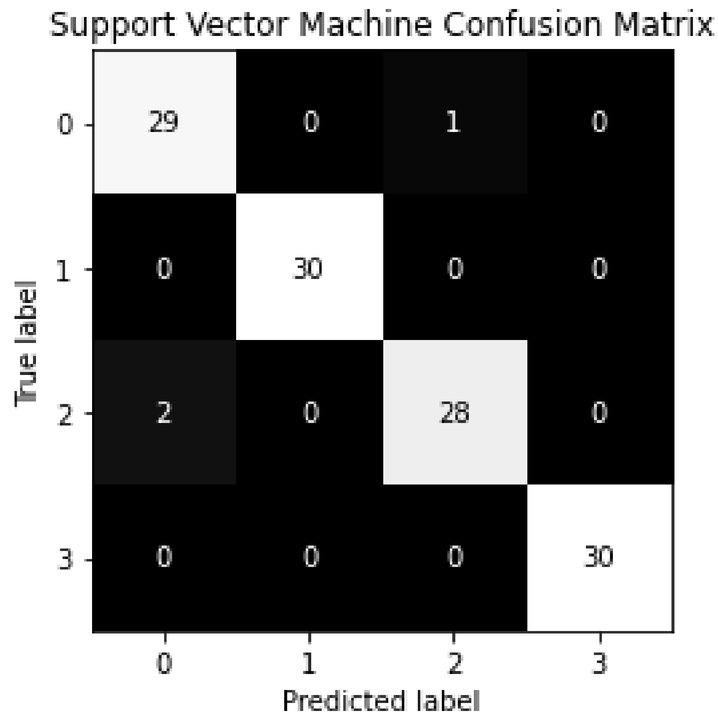
```
print(classification_report(t, y_nbc_predicted))
print(classification_report(t, y_rfc_predicted))
print(classification_report(t, y_svc_predicted))
print(classification_report(t, y_knn_predicted))
print(classification_report(t, y_mlp_predicted))

models = ['Naive Bayes Classifier', 'Random Forest Classifier', 'Support Vector␣
↪Machine', 'K-Nearest Neighbors',
         'Multi-Layer Perceptron']
predictions = [y_nbc_predicted, y_rfc_predicted, y_svc_predicted,␣
↪y_knn_predicted, y_mlp_predicted]
pred_probabilities = [y_nbc_pred_proba, y_rfc_pred_proba, y_svc_pred_proba,␣
↪y_knn_pred_proba, y_mlp_predicted]

plot = 1

for model, prediction, pred_proba in zip(models, predictions,␣
↪pred_probabilities):
    disp = ConfusionMatrixDisplay(confusion_matrix(t.ravel(), prediction))
    disp.plot(
        include_values=True,
        cmap='gray',
        colorbar=False
    )
    disp.ax_.set_title(f"{model} Confusion Matrix")

plt.show()
```

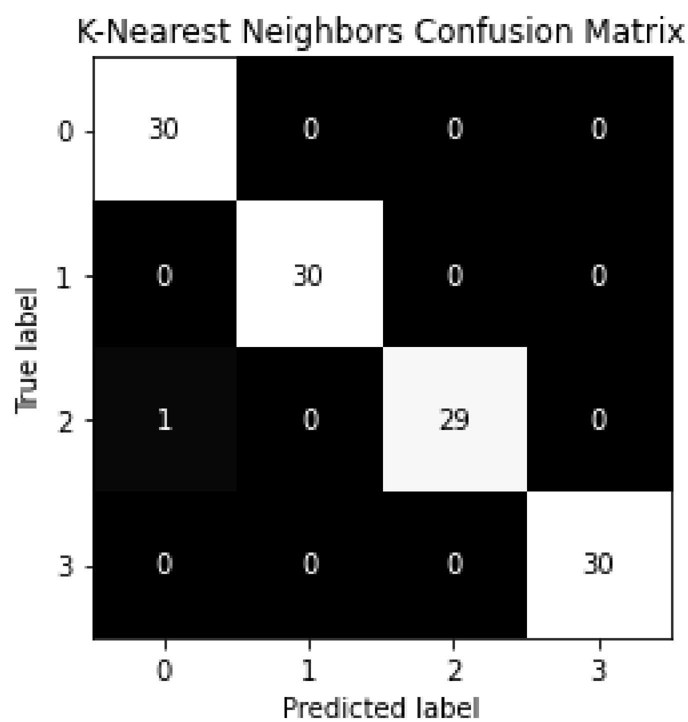
```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        30
           2       1.00      1.00      1.00        30
           3       1.00      1.00      1.00        30
           4       1.00      1.00      1.00        30

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        30
           2       1.00      1.00      1.00        30
           3       1.00      1.00      1.00        30
           4       1.00      1.00      1.00        30

    accuracy                           1.00       120
```

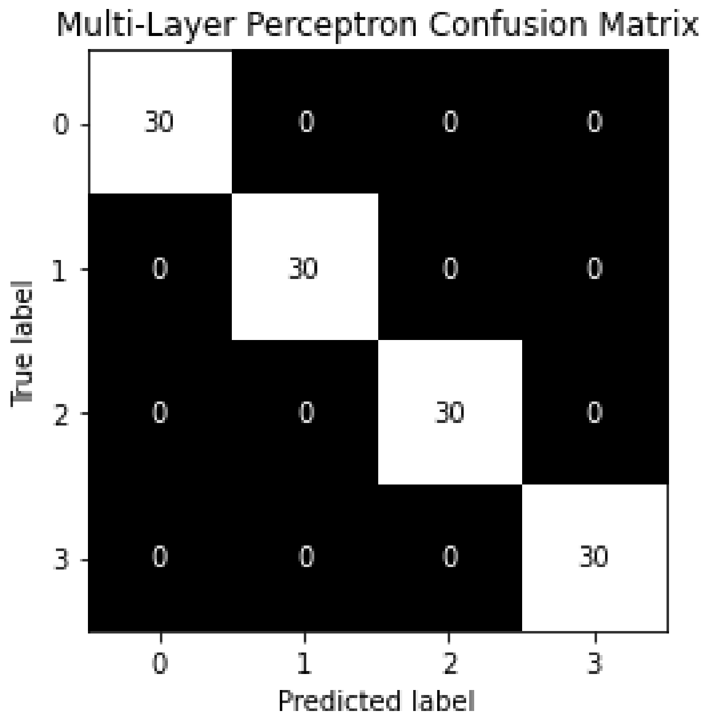
```
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120

              precision    recall  f1-score   support

           1       0.94      0.97      0.95        30
           2       1.00      1.00      1.00        30
           3       0.97      0.93      0.95        30
           4       1.00      1.00      1.00        30

    accuracy                           0.97       120
   macro avg       0.98      0.98      0.97       120
weighted avg       0.98      0.97      0.97       120

              precision    recall  f1-score   support

           1       0.97      1.00      0.98        30
           2       1.00      1.00      1.00        30
           3       1.00      0.97      0.98        30
           4       1.00      1.00      1.00        30

    accuracy                           0.99       120
   macro avg       0.99      0.99      0.99       120
weighted avg       0.99      0.99      0.99       120

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        30
           2       1.00      1.00      1.00        30
           3       1.00      1.00      1.00        30
           4       1.00      1.00      1.00        30

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```


Naive Bayes Classifier Confusion Matrix
True label
Predicted label
0
1
2
3
30
0
30


Random Forest Classifier Confusion Matrix
True label
Predicted label
0
1
2
3
30
0
30


Support Vector Machine Confusion Matrix
29
0
1
0
0
30
0
0
2
0
28
0
0
0
0
30
True label
0
1
2
3
Predicted label


K-Nearest Neighbors Confusion Matrix
30
0
0
0
0
30
0
0
1
0
29
0
0
0
0
30
True label
0
1
2
3
Predicted label


Multi-Layer Perceptron Confusion Matrix
True label
Predicted label
0
1
2
3
30
0

[ ]: