In [1]:

```
# If true, the WAV files will be read and their features will be save
# As this is the most time consuming task, only enable it if you don'
CREATE_CSV_FILES = T
```

In [2]:

```
# Defines the names of the CSV files
TRAIN_CSV_FILE = "train.csv"
TEST_CSV_FILE = "test.csv"
MORE_TRAIN_CSV_FILE = "more_train.csv"
MORE_TEST_CSV_FILE = "more_test.csv"
```

In [3]:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
import librosa
import csv
import os

def extractWavFeatures(soundFilesFolder, csvFileName):
    print("The features of the files in the folder "+soundFilesFolder
    header = 'filename chroma_stft rmse spectral_centroid spectral_ba
    for i in range(1, 21):
        header += f' mfcc{i}'
    header += ' label'
    header = header.split()
    print('CSV Header: ', header)
    file = open(csvFileName, 'w', newline='')
    writer = csv.writer(file)
    writer.writerow(header)
    genres = '1 2 3 4 5 6 7 8 9 0'.split()
    for filename in os.listdir(soundFilesFolder):
        number = f'{soundFilesFolder}/{filename}'
        y, sr = librosa.load(number, mono=True, duration=30)
        # remove leading and trailing silence
        y, index = librosa.effects.trim(y)
        chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
        rmse = librosa.feature.rms(y=y)
        spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
        spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
        rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
        zcr = librosa.feature.zero_crossing_rate(y)
        mfcc = librosa.feature.mfcc(y=y, sr=sr)
        to_append = f'{filename} {np.mean(chroma_stft)} {np.mean(rmse
        for e in mfcc:
            to_append += f' {np.mean(e)}'
        writer.writerow(to_append.split())
    file.close()
    print("End of extractWavFeatures")

if (CREATE_CSV_FILES == True):
    extractWavFeatures("./data/recordings/train", TRAIN_CSV_FILE)
    extractWavFeatures("./data/recordings/test", TEST_CSV_FILE)
    extractWavFeatures("./data/recordings/moreSpeakersTrain", MORE_TR
    extractWavFeatures("./data/recordings/moreSpeakersTest", MORE_TES
    print("CSV files are created")
else:
    print("CSV files creation is skipped")
```

```
In C:\Users\Yana\Anaconda3\lib\site-packages\matplotlib\mpl-data\style
lib\_classic_test.mplstyle:
The text.latex.preview rcparam was deprecated in Matplotlib 3.3 and wi
ll be removed two minor releases later.
In C:\Users\Yana\Anaconda3\lib\site-packages\matplotlib\mpl-data\style
lib\_classic_test.mplstyle:
The mathtext.fallback_to_cm rcparam was deprecated in Matplotlib 3.3 a
nd will be removed two minor releases later.
In C:\Users\Yana\Anaconda3\lib\site-packages\matplotlib\mpl-data\style
lib\_classic_test.mplstyle: Support for setting the 'mathtext.fallback
_to_cm' rcParam is deprecated since 3.3 and will be removed two minor
releases later; use 'mathtext.fallback : 'cm' instead.
In C:\Users\Yana\Anaconda3\lib\site-packages\matplotlib\mpl-data\style
lib\_classic_test.mplstyle:
The validate_bool_maybe_none function was deprecated in Matplotlib 3.3
and will be removed two minor releases later.
In C:\Users\Yana\Anaconda3\lib\site-packages\matplotlib\mpl-data\style
lib\_classic_test.mplstyle:
The savefig.jpeg_quality rcparam was deprecated in Matplotlib 3.3 and
will be removed two minor releases later.
In C:\Users\Yana\Anaconda3\lib\site-packages\matplotlib\mpl-data\style
lib\_classic_test.mplstyle:
The keymap.all_axes rcparam was deprecated in Matplotlib 3.3 and will
be removed two minor releases later.
In C:\Users\Yana\Anaconda3\lib\site-packages\matplotlib\mpl-data\style
lib\_classic_test.mplstyle:
The animation.avconv_path rcparam was deprecated in Matplotlib 3.3 and
will be removed two minor releases later.
In C:\Users\Yana\Anaconda3\lib\site-packages\matplotlib\mpl-data\style
lib\_classic_test.mplstyle:
The animation.avconv_args rcparam was deprecated in Matplotlib 3.3 and
will be removed two minor releases later.

The features of the files in the folder ./data/recordings/train will b
e saved to train.csv
CSV Header:  ['filename', 'chroma_stft', 'rmse', 'spectral_centroid',
'spectral_bandwidth', 'rolloff', 'zero_crossing_rate', 'mfcc1', 'mfcc2
', 'mfcc3', 'mfcc4', 'mfcc5', 'mfcc6', 'mfcc7', 'mfcc8', 'mfcc9', 'mfc
c10', 'mfcc11', 'mfcc12', 'mfcc13', 'mfcc14', 'mfcc15', 'mfcc16', 'mfc
c17', 'mfcc18', 'mfcc19', 'mfcc20', 'label']
End of extractWavFeatures
The features of the files in the folder ./data/recordings/test will be
saved to test.csv
CSV Header:  ['filename', 'chroma_stft', 'rmse', 'spectral_centroid',
'spectral_bandwidth', 'rolloff', 'zero_crossing_rate', 'mfcc1', 'mfcc2
', 'mfcc3', 'mfcc4', 'mfcc5', 'mfcc6', 'mfcc7', 'mfcc8', 'mfcc9', 'mfc
c10', 'mfcc11', 'mfcc12', 'mfcc13', 'mfcc14', 'mfcc15', 'mfcc16', 'mfc
c17', 'mfcc18', 'mfcc19', 'mfcc20', 'label']
End of extractWavFeatures
The features of the files in the folder ./data/recordings/moreSpeakers
Train will be saved to more_train.csv
CSV Header:  ['filename', 'chroma_stft', 'rmse', 'spectral_centroid',
'spectral_bandwidth', 'rolloff', 'zero_crossing_rate', 'mfcc1', 'mfcc2
', 'mfcc3', 'mfcc4', 'mfcc5', 'mfcc6', 'mfcc7', 'mfcc8', 'mfcc9', 'mfc
c10', 'mfcc11', 'mfcc12', 'mfcc13', 'mfcc14', 'mfcc15', 'mfcc16', 'mfc
c17', 'mfcc18', 'mfcc19', 'mfcc20', 'label']
End of extractWavFeatures
The features of the files in the folder ./data/recordings/moreSpeakers
Test will be saved to more_test.csv
CSV Header:  ['filename', 'chroma_stft', 'rmse', 'spectral_centroid',
'spectral_bandwidth', 'rolloff', 'zero_crossing_rate', 'mfcc1', 'mfcc2
', 'mfcc3', 'mfcc4', 'mfcc5', 'mfcc6', 'mfcc7', 'mfcc8', 'mfcc9', 'mfc
c10', 'mfcc11', 'mfcc12', 'mfcc13', 'mfcc14', 'mfcc15', 'mfcc16', 'mfc
c17', 'mfcc18', 'mfcc19', 'mfcc20', 'label']
End of extractWavFeatures
CSV files are created
```

In [4]:

```python
import pandas as pd
import csv
from sklearn import preprocessing

def preProcessData(csvFileName):
    print(csvFileName+ " will be preprocessed")
    data = pd.read_csv(csvFileName)
    data['number'] = data['filename'].str[:1]
    #Dropping unnecessary columns
    data = data.drop(['filename'],axis=1)
    data = data.drop(['label'],axis=1)
    data = data.drop(['chroma_stft'],axis=1)
    data.shape

    print("Preprocessing is finished")
    print(data.head())
    return data

trainData = preProcessData(TRAIN_CSV_FILE)
testData = preProcessData(TEST_CSV_FILE)
moreTrainData = preProcessData(MORE_TRAIN_CSV_FILE)
moreTestData = preProcessData(MORE_TEST_CSV_FILE)
```

```
train.csv will be preprocessed
Preprocessing is finished
       rmse  spectral_centroid  spectral_bandwidth      rolloff  \
0  0.112672         741.829081          758.492178  1438.494873
1  0.090344         635.610880          670.336296  1160.452403
2  0.091456         667.786694          732.606545  1257.180176
3  0.087751         712.304185          731.292437  1449.104818
4  0.096603         844.363886          777.868127  1569.583263

   zero_crossing_rate       mfcc1       mfcc2      mfcc3      mfcc4  \
0            0.034023 -295.578461  189.853683 -19.606564   6.078509
1            0.033458 -339.148743  204.005249  -7.485528  14.297899
2            0.033268 -327.507416  195.596924  -3.994768  21.315845
3            0.035916 -320.809937  200.023743  -8.186146  12.661074
4            0.049465 -315.801300  195.674118 -13.324564   3.544238

       mfcc5  ...     mfcc12    mfcc13     mfcc14     mfcc15    mfcc16
\
0  22.067095  ... -25.725817 -5.172223  -8.323026 -10.299589 -0.144793
1  20.885138  ... -23.106369 -1.200801  -5.515564 -15.416287 -0.405876
```

# Section 2

There are 50 recordings for each digit for each speaker: Jackson, Nicolas and Theo (total 1500 recordings)

Training data has 49 recordings for each digit for each speaker: 1470 recordings total. Test data has 1 recordings for each digit for each speaker: 30 recordings total.

The data used here comes from the recordings stored in:

../data/recordings/train

../data/recordings/test

The model will be trained to predict the spoken digit.

In [5]:

```
# Splitting the dataset into training, validation and testing dataset
from sklearn.model_selection import train_test_split
X = np.array(trainData.iloc[:, :-1], dtype = float)
y = trainData.iloc[:, -1]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3


X_test = np.array(testData.iloc[:, :-1], dtype = float)
y_test = testData.iloc[:, -1]

print("Y from training data:", y_train.shape)
print("Y from validation data:", y_val.shape)
```

```
Y from training data: (1029,)
Y from validation data: (441,)
Y from test data: (30,)
```

In [6]:

```
#Normalizing the dataset
from sklearn.preprocessing import StandardScaler
import numpy as np
scaler = StandardScaler()
X_train = scaler.fit_transform( X_train )
X_val = scaler.transform( X_val )
X_test = scaler.transform( X_test )

print("X from training data", X_train.shape)
print("X from validation data", X_val.shape)
print("X from test data", X_test.shape)
```

```
X from training data (1029, 25)
X from validation data (441, 25)
X from test data (30, 25)
```

In [7]:

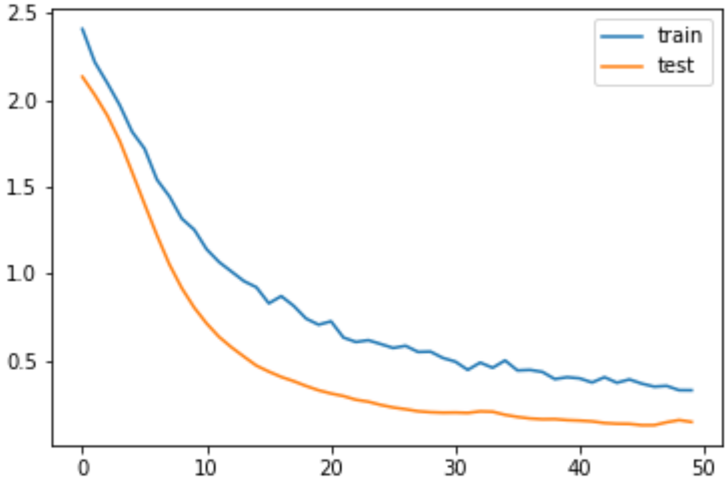
```
#Creating a Model
from keras import models
from keras import layers
import keras

# model 1
model = models.Sequential()
```

```
model.add(layers.Dense(256, activation='relu', input_shape=(X_train.s
model.add(layers.Dropout(0.5))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))

# Learning Process of a model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
#return model

# simple early stopping
from keras.callbacks import EarlyStopping

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)

#Train with early stopping to avoid overfitting
history = model.fit(X_train,
                    y_train,
                    validation_data=(X_val, y_val),
                    epochs=50,
                    batch_size=128)
```

```
Using TensorFlow backend.

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               6656
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                650
=================================================================
```

In [8]:

```python
from matplotlib import pyplot
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
```



In [9]:

```python
def printPrediction(X_data, y_data):
    print('\n# Generate predictions')
    for i in range(len(y_data)):
        prediction = model.predict_classes(X_data[i:i+1])
```

In [10]:

```python
import numpy as np
from keras import backend as K
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix

def report(X_data, y_data):
    #Confution Matrix and Classification Report
    Y_pred = model.predict_classes(X_data)
    y_test_num = y_data.astype(np.int64)
    print('Confusion Matrix')
    conf_mt = confusion_matrix(y_test_num, Y_pred)
    print(conf_mt)
    plt.matshow(conf_mt)
    plt.show()

    print('\nClassification Report')
    target_names = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

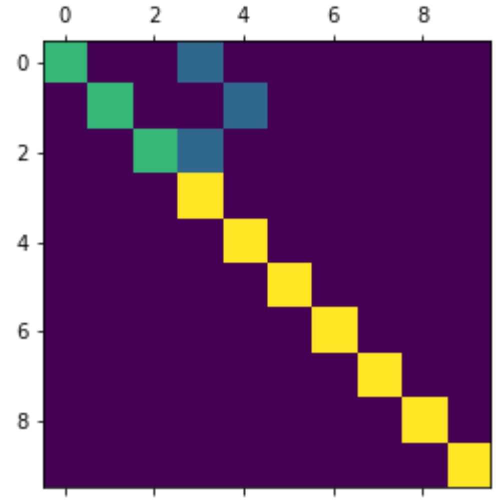
In [11]:

```python
print('\n# TEST DATA #\n')
score = model.evaluate(X_test, y_test)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

# Prediction
# TEST DATA #

30/30 [==============================] - 0s 2ms/step
accuracy: 90.00%

# Generate predictions
y=0, prediction=[0], match=True
y=0, prediction=[3], match=False
y=0, prediction=[0], match=True
y=1, prediction=[1], match=True
y=1, prediction=[4], match=False
y=1, prediction=[1], match=True
y=2, prediction=[2], match=True
y=2, prediction=[2], match=True
```

In [12]:

```
print("Classification Report for Test Data\n")
```

```
Classification Report for Test Data

Confusion Matrix
[[2 0 0 1 0 0 0 0 0 0]
 [0 2 0 0 1 0 0 0 0 0]
 [0 0 2 1 0 0 0 0 0 0]
 [0 0 0 3 0 0 0 0 0 0]
 [0 0 0 0 3 0 0 0 0 0]
 [0 0 0 0 0 3 0 0 0 0]
 [0 0 0 0 0 0 3 0 0 0]
 [0 0 0 0 0 0 0 3 0 0]
 [0 0 0 0 0 0 0 0 3 0]
 [0 0 0 0 0 0 0 0 0 3]]
```



```
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.67      0.80         3
           1       1.00      0.67      0.80         3
           2       1.00      0.67      0.80         3
           3       0.60      1.00      0.75         3
           4       0.75      1.00      0.86         3
           5       1.00      1.00      1.00         3
```

# Section 3

There are 50 recordings for each digit for each speaker: Jackson, Nicolas and Theo (total 1500 recordings) Training data has 49 recordings for each digit for each speaker: 1470 recordings total. Test data has 1 recordings for each digit for each speaker: 30 recordings total.

In addition, there are 2 recordings for each digit for each speaker: Ankur, Caroline and Rodolfo (total 60 recordings) This addition training data has 1 recordings for each digit for each speaker: 30 recordings total. This addition test data has 1 recordings for each digit for each speaker: 30 recordings total.

Therefore the full data set has:

Training: 1500 recordings Training: 60 recordings

The data used here comes from the recordings stored in:

../data/recordings/train ../data/recordings/test ../data/recordings/moreSpeakersTrain ../data /recordings/moreSpeakersTest

In [13]:

```python
# Splitting the dataset into training, validation and testing dataset
from sklearn.model_selection import train_test_split

fullTrainData = trainData.append(moreTrainData)

X = np.array(fullTrainData.iloc[:, :-1], dtype = float)
y = fullTrainData.iloc[:, -1]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3

X_test = np.array(testData.iloc[:, :-1], dtype = float)
y_test = testData.iloc[:, -1]

X_more_test = np.array(moreTestData.iloc[:, :-1], dtype = float)
y_more_test = moreTestData.iloc[:, -1]

print("Y from training data:", y_train.shape)
print("Y from validation data:", y_val.shape)
print("Y from test data:", y_test.shape)
```

```
Y from training data: (1050,)
```

In [14]:

```python
#Normalizing the dataset
from sklearn.preprocessing import StandardScaler
import numpy as np
scaler = StandardScaler()
X_train = scaler.fit_transform( X_train )
X_val = scaler.transform( X_val )
X_test = scaler.transform( X_test )
X_more_test = scaler.transform( X_more_test )

print("X from training data", X_train.shape)
print("X from validation data", X_val.shape)
print("X from test data", X_test.shape)
```

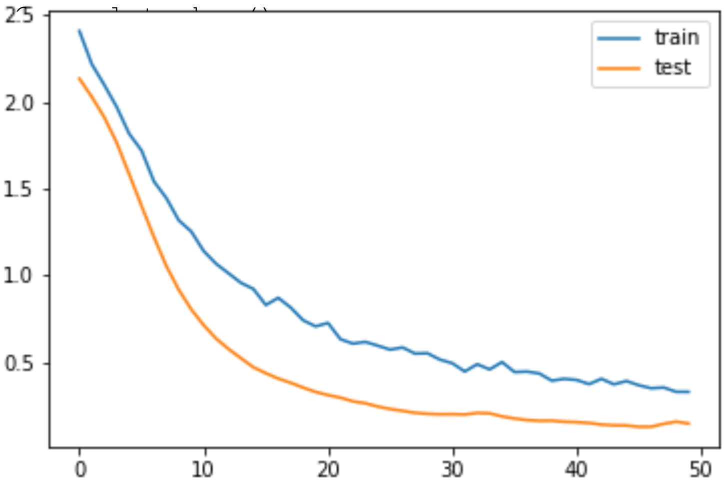
```
X from training data (1050, 25)
X from validation data (450, 25)
X from test data (30, 25)
X from other speakers test data (30, 25)
```

```python
#Creating a Model
from keras import models
from keras import layers
import keras

# model 1
model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_shape=
(X_train.shape[1],)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))

# Learning Process of a model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# simple early stopping
from keras.callbacks import EarlyStopping

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)

#Train with early stopping to avoid overfitting
history = model.fit(X_train,
                    y_train,
                    validation_data=(X_val, y_val),
                    epochs=50,
                    batch_size=128,
```

In [15]:

```python
# plot training history
from matplotlib import pyplot
```

```
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
```



# Present the model performance

In [16]:

```
print('\n# TEST DATA #\n')
score = model.evaluate(X_test, y_test)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

# Prediction
```

```
# TEST DATA #

30/30 [==============================] - 0s 40us/step
accuracy: 90.00%

# Generate predictions
y=0, prediction=[0], match=True
y=0, prediction=[3], match=False
y=0, prediction=[0], match=True
y=1, prediction=[1], match=True
y=1, prediction=[4], match=False
y=1, prediction=[1], match=True
y=2, prediction=[2], match=True
y=2, prediction=[2], match=True
y=2, prediction=[3], match=False
y=3, prediction=[3], match=True
```

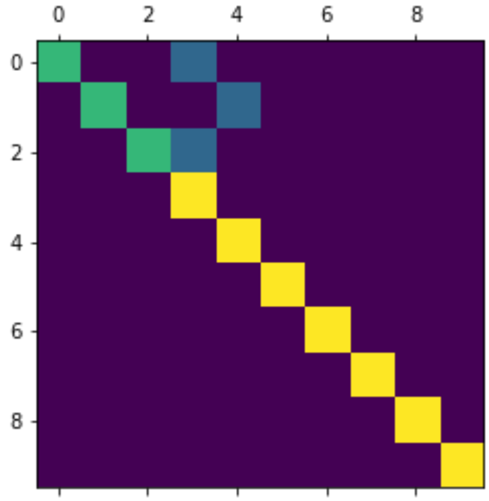
In [17]:

```
print('\n# OTHER SPEAKERS DATA #\n')
score = model.evaluate(X_more_test, y_more_test)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))

# Prediction
# OTHER SPEAKERS DATA #

30/30 [==============================] - 0s 95us/step
accuracy: 26.67%

# Generate predictions
y=0, prediction=[2], match=False
y=0, prediction=[2], match=False
y=0, prediction=[8], match=False
y=1, prediction=[0], match=False
y=1, prediction=[0], match=False
y=1, prediction=[1], match=True
y=2, prediction=[2], match=True
```

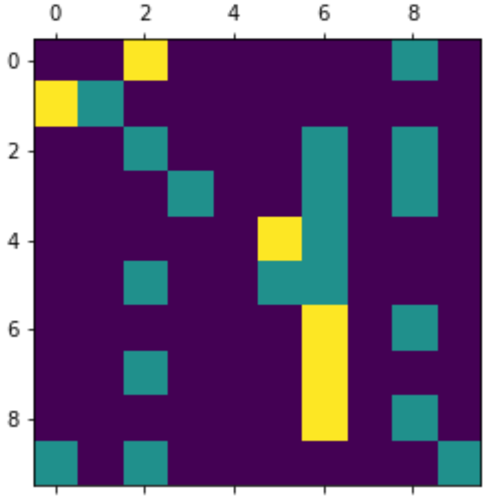
In [18]:

```
print("Classification Report for Test Data\n")
report(X_test, y_test)

print("Classification Report for Other Speakers\n")
```

```
Classification Report for Test Data

Confusion Matrix
[[2 0 0 1 0 0 0 0 0 0]
 [0 2 0 0 1 0 0 0 0 0]
 [0 0 2 1 0 0 0 0 0 0]
 [0 0 0 3 0 0 0 0 0 0]
 [0 0 0 0 3 0 0 0 0 0]
 [0 0 0 0 0 3 0 0 0 0]
 [0 0 0 0 0 0 3 0 0 0]
 [0 0 0 0 0 0 0 3 0 0]
 [0 0 0 0 0 0 0 0 3 0]
 [0 0 0 0 0 0 0 0 0 3]]
```



```
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.67      0.80         3
           1       1.00      0.67      0.80         3
           2       1.00      0.67      0.80         3
           3       0.60      1.00      0.75         3
           4       0.75      1.00      0.86         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         3
           7       1.00      1.00      1.00         3
           8       1.00      1.00      1.00         3
           9       1.00      1.00      1.00         3

    accuracy                           0.90        30
   macro avg       0.93      0.90      0.90        30
weighted avg       0.94      0.90      0.90        30

Classification Report for Other Speakers

Confusion Matrix
[[0 0 2 0 0 0 0 0 1 0]
 [2 1 0 0 0 0 0 0 0 0]
 [0 0 1 0 0 0 1 0 1 0]
```



```
Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       1.00      0.33      0.50         3
           2       0.17      0.33      0.22         3
           3       1.00      0.33      0.50         3
           4       0.00      0.00      0.00         3
           5       0.33      0.33      0.33         3
           6       0.20      0.67      0.31         3
           7       0.00      0.00      0.00         3
           8       0.20      0.33      0.25         3
           9       1.00      0.33      0.50         3
```

```
    accuracy                           0.27        30
   macro avg       0.39      0.27      0.26        30
weighted avg       0.39      0.27      0.26        30

C:\Users\Yana\Anaconda3\lib\site-packages\sklearn\metrics\classificati
on.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defi
ned and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

In [ ]: