Machine Learning for Speech - **Homework 3**

Instructors: Trung Ngo Trong (trung@uef.fi) & Xuechen Liu (xuecliu@uef.fi)

Deadline: 2020.11.18, 9:59 AM (1 minute before the session)

Goals of this exercise

- Learn to process speech dataset
- Train and evaluate a classifier for spoken digits
- Understand unsupervised algorithm using Gaussian Mixture Model

Scoring (total: 10 points)

- Ex1: 4 points
- Ex2: 3 points
- Ex3: 3 points

How to answer questions - **IMPORTANT**

To answer open-ended questions, there is no difference from answering them by pen and paper: you should simply create empty space under the question and provide your answers! Of course, if you want to do some typesetting, you may need some knowledge on markdown and latex equations. For your reference, this note is written in Markdown.

How to make submission - **IMPORTANT**

File → **Download .ipynb** → *Compress the .ipynb file to PDF and send the PDF to submit it to digicampus, or to instructors' email (as a back-up)*.

Note 1: Please re-name the PDF to your name and student number. Any modification/submission after the deadline will be disregarded.

Note 2: Please make sure your answers are included in generated PDF 😂

NOTE: filling all the place marked `#TODO` in the code blocks.

Important: run this block to load all the libraries

```
Processing /root/.cache/pip/wheels/ee/10/1e/382bb4369e189938d5c02e06d10c65181
Requirement already up-to-date: scikit-learn in /usr/local/lib/python3.6/dist
Requirement already satisfied, skipping upgrade: decorator>=3.0.0 in /usr/loc
Requirement already satisfied, skipping upgrade: resampy>=0.2.2 in /usr/local
Requirement already satisfied, skipping upgrade: pooch>=1.0 in /usr/local/lib
Requirement already satisfied, skipping upgrade: scipy>=1.0.0 in /usr/local/l
Requirement already satisfied, skipping upgrade: soundfile>=0.9.0 in /usr/loc
Requirement already satisfied, skipping upgrade: audioread>=2.0.0 in /usr/loc
Requirement already satisfied, skipping upgrade: numpy>=1.15.0 in /usr/local/
Requirement already satisfied, skipping upgrade: numba>=0.43.0 in /usr/local/
Requirement already satisfied, skipping upgrade: joblib>=0.14 in /usr/local/l
```

```
Requirement already satisfied, skipping upgrade: threadpoolctl>=2.0.0 in /usr
Requirement already satisfied, skipping upgrade: six>=1.3 in /usr/local/lib/p
Requirement already satisfied, skipping upgrade: appdirs in /usr/local/lib/py
Requirement already satisfied, skipping upgrade: packaging in /usr/local/lib/
Requirement already satisfied, skipping upgrade: requests in /usr/local/lib/p
Requirement already satisfied, skipping upgrade: cffi>=1.0 in /usr/local/lib/
Requirement already satisfied, skipping upgrade: llvmlite<0.32.0,>=0.31.0dev0
Requirement already satisfied, skipping upgrade: setuptools in /usr/local/lib
Requirement already satisfied, skipping upgrade: pyparsing>=2.0.2 in /usr/loc
Requirement already satisfied, skipping upgrade: urllib3!=1.25.0,!=1.25.1,<1.
Requirement already satisfied, skipping upgrade: idna<3,>=2.5 in /usr/local/l
Requirement already satisfied, skipping upgrade: chardet<4,>=3.0.2 in /usr/lo
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in /usr/l
Requirement already satisfied, skipping upgrade: pycparser in /usr/local/lib/
Installing collected packages: librosa
  Found existing installation: librosa 0.7.2
    Uninstalling librosa-0.7.2:
      Successfully uninstalled librosa-0.7.2
Successfully installed librosa-0.8.0
```

Ex1: Speech dataset processing (4 points)

Task outlines:

1. Extract all .wav files and their metadata (i.e. the digits and speakers ID)
2. Splitting the dataset into 2 partition for: **training** and **testing** (evaluation)
3. Write a function for extract MFCCs features for the wav file
4. Preprocessing MFCCs features for each utterances, understand differences between: **frames-level**, **utterance-level** and **contextual-level** features.

We use [Free Spoken Digit Dataset (FSDD)]()

A simple audio/speech dataset consisting of recordings of spoken digits in `wav` files at 8kHz. The recordings are trimmed so that they have near minimal silence at the beginnings and ends.

- 6 speakers
- 3,000 recordings (50 of each digit per speaker)
- English pronunciations

```
#@title This code will download the dataset and
import os
url = r'https://github.com/Jakobovski/free-spok
filename = '/tmp/fsdd.zip'
if not os.path.exists(filename):
  print('Downloading fsdd dataset ...')
  urlretrieve(url, filename=filename)
path = '/tmp/free-spoken-digit-dataset-master/re
if not os.path.exists(path):
  print('Extracting wav files ...')
  with ZipFile(filename, mode='r') as f:
    f.extractall(path='/tmp')
```

This code will download the dataset and extract all wav files to '/tmp/free-spoken-digit-dataset-master/recordings'

Task 1: Find all wav files and extract its metadata

Find all wav files.

Then extract the digits annotation and speakers annotation for each file.

```
from os import listdir
utterances = list()
metadata = list()
digits = list()
speakers = list()
for filename in listdir(path):
  basename = filename.split('.')[0]
  meta_array = basename.split('_')
  metadata.append([meta_array[0],meta_array[1]])
  digits.append(meta_array[0])
  utterances.append(path+"/"+filename)
  speakers.append(meta_array[1])
#utterances = # TODO find all wav files and extract its metatdata

print(utterances[:5])
print(metadata[:5])
print(digits[:5])
print(speakers[:5])
```

```
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.hist(digits)
plt.title('Histogram of spoken digits')
plt.gca().tick_params(axis='x', rotation=45)
plt.subplot(1, 2, 2)
plt.hist(speakers)
plt.gca().tick_params(axis='x', rotation=45)
_ = plt.title('Histogram of speakers')
```

```
['/tmp/free-spoken-digit-dataset-master/recordings/3_yweweler_7.wav', '/tmp/f
[['3', 'yweweler'], ['4', 'george'], ['2', 'theo'], ['5', 'nicolas'], ['4', '
['3', '4', '2', '5', '4']
['yweweler', 'george', 'theo', 'nicolas', 'nicolas']
```

Histogram of spoken digits Histogram of speakers

Task 2: Splitting the dataset for training and testing

You must follow a strict guideline for splitting the dataset:

- There is no duplicated utterances.
- Training and testing utterances must be mutual exclusive
- None of speakers in test set are included in training set

```
speakers_set = np.unique(speakers)
# TODO splitting the dataset for training and testing
train_speakers = speakers_set[:4]
test_speakers = speakers_set[4:]

train = {}
test = {}
train_utt = list()
train_spk = list()
train_dgt = list()

test_utt = list()
test_dgt = list()
test_spk = list()

for index, speaker in enumerate(speakers):
  if speaker in test_speakers:
    test_utt.append(utterances[index])
    test_spk.append(speaker)
    test_dgt.append(digits[index])
  else:
    train_utt.append(utterances[index])
    train_spk.append(speaker)
    train_dgt.append(digits[index])

train["utt"] = train_utt
train["spk"] = train_spk
train["dgt"] = train_dgt

test["utt"] = test_utt
test["dgt"] = test_dgt
test["spk"] = test_spk
print('All speakers:', speakers_set)
print('Train speakers:', train_speakers)
print('Number train utterances:', len(train['utt']))
print('Test speakers:', test_speakers)
print('Number test utterances:', len(test['utt']))
```

```
All speakers: ['george' 'jackson' 'lucas' 'nicolas' 'theo' 'yweweler']
Train speakers: ['george' 'jackson' 'lucas' 'nicolas']
```

```
Number train utterances: 2000
Test speakers: ['theo' 'yweweler']
Number test utterances: 1000
```

Task 3: Extracting MFCCs features

Steps for extracting MFCCs features:

- Load the audio file
- Resample the audio (if necessary, only if the original sampling rate of wav file is different from provided sampling rate).
- Pre-emphasis
- Perform STFT transform and extract the amplitute spectrogram
- Extract the Mel-spectrogram
- Log compression Mel-spectrogram
- Extract the MFCCs
- Normalize the MFCCs features by mean and standard-deviation to de-correlate the features.

```
Collecting pysoundfile
  Downloading https://files.pythonhosted.org/packages/2a/b3/0b871e5fd31b9a8e5
Requirement already satisfied: cffi>=0.6 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: pycparser in /usr/local/lib/python3.6/dist-pac
Installing collected packages: pysoundfile
Successfully installed pysoundfile-0.9.0.post1
WARNING: The following packages were previously imported in this runtime:
  [soundfile]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

```
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /u
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
Collecting librosa==0.7.2
  Downloading https://files.pythonhosted.org/packages/77/b5/1817862d64a7c231a
     |████████████████████████████████| 1.6MB 5.6MB/s
Requirement already satisfied: audioread>=2.0.0 in /usr/local/lib/python3.6/d
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.6/dist
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: scikit-learn!=0.19.0,>=0.14.0 in /usr/local/li
Requirement already satisfied: joblib>=0.12 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: decorator>=3.0.0 in /usr/local/lib/python3.6/d
Requirement already satisfied: six>=1.3 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: resampy>=0.2.2 in /usr/local/lib/python3.6/dis
Requirement already satisfied: numba>=0.43.0 in /usr/local/lib/python3.6/dist
Requirement already satisfied: soundfile>=0.9.0 in /usr/local/lib/python3.6/d
```

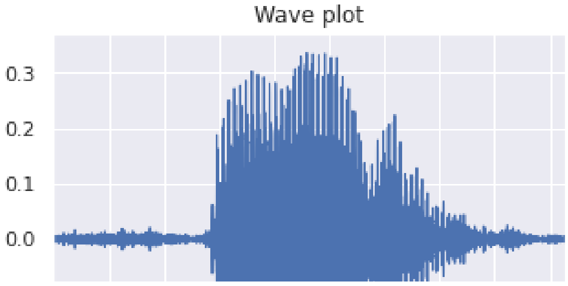
```
import math
import scipy
def preemphasis(signal, coeff=0.95):
  '''pre-emphasis function.
  @signal: input signal, in numpy array
  @coeff: pre-emphasis coefficient

  return: pre-emphasized signal, in numpy array
  '''
  return np.append(signal[0], signal[1:] - coeff * signal[:-1])



def framing(signal, winlen=0.025, winstep=0.01,
            winfunc=np.hamming, srate=16000):
  '''do framing via window function on the signal.
  @signal: input signal, in numpy array
  @winlen: window length, default 25ms
  @winstep: window step length, default 10ms
  @winfunc: type of window function, default Hamming window from numpy
  @srate: sample rate.

  return: framed signal in [num_frames, frame_length]
  '''
  slen = len(signal)
  frame_len = int(winlen * srate)
  frame_step = int(winstep * srate)
```

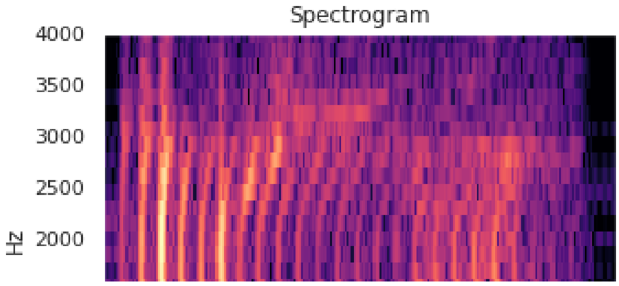
```
  numframes = 1 + int(math.ceil((1.0 * slen - frame_len) / frame_step))

  padlen = int((numframes - 1) * frame_step + frame_len)

  zeros = np.zeros((padlen - slen,))
  padsignal = np.concatenate((signal, zeros))

  indices = np.tile(np.arange(0, frame_len), (numframes, 1)) + np.tile(
              np.arange(0, numframes * frame_step, frame_step), (frame_len, 1)).T
  indices = np.array(indices, dtype=np.int32)
  frames = padsignal[indices]
  win = np.tile(winfunc(frame_len), (numframes, 1))
  return frames * win


def get_spectrum(frames, n_fft=512):
  '''compute STFT of the framed signal and
  hint: you can use build-in functions from numpy/scipy.
  @frames: input framed signal
  @n_fft: number of FFT bins

  return: magnitude spectrum
          (you can return power spectrum as well)
  '''
  complex_spec = np.fft.rfft(frames, n_fft)
  return np.abs(complex_spec)


def compute_filterbanks(spec, n_fft=512, n_filters=40,
                        srate=16000):
  '''compute mel frequency filterbank representation.
  @spec: input spectrum
  @n_fft: number of FFT bins
  @n_filters: number of mel filters
  @low_f: lowest frequency of mel filterbanks (do not change)
  @high_f: highest frequency of mel filterbanks (do not change)

  return: mel frequency representation of signal,
          in [num_frames, n_filters]
  '''

  def __hz2mel(hz):
    return 2595 * np.log10(1 + hz / 700.)

  def __mel2hz(mel):
    return 700 *(10 ** (mel / 2595.0) - 1)

  highfreq = srate / 2
  # compute points evenly spaced in mels
  lowmel = __hz2mel(0)
  highmel = __hz2mel(highfreq)
  melpoints = np.linspace(lowmel, highmel, n_filters+2)
  # our points are in Hz, but we use fft bins, so we have to convert
  #  from Hz to fft bin number
  bin = np.floor((n_fft+1) * __mel2hz(melpoints) / srate)
```

```
    # fetch filterbanks
    fbank = np.zeros([n_filters,n_fft//2+1])
    for j in range(0,n_filters):
        for i in range(int(bin[j]), int(bin[j+1])):
            fbank[j,i] = (i - bin[j]) / (bin[j+1]-bin[j])
        for i in range(int(bin[j+1]), int(bin[j+2])):
            fbank[j,i] = (bin[j+2]-i) / (bin[j+2]-bin[j+1])

    # compute mel representations
    return np.dot(spec, fbank.T)


  def log_dct(fbanks):
    '''perform log compression and DCT on mel filterbanks
    hint: what should you do before applying log?
    @fbanks: input mel frequency representation

    return: MFCCs in [num_frames, n_filters]
    '''
    fbanks = np.where(fbanks <= 0,np.finfo(float).eps, fbanks)
    logbanks = np.log(fbanks)
    feats = scipy.fftpack.dct(logbanks, type=2, axis=1, norm='ortho')
    return feats


  def cms(mfccs):
    '''perform cepstral mean subtraction over mfcc feature matrix
    @mfccs: input mfccs

    return: mean-subtracted MFCCs in [n_frames, n_filters]
    '''
    eps = 2**-30
    rows, cols = mfccs.shape

    # Mean calculation
    norm = np.mean(mfccs, axis=0)
    norm_mfccs = np.tile(norm, (rows, 1))

    # Mean subtraction
    mean_subtracted = mfccs - norm_mfccs
    return mean_subtracted


  import librosa
  def mfccs(file_path: str,
            preemphasis_coeff: float = 0.95,
            n_fft: int = 512,
            winlen: float = 0.025,
            winstep: float = 0.01,
            n_filter_banks: int = 40,
            n_mfccs: int = 20,
            winfunc: callable = np.hamming,
            sampling_rate=8000,
            verbose: bool = False) -> np.ndarray:
    """
```

```
    file_path : str
        path to .wav file
    preemphasis_coeff : float, by default 0.95
    n_fft : int, by default 512
    winlen : float, by default 0.025
    winstep : float, by default 0.01
    n_filter_banks : int, number of Mel filterbanks by default 40
    n_mfccs : int, number of cepstral coefficient, by default 20
    winfunc : callable, windowing function, by default np.hamming
    sampling_rate : int, by default 8000
    verbose : bool, if True plot some debuggin, by default False
    Returns
    -------
    np.ndarray
        Normalized MFCCs features of shape [n_frames, n_mfccs]
    """
    y, sr = librosa.load(file_path)
    if(sr != sampling_rate):
      y = librosa.resample(y, sr, sampling_rate)
      sr=sampling_rate
    preemphed = preemphasis(y)

    frames = framing(preemphed, winlen=winlen, winstep=winstep, winfunc=winfunc)

    spectrogram = get_spectrum(frames, n_fft=n_fft)

    mel_spectrogram = compute_filterbanks(spectrogram, n_fft=n_fft,
                                          n_filters=n_filter_banks,srate=sr)
    mfccs = log_dct(mel_spectrogram)
    mfccs_norm = cms(mfccs)
    hop_length=n_filter_banks
    # TODO: finish the implementation of MFCCs features extraction
    # visualization for debugging
    if verbose:
      plt.figure(figsize=(10, 12))
      librosa.display.waveplot(y, sr=sr, ax=plt.subplot(3, 2, 1))
      plt.gca().set_title('Wave plot')
      librosa.display.specshow(librosa.amplitude_to_db(spectrogram),
                               sr=sr,
                               hop_length=hop_length,
                               fmax=sampling_rate / 2,
                               y_axis='linear',
                               x_axis='time',
                               ax=plt.subplot(3, 2, 2))
      plt.gca().set_title('Spectrogram')
      librosa.display.specshow(librosa.amplitude_to_db(mel_spectrogram),
                               sr=sr,
                               hop_length=hop_length,
                               fmax=sampling_rate / 2,
                               y_axis='mel',
                               x_axis='time',
                               ax=plt.subplot(3, 2, 3))
      plt.gca().set_title('Mels-Spectrogram')
      librosa.display.specshow(librosa.amplitude_to_db(mfccs),
                               sr=sr,
```

```
                            fmax=sampling_rate / 2,
                            x_axis='time',
                            hop_length=hop_length,
                            ax=plt.subplot(3, 2, 4))
    plt.gca().set_title('MFCCs')
    ax = plt.subplot(3, 2, 5)
    ax.plot(mfccs)
    ax.set_title('Individual MFCC features')
    ax = plt.subplot(3, 2, 6)
    ax.plot(mfccs_norm)
    ax.set_title('Normalized MFCCs')
    plt.tight_layout()
  return mfccs_norm.T

# this is for testing
_ = mfccs(train['utt'][0], verbose=True)
```





Task 4: Understand frame-level, utterance-level, contextual-level features

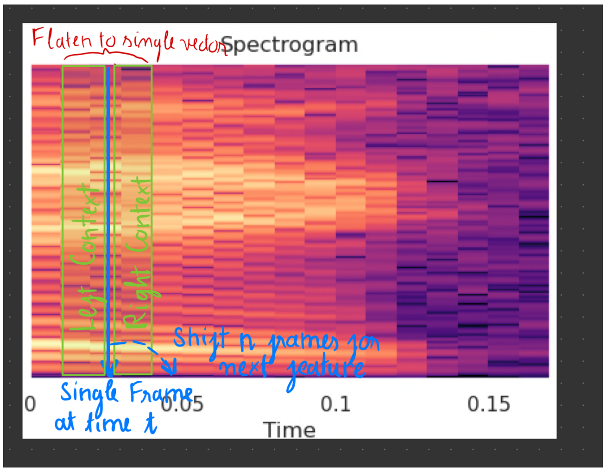
Now our classifier need to map an utterance to its digit.

The input data is a matrix of MFCCs `[n_frames, n_mfccs]` and the output is a single number `0` or `1` or ...

But our algorithm (e.g. LogisticRegression) only accepts a vector input, so

> How do we organize the features into the input for the classifier?

The following figure explain how the contextual-level features work, and how you *"roll"* the frames into a vector for training a classifier.



For reference, you could check out `process_frames_level` and `process_utterances_level` functions.

```
def process_frames_level(
    utts: List[str], dgts: List[str],
    spks: List[str], verbose: bool=False) -> Tuple[np.ndarray, np.ndarray, np.ndarr
  x = []
  y = []
  z = []
  n_frames = []
  for u, d, s in tqdm(zip(utts, dgts, spks)):
    feats = mfccs(u)
    n = feats.shape[0]
    x.append(feats)
    y += [d] * n
    z += [s] * n
    n_frames.append(n)
  x = np.concatenate(x, axis=0)
```

```
    y = np.array(y, dtype=int)
    z = np.array(z)
    assert x.shape[0] == y.shape[0] == z.shape[0]
    if verbose:
      df = pd.DataFrame({'n_frames': n_frames, 'digits': dgts, 'speakers': spks})
      sns.displot(df, x="n_frames", col="digits", bins=20, kind='hist')
      sns.displot(df, x="n_frames", col="speakers", bins=20, kind='hist')
    return x, y, z


  def process_utterances_level(
      utts: List[str], dgts: List[str],
      spks: List[str]) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
    x = []
    y = []
    z = []
    for u, d, s in tqdm(zip(utts, dgts, spks)):
      feats = mfccs(u)
      x.append(np.mean(feats, axis=0, keepdims=True))
      y.append(d)
      z.append(s)
    x = np.concatenate(x, axis=0)
    y = np.array(y, dtype=int)
    z = np.array(z)
    assert x.shape[0] == y.shape[0] == z.shape[0]
    return x, y, z


  def process_contextual_level(
      utts: List[str],
      dgts: List[str],
      spks: List[str],
      n_left: int = 3,
      n_right: int = 3,
      shift: int = 2) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
    # TODO: organizing the features into contextual level
    assert x.shape[0] == y.shape[0] == z.shape[0]
    return x, y, z


  # Example of how to use those functions
  # X_train, y_train, z_train = process_frames_level(train['utt'], train['dgt'],
  #                                                  train['spk'])
  # X_test, y_test, z_test = process_frames_level(test['utt'], test['dgt'],
  #                                               test['spk'])
  # X_train, y_train, z_train = process_utterances_level(train['utt'], train['dgt'],
  #                                                      train['spk'])
  # X_test, y_test, z_test = process_utterances_level(test['utt'], test['dgt'],
  #                                                   test['spk'])
  X_train, y_train, z_train = process_contextual_level(train['utt'], train['dgt'],
                                                       train['spk'])
  X_test, y_test, z_test = process_contextual_level(test['utt'], test['dgt'],
                                                    test['spk'])
  print()

  print(X_train.shape)
```

```
print(X_train[:2])

print(y_train.shape)
print(y_train[:2])

print(z_train.shape)
print(z_train[:2])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-36-20e130d4de8e> in <module>()
      9 #                                                   test['spk'])
     10 X_train, y_train, z_train = process_contextual_level(train['utt'], train['dgt'],
---> 11                                                      train['spk'])
     12 X_test, y_test, z_test = process_contextual_level(test['utt'], test['dgt'],
     13                                                   test['spk'])

<ipython-input-35-e737e869e6da> in process_contextual_level(utts, dgts, spks, n_left, n_right, shift)
     50     shift: int = 2) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
     51   # TODO: organizing the features into contextual level
---> 52   assert x.shape[0] == y.shape[0] == z.shape[0]
     53   return x, y, z

NameError: name 'x' is not defined
```

Ex 2: Train and evaluate spoken digits classifier (3 points)

Task outlines:

1. Train a logistic regression classifier for spoken digits
2. Evaluate the performance of your classifier
3. Fine-tuning your algorithms and try to reach at least 20% accuracy on test set

Task 1: train logistic regression algorithm for spoken digits

Hint: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

```
import torch
from torch.nn import Linear, NLLLoss, LogSoftmax, Sequential
from torch.optim import Adam
def train_logistic_regression(X: np.ndarray,
                              y: np.ndarray,
                              max_iter: int = 1000) -> LogisticRegression:
  # TODO: train and return the logistic regression
  for e in range(max_iter):
    if use_cuda:
        perm = perm.cuda()
    model.zero_grad()
```

```
    lg = model.forward(X)
    loss = criterion(lg,y)
    loss.backward()
    optimizer.step()
  return lg

lg = train_logistic_regression(X_train, y_train)
y_train_pred =lg.forward(X_train)
y_test_pred = lg.forward(X_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-13b66f6da8d3> in <module>()
     16   return lg
     17
---> 18 lg = train_logistic_regression(X_train, y_train)
     19 y_train_pred =lg.forward(X_train)
     20 y_test_pred = lg.forward(X_test)

NameError: name 'X_train' is not defined
```

SEARCH STACK OVERFLOW

Task 2: Evaluate your classifier

The following code show you classifification report for your classifier.

Then write the code for plotting the confusion matrix

Give some comments for the performance of the algorithm.

Why do your model get poor performance?

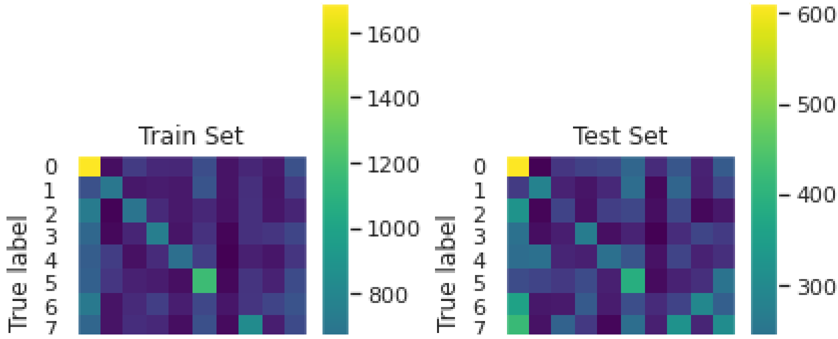
Why is there big difference in performance between training and testing set?

From our points of view, recognizing spoken digits from spectrogram image seems to be very similar to recognize number digits from pixel image, however, the performance of our classifier is much worse compared to any other image classifier. Could you explain why?

In other words, *Why are the major differences between speech and image processing in this case?*

```
print(classification_report(y_true=y_test, y_pred=y_test_pred))

plt.figure()
# TODO: plotting the confusion matrix
plt.tight_layout()
```

```
              precision    recall  f1-score   support

           0       0.20      0.34      0.25      1790
           1       0.29      0.21      0.24      1327
           2       0.13      0.13      0.13      1103
           3       0.20      0.24      0.22      1129
           4       0.25      0.17      0.20      1408
           5       0.20      0.26      0.23      1502
           6       0.18      0.06      0.09      1623
           7       0.19      0.17      0.18      1845
           8       0.24      0.23      0.23      1470
           9       0.22      0.25      0.24      1860

    accuracy                           0.21     15057
   macro avg       0.21      0.21      0.20     15057
weighted avg       0.21      0.21      0.20     15057
```



Task 3: Reach at least 20% accuracy on test set

Fine-tuning your classifier and reach at least 20% accuracy on test set

Hint:

- try different feature level (frame, utterance, contextual level),
- adjust the hyper-parameters of LogisticRegression

```
# TODO:
```

Ex 3: Unsupervised learning with Gaussian mixture model (3 points)

Task outlines:

1. Train GMMs for spoken digit classifier
2. Plotting the GMMs components
3. Evaluating your GMMs classifier and reach at least 40% accuracy on test set

Task 1: Train GMMs for spoken digit classifier

> GMM is an unsupervised learning alorithm, how do we use it for classification task which is a supervised learning problem?

The solution is very intuitive: `train an individual GMM for each digit.`

As a result, our classifier is characterized as follow:

- It is a list contain 10 different GMM
- The first element represent the first digit, and so on

> How do we perform prediction given a list of GMM for the digits?

Hint: you could use [GMM implementation]

```
def train_gmms(
    X: np.ndarray,
    y: np.ndarray,
    n_components: int = 3,
    covariance_type: Literal['full', 'tied', 'diag', 'spherical'] = 'diag'
) -> List[GaussianMixture]:
  # 'full' - each component has its own general covariance matrix
  # 'tied' - all components share the same general covariance matrix
  # 'diag' - each component has its own diagonal covariance matrix
  # 'spherical' - each component has its own single variance
  # TODO: finish your implementation of GMMs classifier
  return all_gmms
```

```
all_gmms = train_gmms(X_train, y_train)
```

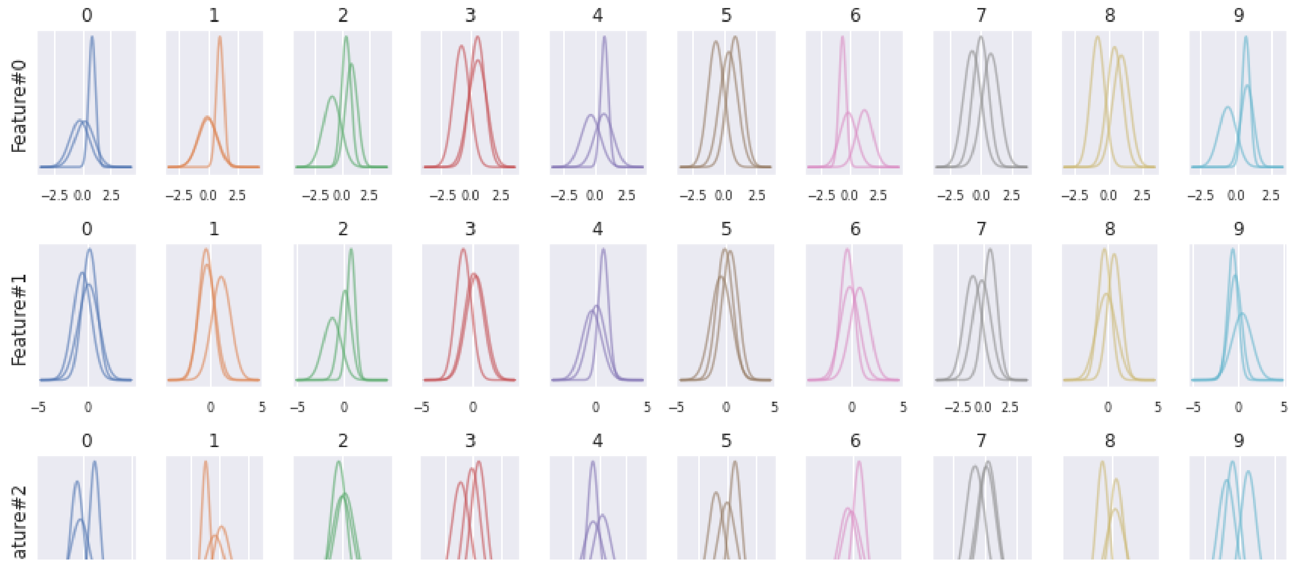
```
    Fitting GMM: 100%|██████████| 10/10 [00:03<00:00,  2.82it/s]
```

Task 2: Visualize the GMMs components for each digit

Plotting the GMMs' components:

- of the first 3 feature dimension and
- for each digits.

```
plt.figure(figsize=(12, 6))
# TODO: your code for plotting the GMM components
plt.tight_layout()
```



Task 3: Evaluating your GMMs classifier

Find out how to perform prediction given a list of trained GMM for each digit.

Then evaluate your model and plotting the confusion matrix.

What do you think about the performance of your GMMs?

Try to reach at least 40% accuracy on test set.

Does your GMMs perform better than the Logistic Classifier? Could you give an explaination why?

```
gmms_predict = lambda x: #TODO: Finish this function to make prediction from the tı
y_train_pred = gmms_predict(X_train)
y_test_pred = gmms_predict(X_test)

print(classification_report(y_true=y_test, y_pred=y_test_pred))

plt.figure()
# TODO: your code for plotting the confusion matrix
plt.tight_layout()
```

```
              precision    recall  f1-score   support

           0       0.66      0.47      0.55      1790
           1       0.61      0.53      0.56      1327
           2       0.26      0.30      0.28      1103
           3       0.33      0.44      0.38      1129
           4       0.60      0.44      0.51      1408
           5       0.48      0.29      0.36      1502
           6       0.41      0.44      0.43      1623
           7       0.32      0.41      0.36      1845
           8       0.48      0.59      0.53      1470
           9       0.45      0.48      0.47      1860

    accuracy                           0.44     15057
   macro avg       0.46      0.44      0.44     15057
weighted avg       0.47      0.44      0.45     15057
```