Homework 3

Problem 1: GMMs for Speaker Recognition

I. Motivation and Background

In this problem, we will use Gaussian Mixture Models for speaker identification, and subsequently use these models to perform speaker classification. Model training and classification will be performed using Mel Frequency Cepstrum vectors, which are computed from speech samples collected from ten speakers. Mel frequency cepstral coefficients (MFCCs) are a way of representing speech signals recorded from human beings, and you can read about them more from (https://en.wikipedia.org/wiki/Mel-frequency_cepstrum) and (https://medium.com/@tanveer9812/mfccs-made-easy-7ef383006040). For this problem, we will not be worried about converting speech samples into MFCC coefficients, we will work with MFCC coefficients directly. To motivate how we fit GMMs to MFCC coefficients, it's useful to understand a high-level picture of how MFCC coefficients are extracted from a speech utterance.

Mel Frequency Cepstral Coefficients

For performing speech recognition, it's very important to extract meaningful features from the audio signal. These features should identify the linguistic content and discard signal noise, emotional cadence etc. Since sounds generated by human beings are filtered by the vocal tract of a person, the shape of the vocal tract manifests in the envelope of the short time power spectrum. MFFCs, by virtue of their formulation, accurately represent this envelope. The broad strokes of the algorithm are as follows

1. Divide the audio signal into short frames (overlapping windows, shown in figure 1).
2. For each frame, calculate the periodogram estimate of the power spectrum.
3. Apply Mel filters to the power spectrum, and sum the energy obtained in each filter.
4. Take the logarithm of the filter energies.
5. Take the DCT of the log filter energies and keep 20 components.

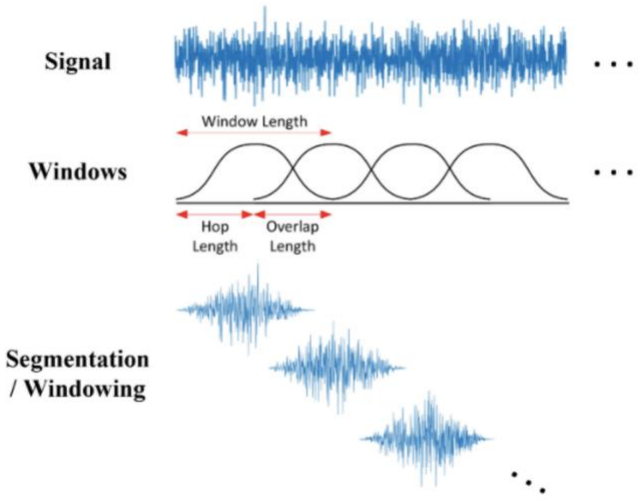

Figure 1: Generating speech frames.

We don't need to worry about the exact algorithm, but it is important to remember that the raw audio signal is divided into multiple overlapping frames, and for each frame we calculate a 20-dimensional MFCC coefficient. The distribution of these MFCC coefficients across all the frames over an audio signal for a speaker are usually powerful discriminative tools. For this problem, you will fit GMMs to the MFCC coefficients for each speaker, which we'll discuss in more detail below.

II. Data

You can download the **HW3_Problem1_template** folder from Canvas. In the **data** folder, you have ASCII-formatted training and testing data for each speaker. You have a train and test folder, along with a utt2spk file, explained below -

1. train: The train directory contains 2 files for each speaker, for a total of 20 files. Each file contains the 20-dimensional MFCC vectors from a speech recording from a subject. Therefore, each file contains data of the form n × 20, where n is the number of frames that the raw audio recording was divided into, and each frame has a 20-dimensional MFCC vector associated with it.
2. test: The test directory contains 10 files and have the same format as the training directory. Each test file contains a varying n number of 20-dimensional feature vectors from the same speaker. You can use Numpy's genfromtxt or any method (built-in or your own) you want.
3. utt2spk: This file contains 2 columns; column 1 represents the filename of the training sample, and column 2 represents the speaker identity. There are a total of 10 speakers whose identities are {110667-m, 106888-m, 3424-m, 102147-m, 103183-m, 101188-m, 4287-f, 2042-f, 7722-f, and 4177-m}. The m and f suffixes refer to the gender of the speaker, and the prefixes are the UIDs of the speakers.

III. Methodology

It is important to note that we use GMMs here in a somewhat different application to those discussed in the lectures. Here, we represent the distribution of parameters for each class (each speaker) using a different GMM model, for a total of 10 GMM models (one for each speaker). Then, given testing data, we calculate the posterior probability of the testing data belonging to each of these 10 classes and classify the testing data as belonging to the class whose GMM gave the highest posterior probability of explaining the data.

1. **EM Algorithm for training GMMs**: Estimate the Gaussian Mixture Model parameters for each speaker separately in the training data (we will have 10 different GMM models).

   - The training data for speaker j is of the form $X^{(j)} = \{X_1^j, X_2^j, X_3^j, \ldots, X_n^j\}$, where $X_i^{(j)} \in \mathbb{R}^{20}$
   - For each speaker, train a 20-dimensional Gaussian Mixture Model with k = 64 Gaussians, each assuming diagonal covariance. Therefore, we're representing each MFCC vector as,

     $$P\left(X_i^{(j)}\right) = \sum_{k=1}^{64} \pi_k^{(j)} G\left(X_i^{(j)} \middle| \mu_k^{(j)}, \Sigma_k^{(j)}\right) \ldots\ldots\ldots (1)$$

     where the multivariate Gaussian distribution for 20 dimensions is defined as

     $$G\left(X_i^{(j)} \middle| \mu_k^{(j)}, \Sigma_k^{(j)}\right) = \frac{1}{\sqrt{(2\pi)^{20}\left|\Sigma_k^{(j)}\right|}} \exp\left(-0.5\left(X_i^{(j)} - \mu_k^{(j)}\right)^T \left(\Sigma_k^{(j)}\right)^{-1}\left(X_i^{(j)} - \mu_k^{(j)}\right)\right) \ldots\ldots\ldots (2)$$

     Here, $X_i^{(j)}, \mu_k^{(j)} \in \mathbb{R}^{20}, \Sigma_k^{(j)} \in \mathbb{R}^{20*20}$
   - The joint probability distribution (likelihood) for the training data of one speaker, assuming each $X^{(j)}$ is independent, is simply the product of marginal probability distributions, given by

     $$P(X^j | \pi^j, \mu^j, \Sigma^j) = \prod_{i=1}^{n} P\left(X_i^{(j)}\right) \ldots\ldots\ldots (3)$$

And therefore,

$$\log P(X^j|\pi^j,\mu^j,\Sigma^j) = \sum_{i=1}^{n} \log P\left(X_i^{(j)}\right) \dots\dots\dots (4)$$

- For performing expectation-maximization, you will want to maximize the log-likelihood with respect to the parameters of the Gaussians $\pi^{(j)}$, $\mu^{(j)}$, $\Sigma^{(j)}$ for k = 1, . . ., 64. **Mathematically derive the closed-forms of the updates** for the Expectation Maximization algorithm for training the parameters of these GMMs. You can simplify the notation here to work with just one speaker for now for your mathematical derivation. Please include your mathematical derivations and the steps of the EM algorithm in your notebook.
- Implement your derived EM algorithm above in Python, and train 10 GMMs, one for each speaker's training data. Run your EM algorithm for maximum 20 steps or until convergence to ε = 1e−4, whichever is first.
- You can use GaussianMixture from sklearn.mixture to compare with your results. Again, use exactly the same parameters; k = 64, and diagonal covariance matrices.

2. **Classification**: We can summarize the algorithm for classification given the trained GMM coefficients as follows
   - Compute the log-likelihood of a test data sequence $X = \{X_1, X_2, \dots, X_n\}$ given the trained speaker GMM parameters $\Theta^{(j)} = \{\pi^{(j)}, \mu^{(j)}, \Sigma^{(j)}\}$ for j=1, …, 10, which is given as

$$\log P(X|\theta^j) = \sum_{i=1}^{n} \log P(X_i|\theta^j) \dots\dots\dots (5)$$

   - We can then classify the test data sequence X by assigning it to the speaker that gives the highest log-likelihood with their GMM model, as follows

$$\arg\max_j \log P(X|\theta^j) \dots\dots\dots (6)$$

   - You can use the score function from GaussianMixture to calculate the average log-likelihoods for the GMMs returned by GaussianMixture to compare with your results.
   - Report classification results for both your implementation and using sklearn's in-built functions for the 10 test samples. You should report your results in a text file test_predictions.txt, where the first column is the name of the test file, the second column is the prediction using your implementation, and the third column is the pre- diction using sklearn's in-built functions. A valid submission would look like the following (assuming your classifications differ for the first test)

     test1 110667-m 2042-f
     test2 110667-m 110667-m
     ...
     test10 106888-m 106888-m

Problem 2: Bayesian Probability, Prior and Posterior

Let X be a random variable representing a biased coin with possible outcomes $X = \{0, 1\}$. The bias of the coin is controlled by a parameter $\theta \in [0, 1]$, so

$$p(X = 1 \mid \theta) = \theta$$

$$p(X = 0 \mid \theta) = 1 - \theta$$

Or, more compactly

$$p(X = x) = \theta^x (1 - \theta)^{1-x}$$

We would like to learn what the parameter θ is, based on experiments by flipping the coin. When there is no knowledge on what θ is before flipping the coin, it is reasonable to assume the prior distribution on θ follows a uniform distribution on the unit interval,

$$p(\theta) = \mathcal{U}([0, 1]) := \mathbb{1}_{[0,1]}(\theta)$$

where $\mathbb{1}_E(x)$ is the indicator function on the set E

$$\mathbb{1}_E(x) := \begin{cases} 1 & x \in E \\ 0 & x \notin E \end{cases}$$

We flip the coin many times, and it is independent and identically distributed, i.e., i.i.d., each time. After each coin flip, we update the distribution for θ to reflect what we learn from the coin flip.
Suppose we observed the sequence $x_{1:3} = 101$ (i.e., we saw '1' followed by '0' and '1' when flipping the coin three times). For each subsequence x_1, $x_{1:2}$ and $x_{1:3}$,

a) Compute the posterior distributions;
b) The expectation value μ;
c) The variance σ^2;
d) The maximum posteriori estimation θ_{MAP}

You should show your derivations step by step and present your final results in the table as shown below. The first one has been done for you.

Posterior	**PDF**	$\boldsymbol{\mu}$	$\boldsymbol{\sigma^2}$	$\boldsymbol{\theta_{MAP}}$
$p(\theta)$	$\mathbb{1}_{[0,1]}(\theta)$	$1/2$	$1/12$	Any θ in $[0, 1]$.
$p(\theta \mid x_1 = 1)$	?	?	?	?
$p(\theta \mid x_{1:2} = 10)$	?	?	?	?
$p(\theta \mid x_{1:3} = 101)$	?	?	?	?

e) Plot each of the posterior distributions above.
f) What behaviour would you expect of the posterior distribution $p(\theta \mid x_{1:n})$ if $x_{1:n} = 1010101010$ …? Hint: You can try to plot the distributions for $n = 1,2,3,4,5$ … and find the trend/behaviour by intuition.

Problem 3: Overcomplete sparse presentation

In this problem, the goal is to build a language recognition system based on I-vector representation. The I-vector presentation approach consists of representing speech segments in single space. This space will felicitate the comparison between all the recordings. In the directory /Problem1/data/, you will find three directories Train, Dev and Eval. These directories correspond respectively to the training data that will be us to train your classifier, development data to tune your classifier and evaluation data to evaluate the different models. In each directory you will find 24 files that correspond to 24 different language classes. Each line of these 24 files corresponds to the I-vector x of dimension 600 and represents a speech recording. The goal of this problem is to build two types of classifier based on Linear Discriminant Analysis (LDA) and sparse dictionary representation.

I) Write the code that applies LDA and cosine scoring to the I-vectors provided in /Problem1/data/. The classification algorithm based on LDA and cosine scoring is described as follow

- Normalizing the length of all I-vectors x (all directories).

$$w = \frac{x}{\|x\|}$$

- **LDA training**

  - Estimate the between class covariance

$$S_b = \sum_{l=1}^{L} n_l (w_l - \bar{w})(w_l - \bar{w})^t$$

  - Estimate the within class covariance

$$S_w = \sum_{l=1}^{L} \sum_{i=1}^{n_l} (w_i^l - w_l)(w_i^l - w_l)^t$$

  where

$\overline{w}$: the mean of the all I-vectors

$w_l = \frac{1}{n_l}\sum_{i=1}^{n_l} w_i^l$ mean for language class l

$n_l =$ number of I-vectors for each language class l

$L =$ total number of language classes

The goal is the estimate LDA matrix V that solves the following Eigen problem $S_b.v = \lambda.S_w.v$ (Use the eigs function to compute the eigenvector). If you have L different classes, LDA will provide L-1 non-zero eigenvalues.

- **Classifier training**
  - Project the i-vectors with LDA matrix V then length normalized the obtained projected I-vectors

    $$w' = \frac{V^t w}{\left\| V^t w \right\|}$$
  - For each class l compute the mean and normalized its length

$$m_i = \frac{\frac{1}{n_l}\sum_{j=1}^{n_l} w'}{\left\| \frac{1}{n_l}\sum_{j=1}^{n_l} w' \right\|}$$

- **Classifier testing**
  - Project the test i-vectors in both dev and eval directories by LDA matrix V

    $$w'_{test} = \frac{V^t w_{test}}{\left\| V^t w_{test} \right\|}$$
  - Compute the dot product of the test i-vector with the normalized mean of each class m_i

    $score_i = w'_{test} * m_i$
  - Select the language class number i that corresponds to do highest score
  - Compute both recognition accuracy rates for dev and eval data

    $$Acc_{dev} = \frac{\text{number of correct classified I-vector in dev directory}}{\text{total number of I-vector in dev directory}}$$

$$Acc_{eval} = \frac{\text{number of correct classified I-vector in eval directory}}{\text{total number of I-vector in eval directory}}$$

II) In the question the goal is to write the code that classifier the I-vector x based on overcomplete space representation (see similar work for face recognition http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4483511&tag=1). If you load each of the training data file in matrix D_c (the column of this matrix correspond to the I-vector which is one line in the training file), you can build your dictionary for the overcomplete sparse presentation by concatenating all the 24 matrices in single large matrix $A=[D_1, D_2, ..., D_{24}]$.
For each test example in Dev and Eval directories uses Lasso algorithm to estimate the new weights in the Dictionary space.

$$\min\left\{\|x - A\alpha_i\|^2 - \lambda\|\alpha\|_1\right\}$$

where $\alpha^t = \left[\alpha_1^t, \alpha_2^t, ..., \alpha_{24}^t\right]$ and each α_c^t is a vector of weights corresponding to the language class c.

The classification in the overcomplete sparse presentation is done based on minimizing the reconstruction of the I-vector x based on each D_c matrix and its weights α_c. This task consists of finding the language class that produces the small reconstruction error for each I-vector $\underset{c}{\operatorname{argmin}}\left\{\|x - D_c\alpha_c\|^2\right\}$ on both directories for Dev and Eval then compute the recognition accuracy rates?

$$Acc_{dev} = \frac{\text{number of correct classified I-vector in dev directory}}{\text{total number of I-vector in dev directory}}$$

$$Acc_{eval} = \frac{\text{number of correct classified I-vector in eval directory}}{\text{total number of I-vector in eval directory}}$$

III) In this question, please redo the same process as question (II) after projecting first all the I-vectors by LDA, which was trained in question (I). Evaluate again the recognition accuracy rates for both Dev and Eval directories? If both accuracies are better than question (II), explain why?

IV) To speed up the process, train a code book using k-means for each language class with k=55. Each language will be only represented by 55 centroids. The classes that have already 55 I-vectors take all of them (do not apply kmeans). For kmeans implementation used the function already predefines in python package.

Apply the overcomplete space representation classification again. Similar to question III use the after projecting first all of them by LDA

V) Apply Gaussian classifier as it will be seen in the class for this task with the assumption of equal prior for all the language classes and shared covariance between all the classes. In this part use the **original I-vector of dimension 600**. Compare your results with previous ones.

For reference look to
http://www.fit.vutbr.cz/research/groups/speech/publi/2011/martinez_interspeech2011_291.pdf

Submission Instructions

Please follow these instructions closely. We will deduct points for not sticking to the template. Solutions should be uploaded to CANVAS.

Your prediction results for Question 1 should go in the problem_1/results/ folder as a text file named test_predictions.txt.

For Question 3, please submit a separate report including theoretical answers and accuracies for all subquestions

Use the Jupyter notebook template for all problems in google collab in the following link-

https://colab.research.google.com/drive/1c5Jj4XqE5jmjemtCoSrBJPDHMbpgH_6o?usp=sharing

Note: Points will be deducted if your code doesn't match the required output format mentioned above. Your code should not output anything else.

Instructions for Python:

You will have to use Jupyter Notebook for Python. We provided the template in Google Colab. Instructions are given in the link. You have to submit the specific deliverables mentioned below. Put them inside "results" folder inside the designated problem folder. You have to submit the "ipynb" file, ".py" file and "Hw3_yourJHID.zip" zip file containing the required results or reports. In the text comments on Canvas, put the link to your Google colab notebook with proper permissions.