Exercise #3:
Generative models and phone classification

Exercise summary:

The third exercise focuses on building a phone classifier for speech feature frames using a parametric probabilistic model. Phone classification refers to the assignment of speech features into phonetic categories. Parametric models are statistical models of speech features defined by the distribution of choice together with its known (or estimated) parameters, such as normal distribution with its mean and variance parameters.

The exercise will start with a single parametric distribution to model each phone class. In the second stage, the approach is extended to a mixture model consisting of several parametric distributions.

Software: MATLAB.

Data: Utterances and phone annotations of LibriSpeech corpus clean-dev section. Please download the data from: *https://www.openslr.org/resources/12/dev-clean.tar.gz* (audio) and https://zenodo.org/record/2619474#.YeULdCzkuL4 (annotations in .TextGrid format). Contact course assistants if you have trouble accessing the dataset.

Tasks:

0) Download and pre-process data (all necessary code pre-provided in E3_main.m)
1) Implement multivariate Gaussian classifier training scripts (trainGaussian.m)
2) Implement multivariate Gaussian classifier inference scripts (testGaussian.m)
3) Implement multivariate Gaussian mixture model classifier training scripts (trainGMM.m)
4) Implement multivariate Gaussian mixture model classifier inference scripts (testGMM.m)

Deliverables: A written report (`E3_surname_report.pdf`) with a title page including your name and student number + content pages with written answers and figures requested in Tasks 1) and 2), answers to Questions 1.1–2.4, and 5 completed MATLAB script files: `E3_main.m`, `trainGaussian.m`, `testGaussian.m`, `trainGMM.m`, `testGMM.m`. Archive all files to `E3_firstname_surname_studentID.zip` and uploaded to Moodle.

Learning goals: Parametric modeling, phone classification, mixture models, EM-algorithm

Note: All exercises of the course, including this one, will consist of a written report and additional files (e.g., code, sounds, annotation files) that are to be created and submitted for evaluation. In the report, always add your name and student ID to the beginning of the report. Use complete English sentences or paragraphs of text to answer the exercise questions, and number each response according to the question numbers in the exercise instructions. Some of the questions have strictly correct answers while others can have multiple valid responses.

Note 2: The exercise will be semi-automatically scored based on the variables and functions defined in the MATLAB code template. Do not change the naming of the key variables or specifications of function templates and calls defined. If you fail to complete the full assignment, in order to get partial points of correct parts, make sure your code runs up to the point that you managed to complete.

Task 0: Download and pre-process data for classification experiments

Task 0.1: Download LibriSpeech audio and alignments from links listed above. Place annotations in `librispeech_alignments` folder under `LibriSpeech` main folder.
Task 0.2: Pre-process the data into features and labels. All code is pre-provided. Please follow the code and comments in `E3_main.m` for pre-processing the LibriSpeech data into speech features and corresponding phone labels.

Task 1: Implement a Gaussian model (GM) for phone classification

In order to perform phone classification, we need a classifier for the data. The goal here is to implement a multivariate Gaussian model estimation in `trainGaussian.m` and model-based speech classification in `testGaussian.m` to be called by the `E3_main.m`.

Input to the training consists of a matrix of MFCC feature vectors (‘`x_train`’) and the corresponding phone class labels (‘`labels_train`’). Output of `trainGaussian()` is a MATLAB struct ‘`GM`’ containing the model parameters for each training phone class *c* together with prior probabilities *p*(*c*) for the classes.

Input to `testGaussian()` consists of test sample MFCC vectors (‘`x_test`’) and the model ‘`GM`’. The script should output relative log-likelihoods ‘`loglik`’ of each class together with class predictions ‘`predicted_labels`’ for each input sample.

Class predictions are fed to an evaluation function `evaluateClassification()` that compares the predicted classes to actual test set phone classes (code provided). Output of the evaluation is accuracy (proportion of samples correctly classified), which we will use as the primary performance metric. The function also outputs unweighted average recall (UAR), which is the mean of class-specific accuracies, hence ignoring how frequent/rare each class is in the test data. In addition, classification confusion matrix is given as an output.

Guidance: See Appendix A.

Task 1.1: Implement `trainGaussian.m` based on the initial template and using full covariance matrices for the Gaussian models.
Task 1.2: Implement `testGaussian.m` based on the initial template.
Task 1.3: Run GM training with the training data and inference with the test data using the above-implemented scripts, and then execute `evaluateClassification()` and then `printConfusionMatrix()` with your GM model class hypotheses for the test data. *In your report*, write down the phone classification accuracy you obtain with the Gaussian classifier while using the pre-defined data split. Also add i) the produced confusion matrix and ii) automatically produced figure on model parameters (with `printModelParameters()`) to the report.

Also answer to the below questions:

Question 1.1: Why is Gaussian distribution a meaningful choice to model speech features such as MFCCs?
Question 1.2: What are the main shortcomings of using a single multivariate Gaussian

distribution to classify speech feature vectors?

Question 1.3: Looking at the confusion matrix, what kind of classification errors seem to be common with the model? (all phones are listed in array '`unique_phones`')

***Task 1.4 (optional, not graded)*:** Implement and test diagonal covariance Gaussian models in addition to full covariance Gaussian models. What happens to the classification performance?

Task 2: Implement a Gaussian mixture model (GMM) for phone classification

In this step, we will extend the concept of Gaussian model to a Gaussian mixture model, i.e., where data from one class is modelled with a mixture of several Gaussians instead of one. Your task is to complete the workflow in `E3_main.m` by implementing `trainGMM.m` and `testGMM.m` functions for GMM model training and testing in the phone classification task. Then you can train a GMM using the given LibriSpeech data and report the performance of the system.

The basic specifications of these functions are equivalent to Task 1, except now the model is stored in struct '`GMM`' which contains a separate GMM for each phone class in the training data. See Appendix B for theory and practical guidance.

Since GMMs do not have a known globally optimum solution, their estimation is based on iterative parameter estimation starting from a set of initial parameters. In `trainGMM.m`, one possible solution for initialization of the model parameters is provided as ready-made code. In addition, a ready-made solution is provided for stopping the iterative training when the model's fit to data no longer improves.

Task 2.1: Implement `trainGMM.m` based on the initial template.

Task 2.2: Implement `testGMM.m` based on the initial template.

Task 2.3: Run GMM training with the training data and inference with the test data using the above-implemented scripts, and then execute `evaluateClassification()` and `printConfusionMatrix()` with your GMM model class hypotheses for the test data. Also run `printLogLiks()` to plot average log-likelihood across the classes as a function of EM iteration number.

In your report, write down the phone classification accuracy you obtain with the GMM classifier while using the pre-defined data split. Also add to the report the i) produced confusion matrix, ii) hyperparameter values of your GMM used to produce the results (maximum number of iterations, number of Gaussian components), and iii) the image produced by `printLogLiks()`.

Task 2.4: Run the code at the end of `E3_main.m`, which produces an example decoding output for a single chosen speech utterance located at `extra_data/` together with its spectrogram. Add the figure based on GMM outputs to your report.

Guidance: See Appendix B.

Answer to the following questions:

Question 2.1: What is the added value of using more than one Gaussian component in modeling the data?

Question 2.2: What are the drawbacks in increasing the number of Gaussian components?

Question 2.3: Are the classification confusions between phone categories similar in the GMM to those of the single univariate Gaussian model?

Question 2.4: If the aim would be to recognize phones from continuous speech signals, what kind of conceptual or technical shortcomings does the current classification approach have?

***Task 2.5 (optional, not graded)*:** You can try to improve your GMM classification performance by improving the parameter initialization (at the beginning of `trainGMM.m`) or by exploring a suitable set of hyperparameters.

Returning the exercise

Return a written report (*`E3_surname_report`*`.pdf`) with a title page including your name and student number + content pages with written answers and figures requested in Tasks 1) and 2), answers to Questions 1.1–2.4, and 5 completed MATLAB script files: `E3_main.m`, `trainGaussian.m`, `testGaussian.m`, `trainGMM.m`, `testGMM.m`. Archive all files to `E3_firstname_surname_studentID.zip` and upload it to Moodle.

Appendix A: Guidance for Task 1

For a single multivariate Gaussian, the probability density function value for sample **x** is defined as:

$$p(\mathbf{x}|\theta) = p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{\exp\left(-0.5(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}} \qquad (1.1)$$

where $\boldsymbol{\mu}$ is the mean, $\boldsymbol{\Sigma}$ the covariance, $|\boldsymbol{\Sigma}|$ the determinant of covariance, and *d* is the dimensionality of **x**. In other words, to "train" a single Gaussian model of some data $\mathbf{x}_i \in \mathbf{X}$, it sufficient to estimate the mean and covariance of the data (`mean()` and `cov()` in MATLAB).

In our classification case, we consider each class $c \in \mathrm{C}$ to have mutually independent training samples, so we can simply estimate a separate mean vector and covariance matrix for each *c*. As a result, we can evaluate $p(\mathbf{x}|\, c, \theta)$ for any sample **x** and class *c*, where θ is the set of model parameters $\{\boldsymbol{\Sigma}, \boldsymbol{\mu}\}$ for each *c*.

However, in order to classify data points into classes $c \in \mathrm{C}$, one is interested in the class posterior probability $p(c \mid x, \theta)$ instead of data likelihood $p(x \mid c, \theta)$. By applying Bayes' rule, this can be obtained with

$$p(c \mid x, \theta) = \frac{p(c)p(x \mid c, \theta)}{p(x)} \qquad (1.2)$$

where *p*(*c*) is simply the prior probability of observing class *c*. This can be empirically estimated from the training data by simply counting and normalizing the relative frequencies of each phone class in the training data, i.e.,

$$p(c) = \frac{\text{frequency}\{c\}}{\sum_c \text{frequency}\{c\}} \qquad (1.3)$$

The denominator in Eq. (1.2) consists of the term *p*(**x**), the overall probability of observing data point **x**. Since this is independent of the class, we can rewrite Eq. (1.2) as

$$p(c \mid \mathbf{x}, \theta) \sim p(c)p(\mathbf{x} \mid c, \theta) \qquad (1.4)$$

and then find the class hypothesis as

$$\arg_c\max\{p(c \mid \mathbf{x}, \theta)\} = \arg_c\max\{p(c)p(\mathbf{x} \mid c, \theta)\} \qquad (1.5)$$

In practical algorithms, dividing and multiplying probabilities can sometimes get numerically very small and cause floating point underflow. Since the logarithm of positive values is a monotonically increasing function, one can re-write Eq. (1.5) as

$$\arg_c\max\{p(c \mid \mathbf{x}, \theta)\} = \arg_c\max\{\ln\big(p(c)\big) + \ln\left(p(\mathbf{x} \mid c, \theta)\right)\}$$

$$= \ln\big(p(c)\big) + \ln\left(\exp\left(-0.5(\mathbf{x}-\boldsymbol{\mu}_c)^{\mathrm{T}}\boldsymbol{\Sigma}_c^{-1}\ (\mathbf{x}-\boldsymbol{\mu}_c)\right)\right) - \ln\left(\sqrt{(2\pi)^d|\boldsymbol{\Sigma}_c|}\right) \qquad (1.6)$$

where $\ln\left(p(\mathbf{x} \mid c, \theta)\right)$ is referred to as the log-likelihood of **x**, given the model for *c*. Since ln() and exp() cancel each other out, this expression can be further simplified if needed (see also https://en.wikipedia.org/wiki/Likelihood_function#Log-likelihood and https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Likelihood_function).

On diagonal covariance. Training of the Gaussian model included the estimation of a covariance matrix $\boldsymbol{\Sigma}$, which is a *d x d* matrix with data dimensionality *d*. One practical advantage of MFCC features over some alternatives, such as log-magnitude spectrum or log-Mel spectrum, is that the

MFCC feature dimensions are partially decorrelated with each other due to the DCT operation in its calculation. Due to this property, it used to be common to use so-called *diagonal covariance* matrices for modeling MFCC-like features: instead of using full pair-wise covariance information between all possible feature dimensions, one simply measures the variances $\boldsymbol{\sigma}^2 = \{\sigma_1^2, \sigma_2^2, \dots, \sigma_d^2\}$ of each individual feature dimension. These values, by definition of covariance matrix, lie on the diagonal of $\boldsymbol{\Sigma}$ (hence the name) and rest of the matrix can be defined as zero.

The advantage of the diagonal covariance approach is that much less data is needed to obtain robust estimates for the model parameters, potentially improving performance on small datasets. However, this naturally limits the modeling power of the approach on larger datasets where full covariance estimation is feasible.

Appendix B: Guidance for Task 2

A GMM is essentially a set of *K* multivariate Gaussian models $N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), k = [1, 2, \dots, K]$, *k* also often referred to as Gaussian *components*, and where each of the components have their own mean $\boldsymbol{\mu}_k$, covariance $\boldsymbol{\Sigma}_k$, and a mixing weight w_k with $\sum_k w_k = 1$.

The likelihood of **x**, given a GMM, is then estimated as a linear mixture of the component-specific likelihoods according to the weights, formally defined as:

$$p(\mathbf{x}|\theta) = \sum_{k=1}^{K} w_k p(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \sum_{k=1}^{K} w_k \frac{\exp\left(-0.5(\mathbf{x}-\boldsymbol{\mu}_k)^{\mathrm{T}}\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right)}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}_k|}} \quad (2.1)$$

i.e., as a weighted sum over probabilities that the data point would be generated by each of the involved Gaussian components with w_k as the component weights.

Despite the apparent simplicity of the model (a collection of Gaussians), such a model can approximate any "smooth"[1] data distribution **x** ~ $f(\phi)$ down to an arbitrarily small error ε with a sufficiently high *K*. However, estimating the parameters of such a model from finite data is another issue, that we will turn into next.

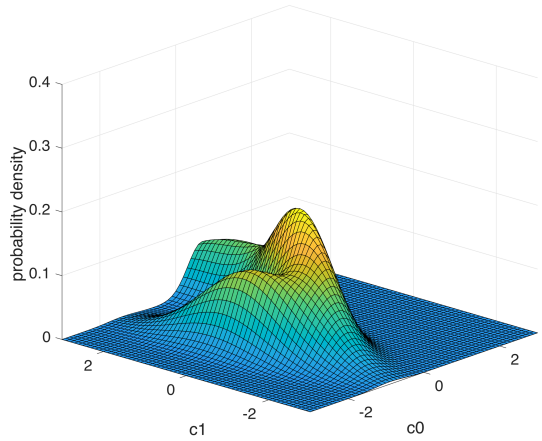

Figure 2.1: A visualization of a three-component GMM for 2-dimensional data.

The standard practice for training GMMs is to use Expectation Maximization (EM) algorithm (Dempster et al., 1977). EM is an iterative process that is guaranteed to converge to a locally optimum solution, where the local optimum depends on the number of components *K*, initial parameters $\boldsymbol{\mu}_k^0, \boldsymbol{\Sigma}_k^0$, and w_k^0 for each $k \in K$, and on input data $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \in \mathbf{X}$.

The "intuitive logic" is as follows: first, on the first iteration *j* = 0, the initial parameters of the model are used to calculate the posterior probabilities $h^0(i,k) = P(k \mid \mathbf{x}_i, \boldsymbol{\mu}_k^0, \boldsymbol{\Sigma}_k^0, w_k^0)$, also known as *responsibilities*, that each of the *K* components would have generated each training observation $\mathbf{x}_i$. This is known as the expectation step, or E-step. New means, covariances, and mixing weights of each component *k* are then estimated from all $\mathbf{x}_i$ by weighting the observations according to the probability $h^0(i,k)$ that the given component *k* generated the observation *i*. This is known as the maximization (M) –step. The new parameters are then again used to estimate the component posteriors etc., repeating the E and M steps in turn until the data likelihood (model's fit to data) no longer improves.

Mathematically, the E-step on iteration *j* can be written as:

[1] This is "common knowledge" in the field, but finding a source for this would be useful!

$$h^j(i,k) = \frac{w_k^j p(\mathbf{x}_i \mid \boldsymbol{\mu}_k^j, \boldsymbol{\Sigma}_k^j)}{\sum_{k=1}^K w_k^j p(\mathbf{x}_i \mid \boldsymbol{\mu}_k^j, \boldsymbol{\Sigma}_k^j)} \quad (2.2)$$

and the M-steps as:

$$w_k^{j+1} = \frac{1}{N}\sum_{i=1}^N h^j(i,k) \quad (2.3)$$

$$\boldsymbol{\mu}_k^{j+1} = \frac{\sum_{i=1}^N h^j(i,k)\mathbf{x}_i}{\sum_{i=1}^N h^j(i,k)} \quad (2.4)$$

$$\boldsymbol{\Sigma}_k^{j+1} = \frac{\sum_{i=1}^N h^j(i,k)[\mathbf{x}_i - \boldsymbol{\mu}_k^j][\mathbf{x}_i - \boldsymbol{\mu}_k^j]^{\mathrm{T}}}{\sum_{i=1}^N h^j(i,k)} \quad (2.5)$$

In practice, GMM training is sensitive to the initial parameters and the number of components. In addition, the training may lead to numerical stability issues. This usually happens with a large *K* where one or more of the components start to specialize to individual data points. As a result, covariance of the component approaches zero and the determinant becomes 0, preventing calculation of $\boldsymbol{\Sigma}^{-1}$. Different heuristic and principled strategies have been developed to tackle the issue of finding the optimal *K*, but are beyond the present scope (for those who are curious: see, e.g., Bayesian non-parametric models and infinite mixture models). In the present exercise, a pre-defined "hack" to ensure numerical stability is provided with the code frame for `trainGMM.m`.

Phone classification with GMMs. In order to perform phone classification with GMM, a separate GMM needs to be trained for each phone class (cf. Task 1, where a separate Gaussian was trained for each phone class). The `trainGMM.m` comes with existing skeleton code for looping across different classes and structuring the models into a cell array, so you only need to worry about 1) implementing the EM-algorithm correctly, 2) converting GMM output likelihoods from Eq. (2.1) to class posteriors similarly to Task 1 in Eqs. (1.2–1.5).

References

Dempster, A. P.; Laird, N. M.; Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society, Series B. 39(1), pp. 1–38.*