CS 461: Machine Learning Principles
Fall 2021

Problem Set #5 (80 Points)

Out: Sunday, November 28

Due: Friday, December 10, 8:00pm

Instructions

How and what to submit? Please submit your solutions electronically via Canvas. Please submit one file:

1. The empirical component of the solution (Python code and the documentation of the experiments you are asked to run, including figures) in a Jupyter notebook file. Rename the notebook ⟨`firstname-lastname`⟩`-sol5.ipynb`.

Late submissions: there will be a penalty of 20 points for any solution submitted within 24 hours past the deadline. No submissions will be accepted past then.

What is the required level of detail? See below.

(a) When asked to **derive** something, please clearly state the assumptions, if any, and strive for balance: justify any non-obvious steps, but try to avoid superfluous explanations.

(b) When asked to **plot** something, please include in the `ipynb` file the figure as well as the code used to plot it. If multiple entities appear on a plot, make sure that they are clearly distinguishable (by color or style of lines and markers) and references in a legend or in a caption.

(c) When asked to **provide a brief explanation or description**, try to make your answers concise, but do not omit anything you believe is important. If there is a mathematical answer, provide it precisely (and accompany by succinct wording, if appropriate).

(d) When **submitting code (in Jupyter notebook)**, please make sure it's reasonably well-documented, runs, and produces all the requested results. If **discussion** is required/warranted, you should include it directly in the notebook (using the markdown cell).

Collaboration policy: collaboration is allowed and encouraged, as long as you (1) write your own solution entirely on your own, and (2) specify names of student(s) you collaborated with in your writeup.

1 Gaussian Mixture Models (GMMs)

Task Our goal will be to use GMMs in a generative classifier for classifying images into the ten Fashion MNIST classes. We have provided most of the code implementing EM for GMM learning (as usual, with some missing pieces you need to fill in).

Dataset We will use the Fashion MNIST dataset. To speed things up and reduce the memory footprint of our code, we have reduced the dimension of the problem by resizing the images to 16×16 pixels, from the original 28×28. While this does lead to some reduction in accuracy for many methods, the images are still possible to classify pretty accurately.

GMM In this section we will learn generative classifiers with Gaussian label-conditional distributions. We will consider learning both a single Gaussian as well as a GMM for each such distribution. We will compare the performance of learning while restricting the covariance matrices to be diagonal for each Gaussian, as well as learning full covariance matrices.

EM To fit the Gaussian mixture models, we will use the *expectation maximization (EM) algorithm*. When we talked about EM in lecture, we focused on an unsupervised learning setting where we only had x's and we were trying to estimate the density $p(x)$ which we parameterized using a GMM. Here, we are doing classification and therefore we are estimating separate densities $p(x|y)$ for each label y. We will use a GMM for each density, so we will be running EM separately for each class's density.

A few practical considerations when implementing EM:

- The EM algorithm is sensitive to initialization. You may find that initializing all the numbers in the model to some small random values will not give good results. A simple method that works well in practice is to randomly assign data points to each Gaussian component and then calculate the corresponding means of those points in each component. Another heuristic is to set the first mean to a randomly chosen data point; the second to the point farthest from the first; the third, to the point farthest from the first two; etc.

- If the values on the diagonal of Σ get close to 0, overfitting and instability can occur. You may need to prevent this "covariance collapse". A common way to do this is to add a small (but not too small) value to the diagonal, or thresholding too small values to be some epsilon, or both.

- You will need calculations of the general form

$$f(x) = \log \sum_i \exp(x_i)$$

  If the values of x have large absolute values, you will run into numerical underflow or overflow as $\exp(x_i)$ approaches zero or infinity. In our case, we are worried about

underflow, since we are dealing with events of very low probability. A common trick is to find an appropriate constant C and realize that the following is mathematically (but not numerically) equivalent

$$f(x) = C + \log \sum_i \exp(x_i - C),$$

where typically $C = \max(x)$. This is such a common practice that there is a function in the Python Scientific Computing package that does this: `scipy.misc.logsumexp`

You will (1) fill in missing code; (2) fit model and report train/val acc; and (3) tune model settings, report best model, discuss trend across settings. Write your answer directly in the markdown cells of the notebook.

Problem 1 **[20 points]**
Fill in the missing pieces of code in Gaussian log probability computation (`compute_log_probs` in `GMM`). You will want to distinguish diagonal vs non-diagonal covariance matrices because the diagonal covariance matrix makes computation much more efficient (e.g., no need to compute full matrix inverse). Computing the log determinant is easy when the matrix is diagonal, as

$$\log\det\left(\text{diag}\left(a_1, \ldots, a_n\right)\right) = \log\left(\prod_{i=1}^{n} a_i\right) = \sum_{i=1}^{n} \log a_i$$

for $a_1 \ldots a_n > 0$. You'll want to use the last formulation, sum the logs instead of taking the log of a product, for numerical stability. For non-diagonal matrices you can use the NumPy function slogdet. Do not use any loops: everything can be done in batch computation. You will want to use the batch matrix multiplication. Pass the unit test.

End of problem 1

Problem 2 **[10 points]**
Implement the mean initialization (`init_centers` in `GMMTrainerEM`).

End of problem 2

Problem 3 **[20 points]**
Implement `update_parameters` in `GMMTrainerEM`, specifically the covariance matrix update for both the diagonal and the non-diagonal cases. Do not use any loops: everything can be done in batch computation using batch matrix multiplication.

End of problem 3

Problem 4 **[10 points]**
Implement the `compute_accuracy` function, which evaluates the accuracy of the GMM-based generative classifier on a dataset.

End of problem 4

Problem 5 **[10 points]**

Now that you have the code for EM, you should be able to use this code to fit a GMM for each class. For each class, fit a single Gaussian and a mixture of 3 Gaussians with diagonal covariance matrices, and report classification accuracy on the training and validation sets. For each class, fit a single Gaussian and a mixture of 3 Gaussians with full covariance matrices, and report classification accuracy on the training and validation sets.

End of problem 5

Problem 6 **[10 points]**

Now, explore the modeling space as you see fit. Required:

- Modify the number of components,
- Restrictions on the covariance matrices, i.e., diagonal vs. full,

Optional:

- Initialization strategy
- Stopping criteria
- Or any other aspect, as long as the resulting model is still a generative classifier with one or more Gaussians per class.

Describe your best configuration in your notebook and the corresponding best validation accuracy. Also, describe in your notebook the trends that you observe across the settings you experimented with, both in terms of the accuracies and the visualized clusters (displayed by the `show_means` function).

End of problem 6