**EIE4105 Multimodal Human Computer Interaction Technologies**

**Lab: Machine Learning for Handwritten Digit Recognition**

**(Google Colab Version)**

**A. Objectives and Outcomes**

After finishing this lab, you should be able to

* Use the Bayes rule and Bayes decision theorem to build pattern classifiers.
* Use Gaussian models and Gaussian mixture models (GMMs) to approximate the distributions of feature vectors.
* Understand the limitations of using a single Gaussian density function to approximate the distributions of feature vectors.
* Understand the impact of using diagonal covariance matrices and full covariance matrices on a Gaussian classifier.
* Understand the curse-of-dimensionality problem.
* Build pattern classifiers based on GMMs for HCI systems.

**B. Assessment Criteria**

* Ability to build pattern classification systems based on the Bayes decision theorem.
* Ability to produce correct results.
* Ability to explain the capability of different machine learning algorithms.
* Clarity of the report.

**C. Submission**

* Copy and paste the graphs and images that you obtain from Google Colab or other Python IDE to a word file and **convert it to PDF**.
* Submit your report to Blackboard before the deadline specified in Blackboard.

**D. Handwritten Digit Recognition**

In this lab exercise, you will develop a handwritten digit recognizer that can recognize the digits (‘0’ to ‘9’) in the MNIST handwritten digit dataset. Some examples of the digits are shown in the figure below.

****

Fig. 1: Handwritten digits in the MNIST dataset

**E. Procedures**

***E.1 Development Environment***

1. We will use Google Colab as the development environment.[[1]](#footnote-1) Colab runs on browsers. You need a Google account to use Colab. If you do not have one, visit https://support.google.com/mail/answer/56256?hl=en. The advantage of Colab is that no installation is required. The disadvantage is that you may need to remount the Google Drive after the session expired.
2. Display the Google Drive (https://drive.google.com/drive/) page in your browser. Create the following directory structures in your Google Drive:

My Drive/Learning/EIE4105/lab1/data

My Drive/Learning/EIE4105/lab1/python

After creating the folders, you should see something like this:

Graphical user interface, application

Description automatically generated

Fig. 2: Directory structure in Google Drive

1. Go to <http://bioinfo.eie.polyu.edu.hk/download/EIE4105/lab1/data/>. Download the .mat files and upload them to your Google Drive under “My Drive/Learning/EIE4105/lab1/data”.
2. Go to <http://bioinfo.eie.polyu.edu.hk/download/EIE4105/lab1/python/>. Download all files in this folder and upload them to your Google Drive under “My Drive/Learning/EIE4105/lab1/python”.

***E.2 Data Visualization***

1. In your Google Drive page, go to “My Drive/Learning/EIE4105/lab1/python”. Then, right click “show\_mnist.ipynb”. Then, select “Open With” and then “Google Colaboratory” to load the Colab iPython file as shown below. If your browser does not have “Google Colaboratory” installed, click “Connect more apps” and search “Colaboratory” to install it first.

Graphical user interface, text, application, chat or text message

Description automatically generated

Fig. 3: Open Google Colab file “show\_mnist.ipynb”

1. Click “+ Code” to create a new command edit box. Then, mount your Google Drive to the IPython Notebook as follows:

A screenshot of a cell phone

Description automatically generated

Click the link and follow the instruction. Put the key in the edit box and press the “Return” key. You should be able to see the following after mounting.

A screenshot of a cell phone

Description automatically generated

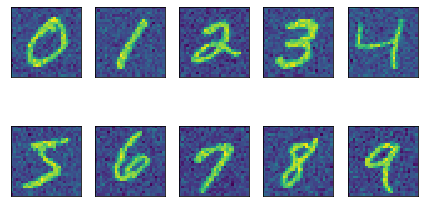
1. Click “+ Code”. Put the following code to change your current working folder to “My Drive/Learning/EIE4105/lab1/python”.

Graphical user interface, text, application

Description automatically generated

1. Continue to click the Logo

   Description automatically generated with medium confidence button to show the noisy digits by executing the show() function. The noisy\_train\_digits is a Python dictionary object containing the data in the Matlab file “noisy\_train\_digits.mat”. The show() function converts row vectors into 28×28 images. Note that in show(), data['trainData'][i][0][k] contains the *k*-th vector of Digit ‘i’. You should be able to see the following image on your screen. Study the Python codes.



***Copy the images and paste them into your report.***

1. Display the histogram of the 400-th pixel for digit ‘0’ by using the following command:

import matplotlib.pyplot as plt

x400 = data['trainData'][i][0][:, 399]

n, bins, patches = plt.hist(x400, bins=256,

density=True,

facecolor='blue', alpha=0.75)

You may find the above code fragment in the file “show\_mnist.ipynb”. Note that Python arrays start from index 0. So, x[0] is the first element of x and x[399] is the 400-th element of x.

***Display the histogram of the 400-th pixel of other digits. Copy the histograms and paste them to your report. Are the distributions of the 400-th pixel identical across all digits? Explain how you would create a handwritten digit recognizer if you are only given the 400-th pixel of the images.*** *[Hints: You may answer this question by using description, diagrams and equations. You may propose what statistical models (Gaussian model) or Gaussian mixture model) are suitable for the classifier and describe how you will train the classifier. The diagram may illustrate the structure of the classifier. Use Microsoft Word’s Equation Editor or Latex to typeset the equations].*

1. Compute the covariance matrix corresponding to Pixel 1 and Pixel 400 of digit ‘0’ and the covariance matrix corresponding to Pixel 400 and Pixel 401 using the following commands:

import numpy as np

y1 = noisy\_train\_digits['trainData'][0][0][:, (0, 399)]

y2 = noisy\_train\_digits['trainData'][0][0][:, (399, 400)]

C1 = np.cov(y1, rowvar=False, bias=True)

C2 = np.cov(y2, rowvar=False, bias=True)

You may find this code fragment in “cov\_matrix.ipynb”. In the above command, y1 contains the first column and the 400-th column of the  matrix, where  is the number of training samples for digit ‘0’. The function np.cov() computes the covariance matrix using the formula:



where



***Record the covariance between Pixel 1 and Pixel 400 and the covariance between Pixel 400 and Pixel 401. Which covariance is larger and why? Note that the covariance between two random variables is a scalar.*** *[Notes: The major mark of this question goes to the explanation]*

1. Compute the covariance matrix of digit ‘0’ and display the covariance matrix based on the following code fragment in “cov\_matrix.ipynb”:

X = data['trainData'][0][0][:]

C = np.cov(X, rowvar=False, bias=True)

img = ax[0].imshow(C, interpolation='nearest')

ax[i].title.set\_text(f'Digit {0}')

***Repeat the same procedure for all other digits and put them into your report. Are the covariance matrices identical? Assuming that the mean of individual digits are zero, is it possible to use of the covariance matrices to differentiate different digits? If yes, how?***  *[Notes: The major mark of this question goes to the explanation of how you would build the classifier. You may use equations to explain why a*  *for all of the digits can still be able to classify the digit.]*

***E.3 Gaussian Classifier***

1. Read the file “get\_gauss\_accuracy.ipynb”. Make sure that you understand how to implement and evaluate the performance of a handwritten digit recognizer. In particular, the following statement computes the mean and covariance matrix of the *k*-th Gaussian:

self.stats[k] = {

"mean": X\_k.mean(axis=0),

"cov": (1/(N\_k-1))\*np.matmul((X\_k-mu\_k).T,

X\_k-mu\_k)+ epsilon\*np.identity(D)

}

The statement

P\_hat[:,k] =

mvn.logpdf(X, s["mean"], s["cov"])+np.log(self.priors[k])

implements the log-likelihood of **x**:



where  and  are the mean vector and covariance matrix of digit, respectively.

***Run get\_gauss\_accuracy.ipynb. Record the accuracy of a Gaussian classifier that uses full covariance matrices and the one that uses diagonal covariance matrices. Explain why one is superior to the other in this recognition task. Which one is superior?***

*[Notes: The major mark of this question goes to the explanation.]*

***E.4 Gaussian-Mixture-Model (GMM) Classifier***

1. Read get\_gmm\_accuracy.ipynb and its classes and functions. Make sure that you understand how to implement and evaluate the performance of a handwritten digit recognizer. Train a GMM classifier with 20 mixture components (n\_components = 20) using the following function:

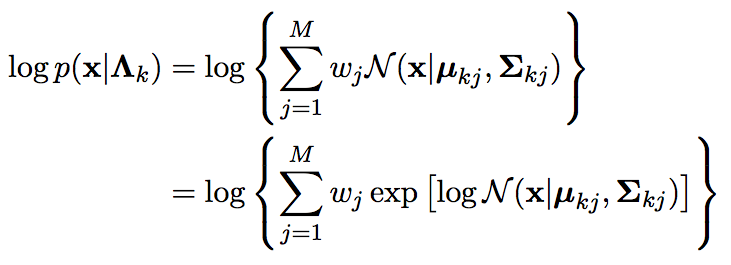
import gmm

k = 20

model = gmm.gmm(n\_components=k, verbose=True)

model.fit(train\_data\_reduced,means\_init\_heuristic='kmeans')

1. Identify the function(s) in “gmm.py” that implements the following log-likelihood function for GMM:



***Run the script get\_gmm\_accuracy.ipynb. Record the accuracy of your GMM classifier. How is the performance of your GMM classifier when compared with the Gaussian classifier?***

Note the following sequence of function calls when computing the log-likelihood of **x**.

• classifier.predict()

• gmm.predict() [No such function in gmm.py, so call its parent in mixture.py]

• mixture.predict()

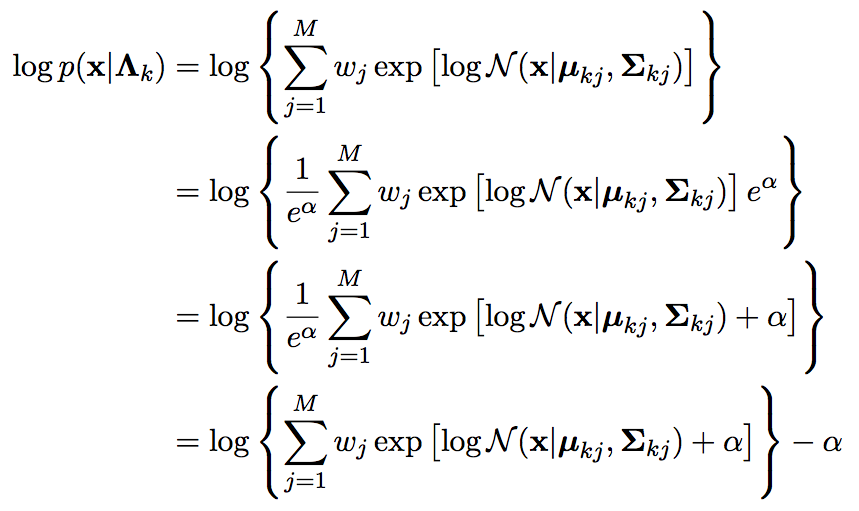
• mixture.\_log\_support() [No such function in mixture.py, so call its child in gmm.py]

• gmm.\_log\_support()

• gmm.\_log\_multivariate\_normal\_density\_full()

1. In theory, given enough data, a GMM classifier with full covariance matrices and many mixture components should perform better than a Gaussian classifier with a diagonal covariance matrix. However, your 20-mixture GMM classifier with full covariance matrices in Step 14 probably performs badly. The main reason of the poor performance is that directly implementing the scoring function in Step 14 will lead to numerical problems. More precisely, when **x** is not close enough to any cluster means in the GMM, the likelihood  will be 0 for all *k*. As a result, we do not have any maximum for making a decision. In such situation, Python will arbitrary select the first one as the maximum. For example, you may observe the array likelihoods in the predict() function of “classifier.py”. You should see that all elements in likelihoods are 0.

To solve the problem, you need to use the following numerical trick in the function ﻿\_log\_multivariate\_normal\_density\_full() in “gmm.py”:



where  is an offset to ensure that the exponent will not overflow/underflow the machine’s numerical representation. To find an appropriate value for , you need to debug into the program to see the value of .

*[Notes: This step requires you to know Python well and is therefore OPTIONAL. You may skip it.]*

***E.5 Classifying Clean Data***

1. Run the code in “get\_gmm\_accuracy.py” to load and display the clean MNIST images.

from mnist import load\_mnist

trainpath = '../../data/noisy\_train\_digits.mat'

testpath = '../../data/noisy\_test\_digits.mat'

train\_data, train\_labels, test\_data, test\_labels =

load\_mnist(trainpath, testpath)

def show(data, labels):

fig, ax = plt.subplots(nrows=2, ncols=5, sharex=True, sharey=True, )

ax = ax.flatten()

for i in range(10):

j = [k for k in range(len(labels)) if labels[k] == i][0]

img = data[j].reshape(28, 28)

img = ax[i].imshow(img, interpolation='nearest')

ax[0].set\_xticks([])

ax[0].set\_yticks([])

plt.tight\_layout()

plt.show()

show(train\_data, train\_labels)

You should be able to see the following image:

Graphical user interface, application, Teams

Description automatically generated

1. Repeat Step 9 to display the histograms. Note that unlike Step 9, train\_data is a 2D Numpy array of shape (60000, 784). You need to use the following code to access different digits in this 2D array. For example to access Digit 3, you may

j = [k for k in range(len(labels)) if labels[k] == 3]

digit3 = train\_data[j,:]

digit3 will be a matrix containing row vectors of digit 3, each with a dimension of 784.

***Record your observations. What are the differences between the histograms of clean and noisy digits?*** *[Notes: You need explain why the histograms of clean digits are different from the histogram of noisy digits.]*

1. Repeat Steps 13-15 to train a Gaussian classifier and a GMM classifier to recognize the clean digits.

***Record your observations. Can you train a Gaussian or GMM classifier? Explain your observations based on the histograms that you obtained in Step 17.*** *[Notes: Majority of the marks goes to the explanation.]*

***E.6 Curse of Dimensionality***

1. This step demonstrate the curse-of-dimensionality problem. Reduce the number of training samples to 100 for each digit by adding the following lines to “get\_gauss\_accuracy.ipynb”:

from mnist import load\_SampleMnist

trainpath = '../data/noisy\_train\_digits.mat'

testpath = 'mat/noisy\_test\_digits.mat'

nSamples = 100

train\_data,train\_labels,test\_data,test\_labels =

load\_SampleMnist(trainpath,testpath,nSamples)

gcf = Gauss\_class\_full()

gcf.fit(train\_data, train\_labels, **epsilon=0**)

Make sure you have set epsilon to 0, which will add epsilon to the diagonal of the covariance matrix:



It is a regularization parameter that helps to make the covariance matrix invertible even though we do not have enough samples. By setting it to 0, we turn off this features.

***Can you train a Gaussian classifier with full covariance matrix? Increase the number of training samples (nSamples) per digit until you can train and test a Gaussian classifier with a “full” covariance matrix. Specifically, if that number is N, the training algorithm will fail to train a Gaussian classifier with full covariance matrix using N – 1 samples. What is that number? It is NOT necessary to explain why you get this magic number. If your math is good, you can find this number without running the program.*** *[Notes: All marks of this question go to this magic number.*

1. Set the number of training samples per class to the magic number that you find in Step 19. Then, train the following classifiers:
2. Gaussian classifier with a full covariance matrix
3. Gaussian classifier with a diagonal covariance matrix
4. GMM classifier with 4 mixtures, each comprising a diagonal covariance matrix

In theory, this magic number should produce non-singular covariance matrices. But, if you still encounter numerical difficulty for some digits, you may add a small number to the diagonal of the covariance matrix as follows:

self.stat[k] = {"mean":X\_k.mean(axis=0),

"cov": (1/(N\_k-1))\*np.matmul((X\_k-mu\_k).T,X\_k-mu\_k)+

epsilon\*np.identity(D)}

where epsilon is a small positive number (e.g., 0.0001). In machine learning, this is called regularization.

***Compare the accuracies achieved by these 3 classifiers. Which one is better? Explain your result.*** *[Notes: Majority of the marks goes to the explanation.]*

-- END --

1. You may also install Python on your computer. For example, “Anaconda 3” contains a full set of Python packages and the core Python. It also has integrated development environments (IDE) such as Spyder and VSCode. If you use Mac or Linux, Python has been pre-installed. But you still need to install an IDE. [↑](#footnote-ref-1)