Expectation Maximisation with Gaussian Mixture Model

By: *Kim Chouard, J114030902*

For: *Intelligent Speech Technology @Shanghai Jiao Tong University*

Presentation

Introduction

This project is about implementing a training and testing algorithms for GMM. The program is written in Python and the code source of this project can be found in this github repository: https://github.com/kimchouard/sjtu-ist-gmm.

More informations on how to use the code can be found in the README, but it is pretty straight forward to get it working once you have clone the repo (or downloaded the .zip):

1. Install dependencies by running:

```
pip install -r requirements.txt
easy_install argparse
```

2. Then either run the main script:

```
python main.py <training input file> <unlabeled input file> <labeled output
file>
```

Or the testing one to run a batch of test:

```
python scripts/tests.py <training input file> <testing input file> <result
output file> <number of iterations>
```

(More details are given on the testing capabilities in the Testing section.)

The data provided for this exercise can be found under the folder data, and it is a good practice to keep all the created file in this folder.

Code repartition

The main code can be found under the *libs/* folder. Here is some details on the code repartition.

- **GMMToolbox.py**

Contains the 2 mains mathematical functions underlying the GMM theory.

  - ➔ *logN* calculate the logarithm (for performance concern) value for a gaussian distribution, given a certain X, mean and covariance.
  - ➔ *gaussianPdf* evaluate the gaussian pdf, useful when trying to assign label once we have trained our models.

- **EM.py**

Contain the so called the EManager. It is in charge to roll the complete algorithm using the Expectation Maximisation method, either:

  - ➔ to train a model based on a training data
  - ➔ to find which label is more accurate based on a trained model.

- **Set.py**

Basically in charge of managing the data. It allows simply to:

  - ➔ to add a point to a dataset
  - ➔ to get data with only a specific label
  - ➔ to affect label if needed based on a model
  - ➔ to plot the data (need to be uncommented on the code source)

The 2 other python files are meant to be executed:

- **main.py**

Offer a simple CLI to be able to execute our program from the command line and, further more, to automate some testing.

- **tests.py**

It can be found under the *script/* folder. It allows you to automate the running of the main command in order to test the program. More information in the Testing section.

GMM Initialisation

The GMM parameters initialisation is located in the *initRun* function of *EM.py*. There is basically 3 parameters to initialize:

- **Weights**

The weight of each mixture can be initialized pretty simply as its sum is equal to 1. Knowing *K*, the number of mixtures, the code create a matrix of size (K, 1) with values *1/K* in order to have an equal repartition at the first place and then tuned little by little by our EM loops.

- **Covariance**

In the same approach than for the weight, we create *K* Identity matrix of size (2,2), so *K* times:

(1, 0
0, 1)

- **Mean**

The first method implemented for the mean initialization is a random sampling from the data. Other solutions are presented in the opening, but because of the good result of this method, they were not implemented in this version.

N.B.: In our case, *K* is equal to 4 as we can see when plotting the data. It is hardcoded for now.

Testing

Method

As said above, *test.py* allows you to measure the accuracy of your program.

```
python scripts/tests.py <training input file> <testing input file> <result
output file> <number of iterations>
```

It will do *"number of iterations"* iterations, and each time:

1. run main.py with the -t option, in order not to import the label from the *"testing input file"* (in our case, dev.txt).
2. Record how much time it has been running.
3. Compare the found result (stored in *data/tempOutput.txt*) to the original *"testing input file"*.
4. Deduce the error rate.

All the result are stored in a csv format under the *"result output file"*. The repository is already containing a CSV result (*data/rest_test.csv*), analyzed in the further section.

Results

Our program was tested on a 2-dimensional data sets. The training set contains 4800 features, and our testing set 400 features. In total, there were 2 different labels.
We runned a test batch of 1000 runs for our script, below is some statistics extracted out of csv result file:

GMMx1000	**Error Rate**	**Time (ms)**
Average	6.5 %	0.214686513
Standard Deviation	2.951354688	0.020236306
Maximum	26.5 %	0.378777027
Minimum	4 %	0.180793047

Statistics out of 1000 runs of the scripts with the Random Mean Init Method

Even if our initialization approach is pretty naive on the means, we end up with pretty promising result. Only 6% error rate in average and 21ms to run. Even if the maximum peak to more the 1 item false out of 4, the standard deviation is only 3 which show that the error rate is pretty stable around 6.5% [+/- 2.95].

Conclusion

Even if this program gave some pretty good result and seems to be pretty efficient, there is some rooms for improvment.

- For example, we can use the K-means method to cluster the different mixtures and deduce the means of each with more accuracy than a random sampling.
- Also, we are now hardcoding the number of mixtures to 4. We should automate the finding of the number K in order to bring this script to more wide applications.

Finally, as required in the instructions, a labeled version of *test.txt* can be found under *data/test_classification.txt*.