PRML Assignment III

Generation of Data

The training data is generated by ***createDataset*** in **data.py**. The ***createDataset*** function uses the means and covariations of Gaussian distributions initialized by ***initializeGaussianDistributions*** in **data.py** to generate samples which compile to Gaussian distributions. Command line options are provided for specifying the number of samples, the path of saving the dataset and corresponding labels, and the number, dimension and separation degree of Gaussian distributions.

Description of Models

GAUSSIAN MIXTURE MODEL

The ***GaussianMixtureModel*** is defined in **model.py** in **handout** directory. It contains an **K-means model** to initialize the means and covariations of gaussian distributions. When training the Gaussian mixture model on a certain dataset, the K-means model is first trained on the dataset, and Gaussian mixture model initialize its means, covariations and proportions of each Gaussian distributions according to the trained K-means model.

The ***fit*** method of ***GaussianMixtureModel*** provides **two ways** to train the Gaussian mixture model. The first is to **calculate the logarithm likelihood function and judge whether it converges**, the second is to **complete certain epochs of training**. Command line options are provided for specifying the training method. In both ways of training, I use the **Expectation Maximization Algorithm** to update the parameters of Gaussian mixture model. The **E-step** fixes parameters and calculates the posterior probability function of distributions. The **M-step** find parameters to maximize the posterior probability function. By applying the **Expectation Maximization Algorithm** iteratively, the posterior probability function will be maximized. Here is the **Expectation Maximization Algorithm** I actually realised:

- E-step: fixes parameters and calculates the posterior probability.

$$\begin{aligned}\gamma_{nk} &= p(z^{(n)} = k \mid x^{(n)}) \\ &= \frac{\pi_k \mathcal{N}(x^{(n)}; \mu_k, \sigma_k)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(x^{(n)}; \mu_k, \sigma_k)}\end{aligned}$$

- M-step: maximize the posterior probability and update the parameters.

$$\pi_k = \frac{N_k}{N}$$

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} x^{(n)}$$

$$\sigma_k^2 = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} (x^{(n)} - \mu_k)^2$$

$$\text{while } N_k = \sum_{n=1}^{N} \gamma_{nk}$$

The ***plot*** method of ***GaussianMixtureModel*** will plot the dataset and ellipse representing distributions dynamically. Colors are randomly specified for each Gaussian distribution and command line options are provided for specifying whether or not to plot the training process of Gaussian mixture model.

K-MEANS MODEL

The ***kMeansModel*** is defined in **model.py** in **handout** directory. It uses **K-means++ algorithm** to choose the initial values of cluster centers. The workflow of **K-means++ algorithm** is to choose the first center randomly and add a new center whose distance to the choose centers is farthest to the class of chosen

centers until k cluster enters are chosen. Here is the **K-means++ algorithm** and **K-means algorithm** I actually realised:

- Choose K cluster centers
  - Choose the first cluster center
  - Choose the farthest from chosen cluster centers as a new center
  - Repeat until K cluster centers are chosen
- Assign each sample in dataset to a nearest cluster center
- Update the cluster center by calculating the mean of each cluster

$$x_c = \frac{1}{N_c} x_{s_c}$$

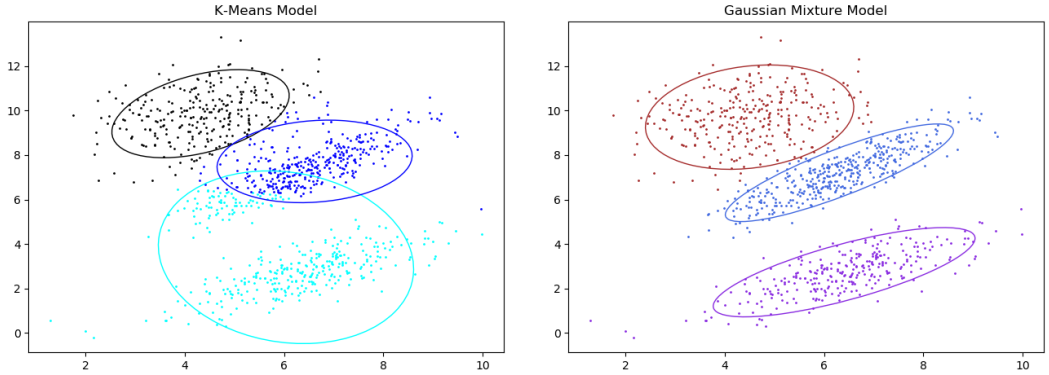
- Repeat until the classification result no longer changes

The ***fit*** method of ***kMeansModel*** train the k-means model on certain dataset.

The ***plot*** method of ***kMeansModel*** will plot the dataset and ellipse representing distributions dynamically. Colours are randomly specified for each Gaussian distribution and command line options are provided for specifying whether or not to plot the training process of k-means model.

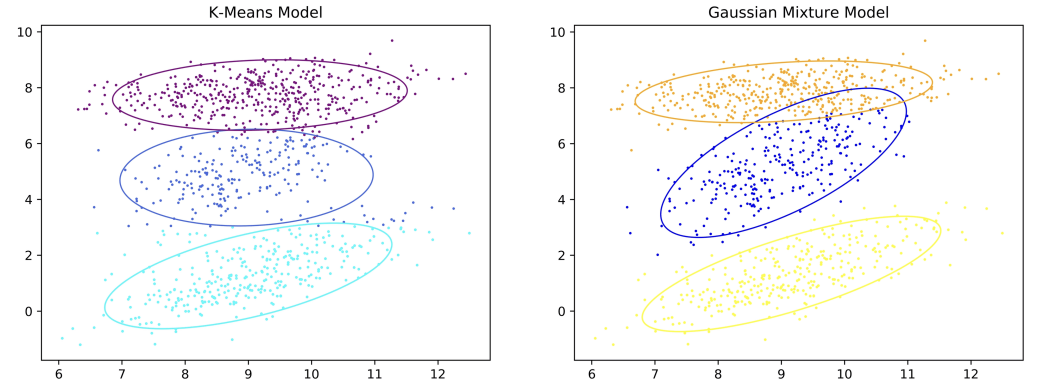
Performance of Models

Create a dataset containing 3 Gaussian distributions and train the Gaussian mixture model. The k-means model and Gaussian mixture model perform differently.



According to the picture, the k-means model assign some samples to the blue class, which evidently belongs to the pink class, while in the result of Gaussian mixture model, it correctly assign them to the sky blue class.

In the following example, the k-means model reaches an accuracy of 89.6%, while the Gaussian mixture model reaches an accuracy of 96.0%.



Both of the examples show that the k-means model considers only the distance between samples and cluster centers, while the Gaussian mixture model considers not only the distance but also the posterior probabilities. As a result, the classification result of k-means model would be closer to a ***circle*** if the mean of Gaussian distributions are close to each other. The clusters try to **expand themselves uniformly in all directions**. However, the classification result of Gaussian mixture model would be closer to a ***ellipse*** since the clusters **expand themselves according to not only the mean of each cluster, but also the covariance of each cluster**. Apparently the Gaussian mixture model has a **better adaptability**.

By the way, the evaluation of unsupervised machine learning models is sometimes not easy. The difficulty in evaluating the k-means and Gaussian mixture models is the label of each sample is unclear. For example, if there are three clusters, samples may be labelled as 0, 1, 2. However the k-means model or Gaussian mixture model may classify the samples belongs to the first cluster as 2, the second as 0, the third as 1 and it is still an accurate classifier. It is improper to compare the initial labels and the classification result directly. I can think of three ways to handle this problem. The first is to calculate the number of samples belongs to each cluster in the initial labels and the classification result and associate them according to the size of clusters. The second is to compare the mean of each cluster according to the initial labels and the classification result, then associate the means of models with nearest cluster centers obtained from initial labels. The third is to sort the cluster centers obtained from initial labels and the means of cluster models according to their modules separately and associate the mean of cluster models with the cluster center obtained from initial labels which is in the same position. I use the first method and the accuracy of models is high in most cases.

Appendix

Run the programme by typing **“python source.py —createDataset True —model GMM”** and specify command line options referring to **source.py**. Here are the command line options.

Options	Type	Default
CreateDataset	bool	FALSE
DatasetPath	str	handout/samples.data’
numOfSamples	int	1000
numOfDistributions	int	3
dimOfDistributions	int	2
separationDegree	int	10
saveLabels	bool	TRUE
labelsPath	str	‘handout/labels.pkl’
model	str	‘GMM’
train	bool	TRUE
plot	bool	TRUE
plotPath	str	handout/’
epsilonSpecified	bool	TRUE
epsilon	float	1e-3
epochsSpecified	bool	FALSE
epochs	int	50

Options	Type	Default
saveModel	bool	TRUE
saveModelPath	str	‘handout/’
resume	bool	FALSE
resumeModelPath	str	‘handout/’
evaluate	bool	TRUE