CSCI 567, Spring'18
Michael Shindler

Programming Set #4

Due: 11:59 pm, April 8, 2018

General instructions

Due date and time April 8th, 11:59pm

Starting point Your repository will have now a directory 'homework4/'. Please do not change the name of this repository or the names of any files we have added to it. Please perform a `git pull` to retrieve these files.

High Level Description

0.1 Tasks In this Programming Set you are asked to implement K-means clustering to identify main clusters in the data, use the discovered centroid of cluster for classification and implement Gaussian Mixture Models to learn generative a model $p(x|z)$ of the data. Specifically, you will

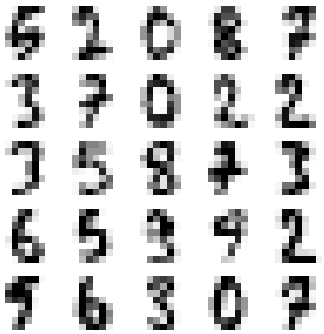
- Implement K-means clustering algorithm to identify clusters in a two-dimensional toy-dataset.
- Implement image compression using K-means clustering algorithm.
- Implement classification using the centroids identified by clustering.
- Implement Gaussian Mixture Models to learn a generative model and generate samples from a mixture distribution.

0.2 Running the code We have provided two scripts to run the code. Run `kmeans.sh` after you finish implementation of k-means clustering, classification and compression. Run `gmm.sh` after you finish implementing Gaussian Mixture Models.

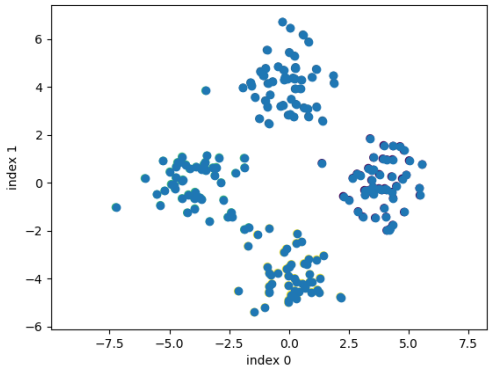
0.3 Dataset Through out the assignment we will use two datasets (See fig. 1) — Toy Dataset and Digits Dataset.

Toy Dataset is a two-dimensional dataset generated from N gaussian distribution with variance $\mathbf{I}_2$ and means equally spaced on perimeter of circle. In this set, we have fixed N=4 and therefore means of 4 gaussian distribution are $(0, 0), (1, 0), (-1, 0)$ & $(-1, -1)$. We will use this dataset to visualize the results of our algorithm in two dimensions.

We will use digits dataset from sklearn [1] to test k-means based classifier and generate digits using Gaussian Mixture model. Each data point is a 8×8 image of the digits. This is similar to MNIST but less complex.



(a) Digits



(b) 2-D Toy Dataset

Figure 1: Datasets

0.4 Cautions Please **DO NOT** import packages that are not listed in the provided code. Follow the instructions in each section strictly to code up your solutions. **Do not change the output format**. **Do not modify the code unless we instruct you to do so**. A homework solution that does not match the provided setup, such as format, name, initializations, etc., **will not** be graded. It is your responsibility to **make sure that your code runs with the provided commands and scripts on the VM**. Finally, make sure that you **git add, commit, and push all the required files**, including your code and generated output files.

0.5 Final Submission After you have solved problem 1 and 2, execute `bash kmeans.sh` command and `bash gmm.sh` command. Git add and push plots and results folder and all the *.py files.

Problem 1 K-means Clustering

In the lecture, we considered the problem of clustering as the problem that given data $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N, \mathbf{x}_i \in \mathbb{R}^D$, we want to identify clusters/groups of data points and establish membership of each $\mathbf{x}_i$ to one of the group. Specifically, the problem of k-means clustering is to find K clusters in the data such that euclidean distance of each point from the center of respective cluster is minimized. Mathematically, let $\mu_k \in \mathbb{R}^D$ be the center of the cluster k and $r_{ik} \in \{0,1\}$, $\sum_{k=1}^K r_{ik} = 1$ be the membership of $\mathbf{x}_i$ to cluster k (i.e. each $\mathbf{x}_i$ can belong to only one cluster), then we define distortion measure as

$$J = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} r_{ik} \|\mu_k - \mathbf{x}_i\|^2 \tag{1}$$

We want to find $\hat{\mu}_k$ and $\hat{r}_{ik}$ such that J is minimized.

$$\hat{\mu}_k, \hat{r}_{ik} = \arg\min_{r_{ik}, \mu_k} J \tag{2}$$

Clearly, we can see for some fixed μ_k,

$$\hat{r}_{ik} = \begin{cases} 1 & k = \arg\min_{k'} \|\mu_{k'} - x_i\|^2 \\ 0 & \text{Otherwise} \end{cases} \tag{3}$$

and for some fixed r_{ik},

$$\hat{\mu}_k = \frac{\sum_{i=1}^N r_{ik} x_i}{\sum_{i=1}^N r_{ik}} \tag{4}$$

We can use eq. 3 & 4 to arrive at the optimal solution to eq. 2. We therefore state the k-means clustering algorithm (see Algorithm 1). We define the algorithm to have converged when distortion measure does not change much.

Algorithm 1 K-means clustering algorithm

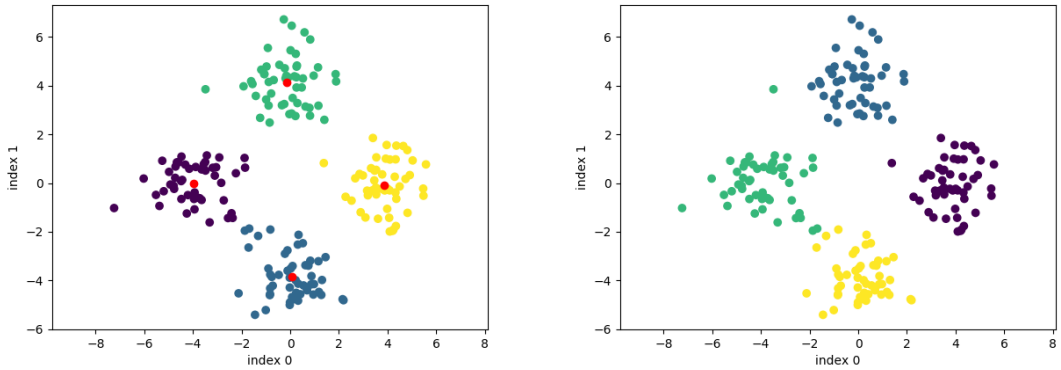
Inputs:
  An array of size $N \times D$ denoting $\{x_i\}_{i=1}^{N}$, N points each of D-dimension, `X`
  Maximum number of iterations, `max_iter`
  Number of clusters, `K`
  Error tolerance, `e`
Outputs:
  Array of size $K \times D$ of means, $\{\mu_k\}_{k=1}^{K}$
  Membership vector `R` of size N, where R[i] = k, such that $r_{ik} = 1$
  ▷ `r` is one-hot encoded version of `R`
Initialize:
  Set means $\{\mu_k\}_{k=1}^{K}$ to `K` points selected from `X` randomly
repeat
  Compute membership r_{ik} using eq. 3
  Compute distortion measure J_{new} using eq. 1
  if $|J - J_{new}| \leq e$ **then**
    STOP
  end if
  Set $J := J_{new}$
  Compute means μ_k using eq. 4
until maximum iteration is reached

Comments

- For K-means clustering, if at some iteration, $\exists k, \forall i, \hat{r}_{ik} = 0$, keep $\hat{\mu_k}$ unchanged.
- For classification with K-means clustering, if some centroid doesn't contain any point, you can set the label of this centroid as 0.
- While assigning a sample to a cluster, if there's a tie (i.e. the sample is equidistant from two centroids), you should choose the first centroid (like what *np.argmin* does).

1.1 Implementing K-means clustering Algorithm You have to implment K-means clustering in the file `kmeans.py`. You should complete the implementation of `KMeans` class sticking to the specifications described in the code.

After you complete the implementation execute `bash kmeans.sh` command to run k-means on toy dataset. You should be able to see three images generated in `plots` folder. In particular, you can see `toy_dataset_predicted_labels.png` and `toy_dataset_real_labels.png` and compare the clusters identified by the algorithm against the real clusters. Your implementation should be able to recover the correct clusters sufficiently well. Representative images are shown in fig. 2. Red dots are cluster centers. Note that color coding of recovered clusters may not match that of correct clusters. This is due to mis-match in ordering of retrieved clusters and correct clusters.



(a) Predicted Clusters (b) Real Clusters

Figure 2: Clustering on toy dataset

1.2 Image compression with K-means In the next part, we will look at lossy image compression as an application of clustering. Recall, k-means clustering tries to minimize the distortion. In image compression, we want to compress the image by reducing the bits to represent information and minimizing the distortion at the same time. One way to limit the bits is to limit the number of colors in the image. Here, we try to limit the number of colors while minimizing the distortion to the image at the same time.

We consider each pixel as a point x_i and try to find K best RGB codes (K best colors) to approximate each pixel (RGB value). For this, part you have to only implement the algorithm to transform the image given K centroids/code vectors. Specifically, complete the function `transform_image` in the file `kmeansTest.py`. We use K=16 in this problem.

After running this file, you should be able to see an image `baboon_compressed.png` in the `plots` folder. You can see that this image is distorted as compared to the original `baboon.tiff`.

1.3 Classification with k-means Recall that in k-nearest neighbour classifier we had to evaluate distance of a test sample from each training point to predict its class. This increased the complexity of the classifier. If we can identify representative sample for each cluster, we can reduce the complexity of prediction. In lectures, we saw that clustering identifies major groups in the data. Each centroid or mean vector μ_k is representative of a particular cluster. We use this intuition to create a classifier using K-means clustering (see algorithm 2).

Algorithm 2 Classification with K-means clustering

1: **Inputs:**
Training Data : {X,Y}
Parameters for running K-means clustering
2: **Training:**
Run k-means clustering to find centroids and membership
Label each centroid with majority voting from its members. i.e. $\arg\max_c \sum_i r_{ik} \mathbb{I}\{y_i = c\}$
3: **Prediction:**
Predict the same label as the nearest centroid (equivalently 1-NN on centroids).

Complete the `fit` and `predict` function in `KMeansClassifier` in file `kmeans.py`. Once completed, run kmeans.sh to evaluate the classfier on a held out test set. For comparison, the script will also print accuracy of logisitic classifier and k-NN classifier. A naive k-means classifier may not do well but it can be an effective unsupervised method in a classification pipeline [2].

Problem 2 Gaussian Mixture Model

We want to solve the problem of learning a generative model of data, i.e. we want to learn some arbitrary distribution of data, $p(x)$ and sample from it.

We assume that x is a mixture of K gaussian distributions (component) and a discrete latent variable chooses which component is x sampled from. Mathematically,

$$p(x|z=k) = \mathcal{N}(\mu_k, \Sigma_k) \quad \& \quad p(x) = \sum_k p(z)p(x|z) \tag{5}$$

Probability of choosing a particular component k given data x is $p(z=k|x)$. Intuitively, this measures how much the component k is responsible for x.

$$\gamma_k = p(z=k|x) = \frac{p(z=k)p(x|z=k)}{p(x)} \tag{6}$$

If we are given N observations $(x_1, ..., x_N)$, using a MLE formulation, one can obtain the relations between model parameters $p(z=k)$, μ_k, and Σ_k as

$$\gamma_{ik} = \frac{p(z=k)\mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_k p(z=k)\mathcal{N}(x_i|\mu_k, \Sigma_k)} \tag{7}$$

$$N_k = \sum_{i=1}^{N} \gamma_{ik} \tag{8}$$

$$\mu_k = \frac{\sum_{i=1}^{N} \gamma_{ik} x_i}{N_k} \tag{9}$$

$$\Sigma_k = \frac{\sum_{i=1}^{N} \gamma_{ik}(x_i - \mu_k)(x_i - \mu_k)^T}{N_k} \tag{10}$$

$$p(z=k) = \pi_k = \frac{N_k}{N} \tag{11}$$

Parameters of GMM can be estimated using EM algorithm described in algorithm 3

2.1 Implementing EM You have to implement EM algorithm (fit function) in file `gmm.py` to estimate mixture model parameters. After implementation execute `bash gmm.sh` command to estimate mixture parameters for toy dataset. You should see a gaussian fitted to each cluster in the data. Representative image is shown in fig. 3. We evaluate two types of mean initialization methods — a) using cluster means estimated by k-means clustering, b) using randomly initialized means. You will notice that initialization with k-means usually converge faster.

Comments

- For k-means initialization, the parameters of k-means method are the same as those of EM's parameters.
- For random initialization, you should generate μ_k randomly (each element uniformly in [0, 1]) and set the covariance matrix $\Sigma_k = I$ and $\pi_k = 1/K$.
- When computing the probability of a gaussian distribution which has covariance matrix Σ, use $\Sigma' = \Sigma + 10^{-3}I$ **when** Σ is not invertible (in case it's still not invertible, add another $10^{-3}I$).

Algorithm 3 EM algorithm for estimating GMM parameters

Inputs:
An array of size $N \times D$ denoting $\{x_i\}_{i=1}^{N}$, N points each of D-dimension, `X`
Maximum number of iterations, `max_iter`
Number of clusters, `K`
Error tolerance, `e`
Init method — K-means or random
Outputs:
Array of size $K \times D$ of means, $\{\mu_k\}_{k=1}^{K}$
Variance matrix Σ_k of size $K \times D \times D$
vector $p(z)$ of size D, `pi_k`
Initialize:
Initialize means randomly
Initialize variance to be identity matrix for each component
Initialize pi_k to be uniform i.e. 1/K
▷ Note: When using k-means for initialization use means and membership from k-means and then variance & pi_k can be estimated
Compute the log-likelihood, $l = \sum_{i=1}^{N} \ln p(x)$
repeat
E Step: Compute responsibilities using 7
M Step:
Estimate means using 10
Estimate variance using 10
Estimate pi_k using 11
compute likelihood l_new
if $|l - l_{new}| \leq e$ **then**
STOP
end if
Set $l := l_{new}$
until maximum iteration is reached

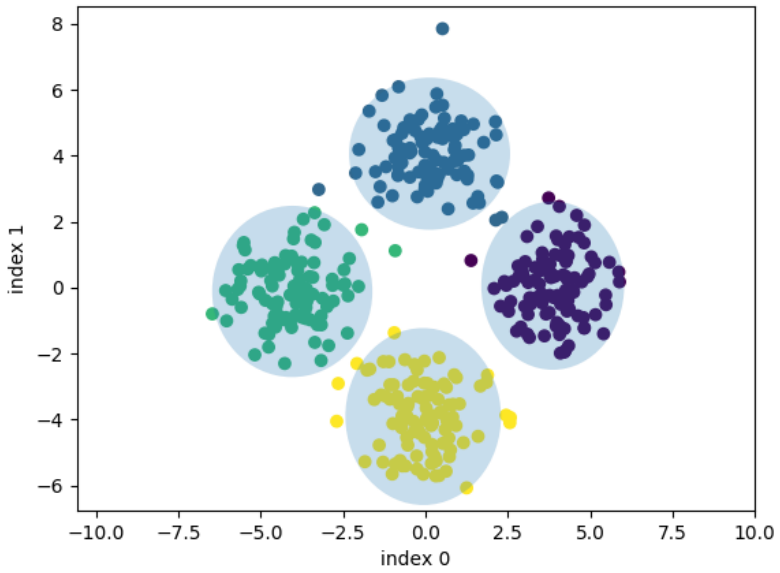
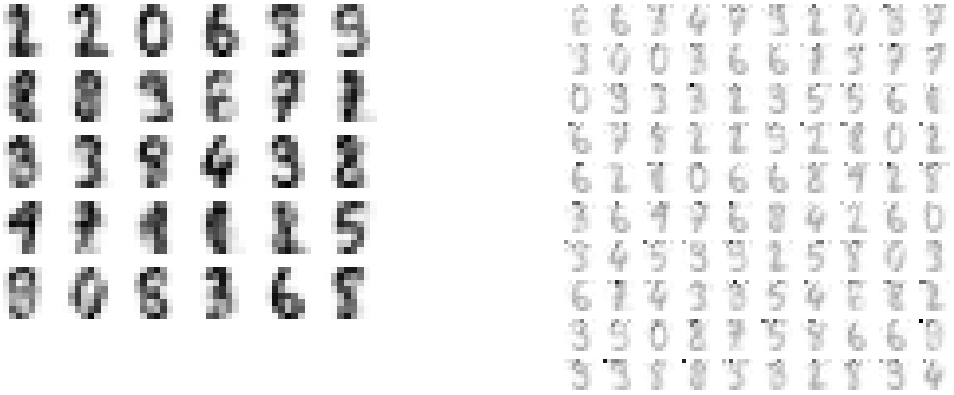


Figure 3: Gaussian Mixture model on toy dataset

2.2 Implementing sampling We also fit a K=30 component gaussian mixture model to digits dataset. A good thing about GMM is we can sample from the learnt distribution and generate new examples which should be similar to the actual data. So in this part you have to implement `sample` function in `gmm.py` which will use mean, variance and responsibility learnt using fit function to generate digits. Sampling from a GMM is a two step process as defined in eq. 5. First we sample k from p(z) (or pi_k variable) and then sample from a gaussian distribution with mean μ_k and variance Σ_k.

After completing this, execute `bash gmm.sh` command. This should produce visualization of means μ_k and some generated samples for the learnt GMM. Representative images are shown in fig. 4.



(a) Means of GMM learnt on digits

(b) Random digits sample generated from GMM

Figure 4: Results on digits dataset

References

[1] sklearn.datasets.digits http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html#sklearn.datasets.load_digits

[2] Coates, A., Ng, A. Y. (2012). Learning feature representations with k-means. In Neural networks: Tricks of the trade (pp. 561-580). Springer, Berlin, Heidelberg.