STAT545 HW3

Yi Yang

9/26/2018

1. Problem 1: The K-means algorithm.

The MNIST dataset is a dataset of 28×28 images of hand-written digits. Download it from http://yann.lecun.com/exdb/mnist/ (you only really need the training images and labels though). To read these images in R, use the script from https://gist.github.com/brendano/39760. Make sure you understand this. Note that the show_digit command displays a particular digit.

1. Since the dataset is quite large, restrict yourself to the first 1000 training images, and their labels. Store these as variables called digits and labels. digits should be a 1000×784 matrix (or its transpose). Include R code.

2. Write a function my_kmeans to perform a k-means clustering of the 1000 images of digits. Use Euclidean distance as your distance measure between images (which can be viewed as vectors in a 784 dimensional space). Your function should take 3 arguments, the matrix digits, the number of clusters K and the number of initializations N. You code should consist of 3 nested loops. The outermost (from 1 to N) cycles over random cluster initializations (i.e. you will call k-means N times with different initializations). The second loop (this could be a for or while loop) is the actual k-means algorithm for that initialization, and cycles over the iterations of k-means. Inside this are the actual iterations of k-means. Each iteration can have 2 successive loops from 1 to K: the first assigns observations to each cluster and the second recalculates the means of each cluster. These should not require further loops. (You will probably encounter empty clusters. It is possible to deal with these in clever ways, but here it is sufficient to assign empty clusters a random mean (just like you initialized them)). Since your initializations are random, make your results repeatable by using the set.seed() command at the beginning (you can also make the seed value a fourth argument). Your function should return:

(a) the cluster parameters and cluster assignments for the best solution
(b) the sequence of values of the loss-function over k-means iterations for the best solution (this should be non-increasing) (recall from the slides that the k-means loss function is the sum of the squared distances of observations from their assigned means)
(c) The set of N terminal loss-function values for all initializations.

Do not hardcode the number of images or their size. Include R code.

3. Explain briefly what stopping criteria you used (i.e. the details of the second loop).

4. Run your code on the 1000 digits for $K = 5, 10, 20$. Set N to a largish number e.g. 25 (if this takes too long, use a smaller number).For each setting of K, plot the cluster means (using show_digit) as well as the evolution of the loss-function for the best solution (you can use a semi-log plot if that is clearer). You do not have to print the other values returned by the function e.g. the cluster assignments, or the values of the cluster means etc., just plots is sufficient

5. For each setting of K, plot the distribution of terminal loss function values (using ggplot's geom_density()).

6. Explain briefly how you might choose the number of clusters K.

Bonus for an extra 20 points:

7. Modify your code to do k-medoids. You only need to change one part of your previous code viz. the part that calculates the cluster prototype given cluster assignments. The cleanest way to do this is to define a function called get_prototype that takes a set of observations and returns the prototype. For k-means this function just returns the mean of the observations. Note that the mean can also be defined as
$$\mu = \arg\min_x \sum_{i=1}^{|D|} (x - d_i)^2$$
Here $D = (d_1, \ldots, D_{|D|})$ is the set of images input to get_prototype, and the mean need not be part of this set. For k-medoids, the prototype is defined as
$$\mu = \arg\min_{x \in D} \sum_{i=1}^{|D|} (x - d_i)^2$$
In other words it finds an element in D that minimizes the sum-squared distance. Include R code for your implementation of get_prototype for k-mediods. You can use as many for loops as you want, but the simplest is to loop over each observation assigned to that cluster, calculate the sum-squared distance when that is set as the prototype, and return the best choice.
8. Run k-medoids for $K = 5, 10, 20$. Since this might take longer, you can use smaller values of N as well as fewer images (e.g. just 100 digits), but report what numbers you used. For each choice of K, show the cluster prototypes. Comment on the quality of the cluster prototypes, as well as the value of the loss function vs k-means.

2. Problem 2: Finite-state Hidden Markov models (HMMs)

(Continued from the problem on Markov chains from the previous homework.)

Suppose now that we do not observe the state S_t of the Markov chain. Instead, at time t we observe Y_t. Y_t can be anything: integers, reals, vectors, images. The only condition is that the probability distribution of Y_t depends only on S_t (and not e.g. on S_{t-1}). Write this as $P_Y(Y_t|S_t)$, with $P_Y(\cdot|S_t = i)$ giving the probability (or probability density) of Y_t given $S_t = i$ (for simplicity, we let this same probability hold for all t). Also write $\mathbf{Y} = (Y_1, ...Y_T)$.

Our HMM model defines a probability distribution over $(\mathbf{S}, \mathbf{Y})$.

1. Write down $P(S_1 = s_1, S_2 = s_2, \ldots, S_T = s_T, Y_1 = y_1, \ldots, Y_T = y_t)$ in terms of π^1, A and P_Y.

For the earlier 'Markov chain' problem, we could quite efficiently calculate $\pi_i^t = P(S_t = i)$, the marginal prior probability of S_t. We will now calculate the marginal posterior probabilities $P(S_t = i|\mathbf{Y})$. Look at the scanned notes for the Kalman filter for reference.

From now onwards, we will fix the values of $\mathbf{Y}$, since these are our observations. Then define an $N \times 1$ vector B^t, with $B_i^t = P(Y_t = y_t|S_t = i)$.