Statistics for Business Analytics 2: R Laboratory Homework 3

Instructions

The goal of this assignment is to learn what the EM algorithm is and what it does. To begin with, I would like you to read the following:

http://ai.stanford.edu/~chuongdo/papers/em_tutorial.pdf

This is a very short 3-page tutorial that explains the EM algorithm at a very high level. In particular, the first half of the paper describes a simple experiment with coin tossing that will form the basis of this section of the assignment. If you have trouble understanding the assignment, let me know. You should submit your markdown file and make sure it runs. Also – please don't write anything in Greek in the markdown file, your submission should be in English.

Let's begin by setting up our investigation. Suppose we have two coins, A and B. The probability of Heads coming up with coin A, is 0.75. The second coin, B, is a fair coin. We are going to do 5 experiments with each coin. Each experiment will involve tossing the coin 10 times and recording the result of each toss. Thus we have 10 experiments in total, with each experiment involving ten coin tosses giving us a total of 100 coin tosses. Make sure you understand this very simple setup before moving on.

Question 1 (2 points)

Tossing a coin is essentially performing a Bernoulli trial. This is a specific case of sampling from the Binomial distribution with a size of 1. Write the command for generating 10 coin tosses of coin A (let's call 0 = Tails and 1 = Heads). Save this to a vector, v. Set the random number seed to 1 before doing this. Hint: Use one of the distribution functions.

Ok so you should now have a vector of length 10 with some 0's and some 1's where 0 represents Tails and 1 represents Heads.

Question 2 (1 point)

Write the simple command for counting the number of Heads that came up in the sample you just drew in the previous step.

Now, let's create a data structure that will contain the outcomes of our entire setup. We need 10 runs of 10 coin tosses. The first five are with coin A, the second are with coin B.

Question 3 (2 points)

Create a matrix M as follows. The rows of the matrix are the experiments and the columns are the coin tosses. The first 5 rows will therefore contain tosses made by coin A and the last five rows will contain tosses made by coin B. This is a 10 by 10 square matrix. Set the seed to 1 before you draw your first random sample.

There are many ways to do the above and not all will generate exactly the same data. Don't worry about this, just make sure you use coin A for the first five rows and coin B for the second five rows.

Question 4 (2 points)

Write the command that counts the number of Heads in each row and returns these counts in a vector. Consequently, I am expecting a vector of length 10 because we have 10 rows in the matrix. Hint: I definitely want to see a function from the apply() family of functions here.

Ok so far we've being playing around with R and setting up our random (albeit very small) experiments. Now let's see what we could do with all this. Suppose for a minute that I gave you the matrix M and I told you how I generated this matrix but I did not tell you the probability of Heads for each coin. I did however tell you that I am using coin A for the first five rows and coin B for the second five rows. Your task is for you to tell me for each coin, your best estimate for its corresponding probability of heads.

Question 5 (2 points)

Using the vector you computed in step 4, write the command that outputs the expected probability of Heads for coin A and the expected probability of Heads for coin B. You can put these two numbers in a single vector.

Notice that this is a very straightforward problem to solve. It is the same idea as predicting the mean of a normal distribution when we have a sample of that distribution for which we are not told the mean. In that case, we would use the mean of the sample as an estimate of the mean of the population. This is the process of parameter estimation using a sample and it is an essential step of many different algorithms that come up during modeling. For example, when we looked at the Naïve Bayes classifier with sentiment detection (Laboratory 5) we wanted to estimate the probability of a particular word appearing in a movie review given that the review was positive. We easily estimated this by computing the ratio of all positive reviews that had the word in question.

This brings us to the ideas of likelihood and maximum likelihood that are essential concepts to understand. Maximizing likelihood is often the criterion used behind popular methods of training models that we have studied in this class. Let's remind ourselves what likelihood is. Suppose we have some data and we want to fit this to a model. The model has certain parameters. In our coin tossing case, the parameters of the model are the probabilities of each coin drawing Heads. The likelihood function is a function of the model parameters that computes the probability that a model with a particular set of parameters generated the data we are considering. Changing the parameters of the model changes this probability because the data becomes more likely under some models and less likely under others. Notice that here that we are treating the data as fixed and allowing the parameters of the model to vary. The concept of maximum likelihood is nothing other than the largest possible value of the likelihood function and we are interested in the parameters that result in this maximum value. Thus, in a nutshell, we basically want to choose the parameters that make the data most likely under a particular model.

Hopefully, I've made this simple idea clear. The form of the likelihood function will differ from model to model, because different models have different structures and different parameters. The data is usually assumed to consist of independent observations and so the form of the likelihood function for some data is usually a product of the likelihood of each data point.

Question 6 (4 points)

Write a function that takes in a vector of coin tosses of arbitrary length (i.e. 0's and 1's as before) and computes the likelihood that the data was generated from a coin tossing model with a specific probability of Heads coming up. The function should look like this:

coinLikelihood <- function(v, headsProb) {

 # Your code here

}

For example, if we specify a vector v <- c(0,1,1,1,0,0,1) and a headsProb of 0.3 we should get back a single number that is the probability of this sequence of heads coming from a coin where the probability of heads is 0.3. Hint: Use a distribution function to compute the probability that each toss of the coin gave the result shown in the vector (remember – 0 = Tails, 1 = Heads), and multiply all these probabilities up. I did this in 2 simple lines (but you don't have to do it in 2 lines).

To visually demonstrate that the ratio of Heads (1's) in the vector is in fact the maximum likelihood estimate of the probability of Heads in the original coin, we can use the likelihood function you just wrote to plot likelihood as a function of different values of this probability.

Question 7 (3 points)

Use the function you just wrote above to plot the likelihood function for the following input vectors:

i) v1 <- c(0,1,0,1,0,0,1,1)
ii) v2 <- c(1, 1, 1, 1, 0, 0, 1, 1, 1,0)
iii) v 3<- c(0, 0, 0 , 0, 0, 0, 0)

Notice once again that we are keeping the data fixed (here the data is just the observation vectors of coin tosses) and giving different values for the headsProb parameter. We suggest you provide a range of different values with fixed increments to obtain a good plot.

The key point to observe here is that the likelihood functions you plotted take a maximum value at a value that is just the percentage of Heads in each vector.

We've talked about likelihood and maximum likelihood and saw an example of this for a simple coin-tossing model. The parameter estimates for the Naïve Bayes model that we saw in Laboratory 5 are another example. This approach works totally fine in both cases because we are in a situation where we have complete information about our model setup. In many cases however, we have partial information because there are certain variables in our system that are latent or hidden. This is where the EM algorithm comes in. *The EM algorithm is a method for estimating the likelihood function (or more importantly, its maximum value parameter settings) when we have latent variables.*

An example of this situation is when we know that we have two coins that are being tossed but not only do we not know the probability of heads for each coin (these are the parameters of the model that we want to estimate) we also don't

know which coin was used in each coin tossing experiment. Let's go back to our original matrix M. Essentially, what we are saying is that there are ten sequences of coin tosses. We don't know which coin was used in which sequence. We are still interested in estimating the probability of heads of each coin. Part of this will obviously involve us having to attribute each toss to a particular coin. It should be clear that this is a harder problem. An analogous problem going back to our sentiment analysis application is to estimate the particular probability of seeing a word in a positive movie review and the analogous probability for negative reviews, given a set of movie reviews for which we are not told whether they are positive or negative. This last example tells you that we can use the EM algorithm to train a Naïve Bayes model when the labels are missing.

Now, let's see how the EM algorithm works. It draws its name from the two steps that it will iterate over again and again. Let's find out what the E-step is. This is the expectation step. Here, the idea is that suppose we have a particular choice of model parameters. In this case, imagine that we have a current guess of what the probability of Heads is for each of the two coins. The goal is to look at our data and for each coin tossing sequence we will compute the likelihood of that sequence under EACH coin. That is, we are estimating the expected likelihood of each sequence under our current model. Next, we take the ratio of these two likelihoods and compute the relative probability that the sequence was generated by coin A compared to the sequence being generated by coin B. Thus, if the likelihood of sequence x being generated from coin A is 0.04, and the corresponding likelihood from coin B is 0.01, then the relative probability that this sequence was generated from coin A is 0.8. Let's compute a function that will perform our E-Step now.

Question 8 (4 points)

Write a function that takes as input a matrix where the rows are sequences of coin tosses. An example of this is the matrix M that we created earlier. This function should also take in a probability of Heads for coin A, and a probability of Heads for coin B i.e.:

```
coinEStep <- function (m, headsProbA, headsProbB) {

  # Your code here

}
```

The output of the function should be a single vector with the relative probability of each sequence in the matrix having come from coin A rather than coin B. Thus, the length of the output vector will be the number of rows of the input matrix m. Use the coinLikelihood function you wrote earlier.

Question 9 (2 points)

If we are given a vector of probabilities that each sequence was generated by coin A as the output of the function we just wrote, write down the (very) simple command for generating the relative probabilities of each sequence being generated by coin B.

Now we are going to discuss the M-Step. This step involves computing the maximum likelihood parameters of our model given our current beliefs about the latent variables. What is particularly interesting here is that we are not going to make a definitive assignment of each sequence to a coin i.e. say that sequence 1 was generated with coin A, sequence 2 with coin B etc… Instead we are going to weigh each sequence with a relative weight based on the relative probabilities we computed in the E-step. Sequences that were deemed more likely to have been generated with coin A will thus more heavily contribute to the computation of coin A's new probability of heads. *Consequently, to compute a new value for the probability of Heads in coin A, we will simply take a weighted sum of the number of heads in each sequence using the weights we obtained in the previous step. We will then divide this by the weighted total number of coin flips made, again using the weights of the previous step.* If you look at how you calculated the probability of Heads when you knew which sequence came from which coin, you basically performed the same calculation but you used binary weights of 0 and 1 for the different coin toss sequences. To compute the updated probability of coin B we will do the same calculation but with a different set of weights that can be computed from the weights obtained from coin A using the command you wrote for question 8 above. If you need a diagram to help visualize this process, look at the paper that I suggested at the beginning of this section. The relevant diagram is on the second page. Let's code up the M-step:

Question 10 (6 points)

Given a vector of relative sequence probabilities for coin A that is produced as output of the previous function, write a function that will recompute the probability of heads for coins A and B. This function also needs the original Matrix m from which you will have to compute the relative number of Heads in each sequence. Here is a signature:

```
coinMStep <- function(m, relativeProbsForCoinA) {

   # Your code here
}
```

Let's quickly recap what an application of the E and M step does. The E-step starts off with a guess for the coin probabilities and produces a weighted assignment of the coin tossing sequences to each coin. The M-step then takes these new weights and uses them to predict new values of the coin probabilities. Thus, in one iteration of the EM algorithm we have obtained a new estimate of our model parameters. It turns out (the math is beyond our scope here) that this algorithm converges to a local maximum so we can let the algorithm iterate until

a new application of the algorithm does not produce a change in the estimate of the model parameters. In practice, we want to set a tolerance level of this degree of change rather than rely on the precision of numbers in our system to do this for us.

Question 11 (2 points)

Write a simple command that takes in two numbers, x and y, and return TRUE when they are the same within a particular tolerance value e. This function will be used to compare two successive estimates of the probability of heads of a particular coin. By specifying e to be, say, 0.001 we are effectively saying that the difference between the two values is smaller than that. There is actually a function in R that does this so either find this function or write it out yourself.

We are now ready to put all our code together and produce a working version of the EM algorithm for our coin tossing experiment.

Question 12 (4 points)

Write a function that takes in a matrix of coin tosses, an initial guess of the probability of heads for coin A, an initial guess of the probability of heads for coin B, and a tolerance level. This function should output a vector containing the estimate of the probability of heads for the two coins. Here is the signature of the function:

```
coinTossEM <- function(m, coinProbsA, coinProbsB, e) {

  # Your code here

}
```

This is a simple function that uses the functions and commands that you wrote in the previous steps. In a nutshell your code should use a while loop that performs an E-step followed by an M-Step and keeps iterating while the new estimates of the coin probabilities have changed at least by an amount equal to the tolerance, e. Remember that in each iteration of the EM algorithm, you'll need to keep track of the current set of parameter estimates and the new set of parameter estimates after the E-Step and M-Step have been applied. This is so at the end, you have the two sets of estimates to compare against the tolerance.

Question 13 (2 points)

Use your function to compute estimates of the probability of Heads in coin A and coin B for the matrix M we constructed at the start of this section and see how close you can get to the correct answer by specifying different tolerances.

Question 14 (2 points)

Can you think of any starting probabilities that, when provided to the function we wrote, will cause problems with the algorithm so that it will not work as intended?

Well done if you made it this far, I know this assignment was a little tricky. You have implemented a complete version of the EM training algorithm albeit for a relatively simple problem. Even for this simple application, there are many improvements we can make to the algorithm and I will mention a few here for those of you who want to study this further:

- Instead of directly computing likelihood we actually tend to compute the logarithm of the likelihood as with very small numbers we might have problems with numerical precision.
- The convergence criterion is more commonly applied to the change in the overall likelihood not the probability estimates (minor issue)
- We could incorporate random restarts (like with k-means) to avoid arriving on a local maximum and we might also want to compute random starting conditions instead of supplying these by hand as the input to the function.
- We could remove inefficiencies such as re-computing the relative number of Heads in the matrix M at every M-step.
- We could add some sanity checking on the inputs for example to ensure that the parameters applied are of the correct type and form.
- Finally, we could generalize to more than two coins.