Programming Assignment 5 – Basic Probability, Computing and Statistics 2017

Jakub Dotlačil and Bas Cornellisen

Submission deadline: Wednesday, May 16th, 8 p.m.

Note: if the assignment is unclear to you or if you get stuck, do not hesitate to contact Jakub or Bas. You can also post a question on Canvas.

1 Assignment

This week we are finally going to implement the expectation maximisation algorithm (EM). This algorithm allows us to estimate the parameters of a mixture model, as discussed in the theoretical part of the course. You can read more about EM in the lecture notes of that course. A more concrete, incremental introduction (using different notation) can be found here. The data we will be working with have been generated from a mixture model whose components are *geometric* distributions. In week 6 of the theoretical part of the course, we discussed EM for this setting in class. We highly recommend you look at the slides of that session and at this sheet with all the computations. As you will see, you can test your EM implementation by checking if the numbers you produce match those in the sheet. However, note that the sheet uses probabilities, and you should use log-probabilities.

As said, our data have been generated from a mixture model. The mixture components are geometric distributions. Your task is to find the mixture weights and parameters of the mixture components. The data were generated from 3 mixture components, so this is the number of components that you should use as well. Recall that EM is sensitive to the starting point (i.e. the initial parameter estimate from which we start optimising). During peer review, we will provide you with a starting point whose resulting parameter estimates we have already computed. You will be assessed according to how well your algorithm does on this particular starting point. This implies that you will have to provide a place in your code (ideally at the beginning) where the reviewers can plug in that fixed starting point.

Implementation We have provided a partial implementation for you in the file `assignment5.py`. You have to implement the E-step and the M-step. The positions where you have to fill in your code are marked with TODO. Before you

start implementing anything, make yourself familiar with the code and ensure that you understand all the data structures it uses. You can also discuss the basic code on the forum. As usual, feel free to add any variables or functions/methods that you deem necessary. If you want to remove any of the data structures or methods that we provide, that's also fine. Also, you might want to use the class `GeometricDistribution`, provided in the assignment file.

Log-probabilities Our calculations are going to involve potentially large products of probabilities. Since probabilities are usually smaller than 1, these products can be very, very small. In fact, they become so small that our computers won't be able to represent them in memory and just turn them into `0`. The standard way to avoid this problem is to work with logarithms of probabilities (logprobs) instead. You should always use logprobs when performing probabilistic computations! Multiplication and division of probabilities is then straightforward because for positive numbers $a, b > 0$ it holds that

$$\begin{aligned}\log(a \cdot b) &= \log(a) + \log(b)\\ \log\left(\frac{a}{b}\right) &= \log(a) - \log(b)\\ \log\left(a^b\right) &= \log(a) \cdot b\end{aligned}$$

But what about addition and subtraction? That is, we are interested in the sum $a + b$ of two probabilities, but we only have $c = \log(a)$ and $d = \log(b)$. How do we get $\log(a + b)$? The naïve way to do addition would be through simple exponentiation

$$\log(\exp(c) + \exp(d)) = \log(a + b),$$

since e.g. $\exp(c) = \exp(\log(a)) = a$. This is inefficient, however, since we need to compute two exponentials and one logarithm. Computers are slow at that. Instead, we choose to exploit the following equivalence which is often called the *log-sum-exp* trick. Without loss of generality, we also assume that $c > d$.

$$\begin{aligned}\log(\exp(c) + \exp(d)) &= \log(\exp(c) \cdot (1 + \exp(d - c)))\\ &= c + \log(1 + \exp(d - c))\end{aligned}$$

There are several advantages to using this trick. First, we only compute one exponential and one logarithm. Second, the logarithm computation is already implemented in many programming languages (including Python) as a function called `log1p` (see here for documentation). The `log1p` function computes $\log(1+\exp(d-c))$ very efficiently when the exponent is small. This is the reason that we want to subtract the bigger number (c according to our assumption). This way, we make sure that the exponent is small and thus `log1p` performs fast computation.

We have provided several functions (in `logarithms.py`) that help you working with log-probabilities. Most importantly, `log_add` implements the log-sum-

exp trick, so if $c = \log(a)$ and $d = \log(b)$, then

$$\texttt{log\_add}(c, d) = \log(a + b) \tag{1}$$

The function `log_add_list` should also be useful; have a look at the code to find out what it does. Do play around with these functions. Understanding how to work with log-probabilities is essential if you want to correctly implement this assignment.

Hint about the log-likelihood In order to compute the log-likelihood for the data set, you need to add the log-likelihoods of all data points (this is ok because we assume that the data points are independent given the mixture components). In order to compute the log-likelihood of the data point x, you need to sum over all possible mixture assignments.

$$\log(P(X = x)) = \log\left(\sum_c P(X = x|Y = c)P(Y = c)\right) \tag{2}$$

Observe that you already compute this sum as part of computing the posterior $P(Y = c|X = x)$. You can hence reuse the result you get there in order to compute the log-likelihood. This implies that the log-likelihood that you print out at after iteration t is the log-likelihood obtained under the parameters of iteration $t - 1$ but that is ok.

Debugging Debug your code using a simple example with two components and the following parameters:

$$w_1^{(0)} = 0.2,\ w_2^{(0)} = 0.8, \qquad \theta_1^{(0)} = 0.2, \theta_2^{(0)} = 0.6, \tag{3}$$

where $w_1^{(0)}, w_2^{(0)}$ are the initial mixture weights and $\theta_1^{(0)}, \theta_2^{(0)}$ are the initial parameters of the two geometric distributions. If you moreover use the example data (`geometric_example_data.py`), the setting is identical to the one discussed in class in week 6 of the theoretical part (see the slides of that session and sheet mentioned above).

Recall that EM is guaranteed to always increase the log-likelihood of the data. Therefore, the code prints the log-likeihood after each iteration. If it goes up after each iteration or stays unchanged, your implementations is probably correct (no guarantees, though). Please make sure to be very strict about this. Even if an EM implementation is wrong, the log-likelihood often still increases during the initial iterations. Pay close attention to later iterations. If they are volatile, you have a bug! EM usually does not need all too many iterations before it converges. Always run 20 iterations, which should be enough for this data set.

2 Grading

You will be graded on your performance from a specific starting point, i.e. a specific setting of initial mixture weights and parameters of mixture components. Make sure that these can be provided to your code. This can be done either by defining list variables at the beginning of your python file (one list of the mixture weights, one list for the component parameters) or, more elegantly, by asking the user to provide a text file in a specific format from which your program reads the initial parameter settings.

1 point All additional classes and functions/methods have docstrings. Award 0 points here if there are one or more classes/functions/methods that do not have a docstring. If the student didn't add any classes or functions, award 1 point by default.

2 points If `initial_mixture_weights` and `initial_geometric_parameters` are provided, they are correctly taken into account in the first step. If these parameters are not provided, random initialisation values are used.

2 point The e-step is correctly implemented. Inspect the code to check this.

2 point The m-step is correctly implemented. Inspect the code to check this.

2 points The log-likelihood increases monotonically when the algorithm is run. It's ok if after a couple of iterations the log-likelihood does not change anymore, as long as it does not go down. (Be strict here!)

1 points Your parameter estimate after 20 iterations for the fixed initial parameter settings is equal to the parameter estimate that we will provide during the peer review period.