Andrew Gordon

CMSC423 Project 2 Writeup

Dr. Patro

4/21/20

Detailing My Findings

Results:

What was the runtime of the full-model versus the equivalence class-based model?

After running the two algorithms independently on the same laptop, the results were that the equivalence class algorithm ran in 97 minutes and the full model EM ran in 49 minutes. I originally predicted that the equivalence class model was going to be the faster algorithm by far, however that was clearly not the case. This definitely was a shock to me, however when I analyzed this further, I realized that the equivalence class model was a lot slower to converge than the full model was. For example, my convergence case was when all values increased or decreased in count by less than one in the current iteration. For the equivalence class, some counts would be 1.498 away from the previous count, and then 1.493 in the next iteration, and then that trend would continue, causing dozens of iterations of the algorithm for that one value to converge. When you take into account that there are two hundred thousand transcripts, this clearly would take a long time to converge at this rate. The full model on the other hand converged relatively quickly when compared to the other model.

Another realization that I had was that I may have sped up the full model algorithm by precomputing all of it's values. My architecture was to parse the data which would take the same amount of time for each model, then run an iteration, check for convergence using the same

function for each model, and then iterate until convergence. In my parsing step, I precomputed the value for p_2, p_3, p_4, and computed the value of those values all multiplied together since this was a constant for each read. By doing this, a lot of time would be saved in computation in both models, however I believe it may have increased the efficiency of the full model more than it did for the equivalence class model.

Lastly, the equivalence class model ran two loops for each set of transcripts which may have slowed it down as well. Part of the calculation for the equivalence class model was that we needed to obtain the sum of all the previous n array values so that we can divide each individual value by it to obtain the new update value for each transcript. This may seem like a small computation, however one thing that I learned in this class was that no computation is small when working with massive datasets. When you combine this with the fact that the equivalence class was slow to converge, this means that this loop was being run a lot of times which could definitely be a bottleneck for time.

Overall, my prediction wasn't correct, however after analyzing it further I realized that simplicity doesn't always correlate with efficiency. The full algorithm definitely wasn't as simple since it included both p_3 and p_4 values as well, however it had less overhead at each step and proved to be the faster algorithm.

What was the difference in the memory used by these variants?

Another big aspect we must consider when writing programs is the memory that they use. When you're working with small datasets, this is less apparent, however when you scale to big data, it's a different story. In this project, we worked with a massive dataset so it was important

to be as efficient as possible with the storage of data and the lookup times required to access that data later on.

In order to compute the full model EM, you need a few more data points than you need for the equivalence class model, namely you need the p_3 and p_4 values. Since I made the decision to precompute the p_2 value for each transcript during the parsing step, I decided that it would make sense to save on storage space and precompute the p_3 and p_4 values as well. From there, it only made sense to multiply those three numbers together since they don't vary, giving us a number that represents the update value for each read with the exception of p_1 which varies each iteration of the EM algorithm. From here, a lot of memory was saved considering that we didn't have to store the values for p_3 and p_4, saving us 2 * total number of reads * the size of a float variable total memory. We still, however, had to store the value obtained by multiplying p_2, p_3 and p_4 for each read which is stored in a list called "precompute" which holds the update value for each read by index of the read. Since there are 1,000,000 total reads, this takes up a lot of memory despite the memory we saved by not storing both p_3 and p_4.

When we look at the equivalence class model, it uses a lot less memory purely because of its simplicity. During parsing, we only need to compute the p_2 value for each transcript and save space by not storing the p_3 and p_4 values. The only other area that we use a chunk of memory is when we create the cluster dictionary for each set of reads, however the maximum size that this can ever reach is 1,000,000 due to that being the total number of alignments and in practice, we know that would never happen. The size of the cluster dictionary tends to be a lot

smaller than that due to the fact that certain transcripts appear more often than others and a lot of alignment reads tend to be generated from the same transcripts.

Overall, the equivalence class EM algorithm uses a lot less memory than the full model EM algorithm does, however that's the tradeoff that we have to make for more accurate estimations.

What was the difference in the quality of the predictions (which prediction was more accurate, and by how much)?

Full EM: Spearman Correlation=0.673, MSE: ~1,686, MAE: 1.495

Equivalence Class EM: Spearman Correlation=0.541, MSE: ~5,978, MAE: 4.686

According to statistics.laerd.com, a spearman result above .7 is considered a strong correlation and my full EM algorithm produced a result that is fairly close to that. In the example that they computed, the correlation between math and english grades improving granted a spearman correlation of .67 and they considered that a very strong correlation. As far as the equivalence class algorithm, the spearman results were a little less favorable, however that was expected. In reality, the correlation between my equivalence class algorithm and the real counts isn't bad at all. A correlation of .541 puts us right around a moderate positive relationship which isn't bad considering how quickly it can be computed, however when comparing it to the full model, it's obviously a lot less accurate.

In terms of mean squared error, we can see a pretty large difference between the two algorithms. First, we have the full EM algorithm that grants us a MSE of ~1,686 which is pretty low considering that this is an estimation and there are over 200,000 transcripts that are being

estimated here. Mean squared error is a tough metric to interpret because it will vary based on the size of your dataset. Luckily, we have two estimations so we can compare them to one another. As you can see, our MSE for the equivalence class algorithm is almost four times larger than the MSE computed for our full model EM, showing just how much more accurate the full algorithm is. According to statisticshowto.com, mean squared error tells us how far away our data lies from the regression line, ultimately telling us how our estimate compares to the true counts. The lower the number is, the better the estimation. Clearly, the full model EM is a lot better of an estimation than the equivalence class estimation was.

The last metric that I chose to use to compare the two estimations was the mean absolute error which I thought was extremely straightforward and easy to interpret. All that it does is take the error for all values in the true and estimated sets which is found by subtracting true from estimate, take the absolute value of that, and then find the mean of the entire set. This gives us the average value that the estimated data is off by, giving some further insight into how great the estimation really was. For more than 200,000 transcript estimated counts, the full model EM algorithm granted a MAE of 1.495 which sounds pretty good if we're being reasonable. That's an extremely large dataset and to have the average count be off by 1.5 is extremely good. On the other hand, we have the equivalence class estimation which turned out to be closer to 4.7, which in my opinion, isn't terrible either. For an algorithm as basic as it is, I was actually a little surprised that it's estimations were this close. When you compare the estimations of the full model, you see that it's not as great, however you also have to consider the difference in speed and memory which is clearly a tradeoff for accuracy.

It's clear that both of these estimation algorithms have a clear purpose and usage. The full model is more accurate, and despite the memory it uses and the speed it runs at, it's sometimes necessary to generate relatively strong estimates. In the case of the equivalence model, sure, the accuracy isn't great, however the memory usage is lower than the other model and the speed is notably better, and sometimes the slight loss in accuracy is shadowed by speed and memory usage. Ultimately, there's a lot to learn from both of these algorithms and it's a great example of a long time debate between speed and accuracy.

Design:

What specific design decisions did you make in your tool, and why?

When I reflected on the last project in this course, I realized my biggest downfall was processing speed. Through my first iteration of the last project, it was going to take over five days to run all of the alignments, then I was able to get that down to three days, and finally I was able to cut it down to a little over a day. I knew that going into this project I would make sure that didn't happen again.

Before writing a line of code, I made sure my understanding of this project was crystal clear so that I was as efficient as possible. I mapped out exactly how I wanted each function to look, the runtime of that function, and begun to think about small things that could be done to save time. The first thing I did was get approval from Dr. Patro on the way I was storing my data. He told me that my approach was adequate and gave me some advice for making my implementation even faster. Through this, I made the decision to reference transcripts by index

instead of name. This made the lookup time a lot faster, considering my original idea was to have a dictionary that mapped names to length (which definitely could've hurt performance).

From then on, I continued thinking with this mentality of speed and utilized these quick lookups in my EM algorithms to ensure no time is lost in that aspect. When I ran my code though, it was still extremely slow. Through the help of Dr. Patro once again, he gave me some advice about precomputing the effective lengths for all possible values and then having a lookup array. This was where I saw a significant speed increase. The step of parsing my transcripts which previously would have taken 8 hours was now running in seconds. At this point, I was certain that my implementation was efficient for both of the EM algorithms and that the time was all going to be spent on the EM algorithms, allowing me to get a true reading of how the full model compares to the equivalence class model in terms of speed.

If you were going to start the project again from scratch, what, if anything, would you do differently?

As I think about this, nothing immediately pops into my head. I learned a lot from the last project and made sure that I wasn't going to put myself through what I endured during that. To prevent that, I planned out everything pretty well, had a really solid understanding of the task we needed to accomplish in this project before writing a line of code and placed efficiency at the top of my list during my implementation of the EM algorithms.

The only thing that I could think of that I would do differently may be to have done this project on my dad's new laptop from the start. I talked about doing this last project since my laptop is four years old and isn't as fast as it used to be, however I never did it. After finishing

my implementation of the code on my laptop, I decided to run the algorithms on my dad's laptop, however it's brand new so installing everything from python and vim to git was excruciatingly painful (Windows is terrible). If I chose to use his laptop from the start, I may have had an easier time developing the code and testing it as I went along since it's seemingly faster and new. Aside from that, I think my approach to this project was pretty good after learning some lessons in the last one and I'm happy with how quickly I was able to finish it.

Roadblocks:

What part of the project did you find the most difficult?

The most difficult part of this project for me had to be the initial understanding of what we were trying to accomplish. I'm not very good at learning through a computer so this semester has proven to be extremely tough for me. During lecture, I can follow what's going on, however, for some reason it's a lot harder for me to actually comprehend it through Zoom as opposed to in a classroom. For this reason, I was pretty lost and the project description was difficult to understand since my base knowledge of EM was missing (due to my inability to follow online lectures).

Luckily, after I put out a post about my struggles with adapting to this change in the semester, Dr. Patro was kind enough to create an entire write up that simplified the process and gave examples that were easy to follow. Without this, I'm not sure that I would be anywhere near finishing this project. It elevated my understanding and allowed me to map out exactly how to tackle implementing these algorithms. Due to this, my understanding of EM is at a level that I believe it would have been if classes were still in person and I'm extremely thankful for that.

Where did you get stuck and how did you (hopefully) overcome this difficulty?

The hardest part of this project for me was going back after finishing it and trying to find where the bottlenecks in speed were. It seemed like speed wasn't going to be an issue in this project since I ensured everything was as efficient as possible and Dr. Patro said this project is a lot less computationally heavy than the last one, however my laptop still couldn't handle it. For that reason, I had to go back and make a lot of changes to increase efficiency. This wasn't just a simple fixing a loop and making it faster. I decided to precompute all values ahead of time so that they were computed 200,000 times instead of millions of times. This dramatically increased the speed of my program.

Aside from that, a big challenge was understanding the project as a whole. It's definitely not an easy topic and the formulas are overwhelming at first. As I stated above, nothing was helping me understand the algorithm and what it was seeking to accomplish until Dr. Patro put out the write up that clarified any and all of my questions. From then on, implementation was pretty fun and interesting, and then the challenge was speeding it up.

All in all, this project was extremely interesting and challenging. I really enjoyed working with large datasets because it allowed me to learn a lot that I can take with me into my career. I've never had to ensure that my code scales with such large datasets and this course helped me realize that planning out your code ahead of time is important. In the past, all of the programs I wrote ran in seconds no matter how complex it was, however that's clearly not the case when there's millions of alignments and transcripts. The challenges of this course only made me a better programmer and was one of the only courses at Maryland that I feel truly

prepared me for the real world so I cherish these experiences and am grateful to Dr. Patro and our TAs for helping me learn these lessons.

Sources:

https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide-2.php

https://www.statisticshowto.com/mean-squared-error/