This is an optional assignment. You do not need to do it.

Assignment 6

MET CS 777 - Big Data Analytics
GMM - Topic Models - Gibbs Sampling
(20 points)

(there is no GitHub link. You need to submit only a zip file)

1 Description

In this assignment, you will be deriving an MCMC algorithm for and implementing and then applying a text clustering model.

There are two sub-assignments: First (1) do the math necessary to derive your algorithm and then write it up (create a PDF file or pencil-and-paper). Then (2) you will implement the algorithm and use it to cluster a corpus of text documents. I'd suggest that you use Spark for your implementation (especially if you are comfortable with Spark at this point), at least to do the initial processing of the input data set, though this is certainly not required. It is also not required to run your learning algorithm on Cloud Amazon/Google. That is fine if your implementation is fast enough to run on your laptop. The data set we are using this time around is significantly smaller than the data sets we've been using for the last few assignments, so EC2 might not be necessary.

NOTE: it is acceptable to work in teams of two or three on this assignment if you would like, though this is optional.

2 Data

You will be dealing with the widely-used "20 newsgroups" data set
http://qwone.com/~jason/20Newsgroups/

A newsgroup post is like an old-school blog post, and this data set has 19,997 such posts from 20 different categories, according to where the blog post was made. The examples of 20 categories are : alt.atheism, comp.graphics, ... talk.politics.mideast, talk.religion.misc

The format of the text data set is the same as the text format used for the documents in A5 and A4; the category name can be extracted from the document's name.

For example, the document with identifier 20 newsgroups/comp.graphics/37261 is from the comp.graphics category. The document with the identifier 20 newsgroups/sci.med/59082 is from the sci.med category. We have prepared the data in a single-line format to be read line by line. Each line is a document.

Direct Download link:
https://raw.githubusercontent.com/trajanov/BigDataAnalytics/master/Spark-Example-Word-Count/20-news-same-line.txt

3 Assignment Tasks

You need to complete four separate tasks to finish the assignment.

3.1 Task 1 : (5 points)

First, your task is to derive and then write up an MCMC algorithm that will allow you to cluster the text documents; if everything works properly, the 20 clusters that you obtain by clustering the documents will roughly correspond to the 20 categories present in the data.

You'll want to use the example MCMC derivation for the mixture of Gaussians from the lecture ”Learning Mixture Models” in slides as the basis for your derivation, though our model will be slightly different. Also, look at the example of how we derived the MCMC algorithm for learning LDA - we did that one in quite a bit of detail, though again, our model here is just a bit different. In your answer, you should give the algorithm in enough detail (including exact formulas) that someone can go off and implement it.

Of course, when you derive an MCMC algorithm for Bayesian machine learning, you always need a generative process that you are trying to reverse. Our generative process will look a lot like the generative process for the Gaussian mixture model, except that the data that our generative process is creating is a list of term count vectors rather than multi-dimensional points as in a GMM. Thus, we will replace our mixture of Gaussians with a mixture of Multinomials.

Given k document clusters or categories, our generative process for n documents (where the number of words in document i is l_i is as follows:

```
$\pi$   $Dirichlet(\alpha)$
for $j = 1$ to $k$ do
    $\mu_j$   $Dirichlet(\beta)$
end for
for $i = 1$ to $n$ do
    $c_i \sim Categorical(\pi)$
    $x_i$   $Multinomial(\mu_{c^i}, l_i)$
end for
```

The result of this generative process is a list of n vectors, where x_i gives us the number of occurrences of each word in the ith document (that is, $x_{i,j}$ is the number of occurrences of word j in document i).

Given this, what I am looking for out of this first task is a precise description of the Gibbs sampler that is going to recover the vector π that gives us the prevalence of each of the categories, the k different μ_j probability vectors that tell us how prevalent each word is in each category in the document, and (finally) the n different c_i values that tell us the category of each document.

3.2 Task 2 - (5 Points)

Next (this is very similar to A4 and A5) you will process the input corpus, build a dictionary, and transform each of the input documents into a count vector that has 20,000 entries—note that last time, we used TF vectors, but since here we have a Multinomial model, we'll be using count vectors. For example, for a particular document, the entry in the 177th value in this vector is an integer that tells us the number of times the 177th most common word in the dictionary appeared in this document.

To get credit for this task, give the number of times each of the 100 most common dictionary words appears in document 20 newsgroups/comp.graphics/37261

Note-1: It is a good idea to map all words to lowercase so that the same word with different capitalization is matched and can be counted up as the same word.

3.3 Task 3 - (5 Points)

Now you will implement the MCMC algorithm that you derived in Task 1, and run it for 200 iterations on the 20 newsgroups data, with the goal of learning the π, μ_j , and c_i values. Learn 20 different mixture components.

To get credit for this task, I will ask you to print out the 50 most important (highest probability) words in each of the 20 mixture components that you learned. Hopefully, these are somewhat meaningful!

Note-2: Every iteration has to do five things:

1) Map where you produce (c_i, x_i) pairs from each x_i.

2) Reduce where you sum up all of the x_i's with the same c_i value.

3) Reduce where you count the number of x_i's assigned to each cluster.

4) Update the various μ's using the result of 2.

5) Update π using the result of 3.

Note-2: And of course, make sure to use NumPy arrays, and don't loop through the arrays... use bulk operations!

Note-3: It is a good idea to save your results every once in a while (after some iterations e.g., each 10 iterations) so you don't lose anything.

Note-4: To help you out, I'm going to supply you with an implementation of a function that, given a document with word count vector x, will use NumPy to compute the vector of probabilities where the jth entry is Pr[document x came from $|$category j x, π, and all $\mu' s$].

The code is quite efficient in that it does not loop through all of the entries in x or in any particular mu. Instead, it uses efficient, bulk NumPy operations.

The Python code (getProbs.py file) is a function that, given a particular data point x_i, as well as the set of μ_j vectors and pi, computes and returns the p vector. So then, if you use the getProb Python function code, all you need to do is to take a sample from a Categorical distribution using the vector getProb() code gives you (using SciPy/NumPy, you would do this by sampling from a Multinomial with one trial — this'll give you a NumPy array like [0,0,0,0,1,0,0], where the 1 indicates what category was returned; you can use numpy.nonzero to find the non-zero element efficiently).

Note-5 Finally, let me make one last comment. One good reason that you might not be able to run this on your laptop is if you don't have enough RAM. Storing all 20,000 document vectors as NumPy arrays is going to take 3.2GB of RAM (each document has 20,000 entries because the dictionary size is 20,000 words; 400M entries X 8B each is 3.2GB). If this exceeds your machines's RAM (or if you just can't get it to run fast enough), **you can reduce the dictionary size to 10,000 words.**

3.4 Task 4 - (5 Points)

Finally, let's see how accurate your learning algorithm is. For each of the 20 mixture components that you learn, look at all of the documents that are assigned to it (the c_i values tell you this assignment). First, print out the number of documents assigned to the cluster. Then print out the top three "real" categories for all of the documents assigned to the cluster and the percentage of assigned documents to the cluster that fall in that category. For example, you might have a cluster that has 3,123 documents assigned to it, where the top three real categories are sci.space (34%), sci.med (12%), and sci.electronics (5%). Print out such information for each of the components that you learn. Ideally, each component will have a high prevalence of a particular category (or at least, it will have a high prevalence of several related categories).

Finally, write a paragraph telling us if you think that your clustering algorithm worked!

Important Considerations

4.1 Machines to Use

One thing to be aware of is that you can choose virtually any configuration for your Cloud Cluster - you can choose different numbers of machines and different configurations of those machines. And each is going to cost you differently! Since this is real money, it makes sense to develop your code and run your jobs locally, on your laptop, using the small data set. Once things are working, you'll then move to Cloud.

As a proposal for this assignment, you can use the **n1-standard-4** or **e2-standard-4** machines on the Google Cloud, one for the Master node and two for worker nodes.

Remember to delete your cluster after the calculation is finished!!!

More information regarding Google Cloud Pricing can be found here https://cloud.google.com/products/calculator. As you can see average server costs around 50 cents per hour. That is not much, but **IT WILL ADD UP QUICKLY IF YOU FORGET TO SHUT OFF YOUR MACHINES**. Be very careful, and stop your machine as soon as you are done working. You can always come back and start your machine or create a new one easily when you begin your work again. Another thing to be aware of is that Google and Amazon charge you when you move data around. To avoid such charges, do everything in the "Iowa (us-cental1)" region. That's where data is, and that's where you should put your data and machines.

- You should document your code very well and as much as possible.
- Your code should be compilable on a Unix-based operating system like Linux or macOS.

4.2 Academic Misconduct Regarding Programming

In a programming class like ours, there is sometimes a very fine line between "cheating" and acceptable and beneficial interaction between peers. Thus, it is essential to fully understand what is and what is not allowed in collaboration with your classmates. We want to be 100% precise, so there can be no confusion.

The rule on collaboration and communication with your classmates is very simple: you cannot transmit or receive code from or to anyone in the class in any way—visually (by showing someone your code), electronically (by emailing, posting, or otherwise sending someone your code), verbally (by reading code to someone) or in any other way we have not yet imagined. Any other collaboration is acceptable.

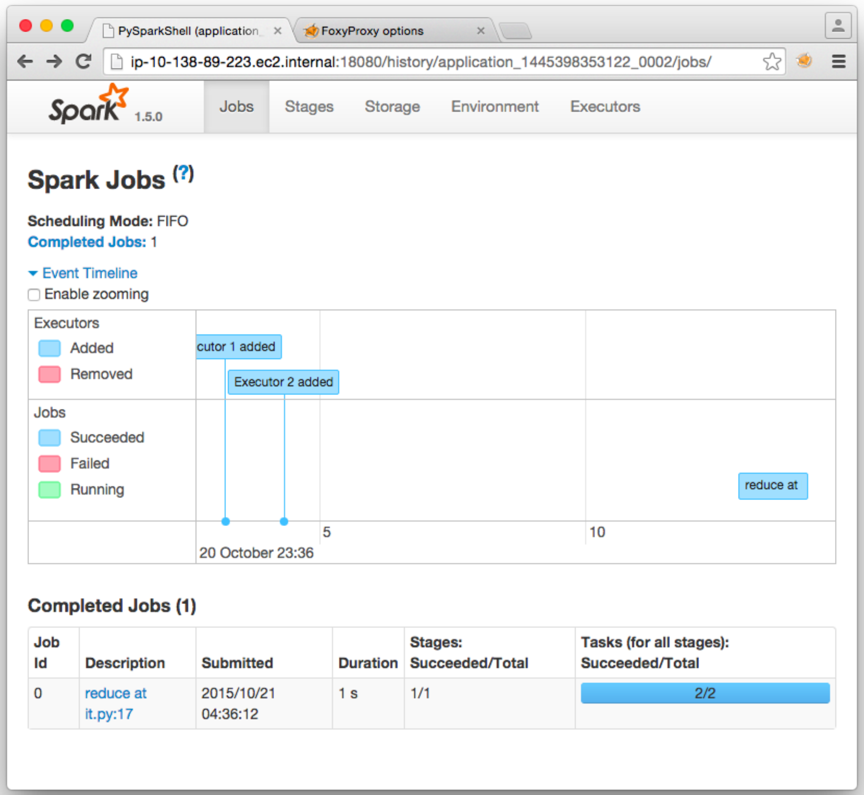
The rule on collaboration and communication with people who are not your classmates (or your TAs or instructor) is also very simple: it is not allowed in any way, period. This disallows (for example) posting any questions of any nature to programming forums such as **StackOverflow**. As far as going to the web and using Google, we will apply the **”two-line rule”**. Go to any web page you like and do any search that you like. But you cannot take more than two lines of code from an external resource and actually include it in your assignment in any form. Note that changing variable names or otherwise transforming or obfuscating code you found on the web does not render the ”two-line rule” inapplicable. It is still a violation to obtain more than two lines of code from an external resource and turn it in, whatever you do to those two lines after you first obtain them.

Furthermore, you should cite your sources. Add a comment to your code that includes the URL(s) that you consulted when constructing your solution. This turns out to be very helpful when you're looking at something you wrote a while ago and you need to remind yourself what you were thinking.

4.3 Turnin

Create a single document that has results for all three tasks.

Also, for each task, for each Spark job you ran, include a screenshot of the Spark History.



Spark Jobs (?)

Scheduling Mode: FIFO
Completed Jobs: 1

Completed Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	reduce at it.py:17	2015/10/21 04:36:12	1 s	1/1	2/2

Figure 1: Screenshot of Spark History

Please zip up all of your code and your document (use .zip only, please!), or else attach each piece of code and your document to your submission individually.

Please have the latest version of your code on GitHub. Zip the GitHub files and submit your latest version of assignment work to Blackboard. We will consider the latest version of the Blackboard.