Assignment 6

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

GitHub Classroom Invitation Link
`https://classroom.github.com/a/_PNcma__`

1 Description

In this assignment, you will be deriving a 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 that you run your learning algorithm on Cloud Amazon/Google. If your implementation is fast enough to run on your laptop, that is fine. The data set that 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 fine 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 exactly the same as the text format used for the documents in A5 and A4; the category name can be extracted from the name of the document.

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 as single line format so that it can be read line by line. Each line is a document.

Direct Download link from AWS
`https://s3.amazonaws.com/metcs777/20-news-same-line.txt`

	Amzon AWS
20 news data - on line (35.2 MB of text)	s3://metcs777/20-news-same-line.txt

Table 1: Data set on Amazon AWS - URLs

	Google Cloud Storage
20 news data - on line (35.2 MB of text)	gs://metcs777/20-news-same-line.txt

Table 2: Data set on Google Cloud Storage - URLs

3 Assignment Tasks

There are four separate tasks that you need to complete 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 own derivation, though our model will be just a bit 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 a MCMC algorithm for Bayesian machine learning, you always need a generative process that you are trying to reverse. Our generative process is going to 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 are going to 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:

$\pi \sim Dirichlet(\alpha)$
for $j = 1$ to k **do**
    $\mu_j \sim Dirichlet(\beta)$
end for
for $i = 1$ to n **do**
    $c_i \sim Categorical(\pi)$
    $x_i \sim Multinomial(\mu_{c_i}, l_i)$
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 that each of the 100 most common dictionary words appear 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 are matched and can be count up as same word.

3.3 Task 3 - (5 Points)

Now you will actually 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 am going to 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 iteration) 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 $j|x, \pi$ 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 documents that are assigned 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!

3.5 Turnin

Create a single document that has results for all three tasks. For each task, copy and paste the result that your last Spark job wrote to Amazon S3. Also for each task, for each Spark job you ran, include a screen shot of the Spark History.

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

Please have the latest version of your code on the GitHub. Zip the files from GitHub and submit as your latest version of assignment work to the Blackboard. We will consider the latest version on the Blackboard but it should exactly match your code on the GitHub