COMP 543 Assignment #5

Due April 16th at 11:55PM.

1 Description

In this assignment, you will be implementing a regularized, logistic regression to classify text documents. The implementation will be in Python, on top of Spark. To handle the large data set that we will be giving you, it is necessary to use Amazon AWS.

You will be asked to perform three subtasks: (1) data preparation, (2) learning (which will be done via gradient descent) and (3) evaluation of the learned model.

2 Data

You will be dealing with a data set that consists of around 170,000 text documents and a test/evaluation data set that consists of 18,700 text documents. All but around 6,000 of these text documents are Wikipedia pages; the remaining documents are descriptions of Australian court cases and rulings. At the highest level, your task is to build a classifier that can automatically figure out whether a text document is an Australian court case. We have prepared three data sets for your use.

1. The *Training Data Set* (1.9 GB of text). This is the set you will use to train your logistic regression model:

   `https://s3.amazonaws.com/risamyersbucket/A5/TrainingDataOneLinePerDoc.txt`

   or as direct S3 address, so you can use it in a Spark job:

   `s3://risamyersbucket/A5/TrainingDataOneLinePerDoc.txt`

2. The *Testing Data Set* (200 MB of text). This is the set you will use to evaluate your model:

   `https://s3.amazonaws.com/risamyersbucket/A5/TestingDataOneLinePerDoc.txt`

   or as direct S3 address, so you can use it in a Spark job:

   `s3://risamyersbucket/A5/TestingDataOneLinePerDoc.txt`

3. The *Small Data Set* (37.5 MB of text). This is for you to use for training and testing of your model on a smaller data set:

   `https://s3.amazonaws.com/risamyersbucket/A5/SmallTrainingDataOneLinePerDoc.txt`

Some Data Details to Be Aware Of. You should download and look at the `SmallTrainingData.txt` file before you begin. You'll see that the contents are sort of a pseudo-XML, where each text document begins with a `<doc id = ... >` tag, and ends with `</doc>`. All documents are contained on a single line of text.

Note that all of the Australia legal cases begin with something like `<doc id = ``AU1222'' ...>`; that is, the doc id for an Australian legal case always starts with `AU`. You will be trying to figure out if the document is an Australian legal case by looking only at the contents of the document.

3 The Tasks

There are three separate tasks that you need to complete to finish the assignment. As usual, it makes sense to implement these and run them on the small data set before moving to the larger one.

3.1 Task 1

First, you need to write Spark code that builds a dictionary that includes the 20,000 most frequent words in the training corpus. This dictionary is essentially an RDD that has the word as the key, and the relative frequency position of the word as the value. For example, the value is zero for the most frequent word, and 19,999 for the least frequent word in the dictionary.

To get credit for this task, give us the frequency position of the words "applicant", "and", "attack", "protein", and "car". These should be values from 0 to 19,999, or -1 if the word is not in the dictionary, because it is not in the top 20,000.

Note that accomplishing this will require you to use a variant of your A4 solution. If you do not trust your A4 solution and would like mine, you can post a private request on Piazza.

3.2 Task 2

Next, you will convert each of the documents in the training set to a TF-IDF vector. You will then use a gradient descent algorithm to learn a logistic regression model that can decide whether a document is describing an Australian court case or not. Your model should use l_2 regularization; you can play with things a bit to determine the parameter controlling the extent of the regularization. We will have enough data that you might find that the regularization may not be too important.

I am going to ask that you *not* just look up the gradient descent algorithm on the Internet and implement it. Start with the LLH function from class, and then derive your own gradient descent algorithm. We can help with this if you get stuck.

At the end of each iteration, compute the LLH of your model. You should run your gradient descent until the change in LLH across iterations is very small.

Once you have completed this task, you will get credit by (a) writing up your gradient update formula, and (b) giving us the fifty words with the largest regression coefficients. That is, those fifty words that are most strongly related with an Australian court case.

3.3 Task 3

Now that you have trained your model, it is time to evaluate it. Here, you will use your model to predict whether or not each of the testing points correspond to Australian court cases. To get credit for this task, you need to compute for us the F1 score obtained by your classifier—we will use the F1 score obtained as one of the ways in which we grade your Task 3 submission.

Also, I am going to ask you to actually look at the text for three of the false positives that your model produced (that is, Wikipedia articles that your model thought were Australian court cases). Write paragraph describing why you think it is that your model was fooled. Were the bad documents about Australia? The legal system?

If you don't have three false positives, just use the ones that you had (if any).

4 Important Considerations

Some notes regarding training and implementation. As you implement and evaluate your gradient descent algorithm, here are a few things to keep in mind.

1. To get good accuracy, you will need to center and normalize your data. That is, transform your data so that the mean of each dimension is zero, and the standard deviation is one. That is, subtract the mean vector from each data point, and then divide the result by the vector of standard deviations computed over the data set.

2. When classifying new data, a data point whose dot product with the set of regression coeffcients is positive is a "yes", a negative is a "no" (see slide 16 in the GLM lecture). You will be trying to maximize the F1 of your classifier and you can often increase the F1 by choosing a different cutoff between "yes" and "no" other than zero. Another thing that you can do is to add another dimension whose value is one in each data point (we discussed this in class). The learning process will then choose a regression coefficient for this special dimension that tends to balance the "yes" and "no" nicely at a cutoff of zero. However, some students in the past have reported that this can increase the training time.

3. Students sometimes face overflow problems, both when computing the LLH and when computing the gradient update. Some things that you can do to avoid this are, (1) use `np.exp()` which seems to be quite robust, and (2) transform your data so that the standard deviation is smaller than one—if you have problems with a standard deviation of one, you might try 10^{-2} or even 10^{-5}. You may need to experiment a bit. Such are the wonderful aspects of implementing data science algorithms in the real world!

4. If you find that your training takes more than a few hours to run to convergence on the largest data set, it likely means that you are doing something that is inherently slow that you can speed up by looking at your code carefully. One thing: there is no problem with first training your model on a small sample of the large data set (say, 10% of the documents) then using the result as an initialization, and continue training on the full data set. This can speed up the process of reaching convergence.

Big data, small data, and grading. The first two tasks are worth three points, the last four points. Since it can be challenging to run everything on a large data set, we'll offer you a *small data* option. If you *train* your data on `TestingDataOneLinePerDoc.txt`, and then *test* your data on `SmallTrainingDataOneLinePerDoc.txt`, we'll take off 0.5 points on Task 2 and 0.5 points on Task 3. This means you can still get an A, and you don't have to deal with the big data set. For the possibility of getting full credit, you can *train* your data on the quite large `TrainingDataOneLinePerDoc.txt` data set, and then *test* your data on `TestingDataOneLinePerDoc.txt`.

4.1 Machines to Use

If you decide to try for full credit on the big data set you should run your Spark jobs three to five c3.2xlarge machines as workers. If you are not trying for the full credit, you can likely get away with running on a smaller cluster. Remember, the costs **WILL ADD UP QUICKLY IF YOU FORGET TO SHUT OFF YOUR MACHINES**. Be very careful, and shut down your cluster as soon as you are done working. You can always create a new one easily when you begin your work again.

4.2 Turnin

Create a single document that has results for all three tasks. Make sure to be very clear whether you tried the big data or small data option. Turn in this document as well as all of your code. Please zip up all of your code and your document (use .gz or .zip only, please!), or else attach each piece of code as well as your document to your submission individually. Do NOT turn in anything other than your Python code and the document that you create.