Data Science Tools and Models: Logistic Regression on Spark

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, using RDDs. 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 150,000 text documents and a test/evaluation data set that consists of around 16,700 text documents. These documents are descriptions of clinical studies. At the highest level, your task is to build a classifier that can automatically figure out whether a text document is a study specifically about obesity. Some documents may mention obesity as a factor, but are not actually studies of obesity.

We have prepared three data sets for your use.

1. The *Training Data Set* (95 Mb of text). This is the set you will use to train your logistic regression model:

   `s3://[DataLocation]/a5KDtrainingV2.txt`

2. The *Testing Data Set* (10 MB of text). This is the set you will use to evaluate your model:

   `s3://[DataLocation]/a5KDtestingV2.txt`

3. The *Small Data Set* (6 MB of text). This is for you to use for training and testing of your model on a smaller data set:

   `s3://[DataLocation]/a5KDsmallsetV2.txt`

Some Data Details to Be Aware Of. You should download and look at the `a5KDsmallset.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 studies that are obesity begin with an "1" `<doc id = "1NCT...">`; that is, the doc id for an obesity study always starts with `1`. You will be trying to figure out if the document is an obesity study 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: Building a Dictionary

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 below. 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.

`["applicant","and","attack","protein","car"]`

Note that accomplishing this will require you to use a variant of your solution from your Spark-kNN assignment. If you do not trust your solution and would like ours, you can post a private request on Piazza.

The top words dictionary MUST be an RDD.

3.2 Task 2: Implementing Logistic Regression

Here you will implement a logistic regression model to classify whether a document is an obesity study. Your feature representation will be TF-IDF vectors (as usual), and you will derive and implement gradient descent for your model. 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.

Do not just look up the gradient descent algorithm on the Internet and implement it. Start with the LLH function from class, and then derive the gradient update formula for gradient descent. We can help with this if you get stuck.

To be clear, you are trying to optimize a function $f(\theta) : \mathbb{R}^d \mapsto \mathbb{R}$ with respect to a parameter vector $\theta \in \mathbb{R}^d$, where d is the number of words in your TF-IDF vectorization (20,000). This function is the log-likelihood of our model, logistic regression. In class we discussed log-likelihoods for Generalized Linear Models; recall that Logistic Regression simply assumes Bernoulli data (p. 15 of GLM Lecture), resulting in the following LLH:

$$\text{LLH}(\theta) = \sum_i y^{(i)}(x^{(i)}\theta) - \log\left(1 + \exp\left\{x^{(i)}\theta\right\}\right) \tag{1}$$

We'd also like you to implement ℓ_2 regularization, penalizing your objective function by the ℓ_2 norm of your parameter vector scaled by a hyper-parameter λ:

$$\text{LLH}(\theta) = \sum_i y^{(i)}(x^{(i)}\theta) - \log\left(1 + \exp\left\{x^{(i)}\theta\right\}\right) - \lambda\,||\theta||_2 \tag{2}$$

It is recommended you get your model working without regularization first.

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 a study on obesity.

3.3 Task 3: Evaluate your Model

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 obesity studies. 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.

Additionally, look at the text for three of the false positives that your model produced (that is, articles that your model thought were obesity studies but were not). Write a paragraph describing why you think it is that your model was fooled. Were the bad documents about obesity? Other similar conditions?

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

3.4 Improving your Model

If you want, you can try a number of things to improve your model's prediction.
You may

- Change the number of top words to use
- Over or undersample data in the TRAINING dataset, using custom rules (not cherry picked, be sure to tell us what rules you used)
- Move the cut-off between positive and negative cases
- Other items mentioned elsewhere in this document
- Experiment with the regularization constant
- Experiment with the gradient descent learning rate

For the purposed of this assignment, please do NOT

- Add additional features beyond the TF-IDF vectors

If you change the number of top words, you must still complete Task 1 as specified – providing the counts for the list of words using the top 20,000 words.

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. Note that you will need to compute the mean and standard deviation from the **training data only**, as that is known apriori. You will then transform the **training and the test data** using the computed mean and standard deviation.

2. When classifying new data, a data point whose dot product with the set of regression coefficients is positive is a "yes", a negative is a "no" (see slides 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.

   1. You might want to explicitly partition the data to fully make use of your powerful machines. Check relevant options for function `textFile`.

   2. Think about caching RDD(s). Although it is simply a function call of `.cache()`, you need to think about which RDD(s) to cache. It obviously does not make sense to cache everything.

   One more 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 30 points each, the last is worth 40 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 `a5KDtestingV2.txt`, and then *test* your data on `a5KDsmallsubsetV2`, we'll take off 5 points on Task 2 and 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 large `a5KDtrainingV2.txt` data set, and then *test* your data on `a5KDtestingV2.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 Turn in

Create a single **pdf** 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, i.e. please do not upload any data.