CS 61C HW 5: Predictive Ratings in Spark (a.k.a **DiStRiBuTeD CoMpuTiNg** on **BiG DaTa** for **MaChInE LeArNiNg**)

TAs: Sean Farhat & Nick Riasonovsky

Due: October 14th, 11:59 PM

Disclaimer: Please read the entire spec thoroughly. If your questions are answered in the spec, we will simply redirect you back here. All updates will be posted on the Piazza thread, so make sure to check back there every once in a while. Thank you.

1 Introduction

In this assignment, you will use the MapReduce programming paradigm to parallelize a simple Naive Bayes classifier with a Bag of Words model in Spark to predict Yelp review ratings.

We will be going over MapReduce + Spark in discussion and lab this week, but they have excellent documentation if you like a refresher (`https://spark.apache.org/docs/2.3.2/rdd-programming-guide.html`). Here, we will outline the important information relevant to this assignment (copied directly from the discussion worksheet):

Resilient Distributed Datasets (RDD) are the primary abstraction of a distributed collection of items. You can think of them as special (key, value) pairs.

Transforms $RDD \rightarrow RDD$

- `map`(f) Return a new dataset formed by calling f on each source element.
- `flatMap`(f) Similar to map, but each input item can be mapped to 0 or more output items (so f should return a sequence rather than a single item).
- `reduceByKey`(f) When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function f, which must be of type $(V, V) \rightarrow V$.
- `aggregateByKey`($zeroValue, seqOp, combOp$) When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value.

Actions $RDD \rightarrow Value$

- `reduce`(f) Aggregate the elements of the dataset *regardless of keys* using a function f.

2 Background

In this homework, you will be developing a program that will be able to estimate the Yelp rating (1, 3, or 5 stars) of a review. This concept, in a more general sense, is a popular open problem in the field of natural language processing, but for this class, the only thing you need to worry about is that we need a huge dataset in order to make our classifier as accurate as possible. It turns out that the mathematics done for each review to train our system is redundant, so we will exploit that for efficiency as we have every other concept in the course! Specifically, we will be employing task-level parallelism using the MapReduce framework we learned in class. One of the most popular libraries that supports this is Apache Spark (developed here in the AMP Lab!) Also, welcome back to Python.

2.1 Naive Bayes

There are many machine learning models/techniques for text classification. Perhaps the simplest (and most surprisingly effective) is a Naive Bayes classification with a Bag of Words model for text.

This can be broken up into three steps:

1. (Training) First, we will calculate the probability of a n star review occurring. This is known as a **prior**:

$$P(n \text{ star review}) = \frac{\text{num of reviews with } n \text{ stars}}{\text{num of total reviews}}$$

   We compute this for all possible values of n.

2. (Training) Then, we need to calculate the probability of a word occurring in reviews of each possible number of stars. This is known as the **likelihood**. Another way of putting this is: I have an n star review, what's the probability of this word appearing?

$$P(\text{word} \mid n \text{ stars}) = \frac{(1 + \text{num of times word appears in } n \text{ star reviews})}{(1 + \text{num of total words in } n \text{ star reviews})}$$

   We compute this for every word that occurs over all n star reviews, for all possible values of n.

3. (Classification) Now that we've trained on our dataset, we can then use our model and start predicting ratings! Given a review (word1, word2, word3,...), for all possible numbers of stars, we will calculate the **joint probability**:

$$P(n \text{ star review, word1, word2} \dots) = P(n \text{ star review}) * P(\text{word1} \mid n \text{ star review}) * P(\text{word2} \mid n \text{ star review}) \dots$$

   Our prediction for the number of stars for the review is then whichever value of n yields the highest joint probability. Pretty cool, right?

2.2 Bag of Words

When thinking about the relationship of words in a sentence to their sentiment or meaning, the sequence of the words seems like a likely factor. However, the Bag of Words text model ignores the ordering of words, and instead considers each word independently. A word in a document (or a review in our case) is represented only by the number of times it appears in the document and nothing else (none of ordering, the word's part of speech, or common phrases is considered).

For a more thorough explanation of these concepts, **including an example**, please look at the Appendix.

3 Your Task

Your task will be to fill in some of the functionality of the Naive Bayes classifier. All of the code for the Naive Bayes classifier is in **classifier/yelpClassifier.py**. The parts for you to fill in are clearly marked. Take some time to understand the main driver functions train and classify and all of the comments in the file.

You are welcome to come up with your own framework for the classifier if you choose. However, we will only be accepting code in classifier/yelpClassifier.py. If your code has any other dependencies and/or does not work together with run-classifier.py, it will not work and you will lose points. The driver Python file is **run-classifier.py**. Feel free to modify this file to debug, but remember that none of your changes to this file will be used when grading.

4 Getting Started

4.1 Logistics

First, this is a partner assignment! You do not have to work with your lab partner, nor do you need to work in a group. If you choose to do so, you only need one submission for the both of you.

Second, there are many depencency requirements to get Spark working. Due to this, you're only guaranteed to get correct functionality when working on the Hive. **However**, training on large datasets (1+ million Yelp reviews) takes a lot of computing power. There are many students in this course and only 32 hives. Therefore, we have included steps on how to set up everything locally. Please consider this option so that we do not overhwhelm the computers that the entire CS department uses. If you are patient, you can do all your testing on your computer, though it may take longer for the large dataset.

4.2 Setup

You should already pulled from the homework starter code repository from Homework 2. It's called **fa18-hw-starter**. All you need to do is pull from here again to get the starter files.

```
$ cd <your homework repository>
$ git pull hw-starter master
```

For this project, we need to be using Python 2.7. Some computers, including the Hive, use Python 3 as their default Python version, so we need to create a new **environment** for our code to run in. To do this run:

```
$ conda create --name 61c-hw5 python=2.7
```

Whenever you are working on the assignment, regardless if you're local or on the Hive, you need to be in this environment. To enter it, run:

```
$ source activate 61c-hw5
```

When you're done and wish to switch back to your computer's default version, run:

```
$ source deactivate 61c-hw5
```

You must be using the appropriate version of Python for your results to match ours, so don't forget to do this every time you start working.

4.2.1 Working locally

Ignore everything here unless you are working locally.

To get the datasets locally, go to `https://www.yelp.com/dataset/download`, enter your information, and download the JSON file. As you can see, the uncompressed file is around 7 GB. In order for the setup script to run correctly, **unzip the file in the same directory as create_datasets.py. If you don't, it will not work properly.** Then you can run:

```
$ python create_datasets.py (sample|small|medium|large)
```

to create whichever dataset you would like. (Note that the large dataset will take 5 minutes to complete.) You should use the sample dataset as a sanity check to make sure everything is working correctly, then small and medium to optimize your code until you are reaching around the staff accuracies, **then** try the large dataset. **Do not test on the large dataset until you are confident your code is mostly correct.** Chances are, your computer will start working pretty hard, so go take a break or something and let it do it's job.

You will also need Spark installed on your local computer; the Hive already has it. If you have pip, this is as simple as doing:

```
$ pip install pyspark
```

READ: Do not push the datasets to the Hive or Github! It will eat up most if not all of your allowed storage.

5 Testing

We are releasing the output of the staff train and classify functions to a sample dataset. This sample dataset contains 80 reviews for training and 20 reviews for classification. Feel free to use this to help you debug the output of each part of your implementation. When run on the sample dataset, **run-classifier.py** will automatically generate output in **your_debug_output.txt**, compare it to the reference staff output, and print out any diffs in **debug_diffs.txt**. Do realize that because it is such a small set of data, do not worry about accuracy. You can run this sample using:

```
$ spark-submit run-classifier.py -d sample
```

Once you have implemented all of the missing parts, we have three sample datasets for you to test out. You can run your Spark code on each dataset by using the command:

```
$ spark-submit run-classifier.py -d (small|medium|large)
```

If you have chosen to test **locally** and followed the steps above to load in the datasets properly, you should run:

```
$ spark-submit run-classifier.py -t local -d (sample|small|medium|large)
```

If you don't know why you are erroring, type:

```
$ spark-submit run-classifier.py -h
```

to ensure that you are passing in the correct arguments.

The dataset breakdown is as follows:

Dataset	Reviews for Training	Reviews for Testing	Staff Accuracy	Staff Timing
small	1280	320	63.8%	several seconds
medium	32,000	8,000	70.51%	1 minute
large	1,920,000	480,000	70.62%	20 minutes

Lastly, Spark 2.0 only works with Java 8 and older. What does this mean for you? **If you have a newer version (Java 9+) on your laptop, you cannot test locally. In addition, the 2nd floor labs have Java 10, so you can't test there either. The Hives, however, have Java 8. To check which version of Java you have, run:**

```
java -version
```

6 Grading

Grading will be simple for this project. The autograder will run your implementation on a set of Yelp reviews not released. If it matches (or exceeds) the staff accuracy and does not take significantly longer to run, you will get full points. If you match the staff benchmark for the three datasets, you should be confident that your code will also match the staff on the unreleased dataset. Partial credit will be given out as follows:

20% - code compiled/did not time out
20% - non-zero accuracy

20% - correct calculate_num_reviews_and_words_per_num_stars()
20% - correct calculate_likelihoods()
20% - correct classify_reviews()

For the last three items, we will test these functions in isolation to award partial credit.

Reminder:

We will only be accepting classifier/yelpClassifier.py. If you make any changes to any other file, including run-classifier.py, it will not be included in grading.

7 Submitting

Congratulations! You just used Spark to provide some insight into a huge Yelp dataset. Yelp even puts out a challenge to any person interested in using tools such as Spark to analyze their data. Feel free to check it out here: `https://www.yelp.com/dataset/challenge`

All submissions will be through Gradescope. You should only modify classifier/yelpClassifier.py. Anything else will be overwritten. If you worked with a partner, remember to include them in your submission.

A Appendix

A.1 Bag of Words

For example, if we had a review "This restaurant is amazing! The best. The food is never bad.", then it would be represented as:

WORD	COUNT
this	1
restaurant	1
is	2
amazing	1
the	2
best	1
food	1
never	1
bad	1

You might think that this representation of text is fairly naive ("never bad" is a lot different than "bad" for instance), but it works surpisingly well.

A.2 Naive Bayes

Next, we'll understand the machine learning classifier we'll be using for this task. Given some data X, Naive Bayes attempts to predict the probability $P(Y|X)$ that the data has a label Y, otherwise known as the posterior probability. (In our case, given the text of a review, we are trying to predict the number of stars that review gave.) In order to calculate this posterior probability, Bayes Rule is applied:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

. However, in practice, although the posterior probability is desired, it is actually proportional to the joint probability $P(Y, X)$, which is more easily calculated; thus, Naive Bayes ultimately attempts to estimate:

$$P(Y, X) = P(X|Y)P(Y)$$

With this goal of estimating $P(Y, X)$, Naive Bayes then tries to estimate $P(X|Y)$ and $P(Y)$ given some set of training data and labels. In our case, our Naive Bayes classifier will be given a training set, the words in the review (data) and the star rating of the reviews that the words appear in (label), and then estimate

$$P(\text{word}|\text{star rating of the review it appears in})$$

and

$$P(\text{star rating})$$

(e.g. P("awesome" — it appeared in a 5 star review) and P(5 star review).
To train our Naive Bayes Classifier, we will estimate $P(X|Y)$, otherwise known as the likelihood, as

$$P(\text{word} \mid n \text{ stars}) = \frac{\text{num of times word appears in } n \text{ star reviews}}{\text{num of total words in } n \text{ star reviews}}$$

To estimate $P(Y)$, otherwise known as the prior, we will estimate it as

$$P(n \text{ star review}) = \frac{\text{num of reviews with } n \text{ stars}}{\text{num of total reviews}}$$

For example, say we only had four reviews:

("I hate the food.", 1 star),
("The food is good.", 3 stars),
("Service is good.", 3 stars), and
("I love the good food.", 5 stars).

Since there are two reviews with three stars, and four reviews total, the prior probability of a review being three stars is $P(3 \text{ stars}) = 2/4$. The same calculations would then be done for one star and five stars, such that we have a table that maps a star rating to its prior probability. For estimating likelihoods, we would estimate the likelihood of "good", given that we know it appeared in a 3 stars review, as

$$P(\text{"good"}|3 \text{ stars}) = \frac{2 \text{ appearances of "good" in three star reviews}}{7 \text{ words total over all three star reviews}} = \frac{2}{7}$$

This calculation would be repeated for every other word that appears at least once in a three stars review, and similarly for the words in one star and five stars reviews. In the end, we would then have a likelihood table for each possible number of stars, where each table maps a word to its likelihood given that table's number of stars. The full prior and likelihood tables (concatenated together for brevity) are shown below:

Priors:

P(1 STAR)	P(3 STARS)	P(5 STARS)
1/4	2/4	1/4

Likelihoods:

WORD	P(WORD | STAR)	P(WORD | 3 STARS)	P(WORD | 5 STARS)
I	1/4	0	1/5
hate	1/4	0	0
the	1/4	1/7	1/5
food	1/4	1/7	1/5
is	0	2/7	0
good	0	2/7	1/5
service	0	1/7	0
love	0	0	1/5

Classification:

Now, for classification. If a Naive Bayes Classifier classifies an unlabeled datum, X, then for all possible labels, $Y = y_1, y_2, y_3...$, the joint probabilities $P(Y = y_1, X), P(Y = y_2, X), P(Y = y_3, X)...$ are all calculated, and then the datum is classified as the label corresponding to the greatest joint probability. Specifically, the joint probability is calculated as

$$P(Y = y, X) = P(Y = y) * P(x_1|Y = y) * P(x_2|Y = y)P(x_3|Y = y)...$$

where Naive Bayes makes the independence assumption that the probabilities of each x_i are independent given the label y. In our case, our labels are 1, 3, and 5 stars, and each datum is a single review, with each word in the review corresponding to an x_i.

More concretely, suppose that we have the same four reviews as earlier, and would like to now predict the number of stars corresponding to the review, "Good food!". For each possible number of stars, we would calculate the probability of that number, given this review, as

$$P(\text{num\_stars, "good", "food"}) = P(\text{num\_stars}) * P(\text{"good"} \mid \text{num\_stars}) * P(\text{"food"} \mid \text{num\_stars})$$

For each of our star ratings:

P(1 star, "good", "food") = (1/4) * (0) * (1/4) = 0
P(3 stars, "good", "food") = (2/4) * (2/7) * (1/7) = 1/49
P(5 stars, "good", "food") = (1/4) * (1/5) * (1/5) = 1/100

Since the joint probability of the review being "Good food!" and being 3 stars is the greatest, Naive Bayes would classify this review as giving 3 stars.

Laplace Smoothing:

Lastly, for our task, we will handle a common problem of using Naive Bayes classification. Suppose we were to classify, "The price is good". To calculate the probability that this review is 3 stars, we would calculate:

P(3 stars, "the", "price", "is", "good") = P(3 stars) * P("the" | 3 stars) * P("price"|3 stars) * P("is"|3 stars) * P("good"|3 stars) = (2/4) * (1/7) * (0) * (2/7) * (2/7) = 0

Since one word, "price", was never found in a 3 star review, the joint probability for this review and a 3 star rating was calculated as zero–despite this review having almost all words that also occur in 3 star reviews. To fix this issue, our Naive Bayes classifier will use Laplace Smoothing, a fancy sounding term for calculating the likelihood as:

$$P(\text{word} \mid n \text{ stars}) = \frac{1 + \text{num of times word appears in } n \text{ star reviews}}{1 + \text{numof total words in } n \text{ star reviews}}$$

This way, words like "price" that are never found in the training set of reviews will be assigned a very small nonzero likelihood instead of zero.

Additionally, multiplying many floating point numbers runs into precision problems (do you remember why?). To handle this, our implementation of classification will take the log of our likelihoods and priors, and add them together (instead of multiplying the likelihoods and priors themselves).