CSE545 Sp20 - Assignment 3 Description

Updated automatically every 5 minutes

Big Data Analytics
Stony Brook University
CSE545 - Spring 2020

Assignment 3.
Assigned: 4/14/2020; Due: 5/1/2020 11:59pm

Overview

Goals.

- Gain experience with a live hadoop-style (hdfs, spark) cluster.
- Implement hypothesis testing with multi-test correction at scale.
- Implement a basic collaborative filtering recommendation system.
- Gain experience navigating a cloud console to spin up a cluster.
- Work with moderately large data.

General Requirements. You must use Python version 3.6 or later, Spark 2.4.4 or later. You will use a cluster for this assignment that comes with Spark already, but you may start development using Spark on your own or non-cluster machines. Everything between input and output must occur within Spark RDDs.

Python Libraries. The only data science, machine learning, or statistics libraries that you may import are those that are listed in this assignment. Of these libraries, you may not use any subcomponents that specifically implement a concept which the instructions indicate you should implement (e.g. hypothesis testing, linear regression, collaborative filtering). Other Python default, non-data science libraries (e.g. sys, basic IO, re, random,csv) may be used -- ask if unsure. All provided method names and classes must be used as provided with the same parameters. However, you may also use additional methods to keep your code clean. The intention is for you to implement the algorithms we have gone over and problem solve in order to best understand the concepts of this course and their practical application. Current allowed data science-related libraries include:

```
numpy as np //for matrices and matrix algebra; not ok
for calling linear regression
```

CSE545 Sp20 - Assignment 3 Description

Updated automatically every 5 minutes

1. a3_cluster_screenshot_[lastname]_[id].png -- console screenshot of your running cluster

2. a3_p1_[lastname]_[id].py -- your python Spark code for part 1.

3. a3_p2_[lastname]_[id].py -- your python Spark code for part 2.

(do not include the brackets [] in your file name -- those are placeholders for your name and id number)

Academic Integrity. Copying chunks of code or problem solving answers from other students, online or other resources is prohibited. You are responsible for both (1) not copying others work, and (2) making sure your work is not accessible to others (now or anytime in the future; exceptions may be made for private sharing with potential employers if instructor is contacted). Assignments will be extensively checked for copying of others' work. Problem solving solutions are expected to be original using concepts discussed in the book, class, or supplemental materials but not using any direct code or answers. Please see the syllabus for additional policies.

Hadoop-Style Cluster (10 points)

Initially, you will have access to a modest class cluster if you wish to test code in such an environment. In time, you will receive information to spin up your own cluster.

To access the class cluster:

- Submit your *public* ssh rsa key to this form.

(May take up to 36 hours to enable access)

- Once a TA acknowledges that you have been added to the server try to ssh in
  a. User name: (TA will provide)
  b. Address: **<ADDRESS>**
     - port: 22
       (use your *private* (id_rsa or *.ppk) key on your end)
  c. Set spark environment variable to python3 (using nano, vi, or emacs)
     - add "export PYSPARK_PYTHON=python3" to .bashrc
     - run "source .bashrc"
  d. Test that spark shell works for you:

```
$ pyspark
Python 3.6.10 ….
...
>>> rdd =
sc.textFile('hdfs:/data/Software_5.json.gz')
>>> rdd.take(2)
```

CSE545 Sp20 - Assignment 3 Description

Updated automatically every 5 minutes

- Spin up a cluster and take screen shot:
  Follow this tutorial: https://www.youtube.com/watch?v=6DD-vBdJJxk .

For now, use the following configuration (You can use any "east" region and zone as long as the zone matches the region):

Name	[you decide]
Region	us-east1
Zone	us-east1-c
Autoscaling	Off
Scheduled deletion	Off
Enhanced flexibility mode	Off
Master node	**Standard (1 master, N workers)**
Machine type	**e2-standard-2**
Number of GPUs	0
Primary disk type	pd-standard
Primary disk size	**64GB**
Worker nodes	**2**
Machine type	**e2-highmem-4**
Number of GPUs	0
Primary disk type	pd-standard
Primary disk size	**32GB**
Local SSDs	0
Preemptible worker nodes	0
Cloud Storage staging bucket	---
Subnetwork	default
Network tags	None
Internal IP only	No
Image version	**1.4.26-debian9**

Image Version: 1.4 (Debian 9, Hadoop 2.9, Spark 2.4)

(configuring a jupyter notebook is optional; note: the tutorial is 1.5 years old; some things look slightly different.)

It may be useful to get Google SDK for your local machine: https://cloud.google.com/sdk/docs/

Alternative setups that have the same total number of VPUs (8) and total memory (64GB) are fine.

[Take a screenshot of console.cloud.google.com/dataproc/clusters to show your cluster "running".]

- Test the cluster.

Set pyspark to use python3:

add "`export PYSPARK_PYTHON=python3`" to .bashrc
(use "`nano .bashrc`" or install your preferred editor)

run "`source .bashrc`"

Launch pyspark: "`pyspark`" and try a few things:

CSE545 Sp20 - Assignment 3 Description

Updated automatically every 5 minutes

```
sudo apt-get -y install python3 python-dev build-
essential python3-pip
sudo easy_install3 -U pip

sudo pip3 install --upgrade google-cloud
sudo  pip3 install --upgrade google-api-python-client
sudo pip3 install --upgrade pytz

sudo echo "export PYSPARK_PYTHON=python3" | sudo tee -a
/etc/profile.d/spark_config.sh /etc/*bashrc
/usr/lib/spark/conf/spark-env.sh
sudo echo "export PYTHONHASHSEED=0" | sudo tee -a
/etc/profile.d/spark_config.sh /etc/*bashrc
/usr/lib/spark/conf/spark-env.sh
nano /etc/spark/conf/spark-defaults.conf # add to
bottom: "spark.executorEnv.PYTHONHASHSEED=0"

sudo pip3 install numpy

#On each worker VM instance (go to
console.cloud.google.com/compute/instances):

sudo apt-get -y install python3 python-dev build-
essential python3-pip
sudo easy_install3 -U pip
sudo pip3 install numpy
```

(This is an updated version of: https://stackoverflow.com/questions/45843960/how-to-run-python3-on-googles-dataproc-pyspark)

You may want to change other settings in `spark-defaults.conf`, such as:

```
spark.executor.instances            3
spark.executor.cores                8
spark.driver.memory                 10000M
spark.executor.memory               9000M
spark.default.parallelism           8
```

- Tell hdfs to replicate a data file 4 times (increases read throughput):
  hadoop fs -setrep -w 4 -R /data/large_sample/Books_5.json.gz
- Shutdown the cluster
  The cluster costs you $TBD/hr when it's up. If you're not using it stop the VM instances (Console> Compute Engine > VM Instances > Select the Instances > "Stop") to save credit. Go to https://console.cloud.google.com/dataproc/cluster and delete the cluster to make sure you do not use up credits on it.

Data

Software_5.json.gz -- small data -- software reviews
Available in class cluster under
hdfs:/data/Software_5.json.gz
N = 12,805 Reviews

Books_5.json.gz -- large data -- book reviews
(warning may take up to an hour to download).
Available in class cluster under
hdfs:/data/Books_5.json.gz
N = 27,164,983 Reviews

The format of the file is JSON, and the following are the fields that will be relevant for this assignment (all others may be filtered out during your first steps):

```
{
  "overall": #rating score from 1 to 5,
  "reviewerID": #string id of the reviewer (e.g. A2SUAM1J3GNN3B),
  "asin": #long integer id of the product (e.g. e.g. 0000013714),
  "reviewText": #string of the review,
  "summary": #summary of the text,
  "verified": #true or false: whether the purchase was verified (assume false if not present)
}
```

Original data is from (Ni, 2018).

Part I: Word Count and Hypothesis Testing (50 Points)

Here, you will attempt to find significant associations between words and ratings by using multi-test corrected hypothesis testing.

Filename: `a3_p1_<lastname>_<id>.py`
Input: Your code should take **one** command line parameter for the review dataset location.
Example: `spark-submit a3_P1_LAST_ID.py 'hdfs:/data/Software_5.json.gz'`

Task Requirements: Your objective is to compute the correlation between each of the 1,000 most common words (case insensitive) across all reviews with the rating score for the reviews, controlling for whether the review was verified or not.

First you must figure out which of all the possible words are the most common. You should consider anything matched by the following regular expression as a word:

```
r'((?:[\.,!?;"])|(?:(?:\#|\@)?[A-Za-z0-9_\-]+(?:\'[a-z]{1,3})?))'
```

Then, you must figure out how common each of the 1k words

observation, and each of the 1,000 words is essentially a hypothesis. Thus, you will have over 1k linear regressions to run representing 1,000 hypotheses to test. Further, you will need to run the tests without and using "verified" as a control (simply including it as an additional covariate in your linear regression as either 0 or 1). You must use Spark such that each of these correlations (i.e. standardized linear regression) can be run in parallel -- organize the data such that each record contains all data needed for a single word (i.e. all relative frequencies as well as corresponding ratings and verified indicators for each review), and then use a map to compute the correlation values for each. *You don't have to worry about duplicate reviews for this one. Assume each review is a separate review.*

You must choose how to handle the outcome and control data effectively. You must implement standardized multiple linear regression yourself -- it is just a line or two of matrix operations (using Numpy is fine). Finally, you must compute p values for each of the top 20 most positively and negatively correlated words and apply the Bonferroni multi-test correction.All together, your code should run in less than 8 minutes on the provided data. Your solution should be scalable, such that one simply needs to add more nodes to the cluster to handle 10x or 100x the data size.

Other than the above, you are free to design what you feel is the most efficient and effective solution. Based on feedback, the instructor may add or modify restrictions (in minor ways) up to 3 days before the submission. You are free to use broadcast or aggregator variables in ways that make sense and fit in memory -- typically 1 row or 1 column by itself will fit in memory but not an entire matrix (at least for the larger dataset).

Output: Your code should output four lists of results. For each word, output the triple: (“word”, beta_value, multi-test corrected (for 1000 hypothesis) p-value)

1) The top 20 word *positively* correlated with rating
2) The top 20 word *negatively* correlated with rating
3) The top 20 words *positively* related to rating, *controlling for verified*
4) The top 20 words *negatively* related to rating, *controlling for verified*

Note: a Bonferroni-correct p-value adjusts the p-value according to the Bonferroni correction. We adjusted the alpha in class. Here, you are adjusting the p-value, so instead of dividing by the number of hypotheses, you will multiply to p-value.

Remember to save your code and delete/terminate your cluster when you're not using it.*

Part II: Recommendation System (40 Points)

CSE545 Sp20 - Assignment 3 Description

Updated automatically every 5 minutes

Filename: `a3_p2_<lastname>_<id>.py`
`(do not include the brackets <> in your file name)`
Input: Your code should take **two** command line parameters: (1) for the review dataset location, and (2) a list of product asins in python list format:

Example:

```
spark-submit a3_p2_LAST_ID.py
‘hdfs:/data/Software_5.json.gz' "['B00EZPXYP4',
'B00CTTEKJW']"
```

Task Requirements: Your objective is to perform **item-item** collaborative filtering over the provided products and ratings. Specifically,

To prepare the system, you will first need to do some filtering:

a. Filter to only one rating per user per item by taking their most recent rating (or their last within the data file; as long as you have one rating per person it is fine)
b. From there, filter to items associated with at least 25 distinct users
c. From there, filter to users associated with at least 5 distinct items

Option: If you have a particular RDD that has less than an order of 1k entries (i.e. a list of reviewerIDs or asins), at that point, it's ok to collect them into a sc.Broadcast variable.

Then, you are ready to apply item-item collaborative filtering to predict missing values for the rows prescribed in the output. Use the following settings for your collaborative filtering:

a. Use 50 neighbors (or all possible neighbors if < 50 have values) with the weighted average approach (weighted by similarity) described in class and the book.
    a. Do not include neighbors:
        i. with negative or zero similarity or
        ii. those having less than 2 columns (i.e. users) with ratings for whom the target row (i.e. the intersection of users_with_ratings for the two is < 2; can check for this before checking similarity).
b. Within a target row, do not make predictions for columns (i.e. users) that do not have at least 2 neighbors with values
c. Only need to focus on the specified items (in practice, you wouldn't store a completed utility matrix, rather this represents querying the recommendation system, given an item, for users that might be interested in the item).

Remember to treat items as "rows" and users as "columns" where the goal is to rate one item based on its similarity to other items. Running from start (reading/filtering data) to finish (printing results should take less than 8 minutes on a cluster with >= 8 vCPUs with 8GB per vCPU and multiple disks for reading the data from HDFS (In reality, such a system could assume that the data was already filtered as that wouldn't need to happen per run but it is fine to happen per run here).

CSE545 Sp20 - Assignment 3 Description

Updated automatically every 5 minutes

Software: B00EZPXYP4 (Norton), B00CTTEKJW (Amazon Music)

Books(Tentative): 0008118922, 1469216051

Remember to save your code and delete/terminate your cluster when you're not using it.