Doug Needham
Data Science Challenge 3 Fall 2014
dougthedataguy@gmail.com

Introduction

This challenge has been a true learning experience for me. It has helped me to uncover some gaps in my knowledge base, as well as confirm other areas of expertise that I already knew I had. I look forward to continuing to challenge myself, as well as sharing the knowledge I have gained with others.

Environment

My environment was a 7 node cluster, with Cloudera manager as a separate node managing the cluster. This cluster was built with Cloudera Director using ami-3218595b (RHEL 6.4 X86_64) on c3.xlarge instances. This was a non-dedicated environment for use in the data science challenge. The individual machine I ran my code on required approximately 8Gig for local storage and processing. I was also the administrator of this environment. The solution to Problem 3 takes quite some time to run in this environment. Optimization of the code was done, however, a larger cluster with more resources should speed up the processing of Problem 3. This particular infrastructure led to the requirement of a few hacks for solving problem2, I will discuss further in that section. For each of the spark-submit commands I ran, I increased the executor-memory parameter to 2G for both Problem1 and Problem3.

Dependencies

Additional packages installed on one machine were: Scipy, tree, and sbt. I had to back-port a number of functions in order to use chi2_contingency from python 2.7 scipy to python 2.6.

The `sbt package` was used to create the jars for the scala code, all build scripts are included with the code in both `DougNeedhamDSC3/problem1/PredictFlights` and `DougNeedhamDSC3/problem3/AnalyzeGraph`

Tools used

Gephi, Python, Scala, Spark, Map-Reduce Streaming, R, awk, grep, uniq, bash, sbt, and the Data Science at the Command Line Toolkit.

General Analysis

Leveraging the Data Science at the Command Line toolkit from Jeroen Janssens, much of the initial analysis was done through simple scripts that allowed me to create a “pseudo map-reduce” job stream. These commands followed this general pattern:

```
cat filename | mapper.py | sort | reducer.py > output.dat
```

Based on the output of this pipeline, I could bring the data into R, Excel, Python, or notepad++ in order to view the results. This method allows me to “boil down” a large corpus of data into something more manageable for my own analysis or, when appropriate as demonstrated below, create a simple visualization for communication of my findings.

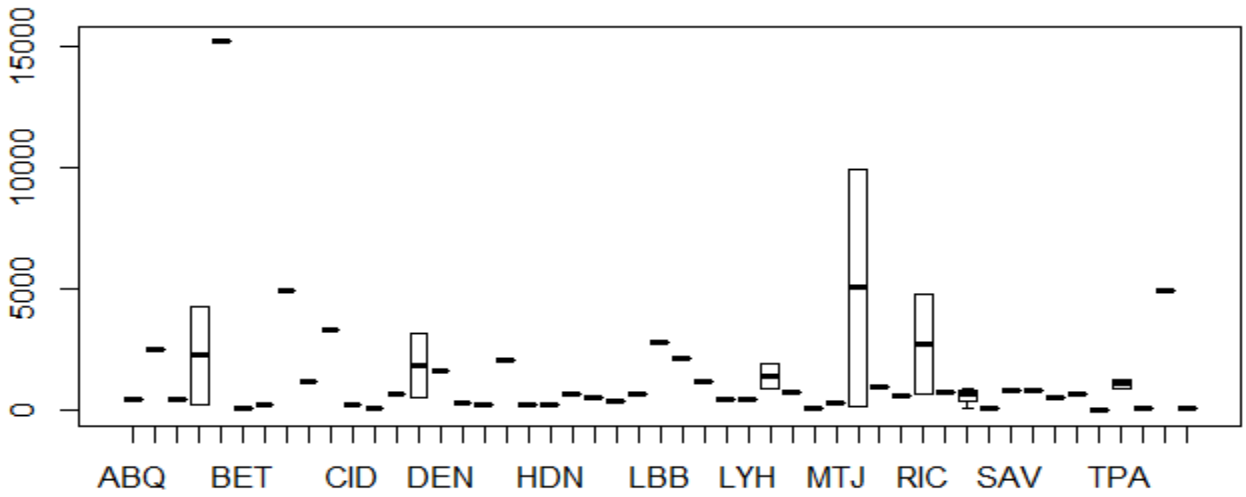
The total time spent per problem is a little difficult to break down. I take an iterative approach solving multiple problems, in order to give myself plenty of time for refactoring and making adjustments based on new information. As a general rule the total time spent per problem does not include the time spent on packaging, and streamlining the shell scripts.

Doug Needham
Data Science Challenge 3 Fall 2014
dougthedataguy@gmail.com

Problem1

Analysis

General analysis was done against all 20 fields looking for visualizations that would rationalize an approach to solve the problem. The visualization that ultimately drove me towards the solution I came up with is the following plot:



This plot was created in R, the shell script and R code for this is under the directory `problem1/analysis`.

With this visualization it appears that having a single predictive model for all points of origin throughout the data set, may not be the correct approach. Unfortunately, this particular analysis was done after attempting to create one single model to predict all delays for all airports. I therefore created a number of python and shell scripts to break the data apart into separate directories in HDFS, and format the data in libSVM format to use within Spark.

I ultimately chose to use SVMwithSGD as the function for this prediction. I performed a number of tests using Logistic Regression, and SVM's. I finally decided upon the SVM model because it consistently provided the highest accuracy and highest area under the ROC curve.

Any features not found in the scheduled file were immediately removed as an option for input to the Support Vector Machine. Year is not a feature for machine learning, since we are effectively training with one year, and predicting with the next year (with some overlap). Since the question at hand is most related to departure delays, the scheduled arrival time is not considered a feature.

Models were created using flight number, and tail number. No significant benefit to any model was found using these features. Therefore these features are not in the final model submitted for the challenge.

The departure delay metric was not used as a feature for the model, as this is considered a dependent variable in this case. Since we are creating separate models based on point of origin, the destination airport, as well as distance travelled, become distinctly influential. If we were doing a graph analysis of this particular set of data the distance could be an edge weight, and the origin and destination airports would be separate vertices.

Doug Needham
Data Science Challenge 3 Fall 2014
dougthedataguy@gmail.com

Assumptions

Any data not in the scheduled file, should not be considered as part of any machine learning. This problem is actually a classification problem, where a flight is classified as delayed or not.

Software and Algorithms

I chose to use Python as a data transformation language, and spark mllib and scala as the predictive software. Along with some awk, and bash control scripts in order to produce the output files as requested.

Time Spent

Overall the amount of time I spent on this problem, including initial coding, refactoring, code cleanup, packaging, and presentation was approximately 70 hours.

Solution

For problem1, I decided to separate out the data into files based on airport, then do an SVM model using the features of scheduled departure hour divided by 100 in order to lower the impact of time on the support vector machine. Month, Day of Month, Day of Week, and distance are all used as features unmodified. For the rest of the features, I converted the "raw" data into the index of dictionaries that are built in the earlier steps prior to the map-reduce job to create the individual airport based files.

The output from the model.predict method in Spark simply returns either a 0.0 or a 1.0 as the score for the prediction. Either 1.0 the flight is delayed, or 0.0 the flight is not delayed. In order to meet the requirement of sorting the data, and realizing the distribution of delays based on airport as evidenced in the plot above, I use the Area under the ROC score from testing the model as a multiplicative factor against the score from the output of model.predict.

In order to say for the airports that generate a model that has a higher degree of accuracy in the prediction those flights are more likely to have a delay since I have a higher degree of confidence in the model to do the prediction.

Problem2

Analysis

In analyzing the spam.log file, it appears that the behavior of spam bots is to land, then do an adclick within at the most 11 seconds. This is the driving decision for identification of bots in the web.log file that is processed for all subsequent calculations.

Assumptions

In the requirements for question 4, in order to use the G-test I wanted to leverage the chi2_contingency with a lambda of "log-likelihood". Since this was a shared cluster using RHEL 6.4 it came with Python 2.6. For a variety of reasons, this is the environment I had to use for the challenge, as such I could not upgrade this to Python 2.7 and use scipy 0.14 which includes the chi2_contingency function. I had to "back-port" the chi2_contingency code along with some subordinate code down to 2.6.

Software and Algorithms

Scipy, Python, portions of scipy.stats backported to Python 2.6 as mentioned in Assumptions. A z-test for sample size calculations should be used for both question 4 and 5.

Time Spent

Overall the amount of time I spent on this problem, including initial coding, refactoring, code cleanup, packaging, and presentation was approximately 40 hours.

Doug Needham
Data Science Challenge 3 Fall 2014
dougthedataguy@gmail.com

Solution

I created a dictionary of UID's that had the bot behavior identified in the Analysis above. This dictionary was used in all subsequent steps. For the number of adclicks, unique UID's and total number of visitors, I created individual files under the data directory that could be used in subsequent python code that performed the various calculations. By storing this data on local disk it allowed me to pass the numbers between the steps in a straightforward manner.

For question 1:

I captured all of the visit_id's for UID's that were in the bot dictionary previously created. This total is saved in the data directory as bot_uids.number. This will be what is reported as the answer for question1. I also capture the visit_id's that are not in the bot dictionary and save this number for use in downstream calculations.

For question 2:

I captured all of the actions that were adclicks, then divided that number by the non-bot visitor count. This is what is reported as the answer for question 2.

For question 3:

This is a multi-step process to answer. I captured distinct campaign data, then summarized it in HDFS. This data was then sent to a local process in order to put the data into a python dictionary. Once this data is stored in a dictionary, I am able to loop through the dictionary to get the highest mean value for a campaign, and query_string. The rest of the output of question3 is formatted into JSON in the format_std_dev.py code.

For question 4:

I captured the experiment for all signup actions, the reduced that down to a count for processing with local python code. Using the ported chi2_contingency function I was able to compare the experiments. I chose to set up the array for the chi2_contingecny test to have a single row per experiment column1 were the signup responses, column2 were the total landed responses minus the signup responses for the experiment.

For question 5:

I captured the experiment for actions: 'order','signup', and 'adclick' then reduced that down to a count for processing with local python code. The local python code loads everything to a dictionary, then loops through the dictionary to summarize the data and divide by the visitor count which comes from the nonbots_visits.number file created in step1.

I used the same approach to answer the number of days question for both question 4 and 5, I took the total responses for experiment 1 and 2 (question 4), and experiment 3 and 4(question 5) to use as a total population. My null hypothesis would be that the responses would be equivalent. In order to get the total days, I divided the population by the days of the experiment (15) in order to get an average daily response. This was the denominator in my equation to calculate the number of days. The numerator was the result of a standard finite population sample size equation.

Problem3

Analysis

The general graph for winklr – winklr_network.csv was subset and brought into gephi for analysis. My method of subsetting the graph was to take the first 1000 vertices, then grep through Winklr-network.csv to find all connections involving those particular vertices. These rows were added to a separate file that could be opened with Gephi. This

generated a graph of 72,575 nodes, and 122,301 edges. The metrics needed from this graph were the Average Path length(12.936), and the network diameter(45). While these numbers are specific to this particular subgraph I was able to use this information to make some assumptions about the overall graph as a whole.

By doing this in Gephi, I was able to get something of an idea of the structure of the graph.

Assumptions

1. In-degree is number of followers,
2. Out-degree is number followed.
3. The further away one vertex is from another the less likely it is that they personally know each other.(*Milgram small-world Experiment)
4. PageRank of the overall graph shows the centrality of each vertex in the Master Graph

Software and Algorithms

Scala, Spark, Graphx and Gephi were used in solving this problem. Gephi was used for analysis of sub graphs generated from the master graph. GraphX allowed the efficient use of Shortest Path, PageRank and Degree counts. This along with some small bash scripts and python code for formatting, looping, and breaking the big problem down to many small problems that could be brought together at the end of the processing.

Time Spent

Overall the amount of time I spent on this problem, including initial coding, refactoring, code cleanup, packaging, and presentation was approximately 80 hours.

Solution

The solution I created for this begins with the followers. I created multiple files, stored in the inGraph directory that have the direct connections of a follower to the vertices that they clicked yielded from the Winklr-topClickPairs.csv file. This breakdown is done with scripts, because raw text manipulation of many files is very straightforward with shell scripts. Using the vertexId of the follower, I am able to analyze the full graph, starting from the point of origin of a follower from the Winklr-topClickPairs.csv graph.

A “happy accident” that fell out of this design is giving the customer the ability to have custom recommendations per user. The requirement is to produce the top 70,000 pairs most likely for user1 to follow user2. We have to throw away the majority of the output data in order to meet this requirement. The method used to create this solution allows this data product to potentially be used on a per-user basis if that becomes a later requirement.

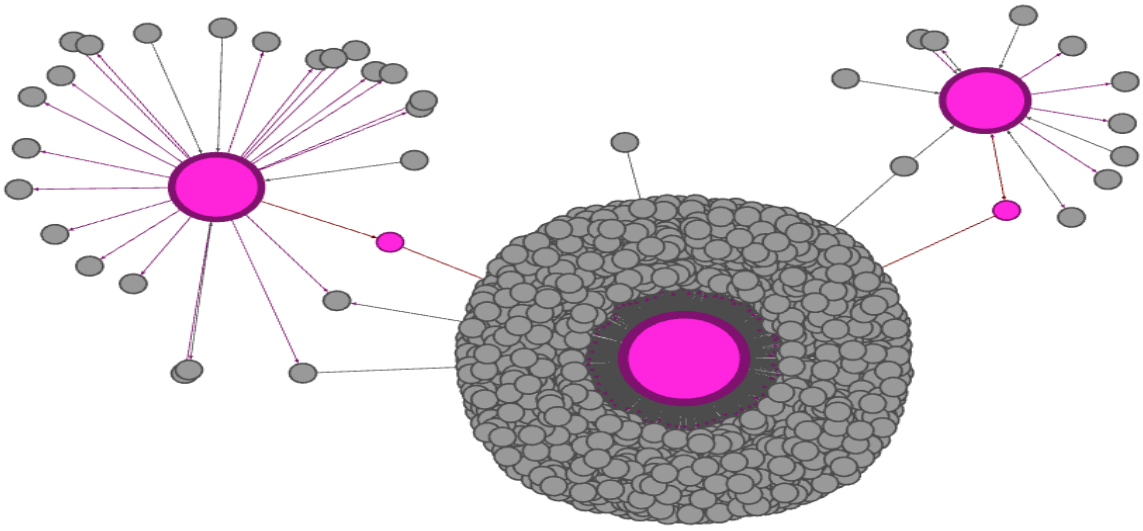
Once I have the shortest path from this point of origin, I mask this graph with the sub-graph that is made up of only the vertices in the click pair graph. Using assumption 1 and 3, I multiply those two numbers to get an “influence” number. The reason for incorporating the shortest path to the calculation is I am using this as a similarity score. It also takes into account the concept of funneling on a per user basis. A person is more likely to ultimately follow a person that one of their friends follows. The reason for inverting the path length is that the “further” away a friend is, the assumption is they are not as influential on this user as someone with a closer path length.

The “influence” graph is then combined with the PageRank of the vertices this particular follower has clicked on. The final step is to eliminate vertices with a path length of Infinity. An Infinite path length in this case simply means no one that this person knows either directly or indirectly has a connection to that particular vertex. Therefore, I want to eliminate that vertex from consideration.

Doug Needham
Data Science Challenge 3 Fall 2014
dougthedataguy@gmail.com

Collecting together the vertices, and the scores that are represent the formula (PR*(inDegree*1/PathLength)) we store that data back out to HDFS. The last few steps in the shell script that is the driver code copies everything out to the file system, formats, sorts, and populates problem3.csv

In order to determine what a network of recommended connections would look like if the user followed the recommendation, and to confirm that the scoring was reasonable. I took a sample in early testing of this methodology and used grep to extract the vertexes that were output from the recommender to create a csv file to load into Gephi. This data was grep'd from winklr-network.csv using the top recommended pairs as output from the pipeline. Once the data was loaded in Gephi, I used the Force Atlas 2 layout to allow the Graph itself to show the structure of the data. The layout was allowed to run until a structure emerged. Once there was some basic structure, rearranged a few of the vertices, then used Gephi to calculate the shortest path between the "From" and "To" vertices that came from the problem3.csv file. The shortest nodes I ran the shortest path between are the purple vertices highlighted below:



The results of this analysis in Gephi gave me a high degree of confidence that my algorithm was doing what was expected. This subgraph analysis could be used as part of a larger presentation to visualize the effects of these recommendations.