One Hot Encoded Taxi: Analyzing Big Data for New York City Taxis

Aakash Moghariya, Ruturaj Patel, Julie Sturgeon

December 9, 2016

1 Introduction

In 2013, the New York City Taxi and Limousine commission released data on taxi trips from yellow, green and for-hire vehicles from 2010 through 2013 through a freedom of information law request. The dataset consists of 1.5 billion records. Compared to the datasets that we had seen up to this point, this was truly big data! The dataset was downloaded from the Illinois Data Bank [1].

The first major obstacle that we faced with this dataset was its sheer size. The Illinois Data Bank allowed us to download all of the data as a zip file of around 28GB, which took about an hour with the super-fast campus wifi. However, this zip file consisted of several other layers of zipped directories and csv files, which quickly ballooned to over 140 GB when decompressed. While this amount of data would easily fit on an external hard drive, we needed the data in a location where it could be easily accessed and processed. We were also wary of working with csv files, preferring a format more conducive to processing (as opposed to just storing) a large amount of data. Our goal, ultimately, was to clean and summarize the massive dataset in order to visualize the final results and find useful patterns.

Initially, we planned on combining the taxi dataset with similar datasets, like Uber or public transportation data from New York City, but found that we had more than enough data to analyze with the taxi data alone.

2 Amazon Web Services

We decided to use Amazon Web Services (AWS) [7] to store and process our data. We chose this technology because it is very popular in industry and we could obtain free AWS credits through our studen Github accounts. Using AWS for both data storage and analysis helped mitigate the costly effects of transferring data for executing Spark programs.

Amazon has many cloud computing solutions, but we only used three of these: Simple Storage Service (S3) for data warehousing, Elastic Computing Cloud (EC2) for data loading and Elastic Map Reduce (EMR) for analysis. These services turned out to be perfect for our needs because of their scalable nature. EMR clusters are made up of EC2 instances, and they provide flexibility in choosing how much computing power goes into an EMR cluster. When we had tasks that seemed to be taking a long time on a less powerful cluster, we could speed up execution by requesting a cluster with more nodes. EMR has many frameworks that can be installed on a cluster, so we were able to generate a Spark cluster in minutes without the hassle of installing Spark ourselves.

While configuring an EMR instance was fairly straightforward, working with S3 posed more challenges. Our first thought was to download and unzip the original CSV files locally, then upload the data into S3 using its user interface. We quickly discovered, however, that this interface is not built for uploading large files. We were logged out before even 1% of a single file could be transferred. So we decided to throw programming and even more AWS at the problem, and sure enough, it worked! We were able to download and unzip the files using EC2 and upload them directly into our S3 bucket.

We also had difficulty in granting access to the S3 bucket to all group members. Amazon takes its data security seriously and there are several hoops to jump through before someone else can have access to your data. We had to create Identity and Access Management (IAM) roles for each group member and then grant each member the specific bucket permissions that they required. This involved manually formatting the code for the bucket policy (in JSON format) and uploading it under the permission section of the bucket.

One of the fundamental problems that comes with processing such a large dataset is to make scalable programs that can finish within a reasonable time. The dataset contained two different types of data files (one for trip information, the other for fare information), each stored as a CSV file with 10-12 columns for every month of data. Loading and joining all of the data in this inefficient format each time we wanted to perform some analysis was not a scalable solution. So we decided to join the dataset and convert it into parquet format. Parquet compresses the data leading to low storage requirement and network overhead when transferring the data. Using parquet files allowed us to select only the columns that we needed to do the analysis, removing the overhead of loading the entire dataset into memory. Most of our Spark scripts were able to finish in under 10 minutes. Some scripts required more resources than others in order to run efficiently, so we learned how to scale the EMR cluster after it was already instantiated. This allowed us to add and remove resources on the go without instantiating the cluster.

To convert the CSV dataset into Parquet, we used EMR to load the CSV files from our S3 bucket, convert them into parquet file, and upload the files back into S3. We created 30 parquet files for each month's worth of data for maximal parallelism in the analysis phase. This process was time-consuming but once the files were converted into Parquet format they were much easier to use. The size of the files decreased drastically after conversion, from over 2 GB per CSV file to less than half a GB for a month's worth of parquet files. Storing the data in a native Spark format allowed us to load the data very quickly into Spark without the need of Databrick's CSV extension. This conversion process took days, but after we had our files converted, we were able to complete analyses over the entire dataset in a reasonable period of time.

Disputed Trips	count per hour, count per month, total per year
Payment Type by Trip	total amount per month, total amount per year
Salary by Driver	average monthly salary, average yearly salary
Speed Analysis	average by month and hour, count of speeds of all trips (rounded)
Tips	average by hour, average by day of month, average by driver
Trip Counts	by week, hour, day of month, day of year, week of year
Trip Distance	by range (0-10, 10-20, 20-30, 30-50, 50-70, 70-100, 100+)

Even though we had some AWS credits, we tried to limit the amount we were spending on AWS services. We looked for spot instances to perform our analysis. With spot instances you can bid on unused CPU time. This option ends up being cheaper, but there are no guarantees of when you are going to get your computing power. We developed our Spark scripts locally, using only one month of data to verify that our scripts runs properly before performing the analysis on the entire dataset residing in S3 using an EMR cluster.

We also explored the AWS bootstrapping service to install software while the EC2 nodes or EMR cluster was being allocated. This method is preferable to manually installation since bootstrapping can install software on all nodes in an EMR cluster (as opposed to just the accessible head node). This came in handy when we were doing area analysis and required a Python library and all of its dependencies to be installed on every node of the cluster. Another way to install software on EMR is by moving the key-pair permission file into the master node. This ssh key pair can be used to log into each of the slave nodes and install software manually.

During this project we learned to manage, install and modify the EMR cluster which is an essential skill to have while working with advanced options on EMR.

3 Analyzing the Data

3.1 Summary Statistics

With such a large amount of data with a variety of features, there was a lot of opportunity for analysis. We generated summary statistics on many different aspects of the data, summarized below:

We performed our analysis with PySpark scripts and used the Databricks CSV extension to write the output into CSV files that could later be loaded into Tableau.

During this analysis, we discovered that this dataset had not been thoroughly cleaned. We found negative values for tips and fares, and in calculating the speed of trips, found that some taxis can apparently travel faster than the speed of light! Initially, we filtered the data as best we could with Spark but had to wait until the visualization stage to discover what further data filtering was necessary.

3.2 Adding Area Information: A Lesson in Failure

One of the exciting features of the taxi dataset was the fact that location was specified for each taxi trip, allowing us to perform geospatial analysis. The data came with latitude and longitude values for the trip pickup and dropoff, but these are continuous values and difficult to visualize, so we wanted to analyze neighborhoods within New York City. Unfortunately, these neighborhoods are not a simple shape and determining which set of coordinates corresponds to which neighborhood requires a bit more complex analysis. We obtained a geojson file[5] of New York City neighborhoods and used Python's Shapely library[6] to assign neighborhoods to pairs of coordinates. The goal was to augment our existing parquet files by adding columns for the neighborhood and borough name for each of the pickup and dropoff locations and saving back the data with the new columns for later analysis.

Ultimately, we were unable to create these new parquet files, even after running the code for 4 hours on a hefty EMR cluster. After some experimentation, we discovered that it wasn't the process of adding the extra columns that was causing the slowdown (we could get the data with the new columns to display after only a few minutes), but the process of writing the new data back to parquet files. After some investigation, we found blog posts from people who had a similar problem of parquet files taking a long time to write to S3. There are a few possibilities here. The default committer ends up writing the files to a temporary location, then moves them to their final location once they've completed[3]. This could explain why running the code on S3 was slower, but we also had difficulties writing out parquet files locally as well. We also found that others were having an issue where the output driver took a long time in writing the common metadata files[2]. There is a way to turn this off (by setting the parquet.enable.summary-metadata flag to false), but that solution did not help us either. We also tried to perform all the analysis requiring neighborhood information in one go by adding the extra columns for location and calculating all the necessary summary statistics in one file. Although we could get the data with the new columns to display, any subsequent operations that we requested to be performed on the new data simply wouldn't work. The overhead of applying costly functions on a large amount of data seems to have done us in.

We were disappointed that we were not able to analyze our data by neighborhood. But as the data has to be generated for each and every row and we were dealing with nearly 1.5 billion rows, we understood that it might take longer time than expected and we considered this task as our future work.

3.3 Building a Model for Surcharge Prediction

In addition to extracting static values from the data, we wanted to see if we could use the data to do something more dynamic. We wanted to see if we could build a model to predict whether or not a taxi trip would include a surcharge (an extra fee). We used logistic regression with BFLGS update from PySpark's mllib package. After performing feature selection, we found that the pickup time, pickup location (in latitude and longitude) and vendor type (there are two large vendors for taxi fare meters in New

York City) provided the best accuracy for a logistic regression model. This model gives very close to 100% accuracy, which makes sense given that a surcharge is a predictable occurrence. Although we did not know the exact formula used to calculate whether or not surcharge should be added, we were able to predict its presence with a high degree of accuracy.

4 Presenting the Data

After using Spark to generate summary statistics in the form of CSV files, we used Tableau in order to visualize the results. This was the real point at which we could draw insights from the data that we had meticulously collected and cleaned. We embedded the resulting Tableau visualizations into a webpage, along with a summary of the project and other interesting results that we obtained. We are currently hosting this website at `http://thedatasets.com/onehotencodedtaxi/`. While Tableau is fairly straightforward and easy to use, we did find some challenges in creating dynamic visualization. We wanted to create visualization where axis of a plot could be changed to view same analysis using different scales. To do that new multidimensional parameter had to be created and queries had to be written to enable data filtering. This filtering was then illustrated by means of drop down menu in the visualization which could change the scale of the plot dynamically.

Although we couldn't break down the trips by neighborhood, we wanted to create at least one visualization of taxi trip location. We chose a single day (December 25, 2010) and visualized the locations of trips throughout the day using Tableau's mapping utility. Using just a single day's worth of data (several thousand trips) almost caused Tableau to crash! But we finally found a pattern from the data that as the day rises, people are travelling out of the city by having longer rides while at night, huge rush is been observed in the center of the city.

We also had to re-run some analyses when we got to the visualization stage. While Tableau is excellent at making visualizations, it is does not have much in the way of data cleaning capabilities. It was only while attempting to make visualizations for some of our aggregated datasets that we realized that we had not done enough to clean the dataset.

5 Self-Marking Scheme

- Getting the data: Acquiring/gathering/downloading: 0
- ETL: Extract-Transform-Load work and cleaning the data set: 3
- Problem: Work on defining problem itself and motivation for the analysis: 0
- Algorithmic work: Work on the algorithms needed to work with the data, including integrating data mining and machine learning techniques: 2

- Bigness/parallelization: Efficiency of the analysis on a cluster, and scalability to larger data sets: 4
- UI: User interface to the results, possibly including web or data exploration frontends: 2
- Visualization: Visualization of analysis results: 4
- Technologies: New technologies learned as part of doing the project: 7

References

[1] Donovan, Brian; Work, Dan (2016): New York City Taxi Trip Data (2010-2013). University of Illinois at Urbana-Champaign. https://doi.org/10.13012/J8PN93H8

[2] https://www.appsflyer.com/blog/the-bleeding-edge-spark-parquet-and-s3/

[3] http://dev.sortable.com/spark-directparquetoutputcommitter/

[4] https://www.ocf.berkeley.edu/~dlevitt/2015/12/13/final-project-nyc-taxi-and-uber-data/

[5] http://catalog.opendata.city/dataset/pediacities-nyc-neighborhoods

[6] https://pypi.python.org/pypi/Shapely

[7] http://aws.amazon.com