Santiago Larrain and Natnaell Mammo
CS 123 Big Data Final Report
The Taxi Project

Motivation
Taxi drivers have a clear incentive to lengthen their trips. The presence of this incentive, along with the uncertainty about the best route, often results in the question that most people who have paid for a cab have probably asked themselves: is this person trying to rip me off? The question is essentially one of fraud detection. The answer requires knowledge of the relevant determinants of an optimal route to a destination, and can also be informed by observation of the past behavior of a particular driver as well as comparisons between drivers along similar routes. WIth this mindset, we set out to identify the most common trip routes and then identify any extreme outliers that would then require further investigation.

Data Description
The intended data set for this project is a collection of all taxi records for New York City in 2013. Data is obtained from a previous FOIL request to NYC's Taxi and Limousine Commission. The files include information about the taxi ID, start and end time of the trip, the latitude and longitude of the pickup and drop-off locations, the fare amount, method of payment, and passenger count. The data is contained in 24 csv files and totals approximately 45 GB. Each month had a pair of files with ~12 million trips.
http://www.andresmh.com/nyctaxitrips/

The data was stored in a the research5 CSIL PC, on a PostgreSQL database as well as on an Amazon S3 bucket.

Data Cleaning
The following steps were taken during the data cleaning process:
- Removed trips that had lat or long = 0, for pickup or dropoff.
- Rounded the coordinates to the third decimal.
- Removed duplicates.
- Removed trips with less than 0.3 miles of distance.
- Removed trips with less than a minute duration.
- Removed trips from outside Manhattan.
- Removed encoded hack licence that was equivalent to 0.
- Joined the 2 files for each month into only one.

Finding the Most Common Trips (MRJob)
Identify groups of similar trips with the same latitude and longitude coordinates for pickup and dropoff, with different N number for each similar group.
We randomly picked 4000 unique trips for each month, and did it for every month.
We then used EMR on AWS to find every similar trip by pickup and dropoff longitude and latitude for each of the unique 48000 random trips.

Flagging Trips and Their Drivers

Now that we had a list of the most common trips, the next step we took was to identify the outliers in these. We decided to further breakdown these similar trips by bucketing on time of day and whether or not it was a weekday. resulting in the following categories:

- Morning Weekday/Weekend
- Afternoon Weekday/Weekend
- Rush Weekday/Weekend
- Evening Weekday/Weekend

For each of these time groups, we identified trips as being suspicious if the fare amount fell three or more standard deviations from the group mean. We also only decided to use groups that had at least 30 trips. The drivers that were responsible for these suspicious trips were then flagged; we then proceeded to pull of their trip history and investigate if this was a recurring behavior.

Pulling All Trips for Each Flagged Driver

Since we had a list of flagged drivers, we used EMR on AWS again, this time to pull every trip that every flagged driver did during 2013. The result were over 37 million trips.

Checking performance of flagged drivers

We had a list of all trips that every flagged driver did on 2013. We then checked for every similar trip for each of those 37 million trips, taking into account not only the coordinates, like before, but also. Then we checked for the distribution of each similar trip, and if the driver was above the standard deviations from the mean of the fare charged, he was possible fraudulent. The output was the sum of times a flagged driver was possibly fraudulent.

In order to compare each trip against the trips made by the flagged drivers (the ones made by the second use of MapReduce), we had to save the output of the second MapReduce into another s3 bucket, to which we read when running this job. This caused (and still causes) a lot of problems. We managed to run this last MapReduce Job locally on a very small subsample of the data, we haven't been able to run it on AWS' EMR without having a Memory Error.

Difficulties

We have encountered lots of problems through out the use of Elastic MapReduce on Amazon Web Services.
Before even starting using MapReduce, we uploaded our data to an EBS volume only to realize we couldn't use EMR on that. So we had to copy it to an S3 bucket.
At first, since we had to check for all similar trips to the random sample we had selected (the sampling was done not in the cloud), we had to use files with the sample data (other files than the 12 files with the taxi drives) and we tried different approaches. At first, we uploaded the files to a web service, and then downloaded them to the instances. In order to do this, we had to

install some packages and in order to do this, we had to install pip on the instances. This was effective, yet inefficient. It took almost 20 hours with the default instances.
So we tried uploading the files directly to the instances, using bootstrap options in the mrjob.config file. This also proved successful and more efficient, yet it was still very slow.
So we then asked to be allowed to run 100 instances instead of the default 20. And this is what we used for the second MapReduce Job, and it proved much faster.
However, the troubled started when we tried to use the last MapReduce Job, where we had to compare every trip, against 4 files that added up to 4 GBs. We had to upload those files to another S3 bucket and read them from there. Yet every time we tried them on AWS failed with memory error. We only managed to run it locally with a very small subsample.

Results
Unfortunately, since the program is still failing on EMR, we haven't been able (yet!) to learn who are the most fraudulent taxi drivers in New York. However, we do not intend to quit this endeavor until MapReduce has been totally tamed.