Milestone Two

Exploratory Analysis

Nae Rong Chang | Tomoki Takasawa | Parker Lopez | Stephen Mazeski

Introduction

Based on our proposal, our group had to change our focus slightly on the project. We were unable to obtain licensing for the Yelp dataset. For this reason, we have changed focus onto the taxi dataset offered in the cluster.

We chose the dataset based on the interesting data points it offered from both a trip price breakdown along with geographic locations being offered from start to finish of an individual trip.

Ultimately, we found cleaning this dataset proved very difficult without being allowed to utilize the libraries we were accustomed to (pandas, numpy, etc..). Because of this, we decided to focus on basic data cleaning, rather than techniques we have been taught in other classes.

Data Exploration

When looking into the taxi dataset, we were able to identify relationships right away. To get an initial feel for the dataset, and to get a working chunk to work with, we simply took the head of each type of csv file and turned them into individual text documents (See appendix). Although this generally is not the best technique in terms of getting a feel for the dataset, we did this more so to construct code for larger runs on HDFS.

We also utilized Pig Latin in order to work with loading a dataset of that scale correctly. We were able to achieve this and that can be found in the appendix as well. We chose to load certain data types as chararrays and doubles based on how we thought they would be altered in further exploration.

Dumping a csv file of that size in Pig Latin is never a good idea. In order to check to make sure our data was loaded correctly we used the LIMIT function within Pig. By using this, we were able to safely dump our chunk of data and check to make sure it was loaded correctly.

Data Cleaning

We used a “try” in our Python code in order to convert all of our string values that we would want to do calculation with in

the future into floating numbers. We were also able to eliminate headers of the files in the same step, creating data cohesion.

Because this is an exploratory analysis, we have set up our data for a join in the next step. The join will be over the medallion category and we hope to compare fare prices with geographic distances based on coordinates given in the database. That is why our key in the MapReduce is the medallion in our code.

Conclusion

Appendix

###Initial test to look at data shape and necessary to construct pig load. Also used for mini test.txt files

```
hdfs dfs -head
/ds410/taxi/faredata/trip_fare_9.csv
```

```
A = LOAD
'/ds410/taxi/faredata/trip_fare_9.csv' USING
PigStorage(',') AS (medallion:chararray,
hack_license:chararray,
vendor_id:chararray,
pickup_datetime:chararray,
payment_type:chararray,
fare_amount:double, surcharge:double,
mta_tax:double, tip_amount:double,
tolls_amount:double, total_amount:double);
```

Load based on how data appears on HDFS

```
describe A;

X = LIMIT A 50;

DUMP X;
```

This is to check that the data is loaded correctly within Pig

---

##Loading data in Python MRJob
##Remove Hack

```
from mrjob.job import MRJob

class mainfunc(MRJob):
    def mapper(self, key, value):

        lines = value.split(",")
        medallion = lines[0]
        hack_license = lines[1]
        vendor_id = lines[2]
        pickup_datetime = lines[3]
        payment_type = lines[4]
        fare_amount = lines[5]
        surcharge = lines[6]
        mta_tax = lines[7]
        tip_amount = lines[8]
        tolls_amount = lines[9]
        total_amount = lines[10]
      #for each category on list
        Try:
      #try to make floats from strings.
In turn eliminated null values and
headers that were unnecessary
intrinsically cleaning the data
            fare_amount =
float(fare_amount)
            surcharge = float(surcharge)
            mta_tax = float(mta_tax)
            tip_amount =
float(tip_amount)
            tolls_amount =
float(tolls_amount)
            total_amount =
float(total_amount)
            yield medallion,
```

```
(hack_license, vendor_id, pickup_datetime,
payment_type, fare_amount,
surcharge,mta_tax,tip_amount,tolls_amount
,total_amount)
            except:
                pass

    def reducer(self, key, list):
        for (a,b,c,d,e,f,g,h,i,j) in list:
            A = a
            B = b
            C = c
            D = d
            E = e
            F = f
            G = g
            H = h
            I = i
            J = j
        yield key,(B,C,D,E,F,G,H,I,J)

if __name__ == '__main__':
    mainfunc.run()
```

```
from mrjob.job import MRJob

class seconfunc(MRJob):
    def mapper(self, key, value):

        lines = value.split(",")
        medallion = lines[0]
        hack_license = lines[1]
        vendor_id = lines[2]
        rate_code = lines[3]
        store_and_fwd_flag = lines[4]
        pickup_datetime = lines[5]
        dropoff_datetime = lines[6]
        passenger_count = lines[7]
        trip_time = lines[8]
        trip_distance = lines[9]
        pickup_long = lines[10]
        pickup_lat = lines[11]
        drop_long = lines[12]
        drop_lat = lines[13]
    #for each category on list
        Try:
    #try to make floats from strings.
In turn eliminated null values and
headers that were unnecessary
intrinsically cleaning the data
        passenger_count=
float(passenger_count)
        trip_time = float(trip_time )
        trip_distance =
float(trip_distance )
        pickup_long =
float(pickup_long )
        pickup_lat = float(pickup_lat )
        drop_long = float(drop_long )
        drop_lat = float(drop_lat  )

        yield medallion,
(hack_license, vendor_id, rate_code ,
store_and_fwd_flag , pickup_datetime ,
dropoff_datetime ,passenger_count
,trip_time ,trip_distance ,pickup_long
,pickup_lat , drop_long , drop_lat )
            except:
                pass

    def reducer(self, key, list):
        for (a,b,c,d,e,f,g,h,i,j,k,l,m) in list:
            A = a
            B = b
            C = c
            D = d
            E = e
            F = f
            G = g
            H = h
            I = i
            J = j
            K = k
            L = l
```

```
            M = m
                yield
key,(B,C,D,E,F,G,H,I,J,K,L,M)

if __name__ == '__main__':
        seconfunc.run()
```