

Exercise 6c

A more complicated Spark problem

Prior Knowledge

Unix Command Line Shell

Simple Python

Spark Python

Simple SQL syntax

Learning Objectives

Pulling together your skills from previous exercises

Geospatial indexing

Statistical correlation

Software Requirements

(see separate document for installation of these)

- Apache Spark 2.0.0
- Python 2.7.12
- Nano text editor or other text editor

Overall plan – Dealing with Incident data from San Francisco Police Department

1. In the Ubuntu VM in the directory ~/datafiles/incidents/ you will find a file sfpd.csv.gz
2. This file contains every SFPD police incident since 2003, dated, located and categorized.
3. Our aim is to take the SF Wind and Temperature data from 2014 and correlate it with the police reports. To do that, we are going to first identify the nearest weather station to each incident.
4. Once we have done that, we can identify how many incidents happened per hour per day by nearest weather station, and then we can also join that with the average temperature and wind speed for those periods.
5. If all that succeeds we can attempt to use a statistical correlation between the number of incidents and the associated weather to see if there is a correlation.

PART A – Processing the wind data

We'd like a full year of this analysis, so let's grab the wind data from 2014. The following snippet is available here: <http://freo.me/oxclo-get2014>

```
cd ~/datafiles
mkdir wind2014
cd wind2014
wget http://freo.me/1LpKbGV -O wd2014.zip
unzip wd2014.zip
rm wd2014.zip
```

I don't suggest loading this data into HDFS, since we are only running on a single node. You can reference it locally within Spark using:

```
df = sqlContext.read.format('com.databricks.spark.csv').\
options(header='true', inferschema='true').\
load('file:///home/oxclo/datafiles/wind2014/*.csv')
```

Hint: if you want to easily parse text into datetime objects in Python:

```
from dateutil.parser import parse
from datetime import datetime

dt = parse(datestring) # returns datetime.datetime
```

In our case we want to produce the date and the hour. The following function takes the date given by the CSV and turns it into a tuple of (String, int) where String is the date e.g. "2014-01-01" and int is the hour from 0-23

```
def date_and_hour(s):
    dt = parse(s.replace('?', ' '))
    hour = dt.hour
    return (dt.strftime("%Y-%m-%d"), hour)
```

In order to do our analysis, we need to calculate the average wind speed and temperature for each 1 hour period, per station.

Another problem we have is bad data. Some records have all the numbers as 0.0, which I take to be a bad sign.

I recommend filtering data out where the wind speed and temperature are 0.0, and also where there are missing values.

There are lots of options, all of which have merit. My approach is to create a key of a tuple (Station, date, hour) where hour in {0-23}. The values of this RDD are (avg vel, avg temp).

You can have a go on your own at this.

Hint: You can either run your code interactively through the shell or submit it as a Python program. I find a combination of both is a successful strategy.

Don't forget the correct headers when you are submitting this as a job:

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext
conf = SparkConf().setAppName("app-name")
sc = SparkContext(conf=conf)
sqlc = SQLContext(sc)
```

And to submit the job:

```
~/spark/bin/spark-submit \
--packages com.databricks:spark-csv_2.11:1.2.0 \
--master local[*] \
your-python.py
```

If you get stuck, there is a sample program for Part A here:

<http://freo.me/oxclo-ws-part-a>

PART B – Locating the incident data.

Each incident has a geo-location (Lat, Long). Our aim is to create an RDD with the same key as the first step, but with value “count of incidents”. In order to do this, we need to associate the incidents to their nearest weather station.

Luckily there is a Python library (actually many!) that supports this. To install this library, on the Ubuntu terminal command line, type:

```
sudo pip install scipy
```

HINT: If you need to use numpy, scipy or other Python tools on **Spark EC2** instead of locally, then you need to install them on all instances (i.e. the slaves as well), not just the master. There is a blog about it here:

<https://datarus.wordpress.com/2014/08/24/how-to-install-python-and-non-python-packages-on-the-slave-nodes-in-spark/>

You will see a lot of build log go by, including a number of warnings. Don't worry!

scipy.spatial includes an algorithm KDTree (https://en.wikipedia.org/wiki/K-d_tree) that will find the nearest point from a set to another point.

If you can create an RDD with the following entry format:

(date, hour, [Y,X])

then the following snippet will remap that into:

(date, hour, location)

where location is e.g. SF04.

For example this will remap:
(“2014-01-01”,09, [37.4834543,-122.3187302])
to
(“2014-01-01”,09,“SF17”)

Snippet: <http://freo.me/oxclo-locate>

I recommend that you filter out only 2014 dates before you apply the location test.

From there it is a simple task to remap this data into:

```
((date, hour, location), 1)
```

and then count using reduceByKey.

If you get stuck, the full code is here: <http://freo.me/oxclo-wind-sfpd-complete>

PART C – Joining the data and looking for correlations.

Finally we need to join this data. Spark has a helpful capability. If you have two RDDs with the same keys then you can join them. So if you have

(k,v) and (k,w) then you will get (k,(v,w))

Finally once you have joined the data you can try some statistics. Spark has a built in test for correlation using either the Pearson or Spearman correlation statistics.

The following code snippet will create a correlation matrix looking for correlation between the incidents and the temperature and wind speed:

```
# assume we have the wind averaged in RDD windaveraged
# and the incident counts in incidentsreduced
# and the keys for both are the same format

joined = windaveraged.join(incidentsreduced)

from pyspark.mllib.linalg import Vectors
from pyspark.mllib.stat import Statistics

#remap the data into a Vector of [t, w, i]
vecs = joined.map(lambda ((s,d,h),((t,w),i)):
    Vectors.dense([t,w,i]))
print(Statistics.corr(vecs))
```

Is there any correlation?

Congratulations! You have completed this lab.