Exercise 9

A 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 3.1.2
- Python 3.x
- Jupyter Notebooks

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`

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.

© Paul Fremantle 2015. Licensed under the This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. See http://creativecommons.org/licenses/by-nc-sa/4.0/

PART A – Processing the wind data

We'd like a full year of this analysis, so let's grab the wind data from 2014.

You don't need to do this, as the data should already be on the VM. If it is missing, you can do this:
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 -o wd2014.zip
rm wd2014.zip
```

We are going to run this on our local machines, not on EC2. I recommend using Jupyter.

You can reference the wind data locally within Spark using:

```
df =
sqlContext.read.csv('file:///home/oxclo/datafiles/wind2014/*.csv',
      header='true', inferSchema='true')
```

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")+":"+ str(hour)
```

This code is in https://freo.me/clo-datetime

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.

© Paul Fremantle 2015. Licensed under the This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. See http://creativecommons.org/licenses/by-nc-sa/4.0/

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

There are lots of options, all of which have merit.

In Python2 my approach was 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). However, the syntax in Python3 is painful using tuples. I won't moan. Oh go on then.

Anyway, in Python3 I tend to do everything as DataFrames using Row() dictionaries. The code is very readable and it's really easy to mix map and sql.

Have a go!

If you get stuck, there is a sample program for Part A here:
https://freo.me/wind-and-crime-1

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 pip3 install scipy
```

It may already be installed.

*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-instal-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.

© Paul Fremantle 2015. Licensed under the This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. See http://creativecommons.org/licenses/by-nc-sa/4.0/

I have pre-written a function that takes a [Y,X] co-ordinate and maps it to the nearest weather station:

```
from scipy import spatial

def locate(l,index,locations):
    distance,i = index.query(l)
    return locations[i]

def map_yx_to_station(yx):
    return locate(yx,        \
    spatial.KDTree(array(     \
    [[37.7816834,-122.3887657], \
    [37.7469112,-122.4821759],  \
    [37.7411022,-120.804151],   \
    [37.4834543,-122.3187302],  \
    [37.7576436,-122.3916382],  \
    [37.7970013,-122.4140409],  \
    [37.748496,-122.4567461],   \
    [37.7288155,-122.4210133],  \
    [37.5839487,-121.9499339],  \
    [37.7157156,-122.4145311],  \
    [37.7329613,-122.5051491],  \
    [37.7575891,-122.3923824],  \
    [37.7521169,-122.4497687]])), \
        ["SF18", "SF04", "SF15", "SF17", "SF36", "SF37",\
        "SF07", "SF11", "SF12", "SF14", "SF16", "SF19", "SF34"] )
```

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 count the number of incidents per station+datehour

If you get stuck, the full code is here: https://freo.me/wind-and-crime

© Paul Fremantle 2015. Licensed under the This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. See http://creativecommons.org/licenses/by-nc-sa/4.0/

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 DFs with the same keys then you can join them.

You need to think carefully about whether you want an inner join or an outer join?

An inner join only finds the rows which match both tables. I think that is wrong because a lack of a row in the incidents table is equivalent to 0 incidents for that station+date+hour.

You may have to think about this a bit. Ask me or the TA to see if your approach agrees with mine. PS you may be right and I may be wrong!

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 a DataFrame called "joined" with columns of temp, wind and incidents:

```
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 row: \
      Vectors.dense([row.temp,row.wind,row.incidents]))
print(Statistics.corr(vecs))
```

Is there any correlation?

Congratulations! You have completed this lab.

© Paul Fremantle 2015. Licensed under the This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. See http://creativecommons.org/licenses/by-nc-sa/4.0/

***Hint**: You can either run your code interactively through the Jupyter or submit it as a Python program.*

If you want to convert your code to run as a job, you need these headers (which are implicit in the Jupyter pyspark context.

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

And to submit the job:

```
unset PYSPARK_DRIVER_PYTHON
unset PYSPARK_DRIVER_PYTHON_OPTS
spark-submit your-python.py
```

© Paul Fremantle 2015. Licensed under the This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. See http://creativecommons.org/licenses/by-nc-sa/4.0/