**Exercise 9**

*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 1.5.1
* Python 2.7.x
* Nano text editor or other text editor

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

1. 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.
6. Simples!

**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. You can reference it locally within Spark using:

df = sqlContext.read.format('com.databricks.spark.csv').\  
options(header='true', inferschema='true').\

load('/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.

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

When prompted for the password, use **oxclo**

*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/*](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.

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)

Snippet: URL goes here

where location is e.g. SF04.

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

From that 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: URL

Finally we need to join this data