THIRD ASSIGNMENT (2 POINTS)

This assignment has two parts:

1. *Machine learning with Spark (1 point).* This can be done in a basic way (0.75 points) and in a more efficient way (+0.25 points)
2. *Programming with Spark* (1 point). This part is also made of two sections, 0.5 points each.

Note: It is not compulsory to do all the parts. For instance, if you decide to do just part 1 but not part 2, your maximum grade will be 1 point.

1. MACHINE LEARNING WITH SPARK (1 point)

We are going to use the same dataset as in the second assignment (wind energy). You can load it into Spark from the supplied .csv file (unzip it first) by using this sentence:

wind_sd = spark.read.csv(path="wind.csv", header=True, inferSchema=True)

First of all, you should go through and understand the python notebook "Decision trees and pipelines in Spark", especially the second and third sections because we are going to use Spark 2.x here (Spark Dataframes). You can also find a complete reference about Spark Dataframe operations here: (http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame).

We are going to use a train/validation/test strategy, as we did in the second assignment. Therefore, you need to create three Spark dataframes: train (for years 2005, 2006), validation (2007, 2008), and test (2009, 2010). Do not forget to remove those columns which cannot be used for prediction ('steps', 'month', 'day', 'hour', 'year').

In this third assignment, we are going to deal with the following issue. The company suspects that there is lots of redundancy among the input attributes. After all, the same meteorological variable is computed at 25 locations in the 5x5 grid. Also, variables like "u-wind component at 100 meters" and "u-wind component at 10 meters" might be quite correlated. A powerful method to remove correlation between the input attributes is PCA (Principal Component Analysis). Typically, the number of PCA components is chosen by means of an unsupervised criterion (e.g. select the number of components that explain 99% of the variance). But in this assignment, you will select the number of components in a supervised way: the basic schema that you will follow is this: (1) compute k PCA components for the training set, (2) train a decision tree with them, (3) evaluate it on the validation set. MAE will be used as performance measure. This can be done for several k and then, select the k value that gets the best MAE on the validation set (note: perhaps checking all possible k values would involve too much computing time, so you will have to reach some sort of reasonable compromise, like testing just k=10, 20, 30, … or testing only the first 50 components, etc.). After that, use train+validation sets with k components for training a decision tree, and evaluate it on the test set. Do we get a better, or at least, a similar MAE using k components than using all the

components? In other words, is there a lot of redundancy in the input attributes that can be removed via PCA?

Everything in the paragraph above can be done using some ideas from the "Decision trees and pipelines in Spark", and you will get **0.75 points** for doing it correctly. However, probably you are computing the same PCA components again and again. That is, if you compute PCA with k=5 and then PCA with k=10, you are computing the first 5 components twice. Try to find a way to compute all (or at least, as many as you need) PCA components just once, and then select the components from there. Try to do this in the most efficient possible way and use operations that you can find in the Spark ML documentation. You can get the remaining **0.25 points** by doing this.

2. PROGRAMMING WITH SPARK (1 point)

The aim of this part is to program in Spark some code that could be used to program the Nearest Neighbor algorithm (KNN). Please, fill in the notebook supplied. Going through the "Programming k-means" notebook could be useful, although it is not completely necessary (the code that you have to write is much simpler, but it will require some thinking). In particular, what you have to do is:

1. (0.5 points) Find the closest instance (this could be used for KNN with k=1)
2. (0.5 points) Find the second closest instance (this could be used for KNN with k=2)

Your code will be just a few lines long, but please, explain clearly in your notebook what your code does. **IMPORTANT:** You are not allowed to sort the RDD (i.e. you cannot use takeOrdered(), top(), sortByKey(), or similar).

WHAT TO HAND IN: In both cases, you will have to hand in a notebook with comments.