Returning your solutions

Return your solutions via Moodle page. We do not count belated returnings. You have to use .pdf file type for any written answers, and .scala/.py for your code. Zip your code and pdf files together and upload. Do not return any project or object files, etc. Mark your **full name and student number** clearly to all your solution files. Each exercise will be graded pass/fail.

Spark API Documentation Links for Scala
http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.package

Spark API Documentation Links for Python
http://spark.apache.org/docs/latest/api/python/index.html

Programming Guidelines

You can download the skeleton file for Scala and Python.

You cannot use an RDD inside another RDD.

This way we can import and run all of the code of all the students in a single Eclipse project.

Exercise 1

Carat Context-factor Dataset Download link . Download the file and unzip it. The size of the csv file is about 1 Gigabyte and the number of rows in the data set is more than 11 million. The data set has the following attributes.

energyRate;batteryHealth;batteryTemperature;batteryVoltage;cpuUsage;distanceTraveled;mobileDataActivity;mobileDataStatus;mobileNetworkType;networkType;roamingEnabled;screenBrightness;wifiLinkSpeed;wifiSignalStrength

CPU load from 0 to 1 (0% to 100%)
Distance traveled: higher than or exactly zero (binary classification)
Screen brightness: from 0 to 255, and -1 as an automatic setting
Wi-fi signal strength: from -100 to 0, exclusive
Battery temperature: higher than or exactly zero

TODO
Consider attribute values which are in the above mentioned range values. You will have to apply a filter function with a number of conditions. Then, write a correlation function, such as pearson correlation (you can follow Eemil's lecture) and apply it for the following attribute pairs (energyRate, cpuUsage), (energyRate, screenBrightness), (energyRate, wifiLinkSpeed), and

(energyRate, wifiSignalStrength). Use RDD to load the file and all the required transformations to compute the correlations. Calculate the time taken for each correlation on attribute pairs to be completed. Report it in your PDF.

Exercise 2: Spark-sql Use the same Carat Data Source from Exercise 1 : Download link

a) Load the data 'carat-context-factors-percom.csv' into a data frame object called caratDF. What's the schema of the data frame caratDF.

   Hint:- Load the data using the function csv. Check if the data you loaded has 14 columns. If it has only one column use option sep=";"

b) Add the following column names to caratDF.
   "energyRate","batteryHealth","batteryTemperature","batteryVoltage","cpuUsage,"distanceTraveled","mobileDataActivity","mobileDataStatus","mobileNetworkType","networkType","roamingEnabled","screenBrightness","wifiLinkSpeed","wifiSignalStrength"

   Hint: Pass these column names as arguments to a function called 'toDF'
   The function is available in all data frames.
   Remember to store the result back into caratDF.
   Check the columns are named using show function.

c) Count the unique elements in the following columns. "batteryTemperature", "batteryVoltage". Also count the outlier data size in the above columns. Use the default value information below:

   Note:
   Battery voltage: from(>=) 2.0 to (<=) 4.35; outside this range are the outliers
   Battery temperature: from(>=) 0.0 to (<=) 50.0; outside this range are the outliers

   Hint:
   If your column data type is not of type Double, you can change it using dataFrame.withColumn function. Refer the following link for example,
   http://stackoverflow.com/questions/29383107/how-to-change-column-types-in-spark-sqls-dataframe/33423959

   http://stackoverflow.com/questions/32284620/how-to-change-a-dataframe-column-from-string-type-to-double-type-in-pyspark

d) From the data frame caratDF, extract the following columns and Store them into a new data frame called carat.
   i) energyRate
   ii) CPU Usage from 0 to 1 (0% to 100%)
   iii) Screen brightness: from 0 to 255, and -1 as an automatic setting
   iv) Wi-fi signal strength: from -100 to 0, exclusive
   v) wifiLinkSpeed

e) Now similar to Exercise 1 but with DataFrame, find correlation for the following pairs of variables in data frame carat. (energyRate, cpuUsage), (energyRate, screenBrightness), (energyRate, wifiLinkSpeed), (energyRate, wifiSignalStrength). Calculate the time taken for each correlation on attribute pairs to be completed. Report it in your PDF. Can you notice any difference in time taken to complete each correlation task similar to Exercise 1. If so, mention the time difference in your PDF.
   Hint :
   For Python, use “corr” function on data frame.
   For Scala, use “stat.corr” function on data frame

http://spark.apache.org/docs/2.1.0/api/python/pyspark.sql.html
http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.DataFrameStatFunctions
http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.package

Exercise 3: GroupByKey, ReduceByKey, and Partition

You are given Carat dataset with 11million data points. The purpose of this exercise is to realize the performance gain with different APIs and partitioning techniques on Spark RDDs.

(1) Filter out rows with screen brightness <=-1 and higher than 255.
(2) GroupByKey the elements with respect to the screen brightness, count the number of elements and find the total energy rate for each brightness level.
(3) Apply reduceByKey for the above problem with respect to the screen brightness.
(4) Apply Hash and Range partitioning with 50 partitions before applying reduceByKey. Compare the performance and discuss the results. *In the case of Python, the Range partitioner is not available. Please write a short note of 10 sentences about Spark partitions and performance optimization.*

Hint: check the optimization slides presented on 28.03.2017.

Exercise 4: Closure & Broadcast

You are given a set of scientific documents here. You are required filter out the stop words and create bag of words for each document. Finally, you will return the filename and size of the bag. In order to filter the stop words, you should follow both of the steps and compare the performance. Download dataset from here. The text files are messy, you may need to clean the unexpected characters and numbers from files with some regex pattern. Have a look into the text file. Download the stop words from here.

(1) Define and initialize a variable with stopwords outside an rdd transformation and utilize the variable inside the transformation.
(2) Broadcast the stop words and use the broadcast variable inside the transformation.
(3) After removing the stop words or other unwanted characters, compute the term frequency (TF), IDF, and TFIDF for each term or word and form an RDD of (word, tf, idf, tfidf).

Hint : In order to load a whole file together as an RDD use sparkcontext.wholeTextfiles. The path is directory of the text files (after unzipping).

```
val rdd = sc.wholeTextFiles(path).map {
    case (file, text) => ({
           // clean the unwanted texts or characters
           // words are the unique words in a document. You write code here for filtering out the stopwords//
           // numwords is the term frequency
     (file,List(word,numwords) )})
 }
```

Term frequency-inverse document frequency (TF-IDF) is a feature vectorization method widely used in text mining to reflect the importance of a term to a document in the corpus. Denote a term by tt, a document by dd, and the corpus by D. Term frequency $TF(t,d)$ is the number of times that term t appears in document d, while document frequency $DF(t,D)$ is the number of documents that contains term t. If we only use term frequency to measure the importance, it is very easy to over-emphasize terms that appear very often but carry little information about the document, e.g., “a”, “the”, and “of”. If a term appears very often across the corpus, it means it doesn’t carry special information about a particular document. Inverse document frequency is a numerical measure of how much information a term provides:

$$IDF(t,D)=\log((|D|+1)/(DF(t,D)+1))$$

where $|D|$ is the total number of documents in the corpus. Since logarithm is used, if a term appears in all documents, its IDF value becomes 0. Note that a smoothing term is applied to avoid dividing by zero for terms outside the corpus. The TF-IDF measure is simply the product of TF and IDF:

$$TFIDF(t,d,D)=TF(t,d) \cdot IDF(t,D).$$

Exercise 5:

(1) How would you construct an effective machine learning pipeline (First Lecture)?
(2) How would you apply collaborative filtering on movielens dataset (Look Around)?
(3) Compare HaLoop, with Hadoop and Spark in 21 sentences (Reading Materials).