Programming for Big Data - Spark Lab 1

In this lab you will use the Databricks Community version of Apache Spark in your browser and run some simple interactive analysis on a set of data.

1. Registration

Go to https://databricks.com/try-databricks and register an account and login to the Web interface for SPARK.

Click “New Notebook”, call it “lab1” and select Scala as language.

To test - create a sequence of 1 to 50, place that sequence in a RDD and then filter for numbers less than 10.

```
val data = 1 to 50
val sparkSample = sc.parallelize(data)
sparkSample.filter(_ < 10).collect()
```

To run this, type the code above in a cell, click Run or press Shift+Enter and confirm to attach to a new cluster.

2. Querying Bram Stoker

Download the following file from Brightspace:

BramStokersDracula.txt

Upload it to DBFS (Data -> Tables -> Upload File). The file should be saved in “/FileStore/tables/BramStokersDracula.txt”.

This is Bram Stoker’s Dracula from Project Gutenberg. Let’s analyse the text using Spark.

```
val lines = sc.textFile(“/FileStore/tables/BramStokersDracula.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

The first line creates a pointer to the file in the file store, the second line maps all the line lengths in the document and the last line reduces the result of the mappers by adding the all the line lengths together. This is a lazy operation so the first two only compute when the last one has been called.

To illustrate the lazy operation, rename the file argument *BramStokersDracula.txt* in *sc.textFile* to *BramStokersDracula1.txt* and rerun the commands. You will see that each

command will complete until you call the reduce function, which will complete the entire chain of commands in a lazy fashion and return a an error because it cannot find the file.

Now let's demonstrate caching of data in memory. Rename the file *BramStokersDracula1.txt* back to *BramStokersDracula.txt* and run the following commands:

```
val lines = sc.textFile(/FileStore/tables/BramStokersDracula.txt")
lines.cache
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

The important line is highlighted in bold letters. Now rename *BramStokersDracula.txt* back to *BramStokersDracula1.txt* and rerun the last two lines and you will get a result.

```
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

This is because *lines* was cached once it was executed correctly and then made available to later operations. Rename *BramStokersDracula1.txt* back to *BramStokersDracula.txt* to continue with the lab.

Now let's see what are the most frequently used words in this document.

```
val tokenized = lines.flatMap(_.split(" "))
val wordCounts = tokenized.map((_, 1)).reduceByKey(_ + _)
val filtered = wordCounts.filter(_._2 >= 150)
filtered.takeOrdered(10)(Ordering[Int].reverse.on(x=>x._2))
```

Let's remove some of the stop words using a second dataset of stopwords and compare the results. A list of stopwords is available at:

stopwords_en.txt

```
val stopWords = sc.textFile("/FileStore/tables/stopwords_en.txt")
val cleaned = tokenized.subtract(stopWords)
val cleanedWordCounts = cleaned.map((_, 1)).reduceByKey(_ + _)
val cleanedFiltered = cleanedWordCounts.filter(_._2 >= 150)
cleanedFiltered.takeOrdered(10)(Ordering[Int].reverse.on(x=>x._2))
```

Q1: What words are different between the two datasets?
(Paste Answer below)

Q2: In the cleaned dataset, what are the ten least frequent words?
(Paste Answer below)

---

3. Querying Auction Data

Now we will use Spark's SQL capabilities to query a dataset of auction bids.

auctions.csv

Download and import the dataset.

Load the csv file into dataframe using the SQL Context:

```
val dfr = sqlContext.read.format("com.databricks.spark.csv").option("header", "true")
val df = dfr.load("/FileStore/tables/auctions.csv")
```

Show all the bidders who have a bidderrate of over 21.

```
df.filter(df("bidderrate") > 21).show()
```

Count all the distinct bidders

```
df.select("bidder").distinct.count
```

The complete guide for interfacing with data frames is available here:

http://spark.apache.org/docs/latest/sql-programming-guide.html

Please play around with the data frame and test out alternatives. This approach, in which you have a wrapper around an SQL table, is called and ORM or Object Relational Mapping and helps keep code clean, reusable and maintainable.

Q3:List all the bidders using a grouped statement
(Paste Answer below)

Now let's register the data frame as a temp table to query with an sql statement.

```
df.createOrReplaceTempView("df")
sqlContext.sql("select * from df").collect.foreach(println)
```

The collect method retrieves all the RDD to the local disk. In this lab it is fine to use but you must be careful because you can topple the heap on a local machine if retrieving a large dataset.

Show the two most frequent bidders.

> *sqlContext.sql("select bidder, count(*) as cnt from df group by bidder order by cnt desc limit 2").collect().foreach(println)*

Q4: Who are the ten least frequent bidders? (Paste Answer below)

----

4. Summary

In this lab we have briefly looked at some of the possibilities of SPARK. In the next labs we will look more closely at SPARK's capabilities.