Assignment 3 - Apache Spark (LDSA)

Important Notes

Submission Guidelines and Marks

This assignment is worth **4 marks in total**. Complete Part 1 for a total of **2 marks**. To pass this assignment, you need to get the marks for Part 1 (sections A and B). The marks are awarded for submitting correct and complete code to each section. Submit two Python Jupyter notebooks, one for each section. Submit with an .ipynb extension for each notebook. Submit the notebook with the output included. Do not submit a PDF file, a .py file, a screenshot, etc.

For example, if the question asks you to count something, the code should do that and the result should be printed out in the notebook. Poorly formatted code or code with many spelling mistakes may be penalized. Do not submit blocks of commented-out code, it makes it difficult for me to know what to mark. I must be able to run the code on the cluster without modifying it.

Part 2 is voluntary to complete. Part 2 comprises sections C and D, **each worth 1 mark**, for a total of 2 further marks. For these sections, submit a single PDF file with your written answers. Label your answers clearly with the question numbers. The document must be well-formatted. Poor spelling, grammar, punctuation may be penalized.

In all, you should submit 2 Jupyter notebooks, and 1 PDF file.

These marks are a guideline, where there are mistakes in some sections I'll need to make a judgement about the overall mark. I strongly recommend that you attempt all questions. You can work together to do the deployment, running the examples from the lectures, and exploring the Web GUIs. **Work individually for all sections of this assignment**.

Deployment

You need to begin by deploying a virtual machine and testing your Python notebook with one of the examples from the lecture. Follow the instructions here:
https://github.com/benblamey/jupyters-public/blob/master/deploy-instructions-2020.txt

Be sure to abide by the guidelines listed in that document for using the cluster cooperatively (and not consuming all the resources). There will be technical problems from time to time with the cluster. When it breaks, we'll need to fix it. Get started early, and aim to finish before the deadline.

Learning Outcomes

Learning Outcome	*Assessed*
Use Apache Spark appropriately for interactive analysis of large datasets, using a range of Map/Reduce operations	Section A, B
Deploy Apache Spark to the Cloud	Prerequisite.
Describe key components in the Spark architecture/execution model, use this understanding to develop applications effectively, and avoid common performance pitfalls	Section C, D
Relate key ideas in the implementation of Spark to concepts in the theory of distributed computing and big data.	Section C

Learning Resources

You will need to use the lecture material, Spark documentation and code examples, to answer the questions. Remember the 'cheat sheet' handout is intended to help you to navigate the Spark documentation - keep it next to you when you are coding! Unless otherwise stated, you must use Spark for all the calculations (not e.g. Excel or Linux tools!).

Debugging

As we discussed in the lecture, Spark will run our Python code remotely on our cluster, and collect results (or errors!) back to our client application. You can expect error messages to be less helpful than with regular Python.

I recommend you use `my_rdd.take(10)` or `my_dataframe.show()` to check the contents of your RDD at each step. Build up working code by adding one step at a time. Convince yourself of your own understanding before moving on.

Web resources like stackoverflow are useful in that they can help you discover the names of functions (especially where you know what you want but don't know what its called). If you use snippets from the web, be sure to take the time to fully understand it, and modify it so it does what you need. Use Python 3 for this assignment.

Part 1

Section A - Working with the RDD API

For this section, we'll use the PySpark RDD API. Use the existing LDSA cluster, and a notebook on your own client machine, which you must deploy yourself.

A parallel corpus is a document, or collection of documents, in multiple languages, where sentence boundaries are aligned between those boundaries. We'll work with a parallel corpus of transcripts from the European Parliament http://www.statmt.org/europarl/ By modern standards, this dataset is very small, but will suit our purposes for this assignment.

Choose a language: I have loaded corpora for all languages into the HDFS cluster, in the directory 'europarl'. Germanic languages like Swedish are likely to work better. For each language there is a pair of aligned files, one in English, and one in the other language.

Question A.1

A.1.1 Read the English transcripts with Spark, and count the number of lines.
A.1.2 Do the same with the other language (so that you have a separate lineage of RDDs for each).
A.1.3 Verify that the line counts are the same for the two languages.
A.1.4 Count the number of partitions.

Question A.2

A.2.1 Pre-process the text from both RDDs by doing the following:

- Lowercase the text
- Tokenize the text (split on space)

Hint: define a function to run in your driver application to avoid writing this code twice.

A.2.2 Inspect 10 entries from each of your RDDs to verify your pre-processing.
A.2.3 Verify that the line counts still match after the pre-processing.

Question A.3

A.3.1 Use Spark to compute the 10 most frequently according words in the English language corpus. Repeat for the other language.
A.3.2 Verify that your results are reasonable.

Question A.4

A.4.1 Use this parallel corpus to mine some translations in the form of word pairs, for the two languages. We'll achieve this by looking for pairs of words that frequently occur in the same position within lines.

For these two lines (from the sv/en parallel corpus):

en: I declare resumed the session of the European Parliament adjourned …
sv: Jag förklarar Europaparlamentets session återupptagen efter avbrottet ….

The pairs would be:
I/Jag, declare/förklarar, resumed/Europaparlamentets, the/session, ...

Clearly, not all the pairs are valid translations. The idea is to look for frequently occurring pairs over a large corpus.

This is a suggested approach - work with the pair of RDDs you created in question A.2.
Hint: make a new pair of RDDs for each step, sv_1, en_1, sv_2, en_2, ...

1. Key the lines by their line number (hint: ZipWithIndex()).
2. Swap the key and value - so that the line number is the key.
3. Join the two RDDs together according to the line number key, so you have pairs of lines with the same line number.
4. Filter to exclude line pairs that have an empty/missing "corresponding" sentence.
5. Filter to leave only pairs of sentences with a small number of words per sentence, this should give a more reliable translation (you can experiment).
6. Filter to leave only pairs of sentences with the same number of words in each sentence.
7. For each sentence pair, map to give a list of word pairs (in order) from the two sentences. We no longer need the line numbers. (hint: use python's built in zip() function)
8. Use reduce to count the number of occurrences of the word-translation-pairs.
9. Print some of the most frequently occurring pairs of words.

Do your translations seem reasonable? Use a dictionary to check a few (don't worry, you won't be marked down for incorrect translations!).

Section B - Working with DataFrames and SQL

For this section, we'll use the PySpark DataFrames/SQL API. Use the existing LDSA cluster, and a notebook on your own client machine, which you must deploy yourself.

We'll work with a large dataset in CSV format. Our dataset is the Los Angeles Parking Citations (https://www.kaggle.com/cityofLA/los-angeles-parking-citations). I have pre-loaded the dataset into the HDFS cluster.

B.1 Load the CSV file from HDFS, and call show() to verify the data is loaded correctly.

B.2 Count the number of partitions in the underlying RDD.

B.3 Print the schema for the DataFrame.

B.4 Count the number of rows in the CSV file.

B.5 Drop the columns Agency Description, Agency, and Route.

B.6 Find the mean fine amount (you need to convert the column to a float).

B.7 Show the top 10 most frequent vehicle makes, and their frequencies.

B.8 Let's expand some abbreviations in the color column. Create a *User Defined Function* to create a new column, 'color long', mapping the original colors to their corresponding values in the dictionary below. If there is no key matching the original color, use the original color.

```
COLORS = {
'AL':'Aluminum', 'AM':'Amber', 'BG':'Beige', 'BK':'Black',
'BL':'Blue', 'BN':'Brown', 'BR':'Brown', 'BZ':'Bronze',
'CH':'Charcoal', 'DK':'Dark', 'GD':'Gold', 'GO':'Gold',
'GN':'Green', 'GY':'Gray', 'GT':'Granite', 'IV':'Ivory',
'LT':'Light', 'OL':'Olive', 'OR':'Orange', 'MR':'Maroon',
'PK':'Pink', 'RD':'Red', 'RE':'Red', 'SI':'Silver', 'SL':'Silver',
'SM':'Smoke', 'TN':'Tan', 'VT':'Violet', 'WT':'White',
'WH':'White', 'YL':'Yellow', 'YE':'Yellow', 'UN':'Unknown'
}
```

B.9 Using this new column, what's the most frequent colour value for Hondas (HOND)?

Part 2

Section C - Concepts in Apache Spark and Distributed Computing

Please answer in full sentence(s) (max 3), per question, as appropriate. Use the handout to help you. Marks will be awarded based on the correct use of terminology for Spark and distributed computing concepts. Do not need to submit code for these questions, but you might want to experiment with your code to help you answer them.

C.1 Anna Exampleson is trying to understand her Spark code by adding a print statement inside her split_line(..) function, as shown in this code snippet:

```
def split_line(line):
    print('splitting line...')
    return line.split(' ')

lines = spark_context.textFile("hdfs://host:9000/king-dream.txt")
print(lines.flatMap(split_line).take(10))
```

When she runs this code in her notebook, she sees the following output:

```
['I', 'am', 'happy', 'to', 'join', 'with', 'you', 'today', 'in',
'what']
```

But, she doesn't see the `"splitting line…"` output in her notebook. Why not?

C.2 Why is it advantageous that data in Spark partitions is immutable?

C.3 *"Calling .collect() on a large dataset can cause my driver application to run out of memory"*
Explain why.

C.4 In what sense are RDDs 'resilient'? How is this achieved?

Section D - Essay Question

"A colleague has mentioned her Spark application has poor performance, what is your advice?"

List 4 clear recommendations, answer in full sentences. I suggest 1 or 2 sentences for each recommendation. Marks will be awarded based on the correct use of Spark terminology. Do not submit code for this question (but you might want to mention API methods, for example). You do not need to include references, but if you feel this supports your answer, please include a short, well-formatted bibliography.