

Exercise 5

Get started with Apache Spark and Python

Prior Knowledge

Unix Command Line Shell
Simple Python

Learning Objectives

Understand the Spark system
Understand the Jupyter Notebook model
Submit Spark jobs locally and using YARN
Write SparkSQL code in Python
WordCount!

Software Requirements

(see separate document for installation of these)

- Apache Spark 2.0.0
- Python 2.7.12
- Jupyter notebooks

Part A. Spark Python Shell (pySpark)

1. We are going to do a wordcount against a set of books downloaded from Project Gutenberg. Wordcount is the definitive Big Data program (sort of Hello World for Big Data) and it is frankly embarrassing that we haven't done one yet.
2. Apache Spark has a useful Python shell, which we can use to interactively test and run code. Since we have our data in HDFS, *we need to ensure HDFS is running*. (Follow the instructions from the Hadoop lab).
3. Let's load some books into HDFS. In a terminal window (Ctrl-Alt-T)

```
hadoop fs -mkdir -p /user/oxclo/books  
hadoop fs -put ~/datafiles/books/* /user/oxclo/books/
```

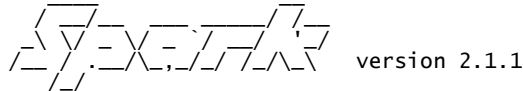
4. Let's make a directory for our code:

```
mkdir ~/pse  
cd ~/pse
```
5. Now start the Spark Python command line tool –

```
~/spark/bin/pyspark
```

- a. You should see a lot of log come up, ending in something like:

```
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
17/07/01 13:51:32 WARN NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
17/07/01 13:51:32 WARN Utils: Your hostname, oxclo resolves to a loopback
address: 127.0.0.1; using 172.16.64.199 instead (on interface ens33)
17/07/01 13:51:32 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
17/07/01 13:51:38 WARN ObjectStore: Failed to get database global_temp, returning
NoSuchObjectException
welcome to
```



```
using Python version 2.7.12 (default, Nov 19 2016 06:48:10)
SparkSession available as 'spark'.
>>>
```

- b. This is the “traditional” Spark Python command line tool. We aren’t going to use this just now.
- c. Type `quit()` to leave.
6. The VM has a “notebook” system called Jupyter configured by default. The result is that instead of starting a command line repl¹, there is a web based editor / evaluator launched instead.
7. To start this, type
`~/pyspark-jupyter.sh`
8. In the command-line you will see

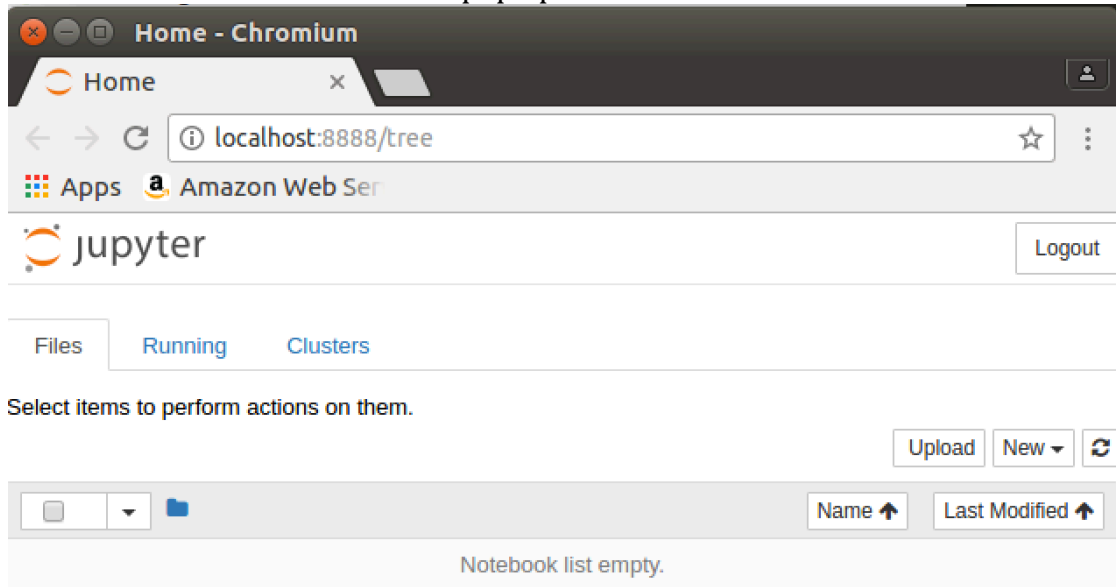
```
[I 13:53:23.865 NotebookApp] Serving notebooks from local
directory: /home/oxclo/pse
[I 13:53:23.866 NotebookApp] 0 active kernels
[I 13:53:23.866 NotebookApp] The Jupyter Notebook is running at:
http://localhost:8888/?token=fd655aab32ed4840ceb47b8b7392b1243a27f5
6350888a91
[I 13:53:23.866 NotebookApp] Use Control-C to stop this server and
shut down all kernels (twice to skip confirmation).
[C 13:53:23.868 NotebookApp]
```

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:

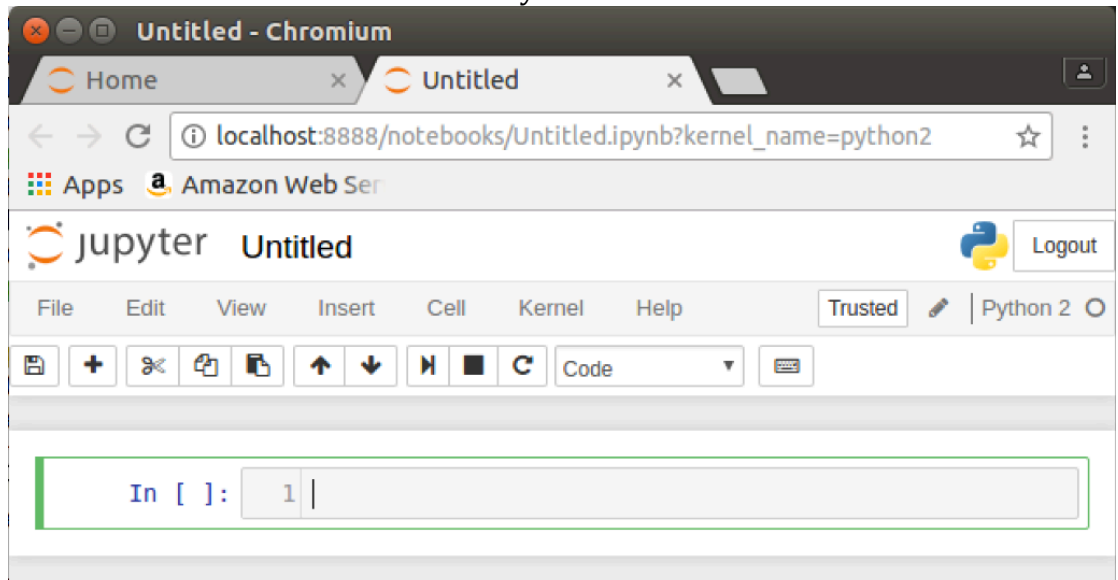
```
http://localhost:8888/?token=fd655aab32ed4840ceb47b8b7392b1243a27f5
6350888a91
```

¹ Read Eval Print Loop

9. And then a browser window will pop up.

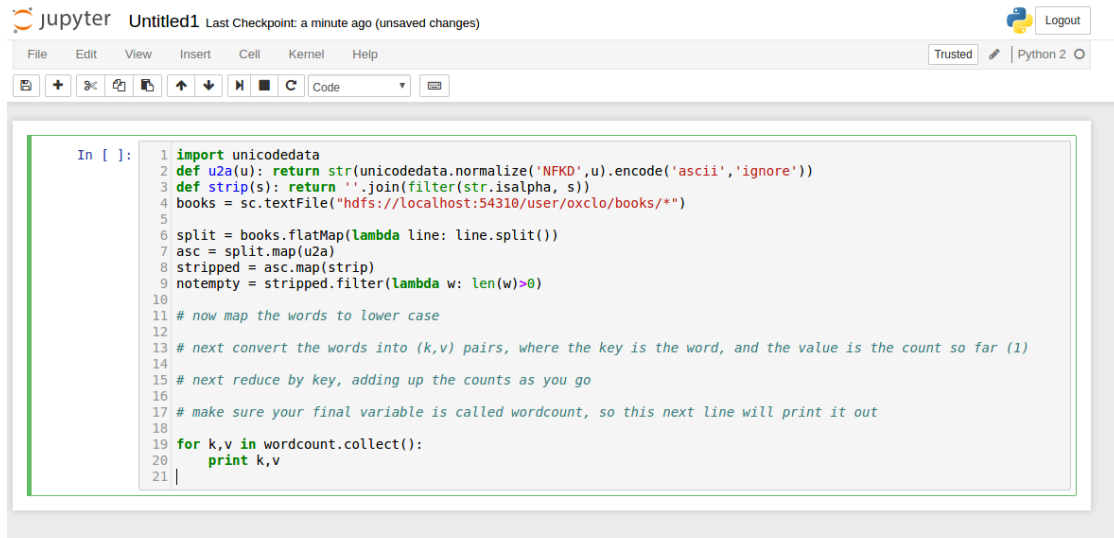


10. Use the **New** button to create a new Python2 notebook:



11. There is a starter of the code you need in the following URL:
<https://freo.me/first-notebook>

12. Paste that into the cell [1] so it looks like this:



13. There are some aspects that are not filled in that you need to write.
Basically this is a data-processing pipeline (also a directed acyclic graph)

14. *Let's look at the parts that are there already.*

15. We already have a SparkContext object defined in the notebook (in a program you need to define one, which we will see later)

16. Unfortunately some of the input is handled as Unicode by Python and we want to get rid of that.

```
import unicodedata
def u2a(u): return str(unicodedata.normalize('NFKD',u).
    encode('ascii','ignore'))
```

17. We also want to remove any non-alphanumeric characters:

```
def strip(s): return ''.join(filter(str.isalpha, s))
```

18. With the preliminaries over, the next line loads the data in:

```
books =
sc.textFile("hdfs://localhost:54310/user/oxclo/books/*")
```

19. Then splits the lines into separate words

```
split = books.flatMap(lambda line: line.split())
```

20. Deals with the Unicode problem

```
asc = split.map(u2a)
```

And removes non-alpha characters

```
stripped = asc.map(strip)
```

and removes empty items:

```
notempty = stripped.filter(lambda w: len(w)>0)
```

21. Now it is time for you to do something!

Convert all the words to lower case, using a map operation. In python, if *str* is a string, then `str.lower()` is the same string in lower case.


22. Now you need to get ready for a reduce. In order to do a reduce, we need some form of *key, value* pairs. I recommend using *tuples* which are simply (k,v) in Python (the brackets group the items into a tuple).

23. Remembering how reduce works, we need each word to have a count. Before reducing, that count is 1. So we need a lambda that takes a word *w* and returns (w,1)

24. Now we can do a reduce that adds all those counts together.

25. Finally, we need to collect the results and print them. In Spark, they may be distributed across different RDD partitions on different machines, so the `collect()` method brings them together.

```
for k,v in wordcount.collect(): print k,v
```

26. Try running the cell, by clicking 

27. Be patient. I suggest you look at the command window and wait until you see spark start working.


28. You should see a word count appear below cell 1:

```
systematic 7
parallelogram 1
sowell 1
presnya 1
four 265
conjuring 1
chamberagain 1
marching 32
sevens 4
awistocwacy 1
trotat 1
canes 1
shipmets 1
understandthat 2
lorn 16
lore 1
inwards 2
wickam 62
utterand 1
slightue 1
```

29. Congratulations!

30. While the pyspark is still running browse to <http://localhost:4040>

31. You will see the Spark web console:

2.1.1

JobsStagesStorageEnvironmentExecutorsSQL

PySparkShell application UI

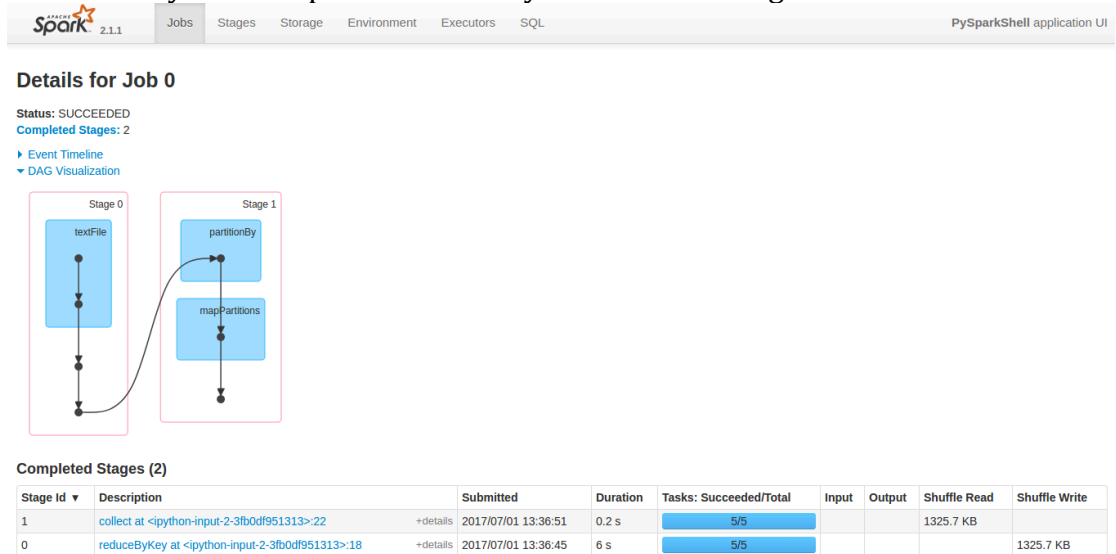
Spark Jobs (?)

User: oxclo
Total Uptime: 6.9 min
Scheduling Mode: FIFO
Completed Jobs: 1
[Event Timeline](#)

Completed Jobs (1)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at <python-input-2-3fb0df951313>:22	2017/07/01 13:36:45	6 s	2/2	10/10

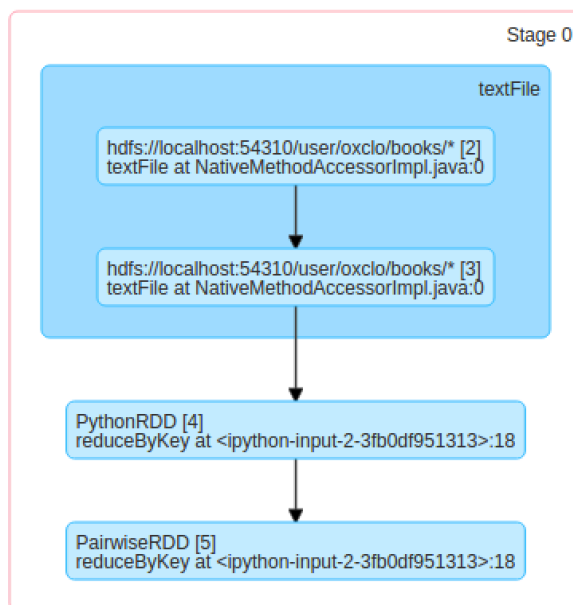
32. Click on the blue link “collect at ipython-input”
This shows you how Spark converted your code into stages:



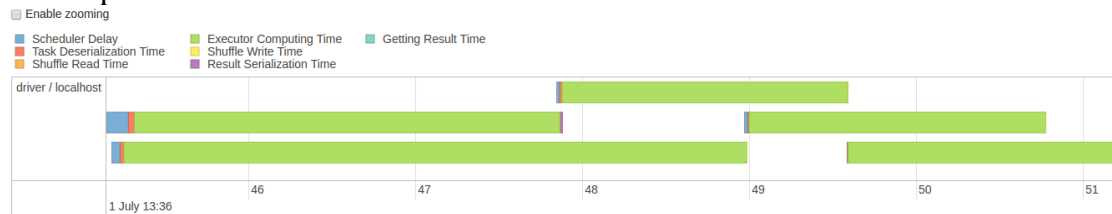
33. Click on Stage 0

Details for Stage 0 (Attempt 0)

Total Time Across All Tasks: 11 s
Locality Level Summary: Process local: 5
Shuffle Write: 1325.7 KB / 330
 DAG Visualization



34. And expand the Event Timeline:



35. Make sure your code is saved from the notebook.

36. Quit the notebook shell by typing Ctrl-C on the command line, and then Y Also close the notebook windows in the browser.

37. Now let's run the same code as a "job" instead of interactively.

38. From <http://freo.me/oxclo-wc-py> copy the code into a file `wc-job.py`

39. You will notice that there is a bunch of "setup" code that we didn't need in the pyspark command line tool. That is because pyspark assumes you want all this and does it for you.

40. We run jobs locally on a single node directly on Spark:

The `local[*]` indicates to use as many threads as you have cores on your system:

```
~/spark/bin/spark-submit --master local[*] wc-job.py
"hdfs://localhost:54310/user/oxclo/books/*"
```

41. Congratulations, the lab is complete!

Extension

42. Re load the code into the Jupyter notebook and now improve it to show the wordcount in descending order, starting with the most common words.