CS/INFO 5304 Assignment 1: Programming in Spark

Due: Feb 14, 2024 11:59 PM

The goal of the assignment is to get you started with Spark. Here you will learn how to write, compile, debug and execute a simple Spark program in your local machine. First part of the assignment serves as a tutorial and the second part asks you to write your own Spark program.

Credit: 10 points
Submission: Submit the code and the results file in GradeScope (File structure explained in detail at the end of the document)
Tip: Read the document fully before you begin working on the assignment

Setting up a stand-alone Spark instance

This section explains how to download and install a stand-alone Spark instance. All operations done in this Spark instance will be performed against the files in your local file system. You may setup a full (single-node) spark cluster if you prefer; the results will be the same. You can find instructions online. If you'd like to experiment with a full Spark environment where Spark workloads are executed by YARN against files stored in HDFS, we recommend the Cloudera Quickstart Virtual Machine: https://www.cloudera.com/downloads/quickstart_vms.html for a pre-installed single-node.

Step 0 (optional): Install Linux VM, if you are using Windows machine

If you are using a Windows machine, you have the option of installing a Linux VM, because much of the documentation for spark online is for a unix environment. You could use VirtualBox to install a Linux image. Download VirtualBox and install it to your device, then install Ubuntu or other Linux system if you prefer within VirtualBox.

Step 1: Check for Java installation

Java installation is one of the mandatory things in installing Spark. Try the following command in your Terminal/Command Prompt to first verify the JAVA version.

```
$java -version
```

If Java is already installed on your system, you get to see the following response −

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

If you do not have java installed in your system, you will get the following msg

```
No Java runtime present, requesting install.
```

In that case, you need to first download the JDK from Oracle's download site:
https://www.oracle.com/technetwork/java/javase/downloads/index.html

For macOS:
```
brew install java
```

Note: It is preferred to use versions later than **JDK 8** for Spark. Also make sure the PATH variable is set rightly to JDK.

Step 2: Python installation verification

Make sure you have installed python in your system

Try the command in your Termina/Command Prompt.

```
Python --version
```

to verify installation. If you do not have python, then download and install python from the following link (we recommend Python 3+)
https://www.anaconda.com/distribution/#download-section

If you choose to use R or Java/Scala instead of python, you will find resources online, but the TA staff may not be able to help you if you run into issues. We highly recommend python as both the language itself and the python Spark API are straightforward.

Step 3: Download and install Spark 3.0 on your machine:

https://dlcdn.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz

Unpack the compressed TAR ball, either by using the following command, or double-click on the package.

```
$ tar xvf spark-3.5.0-bin-hadoop3.tgz
```

OS X

Homebrew users can directly use `brew install apache-spark` to install without downloading the package above.

Linux/OS X

Add the following line to ~/.bashrc, ~/.bash_profile (for old OSX), or ~/.zshrc (newest OS X) file. It means adding the location, where the spark software file (the folder you just unpacked) are located to the PATH variable.

```
export PATH=$PATH:path/to/spark/bin
```

Now if you run

```
spark-shell
```

You will find an output similar to this (version numbers might differ)

```
Spark assembly has been built with Hive, including Datanucleus jars on
classpath
Using Spark's default log4j profile:
org/apache/spark/log4j-defaults.properties
15/06/04 15:25:22 INFO SecurityManager: Changing view acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: Changing modify acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: SecurityManager: authentication
disabled;
   ui acls disabled; users with view permissions: Set(hadoop); users with
modify permissions: Set(hadoop)
15/06/04 15:25:22 INFO HttpServer: Starting HTTP Server
15/06/04 15:25:23 INFO Utils: Successfully started service 'HTTP class
server' on port 43292.
Welcome to
      ____              __
```

```
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.4.0
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java
1.7.0_71)
Type in expressions to have them evaluated.
Spark context available as sc
scala>
```

Alternatively, to start a spark shell for python, run

```
pyspark
```

Windows

Use winzip to unzip the spark file you downloaded, and place the unzipped directory into a “spark” folder of your choosing. From a command prompt, navigate into the spark folder you just created and the unzipped spark directory. The type:

```
bin\spark-shell
```

The spark shell should start (see the section for Mac OS X/Linux for the output).

Alternatively, to start a spark shell for python, navigate to the same directory and type:

```
bin\pyspark
```

If you want to add the bin directory to your path (so you do not have to navigate to the bin folder each time), follow these instructions.

Writing your first spark programs

Step 4: Word count

As discussed in class, the typical “Hello, world!” app for Spark applications is known as word count. The map/reduce model is particularly well suited to applications like counting words in a document. In this section, you will see how to develop a word count application in python.

Write the following program in an editor and save the file as `wordcount.py.`

The first step of every Spark application is to create a Spark context:

```
import sys

from pyspark import SparkContext, SparkConf

conf = SparkConf()
sc = SparkContext(conf=conf)
```

Read data from text file and split each line into words

```
words = sc.textFile(sys.argv[1]).flatMap(lambda line: line.split(" "))
```

Now count the occurrence of each word

```
wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a
+b)
```

Finally save the output to another text file

```
wordCounts.coalesce(1, shuffle=True).saveAsTextFile(sys.argv[2])
sc.stop()
```

To run this application, do the following:

1) You can find a copy of the wikidump here. Download it to your computer
2) Run

```
spark-submit path/to/wordcount.py path/to/wiki.txt
path/to/output_directory
```

If you are running in a folder which contains both wordcount.py and wiki.txt. Run the command below

```
 spark-submit ./wordcount.py ./wiki.txt ./output
```

- `Make sure there are no output folder created before running the command above.`

Step 5: Bigrams

Now you will write your first Spark job to accomplish the following task:

Write a Spark application which outputs the bigram model for the document. A bigram or digram of words is a sequence of two adjacent words like “turn your”, or ”your homework” in a document. A bigram model is a statistical language model that assigns probabilities to the sequences of words. In other words we are trying to find the probability of a word based only on its previous word.

Bigram = $P(w_n|w_{n-1}) = count(w_{n-1}\ w_n) / count(w_{n-1})$

Example: Consider a document with text

San Diego has nice weather. It is raining in San Francisco.

Here the bigram (san diego) = count(san diego)/count(san) = ½ = 0.5

According to this example document if a word san occurs, then there is a 50% chance that the next word is diego. These kinds of statistical models are used frequently in NLP applications.

Implementation

You need to write a Spark job that can compute the **conditional** bigram distribution of the wiki.txt document. Your output should list all the bigrams and their corresponding conditional frequency distributions. In your implementation, you can ignore the letter case, i.e., consider all words as lower case. You can also ignore all non-alphabetic characters.

What to hand-in

Zip together the following files and submit it in GradeScope

a) A README.md file explaining how to run your code, where we can find the output files to step 4 and 5(your google drive link), and how to interpret your output for step 5,
b) The python code for the program in step 5 (step 4 optional, since all given)

Put below files in a google drive folder, and put the link to the folder in the Readme File. Make sure to make the folder available to whoever has the link. Do not modify the folder after the deadline. We will match the result from the code submitted and the output in the folder.

c) Output files for the word count program in step 4
d) Output files containing the count of all bigram words obtained from step 5
e) Output files with the conditional bigram frequency distribution obtained from step 5