# Quiz 5: Using Hadoop for TF-IDF Analysis

[120 points]

## 

*Quiz 5 is about finding the important words in a collection of documents.*

## *Introduction*

[TF.IDF](https://en.wikipedia.org/wiki/Tf%E2%80%93idf), short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. Its usefulness for this purpose is well-established.

The steps in this quiz walk you through a methodology for developing big-data code. We begin with a trivial problem where its solution and the answer are known. Then, replace the various components one at a time so we can easily verify the correctness of each change. In the example below, we begin with a trivial input file, a trivial mapper and a trivial reducer in a shell environment and, through a series of steps, end up with a more sophisticated reducer running against bigger input in the Hadoop environment, verifying the code at each incremental step.

We will be creating a map-reduce version of TF.IDF so it can be used for large corpora. We will do this in four steps:

1. [10 points] Begin In the linux environment or your laptop, (not on Hadoop just yet). Download [Hadoop Streaming code](https://github.com/singhj/big-data-repo) based on Michael Noll’s map-reduce tutorial. Examine the file input.txt and confirm that the result from running the provided mapper.py and reducer.py is correct[[1]](#footnote-0).
2. [50 points] Using the provided [Jupyter notebook](https://colab.research.google.com/drive/1WfVK4zWnGrHwqltHH15ug9Knoa_yHk2s?usp=sharing)[[2]](#footnote-1) as a reference, in the Linux environment, code a new implementation of reducer.py to calculate TF.IDF values. The mapper.py code does not need to be modified — notice that it produces word counts for each document. Confirm that the result from running with your new reducer produces the same result as the reference notebook[[3]](#footnote-2).
3. [20 points] No new python code in this step, run the code from step 1 on a Hadoop cluster[[4]](#footnote-3) with input gs://jsingh-bigdata-public/abc.zip. It should produce the same result as step 1. Then run the code from step 2 on a Hadoop cluster[[5]](#footnote-4)[[6]](#footnote-5) with the same input. It should produce the same result as step 1.
4. [40 points] Finally, find the TF.IDF values of all the words in the given corpus of the last 10 president’s inaugural speeches (Ronald Reagan 1981 to Donald Trump 2017). The corpus is available at gs://jsingh-bigdata-public/ten\_speeches.zip. The net result should look like this table:

| **…** | **Speech i** | **Speech i+1** | **…** |
| --- | --- | --- | --- |
|  | word(i)1, TF-IDF(i)1,  word(i)2, TF-IDF(i)2,  word(i)3, TF-IDF(i)3,  :::  word(i)20, TF-IDF(i)20, | word(i+1)1, TF-IDF(i+1)1,  word(i+1)2, TF-IDF(i+1)2,  word(i+1)3, TF-IDF(i+1)3,  :::  word(i+1)20, TF-IDF(i+1)20, |  |

Each cell should show the 20 words with the highest TF-IDF values in that speech.

1. The command will be python mapper.py < input.txt | sort | python reducer.py [↑](#footnote-ref-0)
2. The Jupyter notebook has an implementation of the TF.IDF algorithm set up to work with the same input as #1. Providing it to you is designed to accomplish two things:

   * 1. Show the inner workings of the TF.IDF algorithm and
     2. Show that the results match those from running SciKit Learn-based TF.IDF analysis and thus validates our implementation of TF.IDF.

   [↑](#footnote-ref-1)
3. The command will be python mapper.py < input.txt | sort | python your\_new\_reducer.py [↑](#footnote-ref-2)
4. The commands will be

   hadoop fs -mkdir /user/inputs/

   hadoop fs -mkdir /user/inputs/abc

   gsutil cp gs://jsingh-bigdata-public/abc.zip .

   unzip abc.zip -d abc

   hadoop fs -put -f abc/\* /user/inputs/abc

   hadoop jar /usr/lib/hadoop/hadoop-streaming.jar -files mapper.py,reducer.py -mapper mapper.py \

   -reducer reducer.py -numReduceTasks 1 -input /user/inputs/abc -output /user/j\_singh/count\_abc [↑](#footnote-ref-3)
5. The commands will be

   hadoop jar /usr/lib/hadoop/hadoop-streaming.jar -files mapper.py,your\_new\_reducer.py \

   -mapper mapper.py -reducer your\_new\_reducer.py -numReduceTasks 1 -input /user/inputs/abc \

   -output /user/j\_singh/tfidf\_abc [↑](#footnote-ref-4)
6. Initially the output had been named /user/j\_singh/count\_abc. But since we are running the new reducer, it makes sense to name the output differently, /user/j\_singh/tfidf\_abc. [↑](#footnote-ref-5)