CprE 419 Lab 4: Clustering Similar Documents

Department of Electrical and Computer Engineering
Iowa State University
Spring 2013

Purpose

In this lab, you will use the Hadoop MapReduce framework for clustering (i.e. grouping together) similar documents, which is a common problem in document analysis. For instance, a web search engine needs to cluster search results so that it will not return multiple webpages that are nearly identical as separate search results. At the end of this lab you will know how to:

- Effectively look through a large collection of documents
- Estimate the Jaccard similarity between documents using minhashing
- Find and group near-duplicate documents

Submission

Create a zip (or tar) archive with the following and hand it in through blackboard**.**

- The resulting output from your program.
- Commented Code for your program. Include all source files needed for compilation.
- A description of the algorithm you used.

Background

Hashing is very useful in determining if two documents are exact duplicates. If the hash values of the documents are unequal, then the documents are definitely not duplicates, and if the hash values are equal, then it is likely the documents are equal – in this case, an additional, more expensive byte-by-byte comparison of the two documents will be needed to confirm that they are equal. Now, suppose that we have two documents that are largely similar but not exactly equal. If you take these two documents and hash them their hash values would likely not match. This property of hash functions makes them great for determining exact duplicates but not very useful for finding near duplicates.

We will now outline a similar tool, based on hashing, for determining if two documents are near duplicates. This is sometimes called as “locality sensitive hashing”. In essence, the probability that two documents have the same hash outputs is equal to their degree of resemblance. If two documents are nearly identical, they are very likely to have the same output. If they are very different, they are unlikely to have the same hash output.

We first introduce a notion of what it means for two documents to be similar. As we have described in class, a document can be viewed as a (long) sequence of bytes. Computing and dealing with metrics among long sequences can be rather expensive, hence we consider a set-based representation of documents based on “shingling”.

For a small positive integer k, the set of k-shingles of a document D is the set of all subsequences of length k that have appeared in D. For instance, the set of all 3-shingles of the

document “hdDUsd5k9es” is {hdD, dDU, DUs, Usd, sd5, d5k, 5k9, k9e, 9es}. The parameter “k” cannot be too small, since this will lead to loss of the document’s structure. It should not be too large either, since that will blow up the size of the representation. We recommend a value k=9.

We now ask ourselves the question, what do we know to be true about the content of documents that are similar? In order for two documents to be similar, by our definition, the content of the two files are nearly duplicate. Hence, the set of k-shingles are also nearly identical. The converse is also mostly true, in that if the set of k-shingles are nearly identical for two documents, the documents are also nearly identical – note that there may be some pathological cases when this may not be true.

We will now introduce a measure of similarity between two sets, called the Jaccard similarity. The Jaccard similarity between two sets A and B is defined as the ratio between the size of their intersection divided by the size of their union, i.e. given by the following formula.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

If two documents are similar, then the Jaccard similarity between their k-shingles will be close to 1, otherwise the similarity will be close to 0.

We now introduce the next concept, known as *minhashing*. Suppose we have a hash function, $H()$, that will take a shingle from our sets as input and generates an integer output. Let's also define $H_{min}(A)$ to be the minimum of the returned hash values for the set A. The minimum values of the two sets will be equal when the minimum value of the union is also within the intersection of the sets.

$$[H_{min}(A) = H_{min}(B)] \; when \; H_{min}(A \cup B) \in (A \cap B)$$

In other words, the probability of the minimum hash values in set A and in set B matching is equivalent to the Jaccard similarity of the sets.

$$Prob[H_{min}(A) = H_{min}(B)] = J(A,B)$$

This is saying is that if two documents are similar then the minhash of the two sets of shingles are likely to be equal, since the sets of shingles have a high Jaccard similarity. This fact will be used profitably in this lab.

Suppose that we have been asked to find sets of similar documents, given a collection of documents. The above theory suggests the following approach:

- Represent each document as a set of its k-shingles.
- Find the minhash of all shingles in the document
- Only inspect documents that have the same minhash for further consideration

This avoids the problem of checking all pairs of documents to check for approximate similarity, and can substantially reduce the time taken for clustering.

Experiment 1 (80 pts)

You will find a single HDFS file at "/datasets/Lab4/group-files." Inside of this file is a large collection of documents. Different documents are split by a newline character and thus are on their own separate line (a document does not itself contain a newline character or a punctuation). For this lab, you will not need to concern yourself with parsing and removing punctuation characters or spaces or tabs from the input documents.

Each line has the following format. There is a document ID at the beginning of the line, which is an integer, followed by a '-', and then followed by the text of the document. Thus, if I have a document with an associated Document ID of 400 and the content of "hdDUsd5k9e" the entry in the file will be as follows:

400-hdDUsd5k9e\n

Within the input document there are approximately 1000 different groups. Each group has approximately 100 documents all of which are near duplicates and can be all considered close to each other. But these documents appear in no particular order, and you cannot expect similar documents to be close to each other in the input file.

Your task is to write a MapReduce program that will produce a list of Document ID's for the documents that belong within a group. Along with this list, the program should also output the content from **one** of those documents. For example, say we have a group of similar documents with the document numbers of 45, 789, 2157, 4500, and 5000. Suppose the content of one of these documents is "hdDUsd5k9e". The output of program should look similar to the line below for that group.

789,45,4500,2157,5000 hdDUsd5k9e

In other words, the key is a comma separated list of the document ID's and the value is the content from one of the documents. Your final output should contain a list of all the groups with one line per group.

You should output the lists of the grouped documents to the directory "/user/<User ID>/Lab4/exp1"