Data Mining

Homework 5

Due: 5/7/2015, 23:59.

Instructions

You must hand in the homeworks electronically and before the due date and time.

Handing in: You must hand in the homeworks by the due date and time by an email to the instructor that will contain as attachment (not links!) a .zip or .tar.gz file with all your answers and subject
`[Data Mining class] Homework #`
where `#` is the homework number. After you submit, you will receive an acknowledgement email that your homework has been received and at what date and time. If you have not received an acknowledgement email within 1 day after you submit then contact the instructor.

The solutions for the theoretical exercises must contain your answers either typed up or hand written clearly and scanned.

The solutions for the programming assignments must contain the source code, instructions to run it, and the output generated (to the screen or to files).

For information about collaboration, and about being late check the web page.

Most of the questions are not very hard but require time and thought. **You are advised to start as early as possible, to work in groups, and to ask the instructor in case of questions.**

Problem 1. Here we ask you to implement Problem 4 of Homework 1, in Hadoop. In particular, write a MapReduce program, executable in Hadoop, to find the top-10 beers with the highest average overall score among the beers that have had at least 100 reviews. Note two points:

- You may preprocess the file so that you have one line per beer so that afterwards you can process it easily with MapReduce.
- You may need more than one rounds of MapReduce to count and to obtain the top-10 beers.

Problem 2. In this problem we will see how we can download data using an API, and practice some of the text processing steps and MapReduce. The goal is to create an inverted index for the abstracts of some articles from the New York Times newspaper.

The NY Times API is available at `http://developer.nytimes.com`. It provides access to various articles, both historic and new. For this assignment we are interested in the *Times Newswire API*, which provides access to articles, blogs, and so on, as they are being produced, and for each article we can obtain directly from the API its URL and its abstract, among other information. Your tasks for this problem are the following:

1. Study the API, download 30,000 *different* articles from the Times Newswire API, and for each of them save the URL and the abstract. Note that because articles are created continually, you may end up downloading some articles multiple times; you should make sure that you do not store each article more than once. For each article assign a unique docID. Note that this part is not completely trivial because, among other issues, you need to deal with the fact that

often the API does not return the expected document, so you need to catch the exceptions thrown, put the right delays, and retry, for some steps.

2. Perform some preprocessing of the article abstracts. In particular, remove all punctuation and numbers, and keep only the words. Convert each word to lowercase, remove stopwords, and stem each word using Porter's stemmer. For this part you may find useful the `NLTK` Python package.

3. After preprocessing the articles, build an inverted index using Hadoop. (This is a small dataset and we could build the inverted index in memory but we want to practice.) The final outcome should be a file in which each line is of the form:

$$\texttt{term}_\texttt{j}\texttt{:doc}_\texttt{1j}\texttt{,tfidf}_\texttt{1j}\texttt{␣doc}_\texttt{2j}\texttt{,tfidf}_\texttt{2j}\ldots$$

   where ␣ is just a space character, $\texttt{term}_\texttt{j}$ is the jth term (alphabetically), $\texttt{doc}_\texttt{ij}$ is the ith article in the posting list of $\texttt{term}_\texttt{j}$ (sorted by docID), and $\texttt{tfidf}_\texttt{ij}$ is the TFIDF score of $\texttt{term}_\texttt{j}$ in article $\texttt{doc}_\texttt{ij}$. You can process the file before and after the *inversion*, for instance, to put each line in the desired format, but the actual work in which the algorithm calculates frequencies and TFIDF scores and builds the inverted index should be done in Hadoop. Note that one map–reduce round is enough.

Problem 3. In this assignment we will implement the k-means algorithm in Hadoop. The specifications of the program are to accept a file where each line corresponds to a document and a value k of the number of clusters, then select k random initial centers, run the k-means algorithm, and return a file where each line corresponds to a cluster.

In more detail, the format of the input file is:

$$\texttt{fileID}_\texttt{i}\backslash\texttt{t term}_\texttt{i1}\texttt{:tfidf}_\texttt{i1}\texttt{,term}_\texttt{i2}\texttt{:tfidf}_\texttt{i2}\ldots$$

where $\texttt{fileID}_\texttt{i}$ is an ID (e.g., the filename) of document i, $\texttt{term}_\texttt{ij}$ the jth term of the ith document, and $\texttt{tfidf}_\texttt{ij}$ the TFIDF value of the jth term in the ith document, after ℓ_2 normalization so that for every i we have that

$$\sum_{j=1}^{n}(\texttt{tfidf}_\texttt{ij})^2 = 1.$$

The main idea of the k-means algorithm in Hadoop should be almost obvious, and it is as follows:

1. First we select the k random centers using MapReduce. To do that we can assign a random number to each document and we select the top k.

2. The main part of the algorithm consists of several map–reduce rounds. In the map step, every document must select its closest center. In the reduce rounds, for every cluster we have a list of the documents assigned to it and we recompute the new center.

3. After convergence, for each cluster we report the documents assigned to it. We can also do this with MapReduce.

The various components of the algorithm are not too hard but you will need to combine them carefully. In particular, you will need three different MapReduce classes (one for choosing the random centers, one to perform an iteration of k-means, and one to report the final clusters) and a main program that controls them.

After you write and test your program, try to cluster some of the books of the Gutenberg project (`http://www.gutenberg.org`). Download the August 2003 CD:

`http://www.gutenberg.org/wiki/Gutenberg:The_CD_and_DVD_Project`

The goal is to cluster all the `.txt` files in the CD. First preprocess the document. Create a single file and for each `.txt` file in the CD:

1. Remove the project Gutenberg header.
2. Remove stopwords, convert to lowercase and stem.
3. Store them in a single file in the format described above.

After you prepare the file, cluster it with $k = 10$, 20, and 50. Try multiple times. Does it end up with the same cluster? Can you suggest a way to combine the different clusterings?

Extra credit: Build an inverted index and/or cluster the Gutenberg DVD.