CS294-1 Behavioral Data Mining Spring 2012

Programming Assignment 3: Clustering via Hadoop

Due: Thursday April 19

The goal of this assignment is to process a moderately large dataset (a recent snapshot of Wikipedia) in distributed fashion using Hadoop. The data are available on the icluster, and accessible in the Hadoop File System at

/user/cs294-1/wikipedia/wikipedia.xml

This is a single large text file (about 37 GB). You should write a mapreduce job or jobs (most likely you will need several mapreduce steps) that tokenizes each article and extracts useful metadata. The output should include:

- A numerical sparse matrix which is a bag-of-features representation of the document. e.g. element Mij should be the count of feature i in document j. Features would typically be individual words, bigrams or trigrams.
- A dictionary mapping integer feature IDs to features (as strings) – in this case the features are words, bigrams and possibly trigrams although you may choose to store only the text of words, treating n-grams as sequences of their term IDs. As in the previous HW, the dictionary should have the integer feature ID as a string as well as the value of the feature, and a word count. The dictionary entries should be sorted in decreasing order of frequency, with the IDs indicating the term's position in the rank ordering. i.e. term 1 should be the most frequent term, and the first term appearing in the dictionary file.
- The category(ies) of the document in wikipedia's hierarchy. Category info is not available in the XML structure of the documents, you have to extract them from the text fields.

You can add other features to the representation, especially if you think they would be useful for clustering the data. You may find it useful to compute a word-ordered representation of the data before the bag-of-features. This is similar to the "tokenized" file from the last HW. Data in this rep are represented as triples of document, word position and word ID.
Wikipedia has quite a lot of syntax in article text – there is both free text and mediawiki markup. In order to extract the meaningful text (and ignore most markup) we recommend you use the Wikipedia tokenizer from the lucene search engine which is available from here:
http://lucene.apache.org/core/old_versioned_docs/versions/3_5_0/api/contrib-analyzers/index.html
This tokenizer can also extract the Wikipedia category information, which is embedded in the article text. The tokenizer will output individual words, which you can then lookup in a dictionary to assign iDs.

Each map-reduce job produces a number of output files equal to the number of reducers. They will be sorted by an order you specify, but only within each reducer. If there are k reducers used in a job, the output will be k file parts, and keys will be sorted only within each of these files. Dictionaries, which have a unique total ordering, therefore have to be computed by jobs that have a single reducer. The main

data files, those containing the bag-of-features data, are very large and are best broken into several files. It's natural to use a number of reducers equal to the number of partitions you want.

There are a few ways to produce this kind of output (which is a *normalized* version of the dataset in the sense of DB normalization). The simplest goes like this:

1. Tokenize the input text in a mapper which outputs a pair (string, integer) = (word, 1) for each word in the document. The reducer sums all counts for the same word. This is exactly the WordCount sample app from the Hadoop docs. This job can use many reducers, and will be faster if it does. Because of power law effects though (remember lecture 9) this job will be quite unbalanced unless you use a combiner.
2. The next job takes the (word, count) key-value pairs from step 1 and reverses their roles to sort in descending order by count. It must use a single reducer for this so all output goes to the same file. This reducer should assign integer IDs in descending order. The result of this job should be the final dictionary representation.
3. The third job should load the dictionary from step 2 into memory on each mapper as a hash table mapping word text to ID. Then it should map over the input XML file to re-tokenize the input, and map each word to its integer ID. If bigrams are to be used, they can be computed on-the-fly by this job. The output value should be a sparse vector of features for each article. The output key can be a string representation of the article ID. You should use multiple reducers for this step so that output files are partitioned and of reasonable size. Category information can be computed and output as a separate file with the same (article ID) key and a value which is a string representation of the category.

It may be necessary to truncate the dictionary to make sure it will fit in memory. Normally this has no effect on clustering when it is at least 10's of thousands of terms.

Given a sparse matrix representation of the data, you should then compute a topical clustering of the articles using an algorithm of your choice. It could be k-means or spectral clustering, or one of the methods described in lecture 21, or any other method that you choose yourself. The goal is to produce a clustering that is roughly similar to Wikipedia's topical clustering system. For that reason, you may want to use hierarchical clustering (which just means you would cluster top-down or bottom-up at several levels, using any clustering method you want).

You can choose whether to run your clustering algorithm in the icluster or outside (e.g. in Matlab). The differences in performance on numerical data between Hadoop and Matlab are such that it may well run faster on a single machine in Matlab. If you do want to cluster in Hadoop, we suggest you try using the BIDMat library which is downloadable from here:

http://bid.berkeley.edu/BIDMat/index.php/

BIDMat provides a Matlab-like interface that runs natively in Scala while using the same native acceleration as Matlab. You can run it standalone to design and debug a program before running it in the cluster.

Since Wikipedia has a topical classification system, you can use it as a reference for unsupervised clustering. This is a very rough way of evaluating a clustering algorithm but at least gives some idea of how well it is doing (and can be used to compare different clusterings). Each document will have a category assignment Bi from its Wikipedia categorization and an assignment Ci from the clustering. To determine the score for a clustering, compute the edge weights Wij as the count of all documents labeled with Bi and Cj. Then treating Wij as a weighted bipartite graph on vertices in B and C, find the max-weight matching between B and C. The total weight of this matching can be referenced to a perfect matching by dividing by the total number of documents (i.e. the number that would be matched if the clustering and classification were the same).

Writeup

Please describe the sequence of steps of your Hadoop job, and the subsequent clustering. There should be enough detail to allow someone else to reproduce the results. Include tables or graphs showing the performance of your clustering. Include a few strongest terms from several of the largest clusters. Hand in a hardcopy and send a zip file with the code and graph data to kenghao@cs.berkeley.edu by Thursday April 19.