Overview
Our goal for this assignment was to recreate Wikipedia’s article groupings into semantic categories by means of clustering. Given an input of Wikipedia’s XML dump, we designed a pipeline of MapReduce jobs aimed at clustering the articles. We then use the categorical groupings of Wikipedia to create a ground truth for clusters, and measure how well our clusters align to the ground truth. At this time we have implemented all stages of this pipeline and have working code that we include with our submission. Unfortunately, we were not able to run our scripts to completion and do not include final results due to challenges with cluster availability.

Procedure
Our procedure consisted of three major steps, which will be described in the following section.

1. Build the bag-of-words term-document matrix from the full dataset.
2. Run k-means over the matrix until it converges.
3. Compare the computed clusters against those that Wikipedia provides with its categories.

Bag of Words Representation of Wikipedia
In order to study the emerging patterns in Wikipedia articles, we used a bag of words representation to model the articles. In particular, we computed a sparse feature matrix where each row corresponds to a document, and each column to a word in the Wikipedia lexicon. For each word in the lexicon, we record the number of times it appears in a document, and use this vector representation to compute the cosine similarity between documents.

We obtained the sparse feature matrix from a series of MapReduce jobs run using a Hadoop installation on the Berkeley EECS iCluster. We now describe the steps taken in this computation.

A count of the number of appearances of all words in the lexicon
Wikipedia articles consist of several sections and include both relevant text and technical wiki markup language. Since our goal in this experiment is to discover patterns in the data, it was necessary to model the documents using semantically meaningful features only. We therefore concentrated on the <title> and <text> sections, which carry the most information regarding the topic of a document. Furthermore, within the <text> section, we only collected all English-only tokens, including category information and links to other pages, that are important for establishing topical similarity between articles. We stemmed all words using the Porter Stemmer [1] and computed the number of appearances of each stemmed word in the entire dataset.

This step was split into two Hadoop jobs. The first job used 551 mappers, combiners and ten reducers to create separate files of word counts in 33 minutes and 38 seconds. The second job
used an identity mapper and a single reducer to combine all ten files into one list of all words in
the lexicon sorted in alphabetical order.

Full lexicon dictionary
Next we used the list of words to create a full lexicon dictionary, or a map from each word to its
unique id and number of appearances in the dataset. To this end we used a single Hadoop job
where the map function flipped the roles of the keys and values in order to sort the words by
number of appearances, and a single reducer to assign a unique id to each word. This job ran
for 1 minute and 3 seconds, and produced a dictionary we could then examine by hand.

Looking at the dictionary, it was easy to identify the power law distribution of the word
appearances in the whole dataset. While few words appeared very often (the most common
word being “the” with 146,273,376 appearances), around forty percent of the words appeared
only once. Since we are interested in similarities between documents, words that appear only
once do not carry significant signal and can be discarded. Following the same argument,
we took the liberty to discard words that appear only two or three times. Similarly, words
that appear very often cannot be used to distinguish between documents or to signify their
similarity, as statistically we would expect them to be uniformly distributed across the corpus.
We therefore discarded the 88 most common words in the corpus, stopping at “football”, the first
word that seemed to carry interesting topical signal.

Sparse feature matrix
In order to parse the Wikipedia corpus in an efficient manner using Hadoop, we used the
Mahout implementation of an XmlInputFormat [http://bit.ly/HKmxtH]. This allowed us to ensure
that whole pages would be mapped using a single mapper. Each mapper loaded the dictionary
of words to unique feature ids into memory, and emitted a sparse feature vector representing a
given document of the following string form. Note that featureId corresponds to the unique word
id encoded in the dictionary, and the count is the number of appearances of that word in the
current document.

    DocumentId <featureId,count> <featureId,count> <featureId,count>...

K-means Clustering
After determining the appropriate cluster labels, we used a distributed k-means algorithm to
iteratively cluster the documents. The general algorithm, implemented in Hadoop, is as follows:

1. Each mapper task receives a single entry from the term-document matrix. Store the
   most up-to-date centroids in a file on HDFS and have all mappers read the file and
determine which centroid the target document was closest to. The mapper then emits
   a key-value pair that consists of the cluster ID (the key) and the document vector (the
   value).
2. The combiner distributes these entries to reducers according to their cluster ID’s using
   the default combiner.
3. Each reducer receives all records for a particular cluster and computes the new centroid,
   measures the intracluster distance between each document as a quality measure, and
   outputs a series of tuples containing <cluster_id, word_id> pairs to HDFS.
4. Determine whether another iteration is needed by determining whether any documents
   switched clusters from the previous iteration. This is performed by running set
   comparisons in a single-node Python script. If the script determines that any nodes
   changed their classification with the latest iteration, repeat all steps. Otherwise the
   clusters have converged.
We seeded our clusters with randomly selected documents as centroids. We did not have time to run the k-means algorithm to convergence but optimized our job to run an iteration across the full data set in approximately 25 minutes with all 20 mapper nodes. The post-processing script then merges all of the iteration’s <cluster_id, doc_id> tuples into a single file and checks for convergence. Our execution scripts were capable of automatically repeating this process until the clustering converged but we did not have time to carry out the necessary iterations.

Generating ground truth clusters

Extracting leaf clusters

The <text> nodes of Wikipedia nodes often contain hand-annotated semantic category information marked as [[Category: XX]]. These categories are organized in hierarchies as a set of overlapping trees as is demonstrated in the image below taken from Wikipedia:Categorization.

![Wikipedia Categorization Diagram](http://bit.ly/HX9oNT)

According to Wikipedia’s general conventions, each article can be annotated with more than one category, all of which are taken from the lowest level possible of the corresponding hierarchy tree. Practically, this means that the categories found on the page itself are low level categories that potentially map into a smaller set of high level categories that we could use to group Wikipedia’s articles into clusters.

As a first step, we collected these low level categories by parsing the Wikipedia xml file. To this end we ran a Map Reduce job using the XmlInputFormat in order to map one page at a time. For each page we collected all categories and emitted a key-value pair where the key is the pageId and the value is the list of categories on that page. For this job we used an identity reducer only in order to collect the output of all mappers into one file.

Top-level category mapping

We used the dataset provided on Piazza to cluster our documents into twenty two high-level categories according to the hierarchy that Wikipedia has defined [http://bit.ly/HX9oNT]. We took the mappings from Piazza and translated each of the extracted categories with their top-level equivalent to generate a vector of all categories that a document was contained within. These categories were then used in our maximum weight matching algorithm as described below.

Cluster similarity

We also wrote a single-node script that could compare the results from our clustering algorithm with the results from the Wikipedia category labeler to pair clusters together using maximum weight matching. The script currently compares the set of documents in each of our automatically extracted clusters with each of the sets in the Wikipedia categories and finds the pairs that have the highest normalized overlap. If a document exists in more than one category, it is included in each of the sets in the matching algorithm in order to avoid selection bias.
The script loads each of our categories separately in order to reduce the memory footprint at the cost of suboptimal performance. This task could have been performed using a MapReduce job but it didn’t seem particularly necessary given the simplicity of scripting the solution and tolerable single-node runtime (assuming that our simulated trials were accurate representations of the runtime of the actual dataset).

**Potential Improvements**

There are a number of approaches that we think would improve the clustering performance but did not have time to implement.

*Other representations of the Wikipedia documents.* While we modelled the Wikipedia documents using a bag of words model, one could conceivably make use of n-grams in the representation, an approach that may capture more of the semantics of the articles than a unigram representation.

*Seed clusters with Wikipedia centroids.* Our approach used centroids that were randomly chosen from the set of all documents. This was simple but not particularly effective at reproducing the topical clusters that occur in Wikipedia naturally. There is little reason to believe that arbitrary centroids would converge to the particular topics that Wikipedia authors isolate, and it is probable that finding centroids according to Wikipedia’s category tags and seeding our system with those would have been significantly more effective.

All of the information that would be required for this technique was available in the original XML dump; it would essentially require a way to label the articles with the appropriate categories (which we can do). Clusters can be selected based on the category tags that are embedded within articles, and the centroids of these articles could be found and used as the seeds for the full clustering procedure.

*Vector distance measures.* We used a normalized Euclidean distance measure because it was straightforward to implement and seemed like a nice starting point. We would have liked to have tested cosine similarity and document length normalization as well and could have done so relatively easily, but would have required another full set of k-means iterations.

**References**