In this assignment, I wrote up two MapReduce jobs to do word counts and k-means clustering on this humongous Wikipedia dataset.

**Word count** In word counts, each mapper takes a document as input and breaks it into words. It then emits a key/value pair of the word. Then, each reducer sums the counts for each word and emits a single key/value with the word and sum. The reducer is also used as a combiner on the map outputs. This reduces the amount of data sent across the network by combining each word into a single record.

**n-gram** n-grams are sequences of n consecutive words. I modified the (unigram) word count code to count unique bigrams and trigrams, too. The mapper maps each bigram/trigram to one term by gluing them, and then the reducers add all the ones corresponding to each bigram/trigram. This is analogous to the method for unigram counting.

**Truncation due to power law** Due to the power law of the textual data, I truncate the terms which were counted less than ten times and have been able to eliminate 40% of the terms.

**Sparse matrix** After we obtain a complete list of unique unigram, bigram, trigram, and their counts, we use these unique words as word indices and call another MapReduce job to convert words to indices for each document. This allows us to form a sparse document-word matrix $X$ for clustering.

**TF-IDF rescaling** We normalize each row of $X$ to be a unit vector to prevent bias toward longer documents where higher counts do not necessarily reflect importance of that term in the document. Thus we obtain the term frequency $\text{tf}(d_i, t_j)$. To eliminate spurious features that are common across many documents such as “the” and “to”, the inverse document frequency is introduced as

$$\text{idf}(D, t_j) = \log \frac{|D|}{1 + |\{d_i \in D : t_j \in d\}|}$$

with $|\{d_i \in D : t_j \in d\}|$ the number of documents which contain term $t_j$, i.e., $\text{tf}(d_i, t_j) \neq 0$ or $x_{i,j} \neq 0$. The one added in the denominator prevents a division-by-zero in case the term is not in the corpus. Finally, $X_{\text{tf-idf}} = \{x'_{i,j}\}$ where

$$x'_{i,j} = \text{tf}(d_i, t_j) \times \text{idf}(D, t_j).$$

**Clustering for the terms** While it seems challenging to evaluate the quality of clustering of Wikipedia documents, it is however more intuitive to cluster for terms and to visualize the term associations where are closer to the centroid of each cluster.

**k-means** I start by taking a small random sample set from the input data set and do a hierarchical clustering within this smaller set. The small-random-sample approach is because hierarchical clustering is not-scaling to large data set. Then, each iteration is implemented
as a MapReduce job. For one, we need a control program on the client side to initialize the centroid positions, kickoff the iteration of MapReduce jobs and determine whether the iteration should end. Within each iteration, most of the processing will be done in the Map task, which determine the membership for each point, as well as compute a partial sum of each member points of each cluster. The reducer did the easy job by aggregating all partial sums and compute the update centroid position, and then out them into a shared store that can be picked up by the MapReduce job of next round.

Figure 1: The terms associated to the centroid of cluster “technology”.
Figure 2: The terms associated to the centroid of cluster "society".