Multi Document Text Summarization Using LSI and SVD

Project Report

Submitted By:
Kushagra Pandey
kp2240

Motivation: Text summarization is a problem that is being very actively worked on and one of the more recent techniques is using Latent Semantic Indexing or LSI. So what I basically set out to do was to use this technique to summarize text from the DUC corpora. I had initially intended to evaluate my system using the rouge tool for comparison with other automatically generated data but I couldn't get the rouge system to work. So instead, I tried to see how varying the parameters such as the desired compression and the number of features affect the text generated. This is totally qualitative but still gives some insights into the process.

Tools/Data: For this project I made use of the Clairlib library and Matlab. The data I used was the DUC corpus, year 2006. Basically I tested on folders D0651F, D0626H. The former was about the impeachment trial of Bill Clinton and the other on the bombing of US embassy in tanzania.

System Description: My program is basically divided into 3 parts. A parser, a sentence clustering engine and a summarizer.
Parser: the parsing of the data is done through a perl script titled stop_test_final11.pl . The basic job is to remove stop words, stem the words and ultimately create a matrix of words*sentences. The basic data structure used is an array `sentences', which stores the sentence number, file name and the sentence boundary in the original file. It also uses a hash `uniqwords' which serves as the vocabulary and a temporary array called `pos' to store the position of each sentence. It writes two files, sentences11.txt, which is the main index and test.txt, which is just the vocabulary. It also builds the sentence*word matrix and stores it in mat_test.txt The reason I build a sentence*word matrix is that is much more natural to build and requires just a single of the corpus. For removing the stop words I use a fixed array of the most popular stop words. For stemming I use Clairlib, and for other maintenance functions, I have a subroutine called punctuate, which contrary to the name, removes all punctuation except for ".", which are my seperators. The reason I need to this is to get the quantity "sentence".
The reason I had to do this and not use ClairLib's inbuilt sentence extractor was that I needed to keep track of the sentence position in the original text and i couldn't figure how to do it with that. For keeping track of sentences, basically I read the file line by line, and append it to string. After this, I split it on "." If more than 2 elements exist, I just pop off the last element, which could be incomplete, and add the others to the sentences array. I also check for the case that the discarded string might itself be a complete sentence.
At the end of the parsing stage, I have the sentence word matrix and the index `sentences'.

Clustering Engine: The core part of the project is in implemented using matlab, both because it is easier and its much more efficient for matrix operations, which I have to perform a lot. This module performs two functions, clustering the sentences and also finding out the centroids of these clusters. The main files are mat.m, similar.m, cluster.m and centroid.m svd_mod.m is a file used for only reducing the dimensionality. The algorithm is as follows:

1. Decompose original matrix A into A* using svd, and reduce dimensionality by using fewer features. I have a variable k, which represents the top 'k' features.
2. Calculate the similarities between each of the formed sentences of the new matrix A*.
3. Cluster the sentences by choosing the 2 most similar, forming a new one by merging them, and then reducing the number of columns of the A*, and again repeating from step 1, replacing A by A*. The compression ratio is a variable comp. the compression ratio I take is by dividing the number of sentences in the original, which is the number of columns of A, by comp.
4. Find the centroid of each cluster once the desired number of clusters are reached.

For finding the centroid, I make use of a table ctr, which is a column*column matrix, initially with only the diagonals as 1. the funda being that the rows represent clusters, the columns the sentences in that cluster. after each iteration, the rows corresponding to the most similar sentences are added, and the row corresponding to the deleted element is deleted. so in the end we are left with the cluster*sentence matrix. This matrix is used by centroid.m to calculate all the centroids of the clusters. The centroids are calculated as those which are closest to all the others. For a measure of similarity, I use the cosine measure. The code has been optimized such that most of the operations are matrix operations, and I have tried to reduce for operation to a minimum to boost speed. However this is the slowest stage, as the clustering algorithm always perform svd till the desired compression ratio is reached, which is a fairly heavy operation.

Summarization: the summarization is done by the script summary.pl. It reads from a file Final.txt, that is generated by matlab, that contains the centroids. The numbers of the centroids correspond with the line numbers of the initial “sentences11.txt” file which has the other information such as the file name and the actual line numbers in the original file, using which we can generate our text. This part is very basic, the only clever bit is that if two sentences are from the same file, then they appear in the same order. Sentence ordering was a problem to which I could not find any easy solutions.

Evaluation: At this stage I can only provide qualitative evaluation as I couldn't test with the rouge system. The text is well formed, however there are some arbitrary spacing schemes. Also, currently all my summarized text is in lower case, as it was easier to handle.
As we vary the k and comp, the best values are centered around k~50, and comp ~ 50. As the values change, there is change in terms of the sentences that are chosen. However, there are 1 or 2 that are always included, which can be seen in the sample ouput providee for D0561F, where there is a sentence about Clinton's impeachment in all the different generated texts. As the compression ratio changes, leading towards less clusters, clusters are quite evenly matched in size, as in usually the largest is significantly larger, but 2,3 or 4 are quite similar in size.

Areas of Improvement:
1. One of the first things I will do is to properly set up the rouge system to gauge how good it actually performs.
2. Change the clustering scheme, preferably go for semantic clustering or y-clustering, rather than the current slow one. This would drastically increase the speed of execution.
3. Another area of improvement is the final arrangement of sentences. This is a significantly harder problem than the other two because it involves arranging the sentences in a paragraph. One way I thought of was to train the machine on the human summarization of the same topics, calculate the “difference” between the sentence structures. First step in this would be to get similar sentences to the ones in the summaries.

   As of now, I think the system performs its task decently. It brings back well formed , relevant sentences.