Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu

Lab report 4

Objective:

Evaluate the efficacy and efficiency of minwise hashing for document similarity.

Procedure:

1) Using the feature vector created in Lab 1, we create a Jaccard similarity matrix for all documents. This will be our true similarity baseline.
2) We create signature for each document using Minhashing (for k=16,32,64,128).
3) Using the signature created in step 2, we create estimate Jaccard similarity matrix for all documents.
4) Compare the estimate Jaccard similarity matrices with baseline Jaccard matrix to find accuracy variance on k.
5) Repeat steps 2 to 4 for different values of k.

Creating Jaccard matrix:

[1] The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

(If A and B are both empty, we define J(A,B) = 1.) Clearly,

$$0 \leq J(A,B) \leq 1.$$

When calculating the baseline, the A and B are the original feature vector values for any 2 documents being compared. However when we compute the Jaccard similarity matrix for documents after minhashing, the Signature values of the documents are used as input.

Minhashing

[2] Converts large variable length sets to short fixed-length signatures, while preserving similarity.

Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu

To calculate this signature or hash-value we used the algorithm discussed in class. The procedure is:

- Create a matrix between words(rows) and documents(columns). The corresponding values denote the tf-ids.
- The words/rows original index values in the above matrix are used for creating different permutations (thus we transform or represent the words into integer values).
- Define a “hash” function $h_p(C)$ = the index of the first (in the permuted order p) row in which column C has value 1:

  $$h_p(C) = \min_p p(C)$$
- We apply the above hash function for each permutation of words. Thus the signature for a document is created.
- The number of permutations done is represented by variable K. Value of K also determines the length of signature.

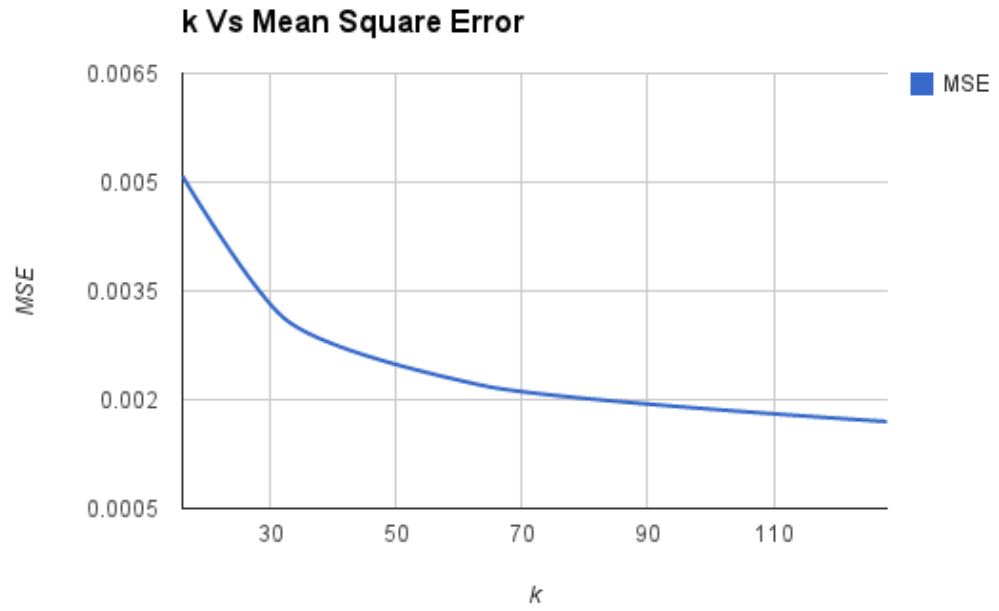
Observations

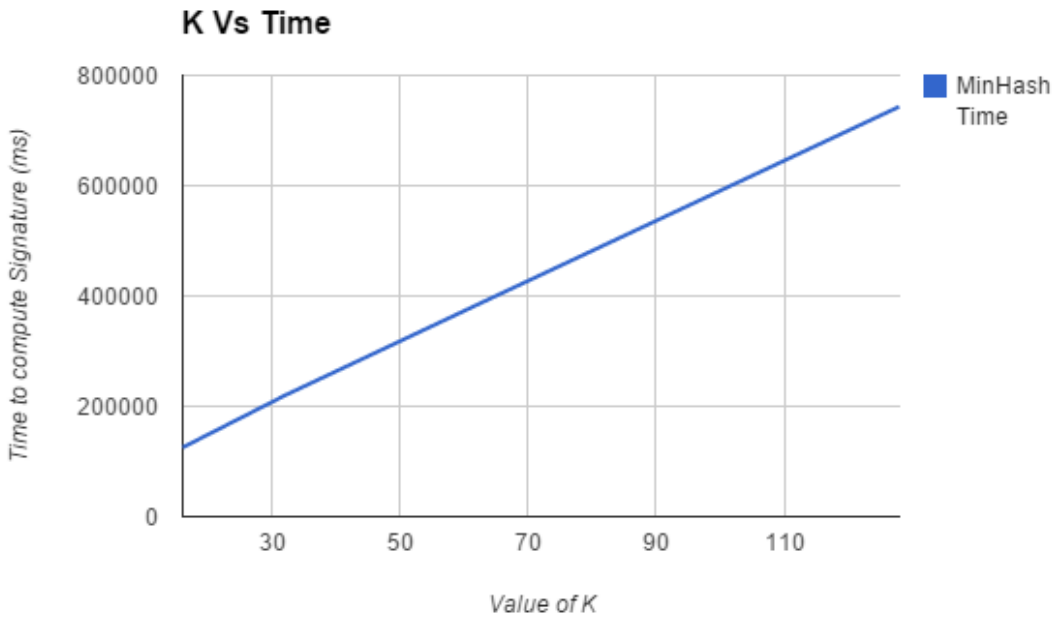
For different values of K, we create the signature for each document and calculate the estimated Jaccard similarity between every pair of documents. We note down the time taken to compute the minhash signature. We find the quality of this estimate Jaccard similarity matrix by comparing it against the baseline Jaccard matrix. MSE is calculated for every K-minhash estimate against baseline similarity matrix.

K	Time(ms)	MSE
16	125350.843	0.005083030165
32	219811.7359	0.003135981759
64	394834.0561	0.002190868497
128	743224.9620	0.001697943106

Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu





Results:

- When K is increased, time to compute K-minhash estimate increases.
- When K is increased, accuracy of estimate improves (MSE decreases), as we are using more integers to represent the data.

Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu

- We tried to find the ideal value of K for the reuters dataset. However for high values of K (e.g. K=200), the program was taking too long to finish (comparing Jaccard matrices) and hence we couldn't find the MSE values for higher values of K.
- One unexpected observation was: changing permutation creation mechanism (hash function) seemed to affect the efficacy! We used the following function to create hash values: (i*x + i*2) % s_n

  Where 'i' is: [0,number_of_documents], x is: [0, K] and s_n is number_of_words

Improving efficiency:

- The baseline Jaccard matrix takes time to compute. Hence we computed the baseline matrix in a separate program and saved the output in a pytext file . We loaded the file in memory when we wanted to calculate the MSE for a K-minhash estimate.
- The most time consuming part of the code was to compare each jaccard estimate with baseline, to speed this up we use multithreading and divided the documents into 4 sub-ranges and ran "four" threads for each range. This reduced the time to compare the documents by a huge difference.
- We also noticed that using a large feature vector (over 4000 words) was causing the algorithm to run very slowly. We increased the tf-idf threshold and re-created the feature vector to only have around 2000 words.

Contributions

Shashank Agarwal & Anurag Kalra both implemented the "minhash" clustering and optimizations. The work overlaps and is hard to be segregated.

References

[1] http://en.wikipedia.org/wiki/Jaccard_index
[2] Class lecture slides