Assignment 5

Jigar Chandra (Chandra.67)
Gaurav Shah (Shah.889)

Goal of the Assignment

To perform Min-Hashing on the Feature Vector generated for the Reuters dataset and observe its efficiency and efficacy.

Base-Line Similarity:

To compute the similarity between two documents, we use the Jaccard similarity metric, and we compute it using the formula:

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

We have used the feature vector 3 from Assignment 1 unchanged
Time to generate the Similarity Matrix: 955.242736178 sec

Min-Hashing:

The min-hashing part comprises of 2 steps

1) Generating the min hash
2) Creating the k-min hash sketch.

Generating the hash:

In order to generate the hash we used the hash function which is of the form

H=Ax+B%dim

Where **A** and **B** are random numbers

x is the number of the row or the row index

dim is the number of unique words that are present.

In order to apply this formula we had to take transpose of the frequency matrix which we had generated as a feature vector in part 1. And then use the row_index of each row within that transposed matrix in the formula stated above.
We needed different permutations for the formula mentioned above. In order to do that we had 2 different lists of values for A and B in the formula and depending on the number of permutations required(16,32,64,128) we would generate the lists of random numbers for A and B for that size. Thus, instead of picking n random permutations of rows, we pick '**k**' randomly

chosen hash functions h1, h2, . . . , hk on the rows. We had stored the hash values as a dictionary where the key was one of the the values from 0 to k-1 and values were list of length equal to the number of unique words. So if k is 16, number of unique words is 1000 then the dict will have the form. Dict={0: [list of 1000 hash values],1:[list of 1000 hash values]….,15: [list of 1000 hash values]}.So each key essentially indicated the permutation number and the value i.e. the list indicated the value of that hash for each row in the transposed matrix

Creating k-minhash sketch:

In order to generate the k-minhash sketch we had created a k X m matrix where 'k' is the same as in k-minhash and 'm' is the number of documents. we initialized the matrix to -1 .we iterate through every row in the transposed matrix we created in above step and for every row we go on checking the k-hashes one after the other. For each hash that we check we see that which indices in the row of the transposed matrix have non-zero values, the corresponding indices in the K-minhash matrix row for that given hash are then replaced by the hash_value obtained from the dictionary computed in step 1. When we move on to the next hash we again check the non-zero indices for the current row. If the hash value at those indices are lower than the current hash value then we don't change it. If however the current hash value is lower, then we change the value at those indices to the current hash value.

The following example will make things clear:

Row	S_1	S_2	S_3	S_4
0	1	0	0	1
1	0	0	1	0
2	0	1	0	1
3	1	0	1	1
4	0	0	1	0

Table 1

The Signature Matrix is of the form as shown below:

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

Table 2

Where S1,S2..S4 stand for the number of documents and h1,h2 are the hash functions.

First, we consider row 0 of table 1. Let us assume that the values of h1(0) and h2(0) are both 1. The row numbered 0 has 1's in the columns for sets S1 and S4, so only these columns of the signature matrix can change. As 1 is less than ∞, we do in fact change both values in the columns for S1 and S4. The current estimate of the signature matrix is thus:

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

Table 3

Results and Performance:
Efficiency:

To compute the estimate for the k-minhash sketches, we observed that the time required increased as the value of k increases. That is, the efficiency decreases as k increases. The reason for this is purely computation based as the size of the signature matrix goes on doubling as the value of 'k' increases from 16 to 32 to 64 and so on and as a result the number of computations also double. Following results were obtained from our implementation of the k-minhash :

Time required to compute the k-minhash sketch

K	Time required (in seconds)
16	275.716798635
32	526.098041809
64	1143.63017076
128	2682.71936879

Time required to compute Jaccard similarity and MSE for the k-minhash sketches:

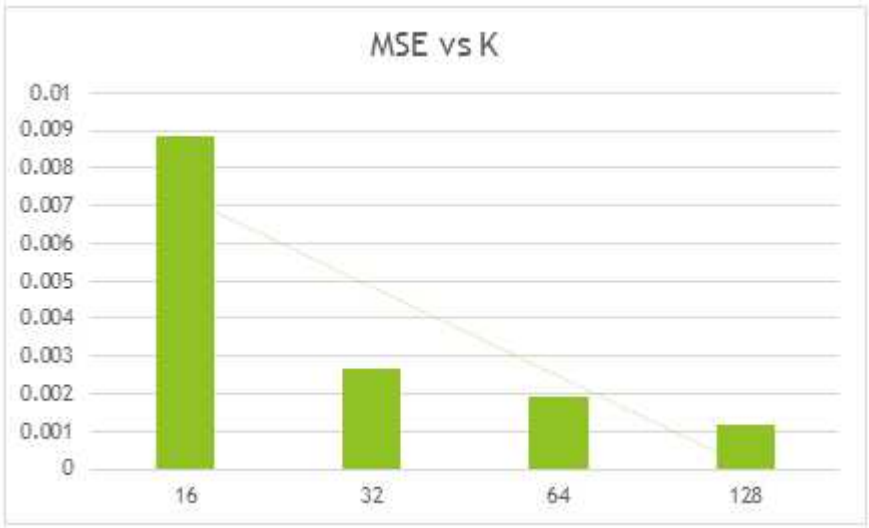
K	Time required (in seconds)
16	979.912647538
32	1662.64883936
64	2766.87042563
128	5143.29239369

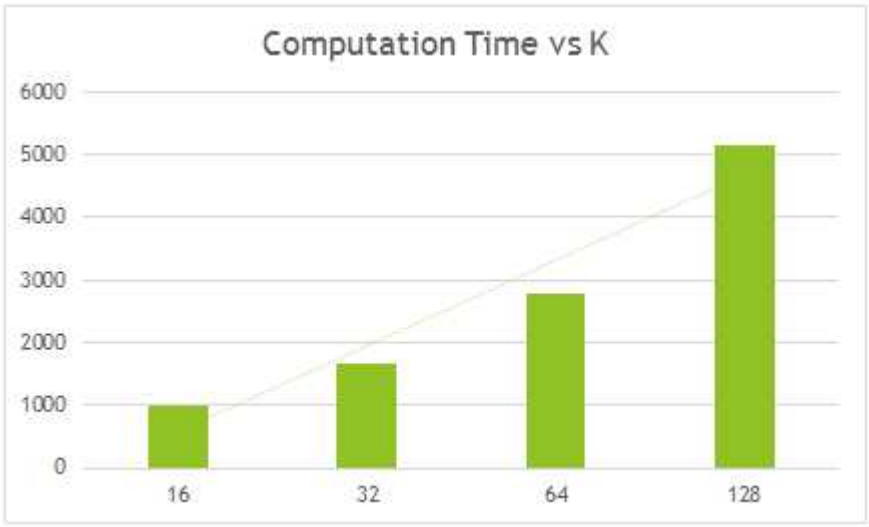
Efficacy:

To report on the efficacy, we compute the Mean Squared Error (MSE) between the estimate and the true similarity. We observe that the value of MSE decreases as k increases. That is, the efficacy improves as k increases. The reason for this is as the value of K increases it becomes more increasingly clear whether they are similar or dissimilar and so we move closer and closer to the baseline version and hence the error reduces. The following results were obtained :

K	MSE
16	0.00882100797724
32	0.00266791108874
64	0.00189296675403
128	0.00117939760072

Following graphs will make the above results easier to interpret:





Individual Contributions

Gaurav implemented the minhashing algorithm and plotting the graphs while Jigar computed the Jaccard similarities and the MSE computations.

References

Min-hashing and the example stated in the section of min-hashing
http://infolab.stanford.edu/~ullman/mmds/ch3.pdf
Jaccard Similarity

en.wikipedia.org/wiki/Jaccard_index