CSE 5243: Data Mining

Assignment 4

Submitted by:

Harsh Gupta (gupta.749@osu.edu)
Dhanvi Athmakuri (athmakuri.1@osu.edu)

PROBLEM STATEMENT

The purpose of this lab assignment is to evaluate the efficacy and efficiency of minwise hashing for document similarity, and analyze the results thus obtained. This report also documents the technical difficulties encountered in the implementation and the methods adopted to tackle them.

IMPLEMENTATION DETAILS:

Project Participants: This implementation has been accomplished by the collaboration of Dhanvi Athmakuri (athmakuri.1) and Harsh Gupta (gupta.749).

Participant Contributions: The layout for the code was discussed together. Harsh Gupta wrote the initial script to preprocess the file, generated the matrix and calculated Jaccard Similarity using the baseline approach. Dhanvi implemented k-minhash approach, calculated estimated Jaccard Similarity and efficacy metrics. Harsh and Dhanvi together optimized the data structure later.

Implementation language and libraries used: We have used python-2.7 programming language to implement our pre-processing methodology. The following libraries have been used to help process the files :

• **sklearn.metrics** – to calculate mean squared error and jacard similarity score

• **sklearn.feature_extraction.text** – to use **CountVectorizer** and **TfidfVectorizer**

Other libraries like urllib2, sys, csv, time, math, collections have been used for utility functions.

We used our own implementation for both Jaccard baseline and minwise hashing.

ASSUMPTIONS – REPRESENTATIVE SAMPLING

All the results stated below were generated by running our code on the complete reuters data set by random sampling 1000 and 2000 documents out of them to give representation of the entire data set. The rationale behind this is to reduce the computation time over the entire set of documents but at the same time not compromising with the quality of the data set. Words occurring too much or too less are ignored for this exercise as they will not provide enough information and lead to a wrong accuracy metric. Both **random sampling and filtering** will help to produce good results. This was a good trade off to execute the code within reasonable time.

PROCESSING METHODOLOGY:

Following is the step-by-step procedure which we have used to process the documents.

1. **Parsing the reuters dataset**

We parsed the reuters dataset to generate input for the other procedures to follow. Parsing was done using BeautifulSoup libraries and we removed the stop words and other unwanted notations from the text of the body tag in each of the documents. After parsing we generated files for trigrams using sklearn library CountVectorizer which gave an output in the form of a sparse matrix which made the work easier to calculate Jacard similarity and minhash similarity.

2. **Creating signature matrix**

We created the signature matrix for all the documents. For computing random permutations (hash functions), we used universal hashing method. We calculated the value of coefficient a, coefficient b and a very large prime number. We used values between 16, 32, 64, 128, and 256 for k. Using these random permutations, we calculated the signature matrix by updating the contents of the signature matrix for a document if the hash value was lesser than the current signature value. For each document we set the initially min_hash value to be as large as the (document size + 1) so that we can update it once we get a smaller value for the document.

3. **Calculating jacard similarity**

We computed the Jaccard baseline manually using the frequency matrix. For this huge number of files calculating Jaccard baseline pairwise, was time consuming. We followed some strategies to make this process fast. First thing we did was the calculation of only upper part of the similarity matrix. Another thing we observed was the document frequency matrix being very sparse. So we decided to use the *set* approach. The index of the words which were present in the document, were put in the set, so the number of comparisons for calculating intersection and union became very less. These tricks helped in calculating the Jaccard similarity quickly. For union rather than computing the actual set union we used an optimization of set theory of [n(A)+n(B)-n(A^B)]. These optimizations helped us compute the massive number of operations a little faster.

4. **Calculating estimated jaccard using minhash**

After calculating Jaccard similarity we had to calculate Jaccard similarity using that signature matrix. This part was bit crucial. Because the signature matrix was completely filled up we could not use the *set* approach. Only the upper part calculation trick was adopted. While calculating similarity each word of a document was compared with each word of another document. So this process was time consuming compared to jaccard baseline calculation though the time taken to do so was comparatively lesser than the jaccard calculation time as the number of features got reduced to the number of shingles.

5. **Calculating Root Mean square error**

We calculated the mean square error using the following formula

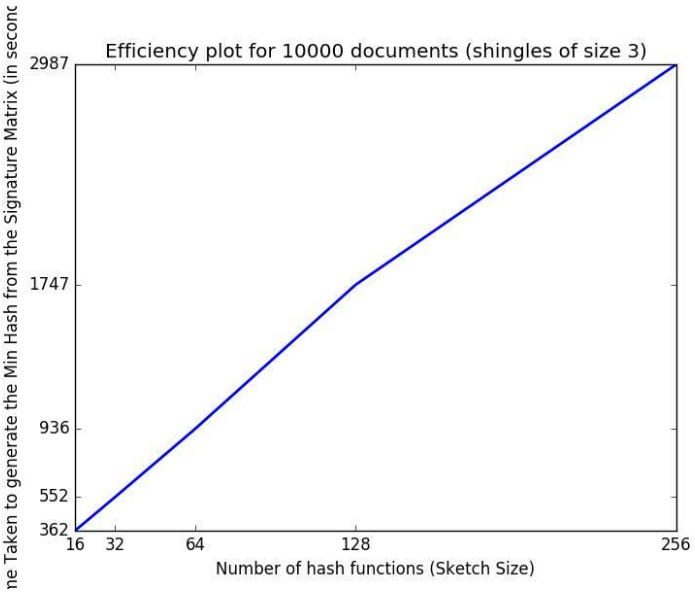
$$\text{RMSE} = sqrt\left(\frac{\sum_1^N (Estimated\ Jaccard - Original\ Jaccard)^2}{N}\right)$$

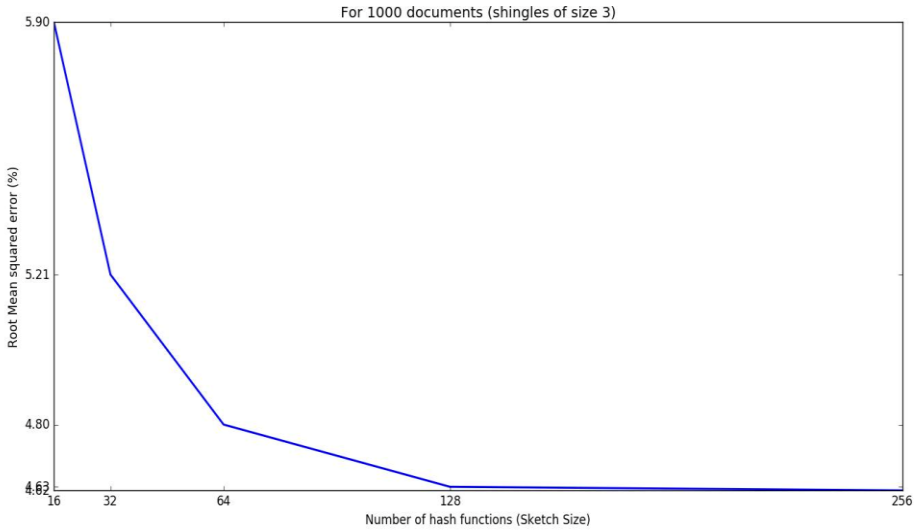
Where N = Number of documents

Since we were only calculating the upper triangular portion of the matrix, the number of calculations for obtaining Mean Square Error was comparatively less.

RESULTS

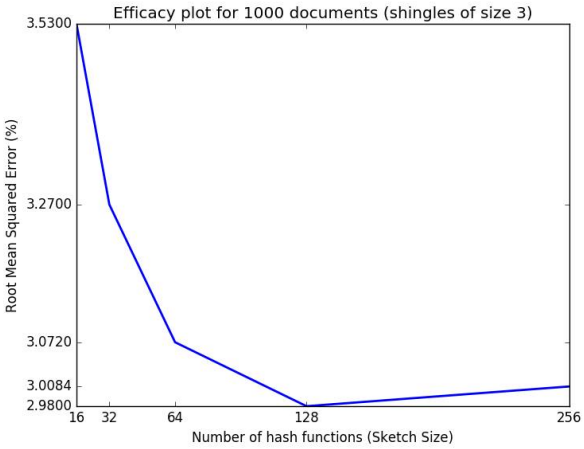
We computed the mean square error of the documents using **trigrams** (based on shingling). We found that as the value of K increases the mean squared error gradually decreases. This validates the statements of the Mining Massive Dataset book. The below graph shows the efficiency plot for **10000 documents.** The average time to calculate the Jaccard similarity is between **200 to 350** seconds.

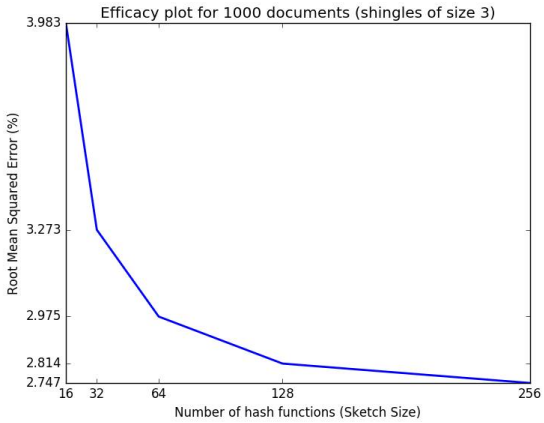


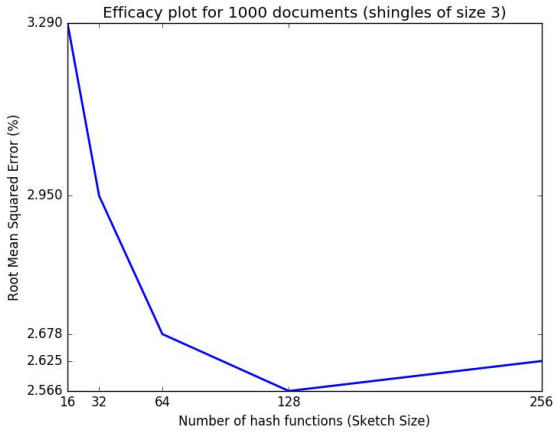


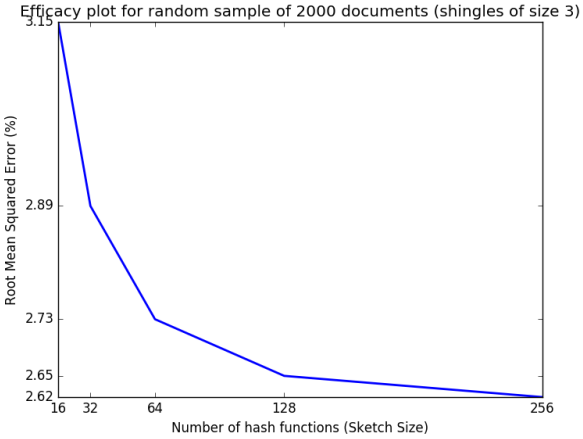
From the above results, one could easily infer that there is a tradeoff between efficiency and efficacy with respect to K. We believe that K = 64 is a good parameter for hash function as it give around 95 % accuracy and takes considerably less time than bigger hash function values.

Some of the **representative sampling** for 1000 documents out of 20000 documents is as follows









For these four **random samples** we see that k = 64 is a good choice for efficacy because after this the error rate may go abrupt and we get higher error than its previous value of hash function.

PROBLEMS FACED:

1. **Calculation of baseline Jaccard similarity**
   Initially calculating Jaccard similarity for even 1000 documents was taking a lot of time because we had to do 1000000 computations to fill the 1000 X 1000 matrix of Jaccard similarity. Therefore we did some optimization stated below.
   - Calculation of only the upper triangular matrix because it can give us the pair wise similarity of entire documents.
   - Using set approach of union and a modified set theory version of intersection to compute jaccard similarity as it reduced the time for computations.

- We tried to parallelize the code, but faced certain difficulties regarding the execution flow of the program, and could not debug it. Hence we decided not to implement.

2. **Calculation of hashing functions:** While calculating the min hash value we were initially calculating the random permutation at the time of comparison with the signature matrix. It was time consuming. Just by calculating all the random permutation at the beginning saved a lot of execution time.

3. **Memory Error:** The entire program is very memory consuming. Total 2 Jaccard matrix each of size (number of documents x number of documents), 5 signature matrix each of size (k x number of documents for k values 16, 32, 64, 128, 256) and 1 data matrix are needed. Initially for k value 64 the program was crashing due to memory error. After certain optimization we were finally able to run the program over the entire dataset without fail.

CONCLUSION:

This assignment was very challenging in terms of program execution time management and memory management. The effects of code optimization was clearly visible. We realized how the task of calculating minwise hashing on big datasets can be challenging. We also realized how the effect of shingling can be helpful in reducing the mean square error value. This assignment not only helped us in writing optimized code but also helped us to understand the minwise hashing in greater detail and how it can be helpful in case of similarity calculation of massive datasets.

REFERENCES:

- http://scikitlearn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
- http://dl.acm.org/citation.cfm?id=2505765