CSE 5243 - Assignment 4

Team Members and Contribution

1. Anu Yadav
2. Pravar Mahajan

We both have worked equally on this assignment

Problem Description and Assumption

In this assignment we are required to explore the efficacy and efficiency of minwise hashing algorithm to compute the similarities between pairs of documents in the Reuters dataset. The Reuters dataset consists of approximately 20,000 documents. Some of the key assumptions are as follows:

(1) We have used the document 'body' for computing the similarity, all other features like title, topics and places have not been considered for computing similarity, and thus have been discarded.
(2) Stop words, punctuations and non-words tokens like numbers and dates have been considered noise and thus filtered from our dataset.
(3) For the purpose of the assignment, we have considered word shingling only. Alternative could be character shingling, which has not been explored here.
(4) Computing similarity between 20,000 is computationally very expensive, leading to memory errors. We have selected a random subset of 17,600 (88% of total) documents for our experiments. Assumption is that these randomly selected documents are an unbiased representation of the entire dataset and will give a good estimate of Mean Squared Errors.

An Overview of the proposed solution

We propose the following steps to solve the problem. We will describe how we performed these steps in the later sections.

(1) Preprocessing - The dataset is in raw XML format, it needs to be cleaned, parsed and noisy words and tokens need to be filtered out.
(2) Shingling - Documents in raw text should be converted into features via the process of shingling
(3) Hash generation - The extracted features are required to be converted into hash functions. These hash functions should be locality sensitive in order to perform similarity computations
(4) Similarity calculations - The hashed documents are used to calculate similarity.
(5) Benchmarking - Jaccard similarity is used as baseline model against which efficiency and efficacy of the minwise hashing algorithm is evaluated.

Details of our Approach to Solve the Problem

The following are the steps involved in analysing the efficacy and efficiency of minwise hashing.

1. Shingling

For previous assignment on classification, we were using the unigram/bigram models to build features for our classifiers. For this assignment, we have rewritten our **own code to create shingles**. The existing code has been generalized to w-shingle word features, for any input 'w'. Previously, a document was represented as vector of counts for each feature. For this assignment, we have changed the count vector to a binary vector notation, where 1 represents presence of a feature and 0 represents absence of the feature. Stopwords, such as 'I', 'the', 'is' etc were removed. The list of stopwords was obtained from Python's NLTK package. The tokens obtained were stemmed using the Porter Stemming algorithm. For our experiments, we have tried with the values of **w=1, 2 and 5**. At the end of shingling, we have approximately 25,000 1-shingle features, 500,000 2-shingle features and 2,500,000 5-shingle features. A

comparison of the performance of the hashing algorithm for different values of number of shingles has been documented in the Results section.

2. Locality Sensitive Hashing

As seen in previous section, the number of features generated for w=1,2 and 5 shingles is very large. Consequently pairwise similarity calculation of 17,600 documents is very slow. We generate k-minhash sketch for **different values of k - 16, 32, 64, 128 and 256**. We have written our own functions to generate these sketches. In the Results section, we have demonstrated that these sketches allow quick calculation of pairwise similarity of documents, without any significant differences in similarities. The running times for generation of these hashes for all the documents in our sampled dataset have been recorded in the table below:

3. Similarity

For the baseline, we have used the raw feature vectors obtained via shingling, and calculated the exact Jaccard similarity between all pairs of documents. Jaccard similarity is defined as:

$$J = \frac{M11}{M11+M10+M01}$$

where,
M11 = number of shingles present in both the documents in the pair
M10 = number of shingles present in document 1 but absent in document 2
M01 = number of shingles absent in document 1 but present in document 2

We will later see that the alternate model, minwise hashing, allows fast approximations of the Jaccard similarity measure as described above. Due to paucity of space, we won't describe the algorithm in detail. In short, the algorithm computes the following probability estimate as an estimate for the Jaccard similarity:

$$Pr[mh_i(A) = mh_i(B)] = Sim(A, B)$$

where, $mh_i(.)$ with $i \in \{1..k\}$ is the function which calculates the i^{th} hash of the input document, out of k hashes. A & B constitute the document pair for which we are computing similarity.

Results

In this section, we document the efficacy as well as the efficiency of the minwise hashing algorithm and evaluate it against our baseline model - Jaccard Similarity. The recorded timings are as observed by running the experiments on Intel Xeon 64-bit E5 processors running at 2600 MHz and with 8GB of RAM.

1. Efficacy

For efficacy or quality, we have chosen to report the mean-squared error between the estimated value of similarity and true value baseline (Jaccard Similarity). The observed values have been recorded in **Table 1**.

$$MSE = \frac{2}{n(n-1)} \sum_{i \neq j} (J_{ij}-X_{ij})^2$$

$i, j \in \{1..n\}$

n = number of documents
X_{ij} = Similarity between documents i and j calculated using the minhashing algorithm.
Y_{ij} = Jaccard similarity between documents i and j

Table 1: Mean Squared Error against baseline (Jaccard), as multiples of 10^{-3}

	1-word shingles	2-words shingles	5-words shingles
16-hashes sketch	5.46	0.561	0.254
32-hashes sketch	5.45	0.560	0.253
64-hashes sketch	5.43	0.557	0.252
128-hashes sketch	5.39	0.553	0.250
256-hashes sketch	5.30	0.544	0.246

2. Efficiency

We will evaluate the efficiency of the minwise hashing algorithm in two parts - Firstly, by considering the time to generate the hashes and secondly, time to calculate the actual similarity. In the following two tables, **Table-2** has observations obtained on the first part and **Table-3** records the observations corresponding to the second part of the evaluation. The timings reported here are wall-clock times and not CPU times, ie they do not reflect the true CPU usage and may have some noise due to competition by parallel process for CPU utilization.

Table 2: Time (in secs) to generate hash functions of different sizes, for different number of shingles

	1-shingle	2-shingle	5-shingle
16-hashes	30	32	39
32-hashes	61	63	75
64-hashes	124	132	133
128-hashes	247	254	270
256-hashes	442	480	546

Table 3: Time (in seconds) to Compute Different Similarity Measures

	1-word shingles	2-words shingles	5-words shingles
Jaccard (true similarity)	951	1113	1216
16-hashes sketch	608	592	610
32-hashes sketch	628	594	704
64-hashes sketch	657	686	699
128-hashes sketch	758	847	860
256-hashes sketch	829	1504	1604

In order to visualize the effects of minwise hashing algorithm to efficacy and efficiency, we have recorded our observations in the form of a graph, in the last page.

Discussions

(1) Efficacy:

(a) As observed from the MSE, minwise hashing algorithm performs reasonably well in calculating the similarities between documents, when compared against the Jaccard Similarity baseline. To put the numbers in perspective, the average Jaccard similarity for all pairs of documents was 0.22 for 1-shingle, 0.25 for 2-shingles and 0.35 for 5-shingles. MSE is very small percentage of these similarity numbers.

(b) The general trend observed is that MSE decreases on increasing the number of hashes, which is an expected outcome. The more hash functions we use, the better will be our probability estimates.

(c) The MSE decreases as we increase the number of shingles, even though the average Jaccard similarity is decreasing. This shows that increasing the number of shingles increases the efficacy of minwise hashing

(2) Efficiency:

(a) In general, we observe that increasing the number of hash functions increases the time to generate the hashes as well as the comparison time, which is expected because more hash functions implies more comparisons. Some minor anomalies observed may be explained by the fact that the experiments were run on stdlinux server which added some noise to the recorded times due to competition for CPU utilization with processes run by other users.

(b) We observe that for 2-shingles and 5-shingles, the time to compute similarity for 256-hashed documents is more than the Jaccard Similarity. On investigating we found a probable explanation for this observation is the fact that average number of features per document in 2-shingle and 5-shingle cases is 76 and 80 respectively, and at some point it would be faster to do the set intersections of the sparse features than to compare 128 or 256 sized arrays.

Our experiments show that minwise hashing algorithms perform reasonably well with respect to Jaccard Similarity in terms of both efficacy and efficiency. In fact, even for small values of k like 16, the approximation is quite good.

Code Details

The script *main.py* is the backbone of our code. It expects two arguments the number of documents and the number of shingles. The script *minwise.py* contains functions to generate hashes for the documents whereas the script *shingling.py* contains functions to generate shingles. Finally, *similarity.py* is used to calculate Jaccard Similarity and k-hash similarity, and has the function to compute the mean squared error.

Libraries Used

Scipy, Numpy, Scikit-learn, NLTK and MatplotLib

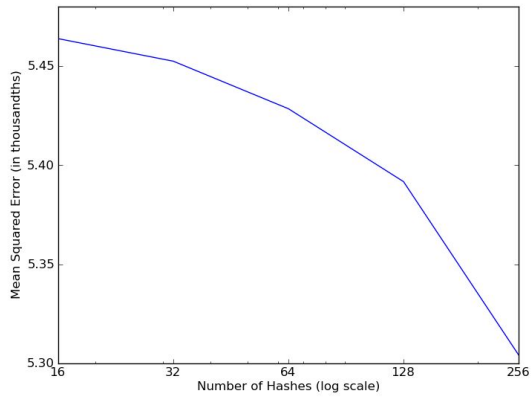

Fig 1a: MSE of true similarity vs approximate similarity for different number of hash functions, with 1-shingle features

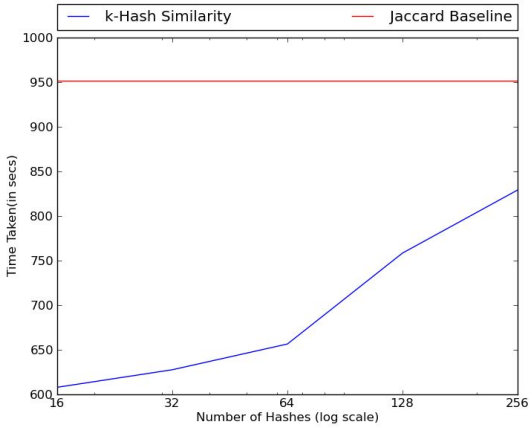

Fig 1b: Time taken (in secs) to compute similarities for increasing values of k, with 1-shingle features.

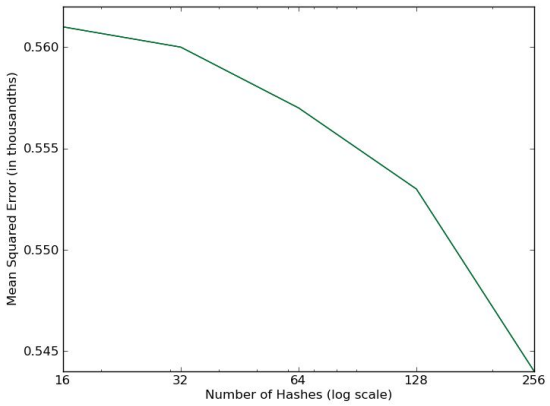

Fig 2a: MSE of true similarity vs approximate similarity for different number of hash functions, with 2-shingle features

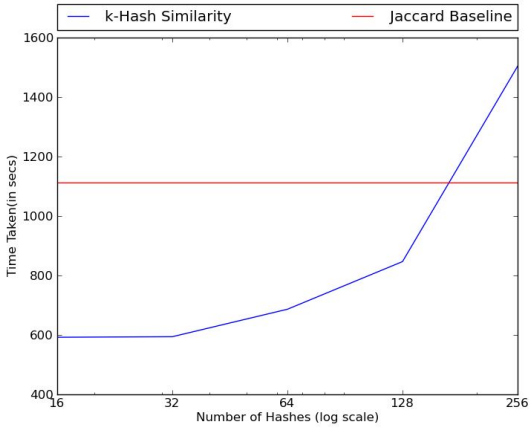

Fig 2b: Time taken (in secs) to compute similarities for increasing values of k, with 2-shingle features.

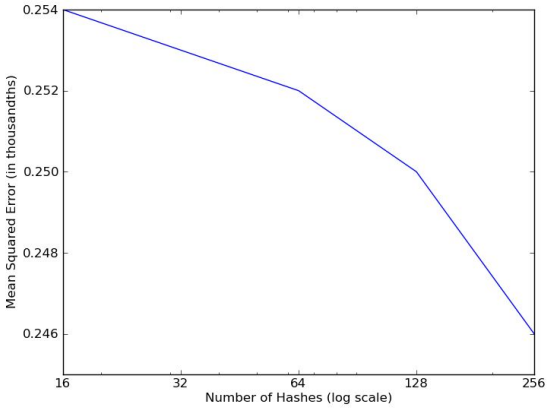

Fig 3a: MSE of true similarity vs approximate similarity for different number of hash functions, with 5-shingle

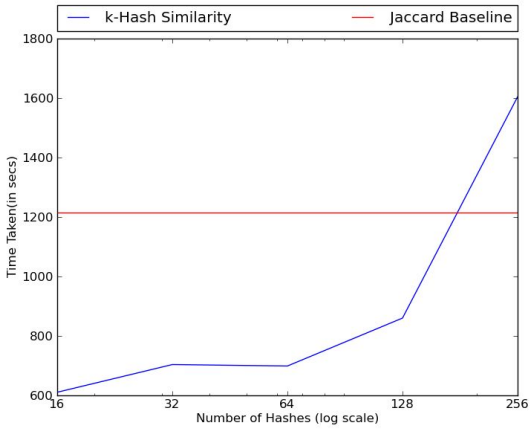

Fig 3b: Time taken (in secs) to compute similarities for increasing values of k, with 5-shingle features.

features