# Assignment 5 Report: Minwise Hashing for Document Similarity

Annelise Wittenberg & Cody Stammer

([wittenberg.18@osu.edu](mailto:wittenberg.18@osu.edu); [stammer.3@osu.edu](mailto:stammer.3@osu.edu))

## Approach

For this assignment, we again took a python approach to build our Jaccard and Minwise hashing functions. Instead of using the word frequency feature vector that we have been using up to this point, we decided to use our bigram feature vector, as it works much more easily with our implementation and will end up taking less space. We chose to run min-hashing on a feature vector of bigrams because the algorithm yields better results for n-grams or shingles. There are 10 bigrams per document, and with duplicate bigrams among documents, the matrix to store the data will be significantly smaller than that of individual words versus documents. This change in feature vector required changing the format of the input file we used for the rest of our functions. We adjusted our code from Assignment 2 to produce an input file of the format:

<Document ID> <Class Label> <Bigrams>

Using this format, we could more easily extract the bigrams from each document for use in comparisons.

We began by writing code to calculate the Jaccard similarity for each document compared with every other document. Since Jaccard(Document1, Document2) is the same as Jaccard(Document2, Document1), we only needed to calculate values for half of the matrix, those above the main diagonal. The main diagonal having Jaccard values of 1.

Next, we created a matrix of (bigrams x Document) which we could use in our Minhash function. The k-minhash implementation determines the minimal hash value for each document based on the algorithm. The algorithm runs k times to produce the signature matrix. Similarities are calculated from the signature matrix to get the estimated Jaccard similarity. This estimated value is compared to the actual value calculated at the beginning to compute the mean square error as a means of calculating the efficacy of the implementation.

## Results

Timing/CPU rate

MSE

Different values of k

Time to generate sketch vs. MSE plot (one for each value of k)

## Issues

We ran into standard development issues while completing this assignment including bugs and syntax errors. The space concerns that were encountered in all of the previous assignments became non-issue in this assignment. We were able to run our full dataset on both a VM with 16GB of RAM and a laptop with 8GB of RAM.

The biggest issue faced was understanding the k-minhash algorithm. It took a few attempts at implementing the algorithm to actually be able to understand what exactly needed to be solved. Once we finally understood what needed to be done, issues just became standard programming and debugging tasks.

Another issue faced was still only having a beginners understanding of python. We have become relatively comfortable with programming in python, but it would perhaps be easier to implement every function necessary if we had better knowledge of different data types or libraries. Once we committed to a data type that we felt comfortable using, we became bound to finding out how to build our functions to accommodate that data type, which wasn’t always easy.

When computing the similarities, we began by working with the top half of the matrix. This approach requires us to store zeros for the bottom half of the matrix underneath the main diagonal in order to preserve matrix structure. Had we used the bottom half, we wouldn’t need to fill the top half with zeros, as the matrix would naturally fill from left to right, top to bottom and preserve the order that we needed. We end up doing the same amount of computations, but doubling the space needed to store the matrix.

## Assumptions

## Work Distribution

Report – Annelise & Cody

Debugging - Annelise

Testing - Cody

Data Transformation - Cody

MSE - Annelise

Min Hash - Annelise