# 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 assignment 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 a 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.

## Results

Timing/CPU rate

MSE

Different values of k

## Issues

## Assumptions

## Work Distribution

import random

import sys

import pprint

featVect = [[1, 0, 0, 1],

[0, 0, 1, 0],

[0, 1, 0, 1],

[1, 0, 1, 1],

[0, 0, 1, 0]]

def hashIt():

p = 2038076783

a = round((random.random())\*10\*\*13, 0)

b = round((random.random())\*10\*\*13, 0)

minSig = p

# Transpose matrix so each row represents a bigram vector of 1s/0s of

# Presence/absence for a particular document

docVects = zip(\*featVect)

# Outer for loop grabs the entire first row

for index, v in enumerate(docVects):

# Calculate the hash value if there's a 1

print(v)

# For each element of row v

for i in v:

if v == 1:

print(index)

# value = ((index of '1')\*a + b)%p%N

value = (index\*a+b)%p%9

print(int(value))

# Update minhash-signature value

if minSig > value:

minSig = value

print(minSig)

pp = pprint.PrettyPrinter()

return int(minSig)

hashIt()

# Len = len(bigramList) = #bigrams

# def minHash(inputMatrix, k, numDocs):

# # Num rows in the hash function will be the # rows in the input matrix

# rows = len(inputMatrix)

# # Cols is the number of columns in the input matrix, ie # docs

# cols = len(inputMatrix[0])

# sigRows = k

# # Sig matrix initialize with series of max int

# sigMatrix = []

# for i in range(sigRows):

# sigMatrix.append([sys.maxint] \* cols)

# for r in range(rows):

# # If value = 1, and signature > hash value, replace signature with hash value

# # According to the k-minhash algorithm.

# for c in range(cols):

# if inputMatrix[r][c] == 0:

# continue

# for i in range(sigRows):

# hashvalue = (random.random()\*r+random.random()%sys.maxint)%numDocs

# if sigMatrix[i][c] > hashvalue:

# sigMatrix[i][c] = round(hashvalue, 2)

# return sigMatrix