CSEP 521, Winter 2021: Homework 8

Krishan Subudhi (ksubudhi@uw.edu) - 2040900

March 5, 2021

Problem 1

Q: Is the $L^{1/2}$ norm a proper distance function (a metric)? Prove or disprove.

$$||(x, y)||_{1/2} = (\sqrt{|x|} + \sqrt{|y|})^2 \quad [1.1]$$

The properties of a proper distance function are:

$$1.d(x, y) = 0 iff x = y$$
$$2.d(x, y) = d(y, x)$$
$$3.d(x, y) \leq d(x, z) + d(z, y)$$
$$4.d(x, y) \geq 0$$

$L^{1/2}$ does not always satisfy property 3 for all combination of x and y. Hence it is not a proper distance function.

Let's take one example in 2d.

$$x = (0, 1)$$
$$y = (1, 0)$$
$$z = (0, 0)$$
$$d(x, y) = (1 + 1)^2 = 4$$
$$d(x, z) = (0 + 1)^2 = 1$$
$$d(y, z) = (1 + 0)^2 = 1$$

$$d(x, z) + d(y, z) = 2$$
$$d(x, y) > d(x, z) + d(z, y)$$

Since for the above example, the property $d(x, y) \leq d(x, z) + d(z, y)$ does not hold true, $L^{1/2}$ norm is not a proper distance function.

Problem 2

Q. Suppose that $|U| = n$ and you select random subsets, $A \subseteq U$ and $B \subseteq U$ with $|A| = m$ and $|B| = m$. What is the expected size of $A \cap B$?

Let X_i be an indicator random variable with values

$$X_i = \begin{cases} 1, & \text{if } i^{th} \text{ element is present in both A and B} \\ 0, & \text{otherwise} \end{cases}$$

$$\begin{aligned} E[|A \cap B|] &= E[\sum_{i=1}^{n} X_i] \\ &= \sum_{i=1}^{n} E[X_i] \\ &= n * E[X_i] \\ &= n * P(X_i = 1) \\ &= n * P(i \ in \ A) * P(i \ in \ B) \\ &= n * m/n * m/n \\ &= \frac{m^2}{n} \end{aligned}$$

Q: Give an expression for the value of the Jaccard similarity of A and B if $m = n/k$

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

if $m = n/k$,

$$\begin{aligned} E[|A \cap B|] &= \frac{m^2}{n} = \frac{n}{k^2} \\ E[|A \cup B|] &= E[|A| + |B| - |A \cap B|] \\ &= \frac{2n}{k} - \frac{n}{k^2} \\ &= \frac{n}{k^2}(2k - 1) \\ J(A, B) &= \frac{|A \cap B|}{|A \cup B|} \\ &= \frac{1}{2k - 1} \end{aligned}$$

Hence $J \alpha \frac{1}{k}$. This means that for uncorrelated data, as k decreases (i.e. subset size increases), Jaccard similarity increases.

Problem 3

In this exercise, I am assuming random distribution of n points. (Due to lack of time, doubts could not be clarified. Hence making this assumption for simplicity.)

I start from j=k and if there are no collisions, I just keep decreasing j until there is a collision . If there is collision, then match with all points inside that bucket. The algorithm will run until there is a collision.

For a new query,

1. set j = k
2. Find the hash of new query for bucket.
   (a) If there is collision,

   find distance from all elements in that bucket and choose the smallest distance element as the nearest neighbour. This is our bound. Search for nearest-neighbours in buckets on left and right if distance from the query to the bucket regions are less than the bound. choose the smallest distance from all 3 buckets as the nearest neighbour.

   (b) If there is no collision, set j = j-1 repeat step 2

Runtime = E[Number of bucket access starting from j =k]

$$\begin{aligned} E[k] &= 1 + P(\text{no collision})\, E[k-1] \\ &= 1 + P(\text{no element in hashed bucket for } j = k)\, E[k-1] \\ &= 1 + \left(1 - \frac{1}{2^k}\right)^n E[k-1] \end{aligned}$$

Since neighbouring buckets are also used, final expectation E'[k] = E[k] +2

Problem 4

Here I downloaded the data and used python collections.Counters to store the BOW data in sparse format. During similarity calculation between X and Y, a union of the sparse dimensions are done. dimensions not present in the original sparse representations will have value 0 when accessed.

Each wordId is a dimension here and count is the value for that dimension.

Similarity heatmap is prepared by calculating the similarity of all documents of group A with all documents of group B and averaging them.

4.1 b) Similarity Heatmap

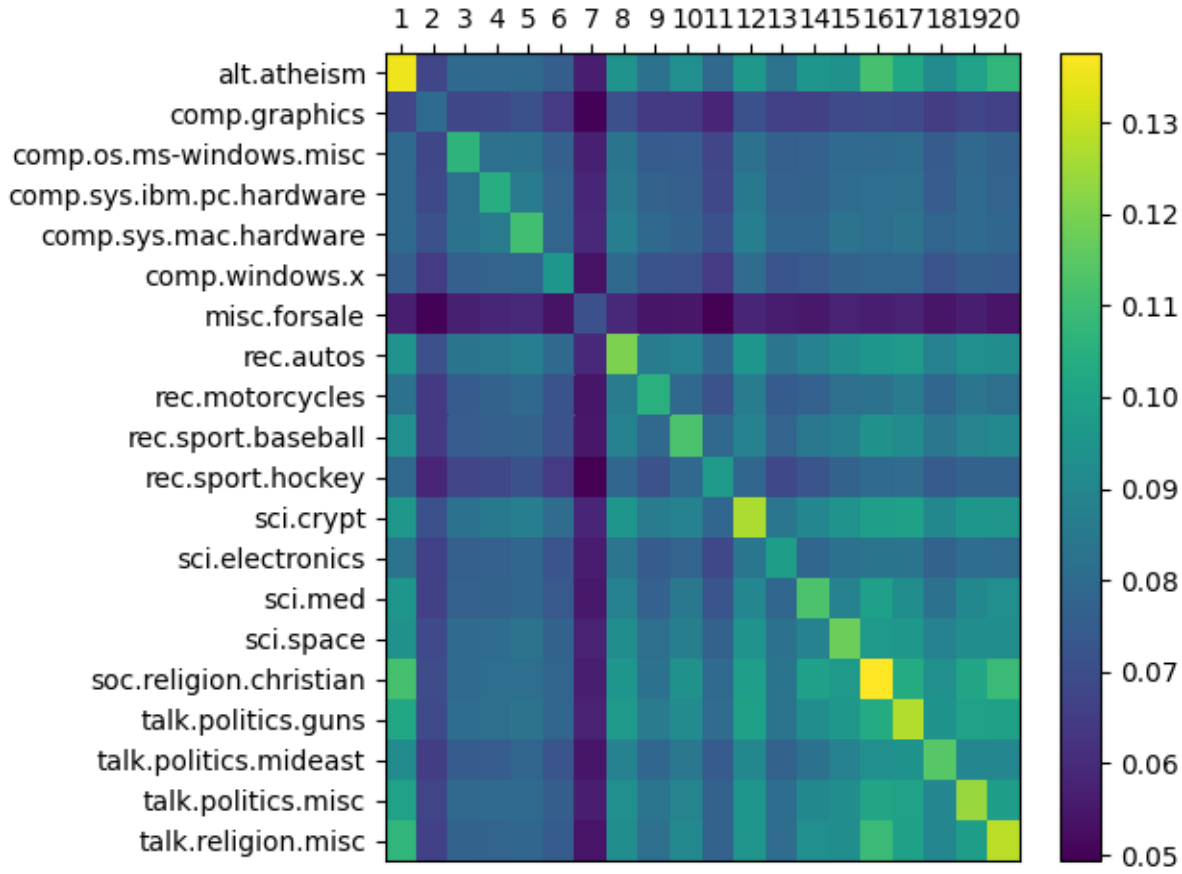

Figure 1: Jaccard similarity heatmap

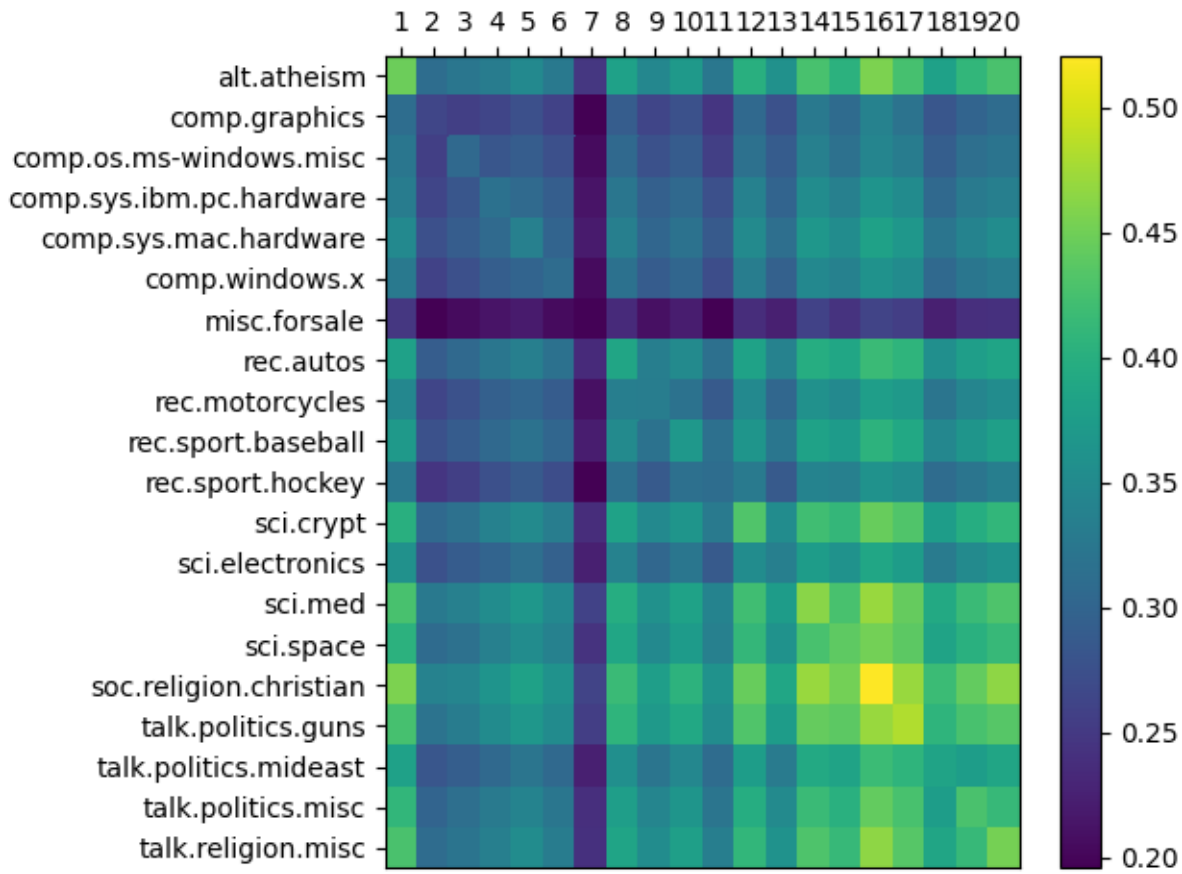

Figure 2: Cosine similarity heatmap

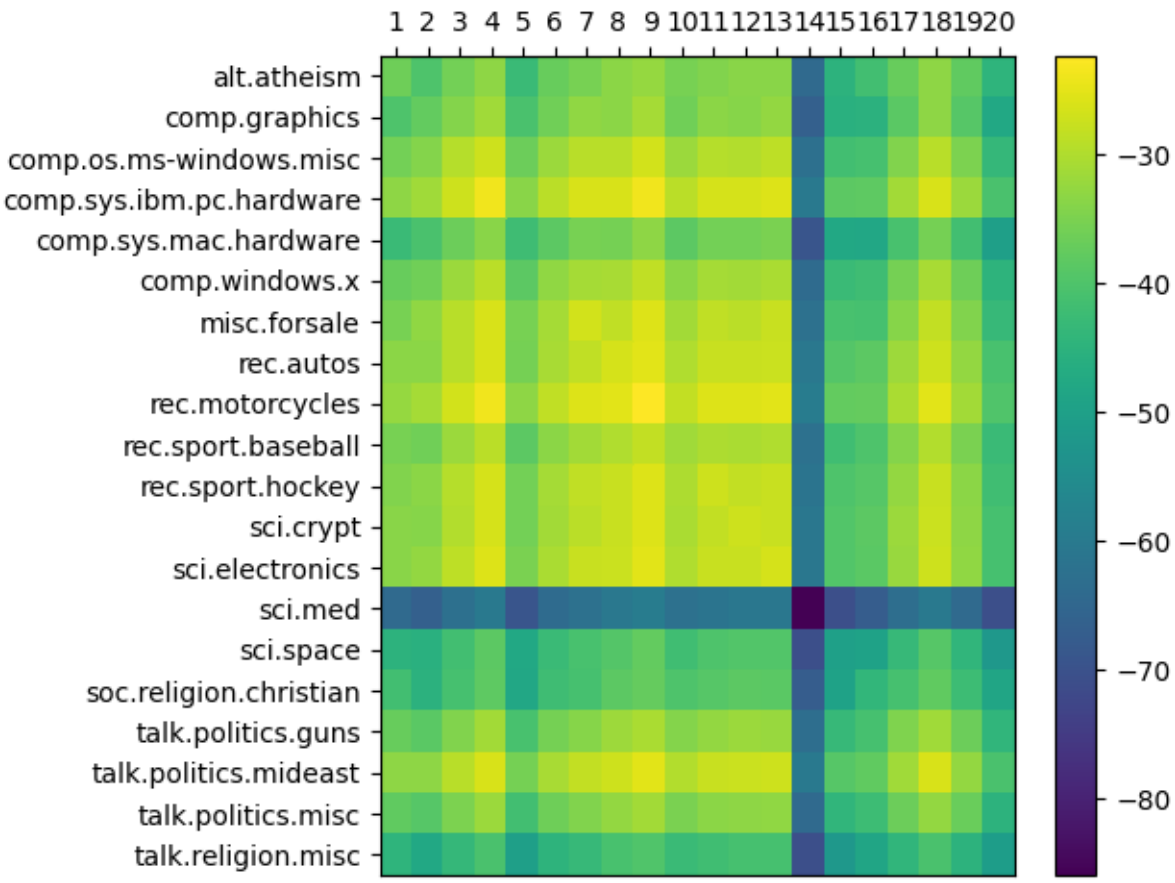

Figure 3: L2 similarity heatmap

Q. which of the measures seems the most reasonable?

The cosine and jaccard similarity seem reasonable compared to L2. Consine similarity seems to be the best since it is somewhat independent of document length and more dependent on the distribution.

Q.Are there any pairs of newsgroups that are very similar? As per both Jaccard similarity and cosine similarity, talk religion.misc and soc.religion.christian seem to be very similar. Overall, the groups have mostly similar documents within themselves.

Q.Would have you expected this? I would have expected to to see more similarity between groups starting with comp. I was also expecting the rec groups to have more similarity too. Overally the talk groups are more similar probably because of the limited vocabulary used in spoken english. But it's just a guess. More analysis is required. misc.forsale is dissimilar with almost every group. This means the documents that belong to this group have completely different distribution.

4.2 Nearest-Neighbour heatmap

The nearest-neighbour heatmap is constructed by calculating the number of nearest neighbours found for all documents in groups other then their own. Jaccard similarity was used as the metric.

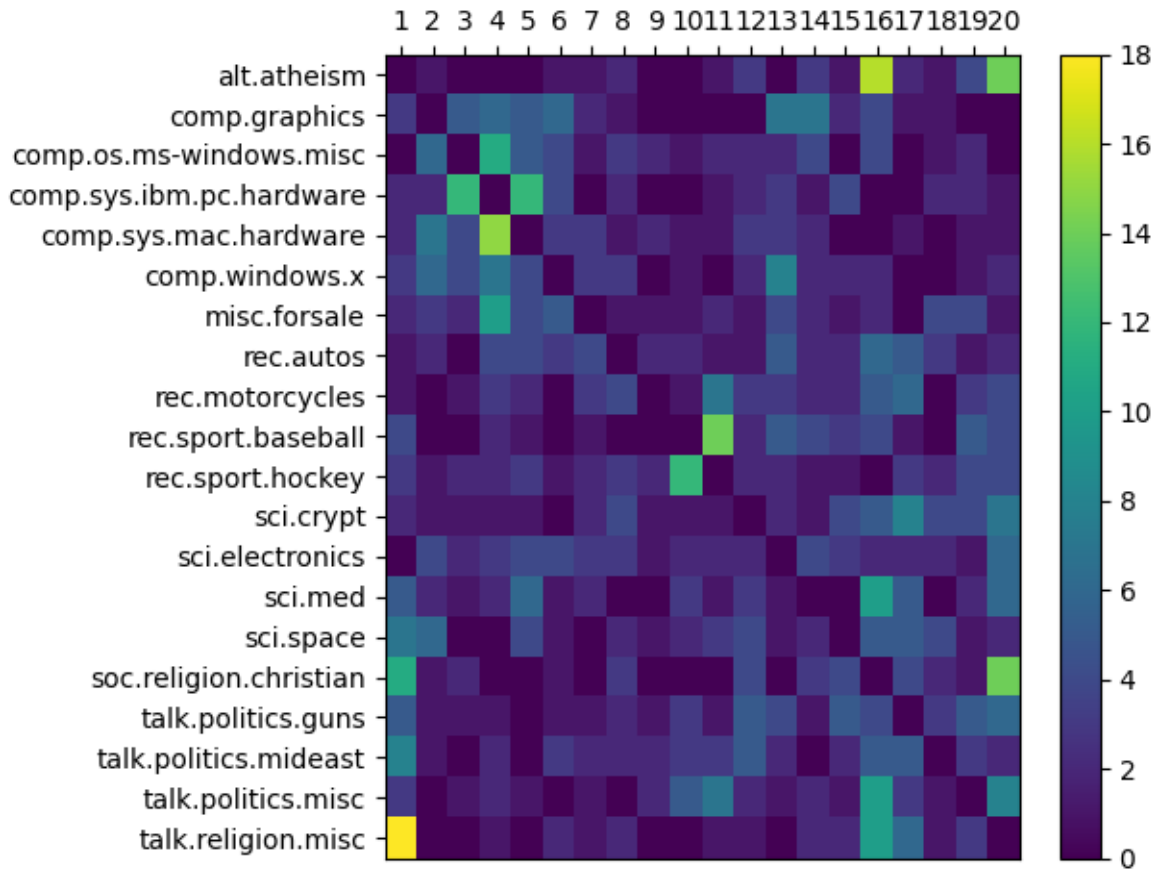

Figure 4: Nearest-Neighbour heatmap

Q.Your plot for part (b) was symmetric, but for part (d) was asymmetric. Explain.

if docA is most similar to docB that does not mean docB is also most similar to docA

example:

doc A: I am Krishan

doc B: I am going to school today with Krishan.

doc C: I will be going to school today.

Which groups seem similar? Compare the plots from parts (b) and (d). Which method seems more suited to comparing newsgroups?

alt.atheism is similar to talk.religion.misc and talk.religion.misc. Also computer hardware groups are more similar with each other. hockey and baseball groups are similar to each other.

While average similarity gives a smoothed heatmap, the differences are hard to distinguish. Nearest neighbour looks like a discrete heatmap but helps in better comparison.

4.3 Source code

```
from heatmap import makeHeatMap # professor's code
import numpy as np
import pandas as pd
from collections import defaultdict, Counter
import math


class SimCalculator:
    def __init__(self,data_file, labels_file,group_file):
        self.doc_bow = self.create_doc_bow(data_file)
        self.label_doc_map = self.create_label_doc_map(labels_file)
        self.doc_label_map = self.create_doc_label_map(labels_file)
        self.label_name_map = self.create_label_name_map(group_file)
        self.labels = range(1, len(self.label_doc_map.keys())+1)#[:5]
        self.label_names = [self.label_name_map[id] for id in self.labels]

    def create_doc_bow(self,data_file):
        doc_bow = defaultdict(Counter)
        with open(data_file,'r') as f:
            lines = f.readlines()
            for line in lines:#remove
                columns = line.split(',')
                doc_bow[int(columns[0])][int(columns[1])] += int(columns[2])
        return doc_bow

    def create_label_doc_map(self, label_file):
        label_doc_map = defaultdict(list)
        with open(label_file,'r') as f:
            lines = f.readlines()
            for i, line in enumerate(lines):
                label_doc_map[int(line)].append(i+1)
        return label_doc_map

    def create_doc_label_map(self, label_file):
        doc_label = {}
```

```
        with open(label_file,'r') as f:
            lines = f.readlines()
            for i, line in enumerate(lines):
                doc_label[i+1] = int(line)
        return doc_label


    def create_label_name_map(self, group_file):
        label_name_map = {}
        with open(group_file,'r') as f:
            lines = f.readlines()
            for i, line in enumerate(lines):
                label_name_map[i+1] = line.strip()
        return label_name_map


    def find_average_similarity(self,groupA, groupB, sim_fun):
        sumscores = 0
        docsA = self.label_doc_map[groupA]
        docsB = self.label_doc_map[groupB]
        for idA in docsA:
            for idB in docsB:
                # print(idA,idB)
                sim = sim_fun(self.doc_bow[idA], self.doc_bow[idB])
                sumscores += sim
                # print(sim)

        return sumscores/(len(docsA)* len(docsB))
    def get_avg_sim_matrix(self, sim_fun):
        rows = []
        for l1 in self.labels:
            print(f'Processing for group {l1}')
            rows.append(
                [self.find_average_similarity(l1,l2, sim_fun) for l2 in self.labels]
            )
        return np.array(rows)

    def get_nn_matrix(self, sim_fun):
        rows = []
        for l1 in self.labels:
            nns = [0]*len(self.labels)#no nn at the beginning
            print(f'Processing for group {l1}')
            for idA in self.label_doc_map[l1]:
                sims = [-np.inf if self.doc_label_map[idB] == self.doc_label_map[idA]
                            else sim_fun(self.doc_bow[idA], self.doc_bow[idB])
                            for idB in range(1, len(self.doc_label_map)+1)]
                sims = np.array(sims)
                most_similar_doc = np.argmax(sims)+1
                most_similar_label = self.doc_label_map[most_similar_doc]
                nns[most_similar_label-1]+=1
            rows.append(nns)
        return np.array(rows)

def get_xy_forsim(x:Counter, y:Counter):
    keys =  set(x.keys()).union(set(y.keys()))
    x = np.array([x[dim] for dim in keys])
    y = np.array([y[dim] for dim in keys])
    return x,y
```

```
def jaccard(x:Counter, y:Counter):
    x,y = get_xy_forsim(x,y)#list of dims
    num = 0
    den = 0
    for xi,yi in zip(x,y):
        num += min(xi,yi)
        den += max(xi,yi)
    return num/den

def cosine(x:Counter, y:Counter):
    x,y = get_xy_forsim(x,y)#list of dims
    num = sum(x*y)
    den = np.linalg.norm(x)* np.linalg.norm(y)
    return num/den

def l2(x:Counter, y:Counter):
    x,y = get_xy_forsim(x,y)#list of dims
    # print(x,y)
    return -np.sqrt(
        ((x-y)**2).sum()
    )

if __name__ =='__main__':

    folder = 'data/'
    c = SimCalculator(folder+'data50.csv', folder+'label.csv', folder+'groups.csv')

    assert jaccard(
        x = Counter({0:2}),
        y = Counter({0:1})
    ) ==0.5

    print('cosine',cosine(
        x = Counter({0:2}),
        y = Counter({0:1})
    )) # should be near to 1

    print('L2',l2(
        x = Counter({0:2}),
        y = Counter({0:1})
    )) # should be near to -1

    #plot heatmap

    for metric in ['jaccard', 'cosine', 'l2']:
        print(metric)
        result = c.get_avg_sim_matrix(eval(metric))
        makeHeatMap(result, c.label_names, 'viridis', f'avg_similarity_{metric}.png')
        print(result)

    nn = c.get_nn_matrix(jaccard)

    makeHeatMap(nn, c.label_names, 'viridis', 'nn_heatmap.png')
    print(nn)
```