Applied Algorithms Assignment 9

Fei Fan (Peter) Chen

March 4, 2021

Exercise 1 (10 pts)

a)

$$
\begin{aligned}
dist(p1, p2) &= \sqrt{(-7-2)^2 + (-4-(-3))^2 + (6-1)^2 + (2-4)^2} = \sqrt{111} = 10.53 \\
dist(p1, p3) &= \sqrt{(-7-1)^2 + (-4-(-10))^2 + (6-1)^2 + (2-3)^2} = 3\sqrt{14} = 11.22 \\
dist(p1, p4) &= \sqrt{(-7-(-5))^2 + (-4-(-4))^2 + (6-(-6))^2 + (2-5)^2} = \sqrt{157} = 12.53 \\
dist(p2, p3) &= \sqrt{(2-1)^2 + (-3-(-10))^2 + (1-1)^2 + (4-3)^2} = \sqrt{51} = 7.14 \\
dist(p2, p4) &= \sqrt{(2-(-5))^2 + (-3-(-4))^2 + (1-(-6))^2 + (4-5)^2} = 10 \\
dist(p3, p4) &= \sqrt{(1-(-5))^2 + (-10-(-4))^2 + (1-(-6))^2 + (3-5)^2)} = 5\sqrt{5} = 11.18
\end{aligned}
$$

b)

$$
\begin{aligned}
p1 \cdot b1 &= 5.96 \\
p1 \cdot b2 &= 15.35 \\
p2 \cdot b1 &= 0.71 \\
p2 \cdot b2 &= 5.38 \\
p3 \cdot b1 &= 6.31 \\
p3 \cdot b2 &= 14.10 \\
p4 \cdot b1 &= -0.70 \\
p4 \cdot b2 &= 7.21
\end{aligned}
$$

c)

$$
\begin{aligned}
dist_r(p1, p2) &= \sqrt{(5.96-0.71)^2 + (15.35-5.38)^2} = 11.26 \\
dist_r(p1, p3) &= \sqrt{(5.96-6.31)^2 + (15.35-6.31)^2} = 9.04 \\
dist_r(p1, p4) &= \sqrt{(5.96-(-0.70))^2 + (15.35-7.21)^2} = 10.51 \\
dist_r(p2, p3) &= \sqrt{(0.71-6.31)^2 + (5.38-14.1)^2} = 10.36 \\
dist_r(p2, p4) &= \sqrt{(0.71-(-0.7))^2 + (5.38-7.21)^2} = 2.31 \\
dist_r(p3, p4) &= \sqrt{(6.31-(-0.7))^2 + (14-7.21)^2} = 9.75
\end{aligned}
$$

d)

$$
\begin{aligned}
p1 \cdot c1 &= -15 \\
p1 \cdot c2 &= 5 \\
p2 \cdot c1 &= 2 \\
p2 \cdot c2 &= 10 \\
p3 \cdot c1 &= -7 \\
p3 \cdot c2 &= 15 \\
p4 \cdot c1 &= 2 \\
p4 \cdot c2 &= -2
\end{aligned}
$$

e)

$$
\begin{aligned}
dist_r(p1, p2) &= \sqrt{(-15-2)^2 + (5-10)^2} = 17.72 \\
dist_r(p1, p3) &= \sqrt{(-15-(-7))^2 + (5-15)^2} = 12.81 \\
dist_r(p1, p4) &= \sqrt{(-15-2)^2 + (5-(-2))^2} = 18.38 \\
dist_r(p2, p3) &= \sqrt{(2-(-7))^2 + (10-15)^2} = 10.29 \\
dist_r(p2, p4) &= \sqrt{(2-2)^2 + (10-(-2))^2} = 12 \\
dist_r(p3, p4) &= \sqrt{(-7-2)^2 + (15-(-2))^2} = 19.23
\end{aligned}
$$

Exercise 2 (10 pts)

a) Assuming a QWERTY keyboard, higher costs will be assigned to letters that are farther away from each other on the keyboard. In fact we can assign cost based on the physical distance (number of keys away) between two letters on a QWERTY keyboard. The cost of addition or deletion would remain 1.

b) We assume we have a *cost(char A, char B)* function that returns the cost on a QWERTY keyboard for a substitution.

```
# Create a 2D array A of size len(stringA+1)*(len(stringB)+1)

Initialize A[0, :] = [ i for i in range(len(stringA+1))]

for i in len(stringA+1):
  for j in len(stringB+1):
    if letters are the same:
      edit_distance = A[i-1]B[i-1]
    if letters are different:
      edit_distance = min(
        A[i-1]B[j-1] + cost(stringA[i],stringB[i]),
        A[i-1]B[j] + 1,
        A[i]B[j-1] + 1,
    )
```

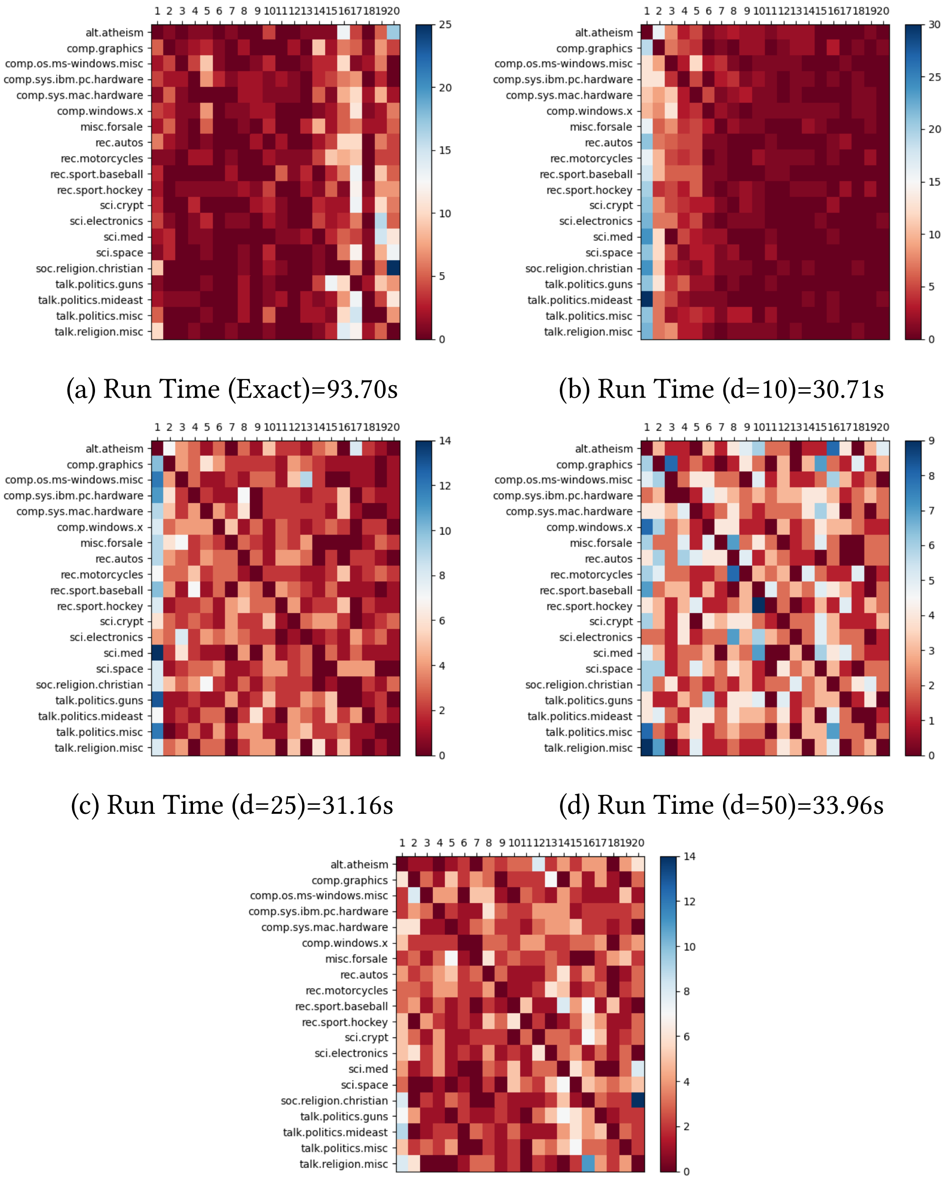
There's some rough optimization to not use $O(n^2)$ memory, but that is elided for simplicity.

Exercise 3 (20 pts)

a) The results in Figure 1 are pretty similar at d=100.

b) The greater the number of dimension in Figure 2 the more fidelity to the original similarity. When d=10, you can see that there is only so many values it can take (e.g., articles hashing to the same fingerprint). As d increases, it allows for a greater fidelity to the original exact cosine similarity. (Results were acquired using SimHash)

c) The results in Figure 3 seem slightly better, although it isn't exactly too noticeable. (Result obtained using SimHash).



(a) Run Time (Exact)=93.70s

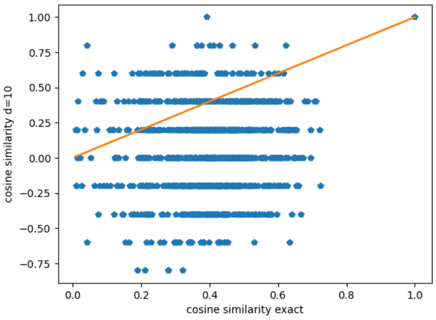
(b) Run Time (d=10)=30.71s

(c) Run Time (d=25)=31.16s

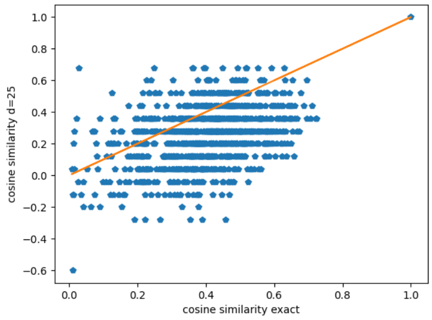
(d) Run Time (d=50)=33.96s

(e) Run Time (d=100)=32.36s

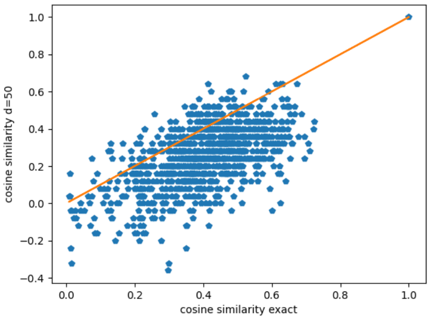
Figure 1: Heatmap of Reduced-Dimension Cosine Similarity



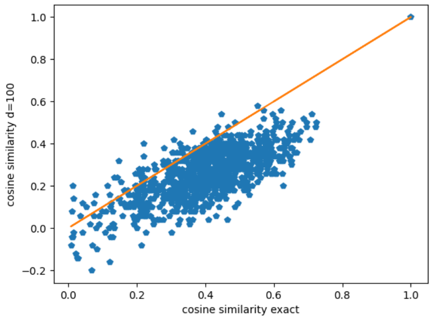
(a) Run Time (d=10)=1.76



(b) Run Time (d=25)=2.48s

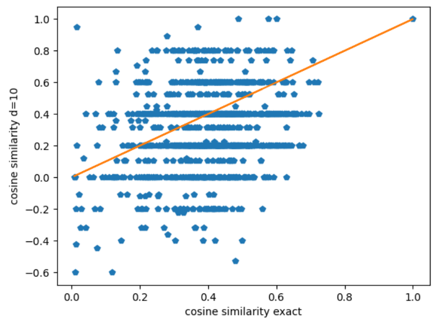


(c) Run Time (d=50)=3.83s

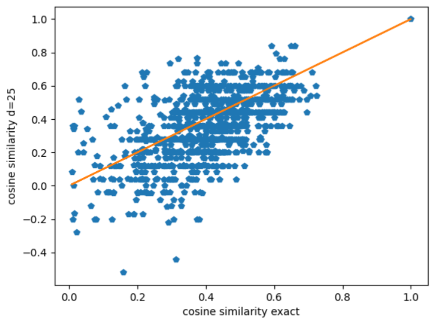


(d) Run Time (d=100)=5.82s

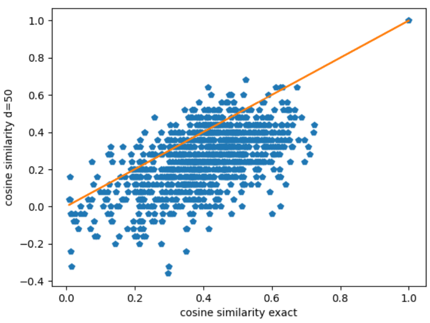
Figure 2: Scatter plot against alt.atheism article 3, M=gaussian



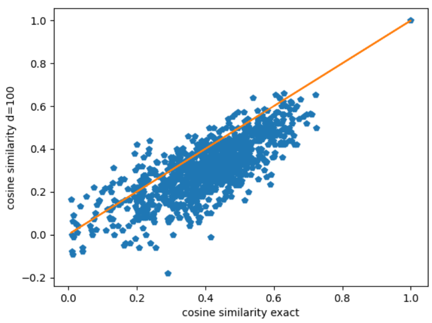
(a) Run Time (d=10)=1.91s



(b) Run Time (d=25)=3.01s



(c) Run Time (d=50)=3.81s



(d) Run Time (d=100)=5.94s

Figure 3: Scatter plot against alt.atheism article 3 M=-1,+1

Code

```
#!/usr/bin/python

import time
import random
import csv
import sys
import matplotlib.pyplot as plt
import math
import numpy as np
from collections import Counter
from heatmap import makeHeatMap

class Similarity():
  def __init__(self, d, reduction="jl"):
    # Parse the input files into a sparse matrix
    data_file = "data50.csv"
    label_file = "label.csv"
    groups_file = "groups.csv"

    n = 0
        # Data Structure
    # Group:
    #   Article: Counter( word: frequency )
    self.data = {}

    group = []
    with open(groups_file) as csvfile:
      groups_reader = csv.reader(csvfile, delimiter = '\n')
      for row in groups_reader:
        self.data[row[0]] = {}
        group.append(row[0])

    label = []
    with open(label_file) as csvfile:
      label_reader = csv.reader(csvfile, delimiter = '\n')
      for row in label_reader:
        label.append(int(row[0]))

    with open(data_file) as csvfile:
      data_reader = csv.reader(csvfile, delimiter = ',')
      for row in data_reader:
        row_id = int(row[0])
        group_id = label[row_id - 1]
        group_name = group[group_id - 1]
        if row_id not in self.data[group_name]:
          self.data[group_name][row_id] = Counter()
```

```
        self.data[group_name][row_id][int(row[1])]= int(row[2])
        if n < int(row[1]):
          n = int(row[1])

        # Data Structure
    # Group:
    #   Article: d-vector fingerprint
    if reduction == "jl":
      # use random normal hash
      M = np.random.normal(0,1, (n, d))
    elif reduction == "uniform":
      M = np.random.uniform(-1,1, (n,d))
      M[M>0] = 1
      M[M<=0] = -1
    self.d = d
    self.reduction = reduction
    self.data_reduced = {}
    for group, articles in self.data.items():
      if group not in self.data_reduced:
        self.data_reduced[group] = {}
      for article, c in articles.items():
        v = np.zeros(n)
        for index, val in c.items():
          v[index-1] = val
        h = v.dot(M)
        h[h>0] = 1
        h[h<0] = -1
        self.data_reduced[group][article] = h

  def plotMostSimilar(self):
    """
    Plot count of articles form B that are most similar
    to any article in A
    based on cosine similarity
    """
    categories = list(self.data_reduced.keys())
    similarity = np.zeros((len(categories), len(categories)))
    most_like_count = { c: {} for c in categories }

    for i in range(len(categories)):
      articlesA = self.data_reduced[categories[i]]
      for articleA, featuresA in articlesA.items():
        most_like_count[ categories[i] ][articleA] = (0, 'unk')
        for j in range(len(categories)):
          if i == j:
            continue
          articlesB = self.data_reduced[categories[j]]
          for articleB, featuresB in articlesB.items():
```

```
          s = self._cosine( featuresA, featuresB )
          if s > most_like_count[categories[i]][articleA][0]:
            most_like_count[categories[i]][articleA] = (s, j)

  for i in range(len(categories)):
    for article, most_similar_article
      in most_like_count[categories[i]].items():
      most_similar_article_category = most_similar_article[1]
      if most_similar_article_category != 'unk':
        similarity[i][most_similar_article_category] += 1

  makeHeatMap(similarity, categories, 'RdBu',
  f'similarity_mostlike_cosine_jl_{self.d}.png')

def scatterplot(self, article=3, group="alt.atheism"):
  ref_reduced_features = self.data_reduced[group][article]
  ref_features = self.data[group][article]

  x = []
  y = []
  for group, articles in self.data.items():
    for article, features in articles.items():
      s = self._cosine_exact( ref_features, features )
      x.append(s)
      s = self._cosine( ref_reduced_features,
         self.data_reduced[group][article] )
      y.append(s)

  plt.figure()
  plt.plot(x, y, 'p')
  plt.plot(x, x, '-')
  plt.ylabel(f"cosine similarity d={self.d}")
  plt.xlabel("cosine similarity exact")
  plt.savefig(f"scatter_{self.reduction}_{self.d}.png",
    format='png')

def _cosine_exact(self, A, B):
  X2 = 0
  Y2 = 0
  dotXY = 0

  checked = set()
  for word, count in A.items():
    Bcount = B[word]
    dotXY += Bcount * count
    X2 += count * count
    Y2 += Bcount * Bcount
    checked.add(word)
```

```
    for word, count in B.items():
      if word not in checked:
        Y2 += count * count

    return dotXY / ( math.sqrt(X2) * math.sqrt(Y2) )

  def _cosine(self, A, B):
    if np.linalg.norm(A) * np.linalg.norm(B) == 0:
      return 0.0
    return A.dot(B) / (np.linalg.norm(A) * np.linalg.norm(B))


if __name__ == "__main__":
  st = time.time()
  #s = Similarity(d=10)
  s = Similarity(d=100, reduction="uniform")
  #s.plotMostSimilar()
  s.scatterplot()
  print("runtime", time.time() - st
```