Applied Algorithms Assignment 8

Fei Fan (Peter) Chen

February 25, 2021

Exercise 1 (10 pts)

For simplicity, we expand the norm $\|(x, y)\|_{1/2} = |x|^2 + 2 \cdot \sqrt{|x||y|} + |y|^2$. A norm has to satisfy three conditions [1]:

Condition 1.

$$p(\mathbf{u} + \mathbf{v}) \leq p(\mathbf{u}) + p(\mathbf{v})$$
$$|x_1 + x_2| + 2 \cdot \sqrt{|x_1 - x_2||y_2 - y_2|} + |y_1 + y_2| \leq |x_1| + |x_2| + |y_1| + |y_2| + 2 \cdot \sqrt{|x_1||y_1|} + 2 \cdot \sqrt{|x_2||y_2|}$$

Each component satisfies the triangle inequality thus the norm satisfies triangle inequality.

Condition 2.

$$p(a\mathbf{u}) = |a|p(\mathbf{u})$$
$$|ax| + 2 \cdot \sqrt{|ax||ay|} + |ay| = |a|(|x| + 2 \cdot \sqrt{|x||y|} + |y|)$$

Condition 3.

$$p(\mathbf{u}) = 0 \ if \ \mathbf{u} = 0$$
$$|0| + 2 \cdot \sqrt{|0||0|} + |0| = 0$$

Therefore, $L_{1/2}$ is a valid norm.

Exercise 2 (10 pts)

Assume that X take the value of 1 if $X_k \in A \ and \ X_k \in B$, then:

$$
\begin{aligned}
E[A \bigcap B] &= E[\sum_{k=0}^{n} X_k] \\
&= \sum_{k=0}^{n} E[X_k] \\
&= \sum_{k=0}^{n} \frac{m}{n}\frac{m}{n} \\
&= n \cdot \frac{m^2}{n^2} \\
&= \frac{m^2}{n}
\end{aligned}
$$

Similarly, if we let X take the value of 1 if $X_k \in A \ or \ X_k \in B$, then:

$$
\begin{aligned}
E[A \bigcup B] &= E[\sum_{k=0}^{n} X_k] \\
&= \sum_{k=0}^{n} E[X_k] \\
&= \sum_{k=0}^{n} \frac{m}{n} + \frac{m}{n} \\
&= n \cdot \frac{2m}{n} \\
&= 2m
\end{aligned}
$$

Then the expected Jaccard similarity (given $m = \frac{n}{k}$) of A and B is:

$$
\begin{aligned}
\frac{|A \bigcap B|}{|A \bigcup B|} &= \frac{m}{2n} \\
&= \frac{1}{2k}
\end{aligned}
$$

Exercise 3 (10 pts)

We create a hash table of size n where $n << 2^k$. Each index i will be hashed to a position in the hash table. Each query point will first look in the hash table index of the nearest midpoint in the range of $|i2^{-j}, (i+1)2^{-j}|$. If a closet point is found, stop. If not, look at the hash table index of bucket i-1 and i+1.

Since there are only n buckets, with O(1) time hash operation, the expected number of buckets you need to search is at most n given that each index i has equal probability of being hashed to any bucket. Further, since there are n points and each has the probability of $\frac{1}{n}$ of hashing to any bucket, the expected number of items in each bucket is 1. Which means that the expected number of bucket access to find a candidate for nearest neighbor is 3 (one to find the bucket you are in and 2 to search the two buckets beside you). The farther out you go after first finding a candidate, the smaller your error gets until you arrive at a point that is actually in the bucket i you've just hashed (instead of an empty bucket i, but hashing to the same index in the hash table as a farther away point in another bucket since there are 2^k buckets for only n hash table indices).

However you can also use $k \cdot n$ space composed of multiple hash tables going from less granular (dividing the 1-D space in half), to more granular (dividing the 1-D space in quarters)...so on till 2^k buckets. This will allow you to locate exactly the buckets that actually have points close by. The expected number of bucket accesses here for any query would be k.

Exercise 4 (20 pts)

a) See code.

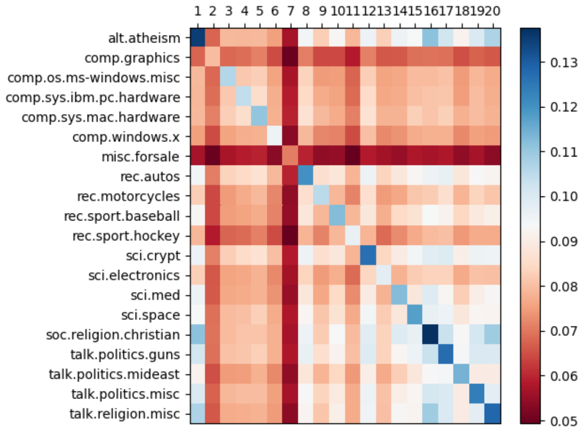
b) See Figures 1a, 1b and 1c.

c) From casual visual inspection, it seems that using cosine distance produces the most intuitive average similarity results. As we have some interesting related categories e.g., religion-based categories such as alt.atheism, soc.religion.christian, talk.religion.misc and clear lack of correlation for misc.forsale. The other two measures seem overly optimistic or pessimistic when judging similarity of articles between categories.

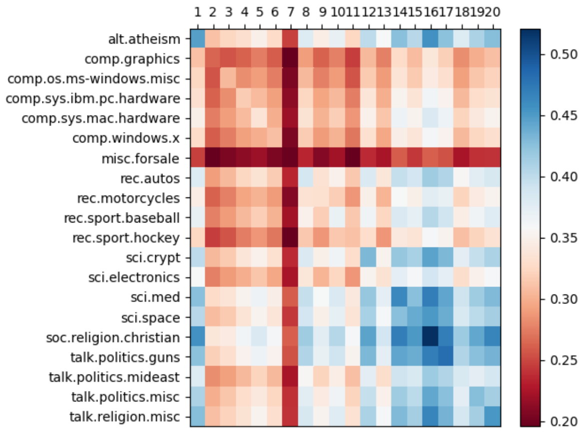
d) See Figure 2.

e) The plot for part b) looked at similarity of all articles between categories A and B, therefore it is symmetric. In part d) we looked at the count of articles that are most like the articles in A that are in B, this does not have to equal the count of articles that are most like articles in B that are in A, hence asymmetric.

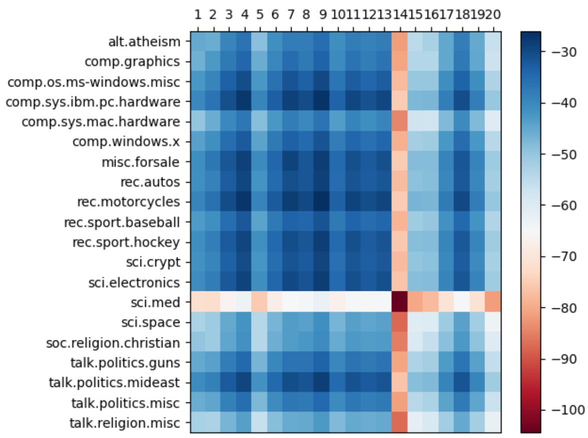
f) From d), religion and computer articles seem to have high similarity in the bag-of-words that they use. I would honestly use neither of these similarities as they all have pretty low recall or super low precision.



(a) Jaccard



(b) Cosine



(c) L2

Figure 1: Average Similarity based on various norms.

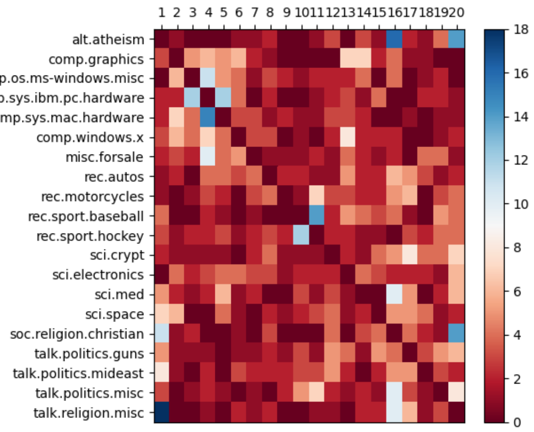


Figure 2: Most-likely article category count: Jaccard

Code

```
#!/usr/bin/python

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):
    # Parse the input files into a sparse matrix
    data_file = "data50.csv"
    label_file = "label.csv"
    groups_file = "groups.csv"

    # 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])

  def plotAvg(self, measure="jaccard"):
    """
    Plot average similarity between items in group A and B
    measure:
      jaccard: jaccard similarity
      cosine : cosine similarity
      l2     : L2 similarity
    """
    categories = list(self.data.keys())
    similarity = np.zeros((len(categories), len(categories)))
    for i in range(len(categories)):
      for j in range(len(categories)):
        articlesA = self.data[categories[i]]
        articlesB = self.data[categories[j]]
        avg = 0
        for articleA in articlesA:
          for articleB in articlesB:
            avg += self._norm( articlesA[articleA],
              articlesB[articleB], measure )
        avg = avg / (len(articlesA) * len(articlesB))

        similarity[i][j] = avg

    makeHeatMap(similarity, categories, 'RdBu',
      'similarity_avg' + measure + ".png")

  def plotMostSimilar(self):
    """
    Plot count of articles form B that are most
    similar to any article in A
    based on Jaccard similarity
    """
    categories = list(self.data.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[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[categories[j]]
          for articleB, featuresB in articlesB.items():
            s = self._norm( featuresA, featuresB, "jaccard" )
            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',
      'similarity_mostlike_jaccard.png')

  def _norm(self, A, B, measure):
    """
    Calculate the norm given measure
    """
    if measure == "jaccard":
      return self._jaccard(A, B)
    elif measure == "cosine":
      return self._cosine(A,B)
    elif measure == "l2":
      return self._l2(A,B)
    else:
      sys.exit("invalid measure")

  def _jaccard(self, A, B):
    minCount = {}
    maxCount = {}
    for word, count in A.items():
      if count > B[word]:
        minCount[word] = B[word]
        maxCount[word] = count
      else:
        minCount[word] = count
        maxCount[word] = B[count]
```

```
    for word, count in B.items():
      if count > A[word]:
        minCount[word] = A[word]
        maxCount[word] = count
      else:
        minCount[word] = count
        maxCount[word] = A[word]

    sumMin = 0
    sumMax = 0

    for _, count in minCount.items():
      sumMin += count

    for _, count in maxCount.items():
      sumMax += count

    return sumMin/sumMax

  def _cosine(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 _l2(self, A, B):
    checked = set()
    norm_sum = 0
    for word, count in A.items():
      Bcount = B[word]
      norm_sum += ( count - Bcount ) * ( count - Bcount )

    for word, count in B.items():
      if word not in checked:
```

```
        norm_sum += count * count

    return -math.sqrt(norm_sum)

if __name__ == "__main__":
    s = Similarity()
    s.plotAvg()
    s.plotAvg('l2')
    s.plotAvg('cosine')
    s.plotMostSimilar()
```

References

[1] Norm (Mathematics. https://en.wikipedia.org/wiki/Norm_(mathematics).