Worksheet 04

Name: Mohit Sai Gutha UID: U48519832

Topics

- Distance & Similarity

Distance & Similarity

Part 1

a) In the minkowski distance, describe what the parameters p and d are.

In the Minkowski distance formula, the parameter "p" must be equal to or greater than 1 and is subjet to choice. Specifically, "p" influences the type of distance calculated: when "p" equals 2, the distance corresponds to Euclidean Distance, and when "p" equals 1, it corresponds to Manhattan Distance. Meanwhile, "d" denotes the dimensionality of the space under consideration. For instance, if "d" equals 3, we are working within a three-dimensional space.

b) In your own words describe the difference between the Euclidean distance and the Manhattan distance.

Manhattan Distance measures the shortest distance in a grid based space. It is basically the minimum number of movements required to get from point A to point B moving along a grid like path (either in zig zag line or a straight line).

Euclidean Distance is the shortest distance between the two points i.e. the length of the line segment connecting the two points in 2D space.

Consider A = (0, 0) and B = (1, 1). When:

- p = 1, d(A, B) = 2
- p = 2, d(A, B) = $\sqrt{2}$
  $= 1.41$
- p = 3, d(A, B) = $2^{1/3}$
  $= 1.26$

- p = 4, d(A, B) = $\begin{aligned} &2^{1/4} \\ &= 1.19 \end{aligned}$

c) Describe what you think distance would look like when p is very large.

As p grows very large, the distance approaches 1.

d) Is the minkowski distance still a distance function when p < 1? Expain why / why not.

When p < 1, the minkowski distance isn't a distance function as it then violates the triangle inequality because in this situation the direct distance from B to C will be greater than the distance between B to C through A.

e) when would you use cosine similarity over the euclidan distance?

When the direction is of more importance to the context than the magnitude, we use cosine similarity over euclidan distance.

f) what does the jaccard distance account for that the manhattan distance doesn't?

Jaccard distance accounts for the differences of data sets in terms of the size of the intersection while the manhattan distance doesn't.

Part 2

Consider the following two sentences:

In [56]:

```
s1 = "hello my name is Alice"
s2 = "hello my name is Bob"
```

using the union of words from both sentences, we can represent each sentence as a vector. Each element of the vector represents the presence or absence of the word at that index.

In this example, the union of words is ("hello", "my", "name", "is", "Alice", "Bob") so we can represent the above sentences as such:

In [57]:

```
v1 = [1,      1, 1,     1, 1,      0]
```

```
v1 = [1,     1, 1,    1, 1,    0]
#      hello my name is Alice
v2 = [1,     1, 1,    1, 0, 1]
#      hello my name is     Bob
```

Programmatically, we can do the following:

In [58]:

```
corpus = [s1, s2]
all_words = list(set([item for x in corpus for item in x.split()]))
print(all_words)
v1 = [1 if x in s1 else 0 for x in all_words]
print(v1)
```

```
['Alice', 'hello', 'name', 'is', 'my', 'Bob']
[1, 1, 1, 1, 1, 0]
```

Let's add a new sentence to our corpus:

In [59]:

```
s3 = "hi my name is Claude"
corpus.append(s3)
```

a) What is the new union of words used to represent s1, s2, and s3?

In [60]:

```
all_words = list(set([item for x in corpus for item in x.split()]))
print(all_words)
```

```
['Alice', 'hello', 'name', 'Claude', 'is', 'hi', 'my', 'Bob']
```

b) Represent s1, s2, and s3 as vectors as above, using this new set of words.

In [61]:

```
v1 = [1 if x in s1 else 0 for x in all_words]
print(v1)
v2 = [1 if x in s2 else 0 for x in all_words]
print(v2)
```

```
v3 = [1 if x in s3 else 0 for x in all_words]
print(v3)
```

```
[1, 1, 1, 0, 1, 0, 1, 0]
[0, 1, 1, 0, 1, 0, 1, 1]
[0, 0, 1, 1, 1, 1, 1, 0]
```

c) Write a function that computes the manhattan distance between two vectors. Which pair of vectors are the most similar under that distance function?

In [62]:

```
def minkowski(x, y, p):
    if p < 1:
        raise ValueError("p must be greater than 1")
    if len(x) != len(y):
        raise ValueError("The dimensions are supposed to be the same for both x and y")
    res = 0
    for i in range(len(x)):
        res += abs(x[i]-y[i]) ** p
    return((res**(1/p)))

def manhattan(x, y):
    return minkowski(x, y, 1)

print(manhattan([0,0], [1,1]))
print(minkowski([0,0], [1,1], 2))
```

```
2.0
1.4142135623730951
```

d) Create a matrix of all these vectors (row major) and add the following sentences in vector form:

- **"hi Alice"**
- **"hello Claude"**
- **"Bob my name is Claude"**
- **"hi Claude my name is Alice"**
- **"hello Bob"**

In [63]:

```
s1 = "hi Alice"
s2 = "hello Claude"
s3 = "Bob my name is Claude"
s4 = "hi Claude my name is Alice"
s5 = "hello Bob"
corpus = [s1, s2, s3, s4, s5]
all_words = list(set([item for x in corpus for item in x.split()]))

matrix = []
for s in corpus:
    vector = [1 if x in s else 0 for x in all_words]
    matrix.append(vector)
    print(vector)
```

```
[1, 0, 0, 0, 0, 1, 0, 0]
[0, 1, 0, 1, 0, 0, 0, 0]
[0, 0, 1, 1, 1, 0, 1, 1]
[1, 0, 1, 1, 1, 1, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 1]
```

e) How many rows and columns does this matrix have?

In [64]:

```
rows = len(matrix)
columns = len(matrix[0])
print("The matrix has " + str(rows) + " rows and " + str(columns) + " columns.")
```

```
The matrix has 5 rows and 8 columns.
```

f) When using the Manhattan distance, which two sentences are the most similar?

In [65]:

```
minimum = manhattan(matrix[0], matrix[1])
vectors = (0, 1)
for i in range(1, len(matrix)):
    for j in range(i, len(matrix)):
        if i != j:
            distance = manhattan(matrix[i], matrix[j])
            if minimum > distance:
                minimum = distance
                vectors = (i, j)
print(f"When using Manhattan distance, sentences \"{corpus[vectors[0]]}\" and \"{corpus[vectors[1]]}\" are similar.")
```

```
When using Manhattan distance, sentences "hello Claude" and "hello Bob" are similar.
```

Part 3 Challenge

Given a set of graphs $\mathcal{G}$, each graph $G \in \mathcal{G}$ is defined over the same set of nodes V. The graphs are represented by their adjacency matrices, which are 2D arrays where each element indicates whether a pair of nodes is connected by an edge.

Your task is to compute the pairwise distances between these graphs based on a specific distance metric. The distance $d(G, G')$ between two graphs $G = (V, E)$ and $G' = (V, E')$ is defined as the sum of the number of edges in G but not in G', and the number of edges in G' but not in G. Mathematically, this can be expressed as:

$$d(G, G') = |E \setminus E'| \\ + |E' \setminus E|.$$

Requirements:

1. **Input: Should take a list of 2D numpy arrays as input. Each array represents the adjacency matrix of a graph.**
2. **Output: Should output a pairwise distance matrix. If there are n graphs in the input list, the output should be an $n \times n$ matrix where the entry at position (i, j) represents the distance between the i^{th} and j^{th} graph.**

In [66]:

```
import numpy as np

def compute_pairwise_distances(graphs):

    n = len(graphs) #number of graphs in the input list
    pairwise_distance_matrix = np.zeros((n, n), dtype=int) #initializing an n x n output matrix
    for i in range(n):
        for j in range(n):
            pairwise_distance_matrix[i, j] = np.sum(np.abs(graphs[i] - graphs[j]))

    return pairwise_distance_matrix

#my own test code as suggested by professor
#three 2x3 graphs
#expected output - [[0,2,3], [2,0,3], [3,3,0]]
graph1 = np.array([[0, 1, 0],
                    [1, 0, 1]])

graph2 = np.array([[0, 1, 1],
                    [1, 0, 0]])

graph3 = np.array([[0, 0, 1],
                    [0, 0, 1]])


input_list = [graph1, graph2, graph3]

# Compute pairwise distances
pairwise_distance_matrix = compute_pairwise_distances(input_list)
print("Pairwise Distance Matrix:")
print(pairwise_distance_matrix)
```

```
Pairwise Distance Matrix:
[[0 2 3]
 [2 0 3]
 [3 3 0]]
```