# Worksheet 04

Name: Jeong Yong Yang, Junho Son UID: U95912941, U64222022

## Topics

- Distance & Similarity

## Distance & Similarity

### Part 1

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

The parameter p is tune distance (parameter that we are able to tune). If p = 2, it is Euclidean distance and if p = 1, it is Manhattan distance.

The parameter d is the dimension. In other words, it represents d-dimensional space.

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

Manhattan distance measures the distance horizontally and vertically but cannot go diagonal so when measuring distance between two points, it has to add the horizonal and vertical aspects. However, Euclidean distance can measure the distance diagonally and can measure the direct line distance from two points. Therefore, Manhattan distance is always greater than or equal than the Euclidean distance.

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) = $2^{1/4} = 1.19$

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

As p goes very large, the distance between A and B decreases and would be closer to 1 since it becomes 2^(very small number). In other words, as p increases, the distance between A and B goes closely to 1 but never reaches 1.

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

No it cannot be a distance function when p < 1 because once p < 1, it violates the triangle inequality rule of d(i, j) <= d(i, k) + d(k, j).

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

We would use cosine dissimilarity over the Euclidean distance when direction matters more than magnitude.

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

Jaccard distance accounts for the size of intersection, which Manhattan distance does not take into account.

### Part 2

Consider the following two sentences:

```
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:

```
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:

```
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)
```

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

Let's add a new sentence to our corpus:

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

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

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

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

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

```
vector1 = [1 if x in s1 else 0 for x in union_of_words]
vector2 = [1 if x in s2 else 0 for x in union_of_words]
vector3 = [1 if x in s3 else 0 for x in union_of_words]

print(vector1)
print(vector2)
print(vector3)
```

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

```
def manhattanDistance(vec1, vec2):
  value = 0
  for x in (range(len(vec1))):
    value += abs(vec1[x] - vec2[x])
  return value

man1 = manhattanDistance(vector1, vector2)
man2 = manhattanDistance(vector1, vector3)
man3 = manhattanDistance(vector2, vector3)

print(man1)
print(man2)
print(man3)
```

```
2
4
4
```

Vector1 and vector2 (vector from "hello my name is Alice" and vector from "hello my name is Bob", respectively) are most similar under Manhattan distance.

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"

```
sen1 = "hi Alice"
sen2 = "hello Claude"
sen3 = "Bob my name is Claude"
sen4 = "hi Claude my name is Alice"
sen5 = "hello Bob"

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

```
8
['my', 'name', 'Claude', 'hi', 'is', 'Bob', 'Alice', 'hello']
```

```
#       my,   name,   Claude,   hi,   is,   Bob,    Alice,    hello
vec1 = [0,    0,      0,        1,    0,    0,      1,        0]

vec2 = [0,    0,      1,        0,    0,    0,      0,        1]

vec3 = [1,    1,      1,        0,    1,    1,      0,        0]

vec4 = [1,    1,      1,        1,    1,    0,      1,        0]

vec5 = [0,    0,      0,        0,    0,    1,      0,        1]

matrix_of_vecs = [vec1, vec2, vec3, vec4, vec5]
print(matrix_of_vecs)
```

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

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

```
print(f"There are {len(matrix_of_vecs)} rows in the matrix.")
print(f"There are {len(matrix_of_vecs[0])} columns in the matrix.")
```

```
There are 5 rows in the matrix.
There are 8 columns in the matrix.
```

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

```
sim_vec1 = []
sim_vec2 = []
index1 = 0
index2 = 0
smallestManD = manhattanDistance(vec1, vec2)
for vv1 in range(len(matrix_of_vecs)):
  for vv2 in range(vv1+1, len(matrix_of_vecs)):
    if manhattanDistance(matrix_of_vecs[vv1], matrix_of_vecs[vv2]) < smallestManD:
      index1 = vv1 + 1 # add 1 since index starts from 0
      index2 = vv2 + 1 # add 1 since index starts from 0
      sim_vec1 = matrix_of_vecs[vv1]
      sim_vec2 = matrix_of_vecs[vv2]
      smallestManD = manhattanDistance(matrix_of_vecs[vv1], matrix_of_vecs[vv2])
print(sim_vec1, sim_vec2)
print(index1, index2)
```

```
[0, 0, 1, 0, 0, 0, 0, 1] [0, 0, 0, 0, 0, 1, 0, 1]
2 5
```

Vector2 and vector5 (vector from "hello Claude" and vector from "hello Bob", respectively) are most similar under Manhattan distance.

## 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.

```python
def graph(graph):
  num_graphs = len(graph)
  distance_matrix = [[0 for _ in range(num_graphs)] for _ in range(num_graphs)]
  for x in range(len(graph)):
    for y in range(x+1, len(graph)):
      distance = 0
      for r in range(len(graph[x])):
        for c in range(len(graph[x][r])):
          if graph[x][r][c] != graph[y][r][c]:
            distance += 1
      distance_matrix[x][y] = distance
      distance_matrix[y][x] = distance
  return distance_matrix

gra1 = [[1, 1, 1],[0, 0, 1], [1, 0, 1]]
gra2 = [[1, 0, 0],[1, 1, 0], [0, 1, 1]]
gra3 = [[0, 0, 1],[0, 1, 0], [1, 1, 0]]

graphs = [gra1, gra2, gra3]
print(graph(graphs))

gra1 = [[1, 1, 1],[0, 0, 1], [1, 0, 1], [0, 0, 1]]
gra2 = [[1, 0, 0],[1, 1, 0], [0, 1, 1], [1, 1, 1]]
gra3 = [[0, 0, 1],[0, 1, 0], [1, 1, 0], [1, 0, 1]]

graphs = [gra1, gra2, gra3]
print(graph(graphs))

```

```
[[0, 7, 6], [7, 0, 5], [6, 5, 0]]
[[0, 9, 7], [9, 0, 6], [7, 6, 0]]
```

```

```