Worksheet 04

Name: Maha Alali UID: U84088912

Topics

- Distance & Similarity

Distance & Similarity

Part 1

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

- p: a generic parameter that we can choose and change depending on the objective we are trying to acheive.
- d:the dimensional space (fixed from the dataset)

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

- Euclidean distance: it is when p=2 in the Minkowski distance.
- Manhattan distance:it is when p=1 in the Minkowski 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 get's larger, the value d(b, c) in a triangle inequality d(b, c) <= d(a, b) + d(b, c) for any point a, b, c in a dataset would decrease and when p is infinite, it becomes a square (Chebyshev's distance).

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

No, becuase it violates the triangle inequality.

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

When direction matters more than magnitude.

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

The size of the data.

Part 2

Consider the following two sentences:

In [1]:
```
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 [2]:
```
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 [3]:
```
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', 'name', 'my', 'hello', 'is', 'Bob']
[1, 1, 1, 1, 1, 0]
```

Let's add a new sentence to our corpus:

In [4]:
```
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 [5]:
```
all_words = list(set([item for x in corpus for item in x.split()]))
print(all_words)
```

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

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

In [11]:
```
v1 = [1 if x in s1 else 0 for x in all_words]
print(v1, s1)
v2 = [1 if x in s2 else 0 for x in all_words]
print(v2, s2)
v3 = [1 if x in s3 else 0 for x in all_words]
print(v3, s3)
```

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

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 [ ]:

```
def minkowski(x, y, p):
    if (len(x) != len(y)):
        raise RuntimeError("x and y should be the same dimension")

    res = 0
    for i in range(len(x)):
        res += (abs(x[i] - y[i]))**p

    return res ** (1/p)

print(minkowski([0, 0], [1, 1], 2)) # expect sqrt of 2
print(minkowski([0, 0], [1, 1], 1)) # expect 2
```

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 [42]:

```
s1 = "hi Alice"
s2 = "hello my name is Bob"
s3 = "hi Alice"
s4 = "hello Claude"
s5 = "Bob my name is Claude"
s6 = "hi Claude my name is Alice"
s7 = "hello Bob"

corpus = [s1, s2, s3, s4, s5, s6, s7]
matrix = []
all_words = list(set([item for x in corpus for item in x.split()]))
print(all_words)

matrix = [[1 if x in s else 0 for x in all_words] for s in corpus]

for i in range(len(corpus)):
    print(matrix[i], corpus[i])


# matrix = [[item for _ in range(columns)] for _ in range(rows)]

# class Matrix:
#     def __init__(self):
#         self.matrix = [[item for _ in range(columns)] for _ in range(rows)]
#     def insert:
#     def remove:
```

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

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

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

In [28]:
```python
print("columns:", len(matrix[0]))

print("rows:", len(matrix))
```

```
columns: 8
rows: 7
```

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

In [63]:
```python
alldist = []
minpair = (corpus[0], corpus[1])
minval = float('inf')
minindex = (0, 0)
for i in range(7):
    for j in range(7):
        v1 = matrix[i]
        v2 = matrix[j]

        if i == j:
            continue

        dist = sum(abs(a - b) for a, b in zip(v1, v2))
        alldist += [[(corpus[i], corpus[j]), dist]]
        if dist < minval:
            minval = dist
            minpair = (corpus[i], corpus[j])
            minindex = ("s"+str(i+1),"s"+str(j+1))


print("\n\n")
print("The two most similar sentences are",minindex[0],"and",minindex[1],"=",mir

# print("\n\n")
# for i in range(len(alldist)):
#     print(alldist[i], "\n")
```

```


The two most similar sentences are s1 and s3 = ('hi Alice', 'hi Alice') with Man
hattan distance: 0
```

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.