Worksheet 04

Name: Rishab Sudhir

UID: U64819615

Topics

- Distance & Similarity

Distance & Similarity

Part 1

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

Minkowski Distance - This distance metric is a generalization of the Euclidean and Manhattan distance metrics.

p - hyperparameter (p >= 1)

d - the dimensionality (dimensional space), fixed by the data you're working with

when p = 2 -> euclidean distance

when p = 1 -> manhattan distance

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

Euclidean distance is a straight line distance from point a to b. Manhattan distance is the x y distance to from point a to b (like a triangle)

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.

When p is very large, the Minkowski distance will approach the maximum of the absolute differences of their coordinates. In the example, the differences are 1 and 1 along both dimensions, so the Minkowski distance will converge to 1, which is the maximum difference

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

When p is less than one then as p decreases the distance value increases. For values of p less than 1, the formula does not define a valid distance metric as the triangle inequality is not satisfied.

Let x:=(0,0) , y:=(1,1) , z:=(0,1) .

Then:

$d(x,y)$=2^{1/p},

which when p<1 is more than 2.

$d(x,z)=d(y,z)$=1.

So $d(x,y)>d(x,z)+d(z,y)$.

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

When direction matters more than magnitude. (if documents are of different sizes, direction gives us more information than magnitude)

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

The size of the document

Part 2

Consider the following two sentences:

In [4]:
```
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 [5]:
```
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 [6]:
```
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)
```

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

Let's add a new sentence to our corpus:

In [9]:
```
s3 = "hi my name is Claude"
corpus.append(s3)
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)
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)
```

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

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

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

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

In [ ]:
```
[1, 1, 0, 1, 1, 0, 1, 0]
[1, 1, 1, 0, 1, 0, 1, 0]
[1, 0, 0, 0, 1, 1, 1, 1]
```

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 [14]:
```
corpus = []
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)


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

    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 sq root 2
print( minkowski([0,0], [1,1], 1))
# expect 2
```

```
1.4142135623730951
2.0
```

Type *Markdown* and LaTeX: α^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 [19]:
```
mat = [
        ["hi","Alice"],
        ["hello","Claude"],
        ["Bob","my","name", "is","Claude"],
        ["hi","Claude","my","name","is", "Alice"],
        ["hello","Bob"]
      ]

#class matrix:
#    def __int__(self):
#        self.matrix = [[item for _ in range(columns)] for _ in range(rov

#    def insert:

#    def remove:
```

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

In [18]:
```
rows = 5
columns = 6
```

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

In [35]:
```
1 = "hi Alice"
2 = "hello Claude"
3 = "Bob my name is Claude"
4 = "hi Claude my name is Alice"
5 = "hello Bob"
orpus = ["hi Alice","hello Claude","Bob my name is Claude","hi Claude my
ll_words = list(set([item for x in corpus for item in x.split()]))
rint(all_words)
1 = [1 if x in s1 else 0 for x in all_words]
2 = [1 if x in s2 else 0 for x in all_words]
3 = [1 if x in s3 else 0 for x in all_words]
4 = [1 if x in s4 else 0 for x in all_words]
5 = [1 if x in s5 else 0 for x in all_words]

at = [v1,v2,v3,v4,v5]
im1 = mat[0]
im2 = mat[0]
andist = 100000
or x in range(0, len(mat)):
   for y in range(x+1,len(mat)):
       dist = minkowski(mat[x], mat[y], 1)
       if dist < mandist:
           sim1 = corpus[x]
           sim2 = corpus[y]
           mandist = dist

rint("the most similar sentances are ''" + str(sim1) + "' and '" + str(si
```

```
['name', 'hello', 'Bob', 'Alice', 'is', 'Claude', 'my', 'hi']
the most similar sentances are ''hello Claude' and 'hello Bob' with a m
an dist of 2.0
```

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