index

March 14, 2022

1 Distance Metrics

1.1 Introduction

In this lesson, you'll learn about various kinds of distance metrics that you can use as a way to quantify similarity!

1.2 Objectives

You will be able to:

- Calculate Manhattan distance between two points
- Calculate Euclidean distance between two points
- Compare and contrast Manhattan, Euclidean, and Minkowski distance

1.3 Relationship between similarity and distance

In this section, you'll be focusing on a foundational ***Supervised Learning*** algorithm, the ***K-Nearest Neighbors***, or ***KNN*** for short. In order to understand how this algorithm works, you first have to understand some ***distance metrics***, and how you can use them to tell us how similar two objects are.

The assumption that distance-based classifiers like KNN are built on is that ***distance helps us quantify similarity***. Objects that are more alike are more likely to be the same class. By treating each column in your dataset as a separate dimension, you can plot each data point that you have and measure the distance between them!

You might remember learning about how to calculate the distance between two points on a Cartesian plane from school. That's certainly an important distance metric that you'll be using – however, there is more than one distance metric you can use, and all of them can be useful depending on the context of the problem. In this lesson, you'll learn how to calculate different distance metrics so that you have a tool to evaluate how similar or different data points are from one another when we begin exploring the KNN algorithm!

1.4 Manhattan distance

The first (and easiest) distance metric you'll cover is ***Manhattan distance***. Manhattan distance is aptly named, because it measures the distance from one point to another traveling along the axes of a grid. Take a look at the following image:

In this image, all the lines except for the green line measure the Manhattan distance between the two points. You'll notice that if you start from the point in the bottom left corner of the grid and count the number of units moved in the X dimension (left and right) and the number of units moved in the Y dimension (up and down), the distance is the for the red, blue, and yellow lines.

The easiest way to remember Manhattan distance is to use the analogy that provides this distance metric it's name – look at the picture above, but picture this grid as the famous grid of streets in Manhattan. If you were to take a taxi from point A (bottom left) to point b (top right), how many blocks east or west would the taxi have to travel? How many blocks north or south? How many blocks total? By calculating the total number of blocks we have to drive, we're actually calculating the Manhattan distance between point A and point B.

So far, this discussion has explored Manhattan distance in a 2-dimensional space. However, all of the distance metrics you're going to learn can generalize to an n-dimensional space. For instance, in 3 dimensions, it's no harder to calculate the distance from one square on a Rubik's Cube to any other square – all you do is take into account how many squares we need to move towards or away from ourselves to measure depth, as well as left/right and up/down. Once you know the total units you need to move in each of these 3 dimensions, you just sum them to calculate the Manhattan distance in 3 dimensions.

Here's the formula for Manhattan distance:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Let's break this formula down:

The left side of the equals sign just means "the distance between point x and point y".

The right side of the equals sign looks like it has a bit more going on, but don't let the mathematical notation scare you. In basic English, it essentially means "calculate the absolute number of units you move in each distinct dimension, and then sum them all up".

The $\sum$ just means "the cumulative sum of each step". In a given step, you take a dimension, and then look at the corresponding values for that dimension on point X and point Y. You then compute the absolute value of the difference between them by subtracting Y's value for that dimension from X's value for that dimension, and then add it to our total.

If the rationale here seems unclear to you, think of it this way – from a practical standpoint, walking 3 blocks to the west is the same as walking 3 blocks to the east. However, when you're doing this on a grid, you still want movements in the opposite (negative) direction to count! In order to make them count, you calculate the absolute difference between them. This makes it so that a move from 0 to -3 and a move from 0 to 3 measure the same amount of distance.

Let's think about what this would look like in code. Let's assume that each point is stored as a tuple of values. If you wanted to calculate the distance between them, you can easily do this with a `for` loop!

[1]:
```
# Locations of two points A and B
A = (2, 3, 5)
B = (1, -1, 3)
```

```
manhattan_distance = 0

# Use a for loop to iterate over each element
for i in range(3):
    # Calculate the absolute difference and add it
    manhattan_distance += abs(A[i] - B[i])

manhattan_distance
```

[1]: 7

1.4.1 A hint on turning mathematical notation into code

Anytime you see mathematical notation that includes the $\sum$ symbol, this can reliably be represented as a `for` loop! The math to the right of the symbol tells you what the body of the `for` loop should look like (what unique things you're doing in each given step). Similarly, the numbers on the bottom and top of the $\sum$ sign tell you the starting and stopping indexes, respectively. In the case of the Manhattan distance equation above, the n means "length n", the length of the entire number of dimensions, because we want to count them all in our calculations.

> ***NOTE***: Be careful interpreting the starting dimensions. Remember that although mathematicians start counting at 1, computer scientists start counting at 0! Whereas from a mathematical perspective, you are starting at dimension 1 and counting up to and including dimension 3, in your code, you start at dimension 0 and count up to and including dimension 2.

1.5 Euclidean distance

The next and most common distance metric is ***Euclidean distance***. The equation at the heart of this one is probably familiar to you: $a^2 + b^2 = c^2$, or the ***Pythagorean theorem***! Take a look at the diagram above with all the colored lines on it. The green line measures the Euclidean distance between the two points, by moving in a straight line. If you take the horizontal and vertical sections of the red line and treat them as a and b, then you can easily use the Pythagorean theorem to calculate the length of the green line to calculate the euclidean distance between the two points.

In this case, that is: $6^2 + 6^2 = \sqrt{72} \approx 8.485$

1.5.1 Working with more than two dimensions

Just as with Manhattan distance, you can generalize this equation to n dimensions. You just add any extra dimensions in, same as we did with the first two. For instance, if you wanted to calculate the Euclidean distance between two points in a 3-dimensional space, then the formula would be $d^2 = a^2 + b^2 + c^2$, with d being the Euclidean distance.

Let's take a look at formula for the Euclidean distance:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

This is a straightforward equation – for each dimension, you subtract one point's value from the other's to get the length of that "side" of the triangle in that dimension, square it, and add it to our running total. The square root of that running total is our Euclidean distance.

In Python, you can easily calculate Euclidean distance as follows:

[2]:
```
from math import sqrt

# Locations of two points A and B
A = (2, 3, 5)
B = (1, -1, 3)

euclidean_distance = 0

# Use a for loop to iterate over each element
for i in range(3):
    # Calculate the difference, square, and add it
    euclidean_distance += (A[i] - B[i])**2

# Square root of the final result
euclidean_distance = sqrt(euclidean_distance)

euclidean_distance
```

[2]:
```
4.58257569495584
```

1.6 Minkowski distance

A final distance metric you should be familiar with is ***Minkowski distance***. The Minkowski distance is a generalized distance metric across a *Normed Vector Space*. A Normed Vector Space is just a fancy way of saying a collection of space where each point has been run through a function. It can be any function, as long it meets two criteria: 1. the zero vector (just a vector filled with zeros) will output a length of 0, and 2. every other vector must have a positive length

Don't worry too much about the specifics of the mathematical definition above. Instead, try to gain an intuition for what Minkowski distance actually measures. Both the Manhattan and Euclidean distances are actually *special cases of Minkowski distance*. Take a look:

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^c\right)^{\frac{1}{c}}$$

Do you see it? It's the exponent! In this case, the function is just an exponent. If you were to define a value for the exponent, you could say that:

Note to myself: The equations are not correct

```
# Manhattan Distance is the sum of all side lengths to the first power
manhattan_distance = np.power((length_side_1 + length_side_2 + ... length_side_n)**1 , 1/1)
```

```
# Euclidean Distance is the square root of the sum of all side lengths to the second power
euclidean_distance = np.power((length_side_1 + length_side_2 + ... length_side_n)**2, 1/2)

# Minkowski Distance with a value of 3 would be the cube root of the sum of all side lengths t
minkowski_distance_3 = np.power((length_side_1 + length_side_2 + ... length_side_n)**3, 1/3)

# Minkowski Distance with a value of 5
minkowski_distance_5 = np.power((length_side_1 + length_side_2 + ... length_side_n)**5, 1/5)
```

> **NOTE**: You'll often see Minkowski distance used as a parameter for any distance-based machine learning algorithms inside `sklearn`.

1.7 Summary

Knowing what you now know about Minkowski distance, and its two special cases, Manhattan and Euclidean distances, think about how you could write a generalized distance function that can calculate any of these, because it's exactly what you'll be doing in our next lab!

[ ]: