

## Day 14: $k$ -Nearest Neighbors

# How are homes appraised?

When a house is bought or sold, an appraiser typically evaluates the expected value of the home.

- Number of bedrooms
- House amenities
- Square footage
- Many other features...

The features of the marketed home are then compared to similar homes that have sold recently.

# What do we mean by similar?

How do we measure similarity?

We can measure it as a distance!

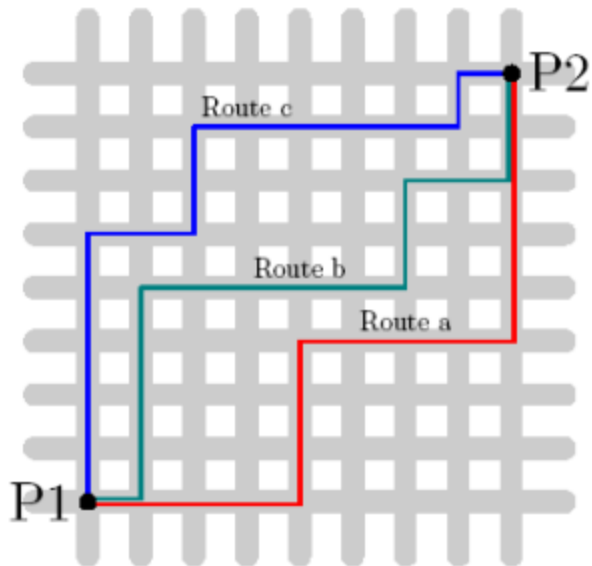
# How do we measure distance?

- City blocks
- "As the crow flies" (shortest line)
- Distance on a sphere (ie - shortest flight path)
- Travel time

# City Blocks

The measurement of distance by city blocks is frequently referred to as **manhattan distance**.

$$\text{Manhattan Distance} = X \text{ Blocks} + Y \text{ Blocks}$$



# Calculating Manhattan Distance

## Exercise:

Given two points with  $n$ -dimensional coordinates, generate a function that will return the manhattan distance between those two points.

**Bonus:** Include a check to make sure that each vector has the same dimensionality.

# Exercise Answer

```
import numpy as np

def manhattan(p1, p2):
    d = 0
    for i in range(len(p1)):
        d+=np.abs(p1[i]-p2[i])
    return d
```

OR

```
def manhattan(p1,p2):
    return np.sum([np.abs(p1[i]-p2[i]) for i in range(len(p1))])
```

Note that we need to use the absolute value, since negative distances in a given dimension must still be travelled in the same way as positive distances (no wormholes here).

# As the crow flies...

This is the measurement that we most often think of as distance. It is referred to as **Euclidean distance**, and is calculated with the Pythagorean Equation.

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^N (x_{i1} - x_{i2})^2}$$

In two dimensions:

$$\begin{aligned} &= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \\ &= \sqrt{a^2 + b^2} \end{aligned}$$



# Calculating Euclidean Distance

## Exercise:

Given two points with n-dimensional coordinates, generate a function that will return the euclidean distance between those two points.

# Exercise Answer

```
import numpy as np

def euclidean(p1, p2):
    d = 0
    for i in range(len(p1)):
        d+=(p1[i]-p2[i])**2
    return np.sqrt(d)
```

OR

```
def euclidean(p1,p2):
    return np.sqrt(np.sum([(p1[i] - p2[i])**2 for i in range(len(p1))]))
```

# Nearest Neighbor

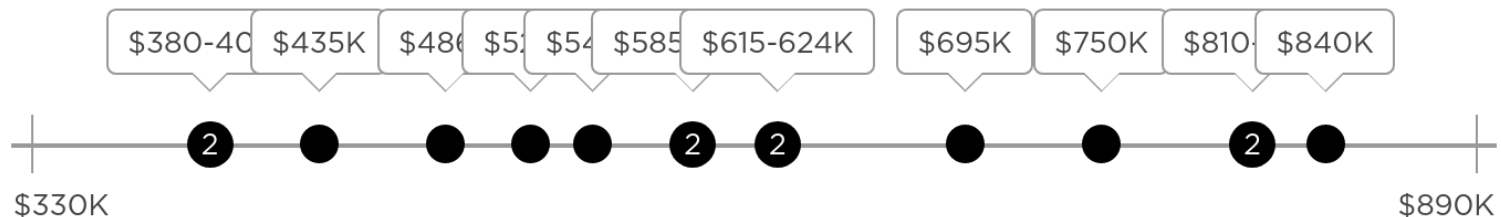
One way to make inference about a new, **unlabeled** observation is to compare it to the most similar (and labeled) observation(s).

- Do this using distance metrics!
- Find the observation with the smallest **distance**, and make inference about our new point

## Price This Home



Homes like this sold for \$380-840K.



# $k$ -Nearest Neighbors

What if there are a lot of similar observations? We can choose a number of comparisons to make! This algorithm is called  $k$ -Nearest Neighbors. If  $k = 1$ , then we simply compare the single nearest observation.

- Increasing  $k$  will decrease variance (overfitting), but may also increase bias

# Using $k$ -Nearest Neighbors

**Step 1** - Collect all labeled data and store as "coordinates" of observations, with each label as the value at a given coordinate.

- We don't have to do ANY up-front calculations or modeling when we use  $k$ -Nearest Neighbors
- We have no way of knowing which stored observations will matter until we see the coordinates of the test observation

# Using $k$ -Nearest Neighbors

**Step 2** - When you receive a test observation (or many), calculate the distance from the new observation to **every** stored observation.

**Step 3** - Sort the distances, and select the  $k$  observations with the lowest distance value.

**Note:** The calculations that must be performed for every test observation are the same, and the estimation is therefore "computationally expensive."

# Using $k$ -Nearest Neighbors

**Step 4** - Use some sort of average (typically weighted by distance) of the outcomes for the  $k$  nearest neighbors of the new observation to determine the predicted label of the new observation.

# Using $k$ -Nearest Neighbors

## Positives:

- No up-front training necessary!
- Have control over how many observations affect prediction

## Negatives:

- Relatively slow to generate a label, since all training must be done *after* observing test data
- Fitting must be done for **every** new observation



# kNN in Python

```
# Import our typical libraries, and the kNN Classifier
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Import our data, and separate our dependent variable
data = pd.read_csv("passFailTrain.csv")
y = data['G3']
x = data.drop(['Unnamed: 0', 'G3'], axis=1)
```

# kNN in Python

```
# Create train and testing data
x, xt, y, yt = train_test_split(x, y,
                                test_size=0.1, random_state=42)
# Declare our classifier and its parameters
model = KNeighborsClassifier(n_neighbors=10,
                             metric='euclidean')
# 'Fit' the model to the data
reg = model.fit(x, y)
# Generate predicted labels for our test data
pred = reg.predict(xt)
# Calculate accuracy score
accuracy_score(pred, yt)
```

Produces an accuracy of 83.3%



# Dawn of The Final Day

~~24 Hours Before~~

EXAM!

# Identifying Terrorists

The Tehrik-i-Taliban (TTP) have been an active terrorist organization in Pakistan for years. Using data on terrorist attacks (some labeled, some not), you need to identify whether or not **unclaimed** terrorist attacks were perpetrated by the TTP.

# Let's look at the data

Data and a dictionary of variables are available in Canvas

# The rubric

Also in Canvas

# Lab Time!

Good luck! 😊