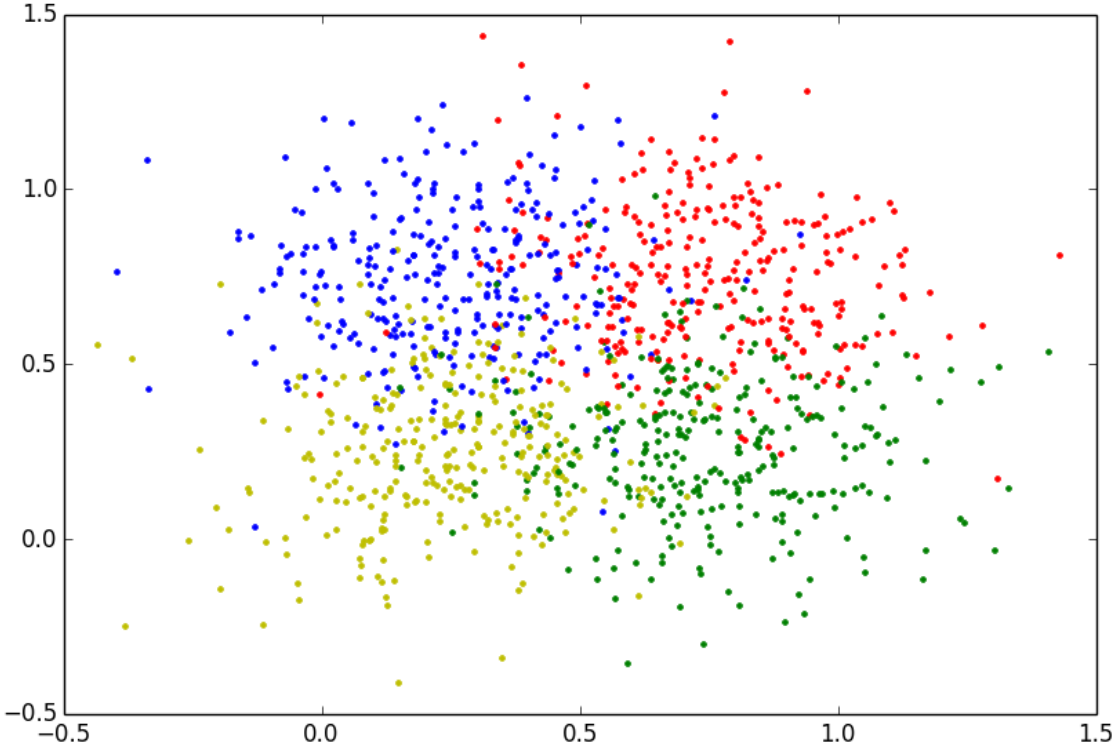
# Introduction to Data Science

## K-NEAREST NEIGHBORS

BRIAN D'ALESSANDRO

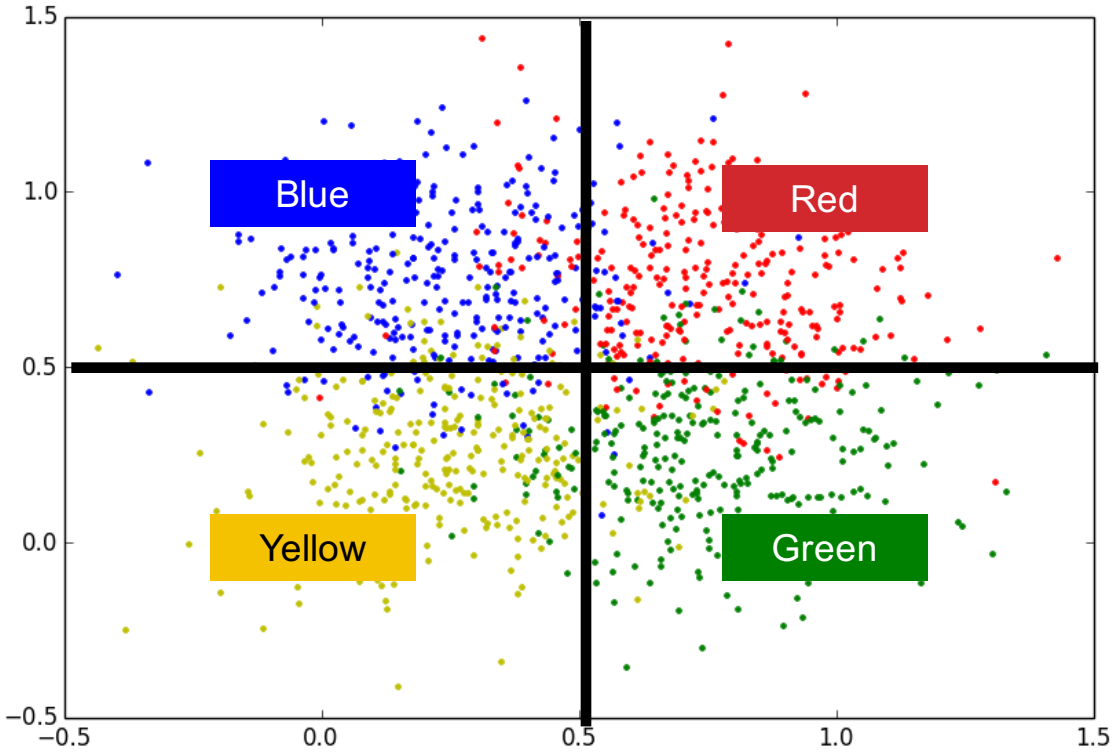
# THE IDEA

How would you decide what color to give a point?



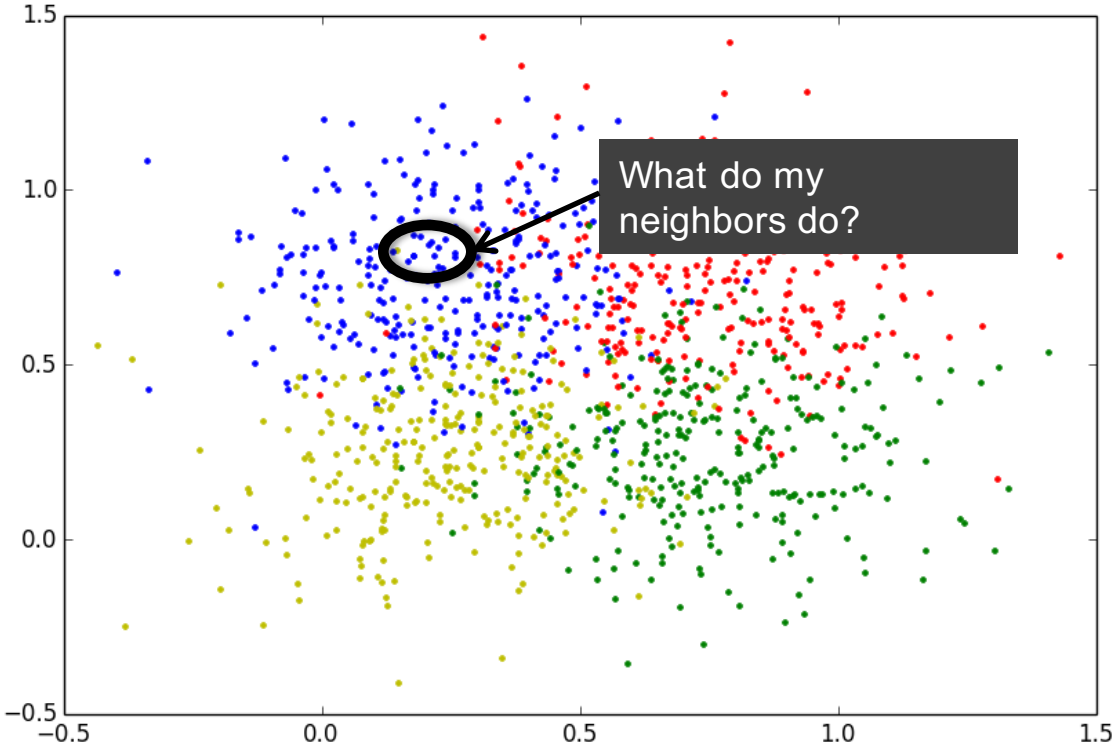
# THE IDEA

Many algorithms focus on the shape/structure of the decision boundary and use functions or rules to draw them



# THE IDEA

K-NN on the other hand ignores any global structure and just focuses on local information.



# DEFINITION

More formally…

If we have a data set $T=<X,Y>$, where $X=<x_1,x_2,\ldots x_k>$ is a vector of $k$ features and $Y$ is a real valued number (either a continuous number or binary indicator of class membership).

Let $N_k(x)$ be the neighborhood of the k nearest neighbors in some metric space to an instance of interest. Then the regression or classification estimate for the given instance is:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

# SOME QUALITIES

## Non-parametric

You don't have to make any assumptions about the functional form that estimates *E[Y|X].* This makes it very powerful for estimating any arbitrary decision curve, but extreme flexibility always risks overfitting.

## Instance-based learning

You technically don't need to train this. Estimation of *E[Y|X]* is done locally and a scoring time by taking the average of *Y* of the k nearest neighbors of the instance. There is effectively no model, just all of the training data stored in memory.

# WHAT IS A NEIGHBORHOOD?

A *neighborhood* is a set of examples that are *close* to the given instance.

To be *close* we need some *metric* that defines *distance* between two examples.

# DISTANCE METRIC

Definition: A metric is a function that defines a distance between elements of a set.

Properties:

Given two points a and b, and a distance function d()

1. $d(a,b) \geq 0$ … non-negativity
2. $d(a,b)=0$ only if and only if $a=b$
3. $d(a,b)=d(b,a)$ … symmetry
4. $d(a,c) \leq d(a,b)+d(b,c)$ … triangle inequality

*Understanding these properties is helpful for using and validating various available distance metrics.*

*The metrics we'll explore in our examples is Euclidean Distance, though Hamming Distance and Mahalanobis Distance are often used.*

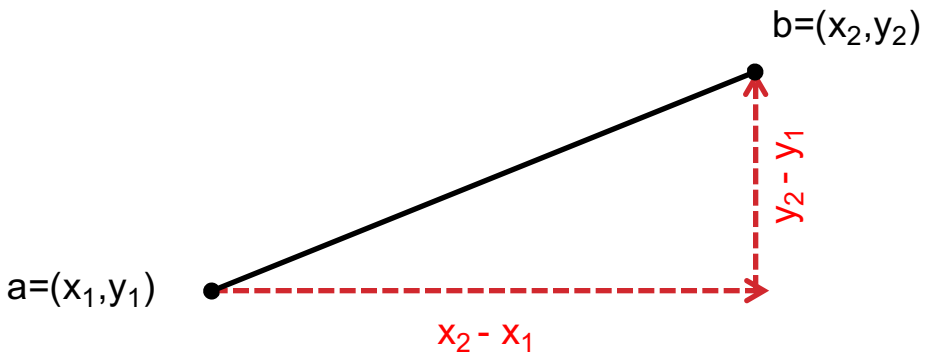
# EUCLIDEAN DISTANCE

This metric derives from basic geometry, and is the way distance is often defined in physical coordinate systems.

Let ***a*** and ***b*** be two k-dimensional vectors in Euclidean space. The Euclidean distance between them is defined as:

$$d(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_k - b_k)^2} = \sqrt{\sum_{i=1}^{k} (a_i - b_i)^2}$$

The two dimensional case is the famous Pythagorean Theorem:



# IMPLEMENTING THE "MODEL"

1. Build a training set – must have a label and numeric features.

| User | Y | X1 | X2 | X3 |
|---|---|---|---|---|
| 1 | 1 | 0.64 | 0.72 | 0.48 |
| 2 | 0 | 0.02 | 0.01 | 0.21 |
| 3 | 0 | 0.19 | 0.20 | 0.87 |
| 4 | 0 | 0.39 | 0.16 | 0.29 |
| 5 | 1 | 0.87 | 0.69 | 0.17 |
| 6 | 0 | 0.41 | 0.09 | 0.54 |
| 7 | 0 | 0.54 | 0.46 | 0.12 |
| 8 | 1 | 0.97 | 0.26 | 0.35 |
| 9 | 0 | 0.56 | 0.52 | 0.35 |
| 10 | 1 | 0.96 | 0.29 | 0.84 |

# IMPLEMENTING THE "MODEL"

2. Choose a non-training instance and compute distance (we'll use Euclidean) from every member of the training set.

| User | Y | X1 | X2 | X3 |
|---|---|---|---|---|
| 11 | ? | 0.57 | 0.43 | 0.95 |

| User | Y | X1 | X2 | X3 |
|---|---|---|---|---|
| 1 | 1 | 0.64 | 0.72 | 0.48 |
| 2 | 0 | 0.02 | 0.01 | 0.21 |
| 3 | 0 | 0.19 | 0.20 | 0.87 |
| 4 | 0 | 0.39 | 0.16 | 0.29 |
| 5 | 1 | 0.87 | 0.69 | 0.17 |
| 6 | 0 | 0.41 | 0.09 | 0.54 |
| 7 | 0 | 0.54 | 0.46 | 0.12 |
| 8 | 1 | 0.97 | 0.26 | 0.35 |
| 9 | 0 | 0.56 | 0.52 | 0.35 |
| 10 | 1 | 0.96 | 0.29 | 0.84 |

| D |
|---|
| 0.56 |
| 1.02 |
| 0.46 |
| 0.74 |
| 0.88 |
| 0.56 |
| 0.83 |
| 0.74 |
| 0.61 |
| 0.43 |

# IMPLEMENTING THE "MODEL"

3. Find the k nearest neighbors (and k should be chosen in advance), and average the labels Y in the training set.

$$E[Y|X_{11}] = 1/3$$

| User | Y | X1 | X2 | X3 |
|---|---|---|---|---|
| 11 | ? | 0.57 | 0.43 | 0.95 |

| User | Y | X1 | X2 | X3 | D |
|---|---|---|---|---|---|
| 1 | 1 | 0.64 | 0.72 | 0.48 | 0.56 |
| 2 | 0 | 0.02 | 0.01 | 0.21 | 1.02 |
| 3 | 0 | 0.19 | 0.20 | 0.87 | 0.46 |
| 4 | 0 | 0.39 | 0.16 | 0.29 | 0.74 |
| 5 | 1 | 0.87 | 0.69 | 0.17 | 0.88 |
| 6 | 0 | 0.41 | 0.09 | 0.54 | 0.56 |
| 7 | 0 | 0.54 | 0.46 | 0.12 | 0.83 |
| 8 | 1 | 0.97 | 0.26 | 0.35 | 0.74 |
| 9 | 0 | 0.56 | 0.52 | 0.35 | 0.61 |
| 10 | 1 | 0.96 | 0.29 | 0.84 | 0.43 |

# CHALLENGES

## Expensive Search

The algorithm becomes more accurate when the number of samples N increases. However, as N increases the search cost increases. Scoring a new instance requires $O(N_{train})$ searches when a brute force search is used (although there are fast approximate search methods that solve this in practice).

When SkLearn 'fits' a KNN model, it is not fitting in the ERM sense. Rather, it parses the data using tree search methods to allow for efficient nearest neighbor search upon scoring.

**algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'***

Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

# CHALLENGES

## Distance Metrics

**The Curse of Dimensionality (CODA)** – when using Euclidean distance (as well as other Lp-norm distance measures, avg. distances increase as dimensionality increases. In a high-dimensional space, most parts are far from all other points.

**Scale matters** – features with higher avg. magnitude tend to dominate distance metrics. It may be required to normalize data first.

Since Euclidian Distance is the sum of d non-negative terms, we can see why distance increases with *d.*

Also, if we were to multiply a feature by 1000, its predictive power does not change, but we can see from above how it would dominate the distance formula

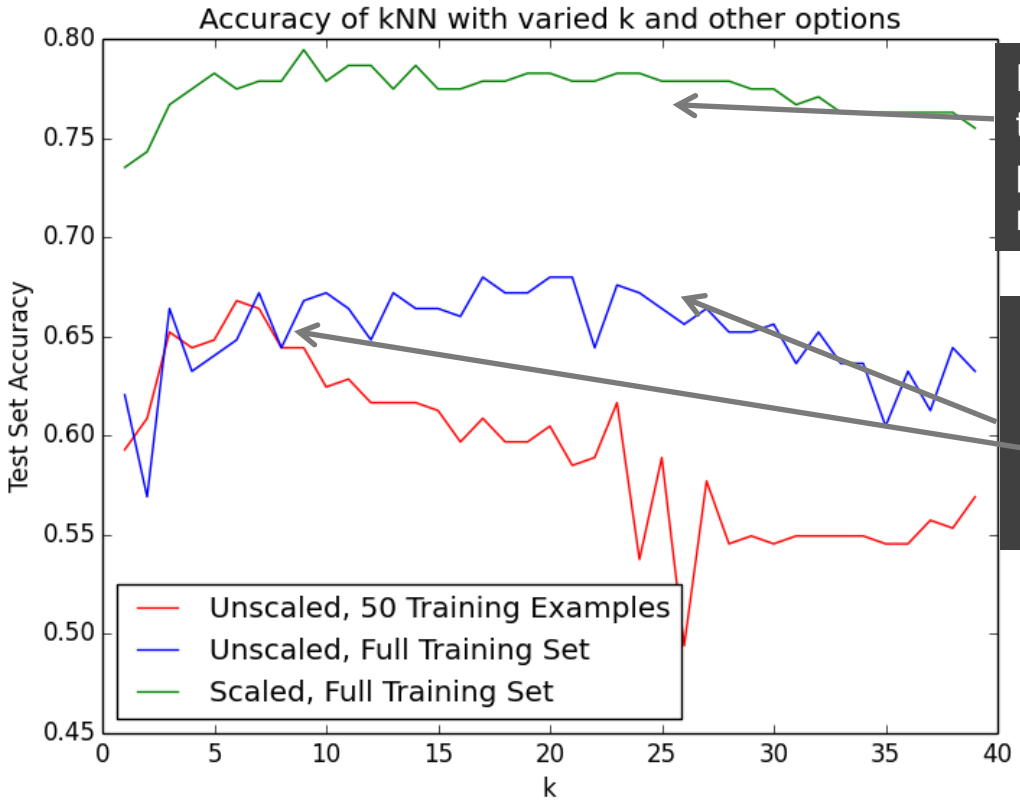
$$d(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_k - b_k)^2} = \sqrt{\sum_{i=1}^{k} (a_i - b_i)^2}$$

Let $\text{dist}_{\text{max}}$ and $\text{dist}_{\text{min}}$ be the farthest and closest point to the center. As *d* increases the ratio of the difference between max and min to the min tends to 0. This means all points are equally far apart, leaving no nearest neighbors to learn from.

$$\lim_{d \to \infty} \frac{\text{dist}_{\text{max}} - \text{dist}_{\text{min}}}{\text{dist}_{\text{min}}} \to 0$$

# EXAMPLE – KNN CLASSIFIER

This plot shows accuracy vs. k on a test set for kNN classification with several design options. This sort of analysis is how we would normally choose design parameters and hyper-paramaters.




Rescaling the data performs much better

The optimal k depends on how much training data we have.