k-Nearest Neighbors
(kNN)

Chris Reger

Original Slides: Ryan Henning, Brandon Martin-Anderson, Lori Bordzuk

galvanize

Today's Objectives

- Implement KNN algorithm
- Explain the difference between KNN for regression vs. classification
- Understand KNN hyperparameters
    - How does changing them affect the model?
- Describe the curse of dimensionality

Supervised vs. Unsupervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.

X, y -> predicting y based on the values in X

X, y a.k.a
features, target
independent, dependent
exogenous, endogenous
predictors, response

Example capstone: [Avalanche Prediction]

Unsupervised learning is a type of self-organized … learning that helps find previously unknown patterns in data set without pre-existing labels.

X -> understanding structure in X

Example capstone: [Spice Blends]

galvanize

Parametric vs Non-parametric models

A machine learning algorithm can be supervised or unsupervised, and parametric or non-parametric.

A **parametric** algorithm

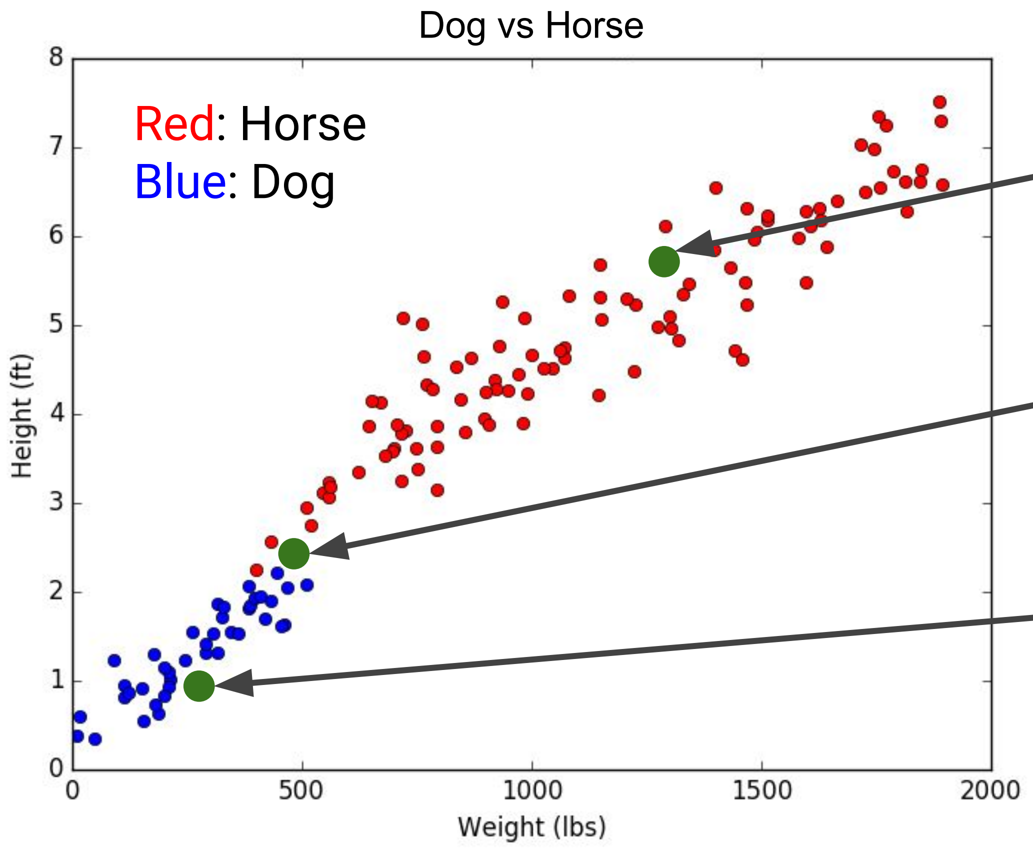
- has a fixed number of parameters
- makes assumptions about the structure of the data
- will work well if the assumptions are correct!
- common examples: linear regression, neural networks, statistical distributions defined by a finite set of parameters

A **non-parametric** algorithm

- uses a flexible number of parameters, and the number of parameters often grows as it learns from more data.
- makes fewer assumptions about the data
- common examples: K-Nearest Neighbors, decision trees

source - Jason Brownlee

Big dog or small horse?



New datapoint 1:
Is it a dog or a horse?

Horse

New datapoint 3:
Is it a dog or a horse?

Great Dane?

Pony?

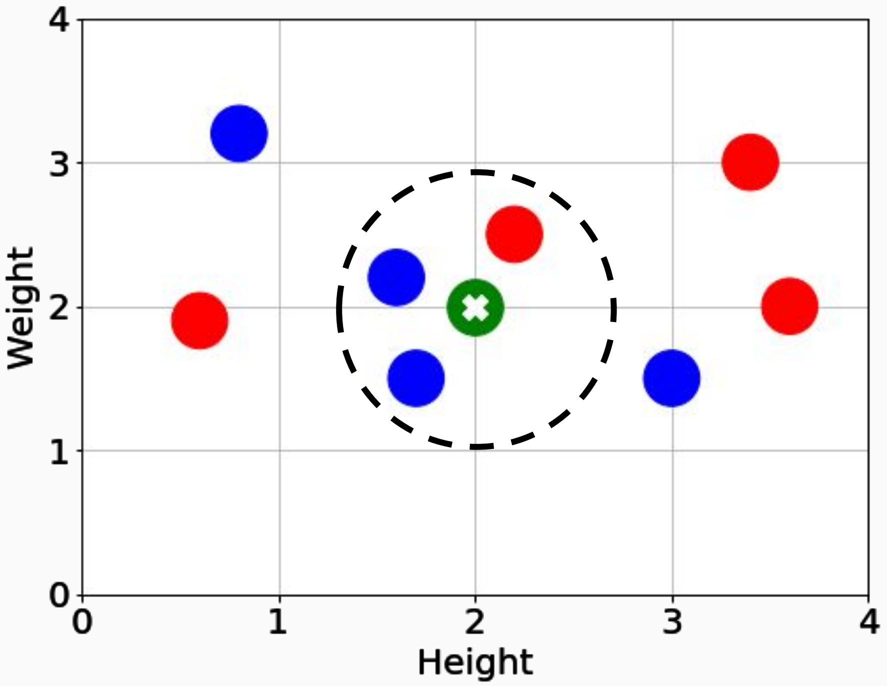
New datapoint 2:
Is it a dog or a horse?

Dog

The k-Nearest Neighbors: Classification

For a new input *x*, predict the **most common label** amongst its *k* closest neighbors

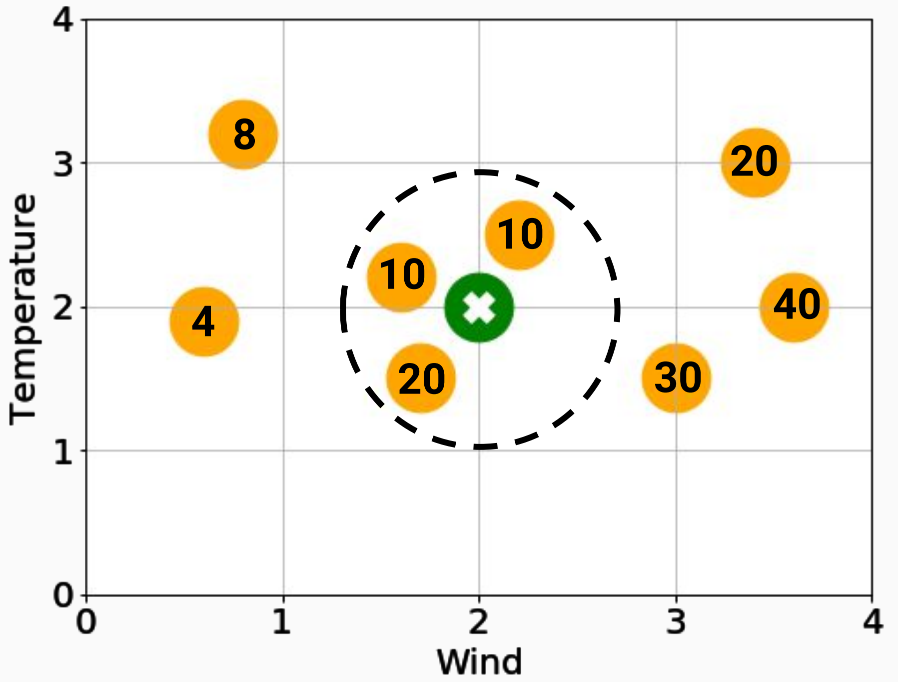
Image on right:
k = 3
Predict **BLUE**



The k-Nearest Neighbors: Regression

For a new input x, predict the **average label** amongst its k closest neighbors

Image on right:
k = 3
Predict **13.3**



The k-Nearest Neighbors Algorithm

Training algorithm:

1. Store all the data.

Prediction algorithm (predict the class of a new point x'):

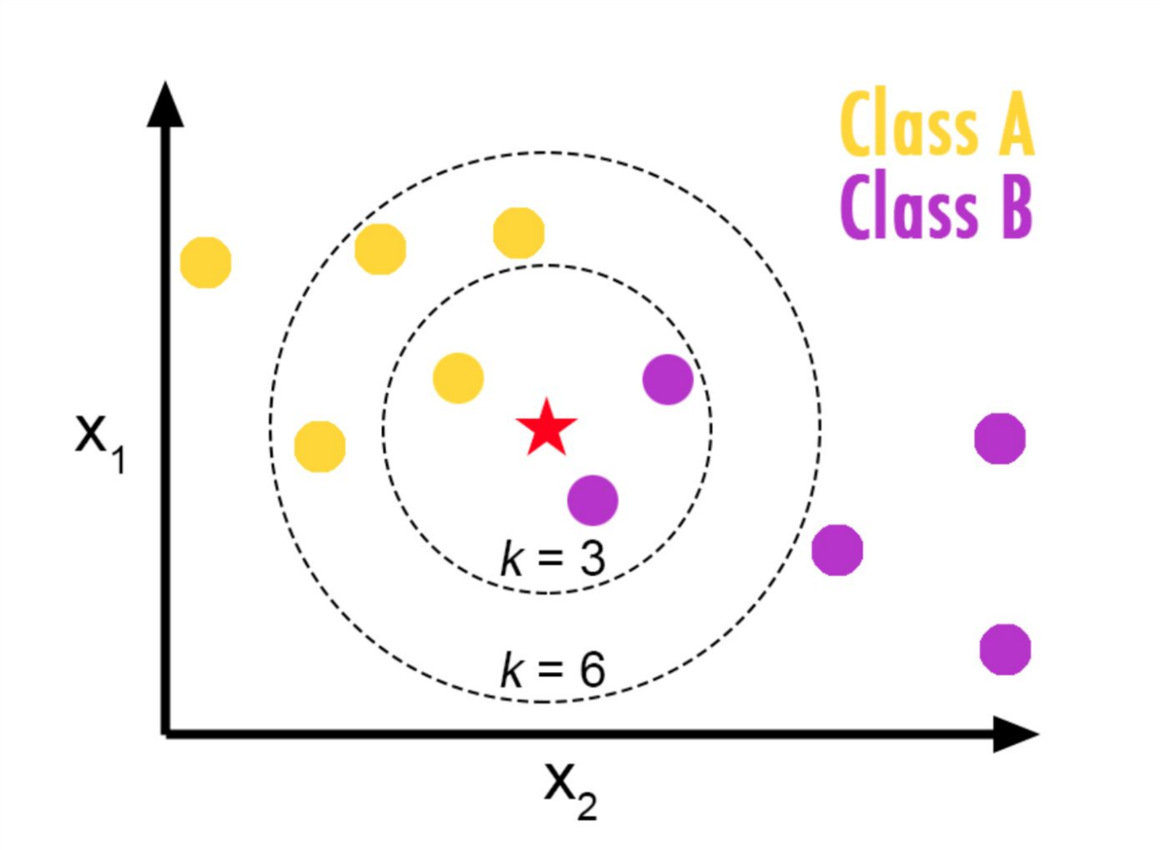
1. Calculate the distance from x' to all points in your dataset.
2. Sort the points in your dataset by increasing distance from x'.
3. Predict the majority label of the *k* closest points.

kNN Hyperparameter: Distance Metrics

Euclidean Distance (L2): $$\sum_i (a_i - b_i)^2$$

Manhattan Distance (L1): $$\sum_i |a_i - b_i|$$

Cosine Distance = 1 - Cosine Similarity: $$1 - \frac{a \cdot b}{||a||\,||b||}$$



What is the prediction when k=3?

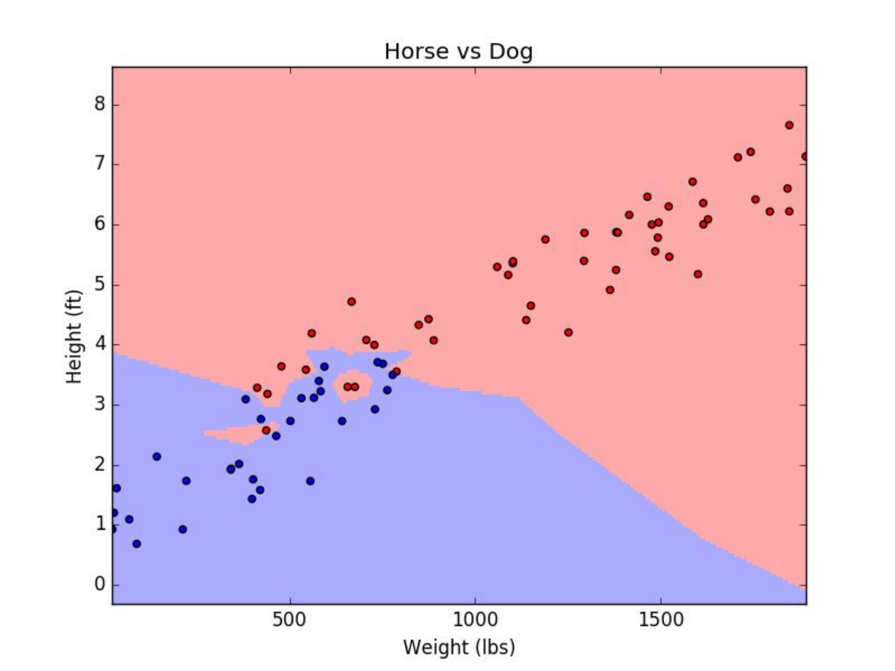
Class B

What is the prediction when k=6?

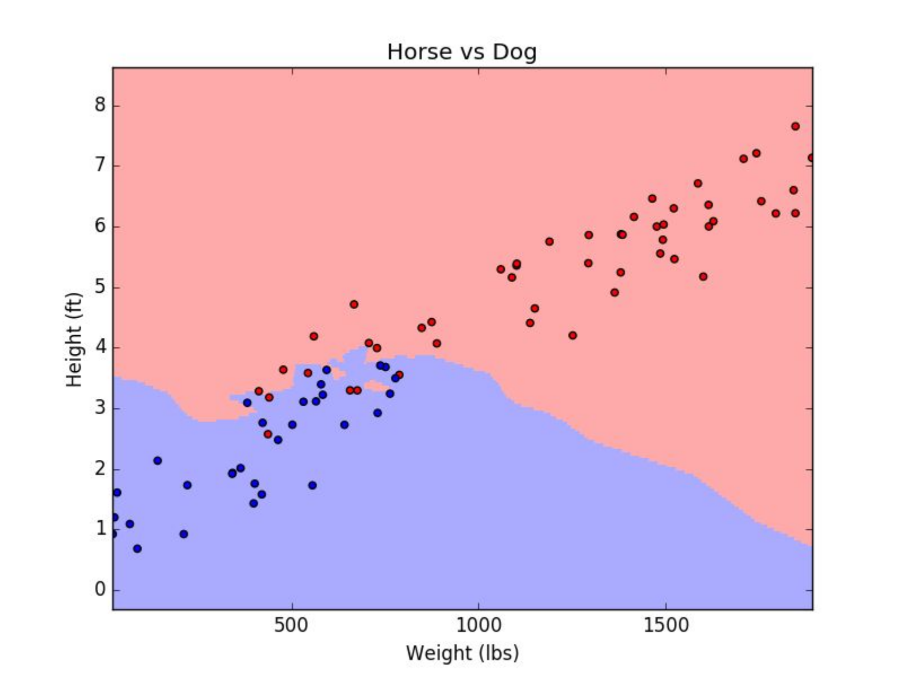
Class A

Hyperparameter *k*: the number of nearest neighbors to consider

k=1



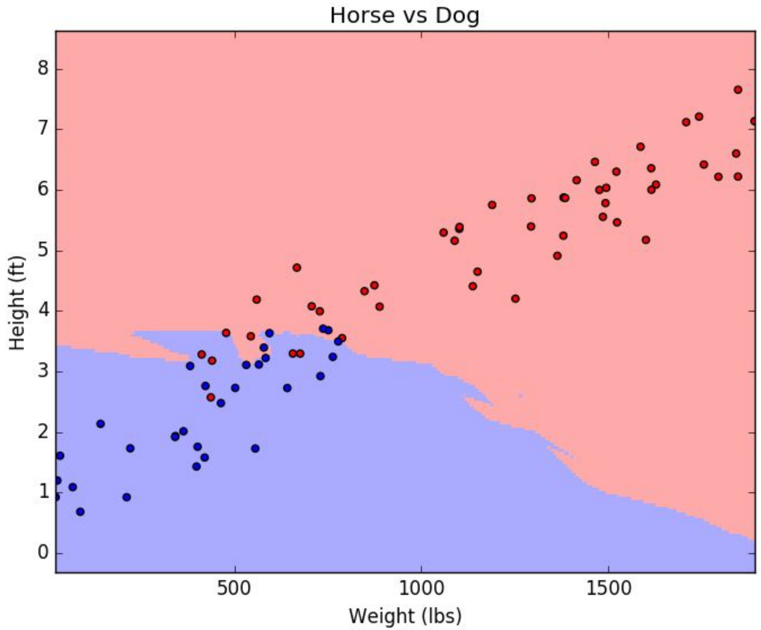
k=5



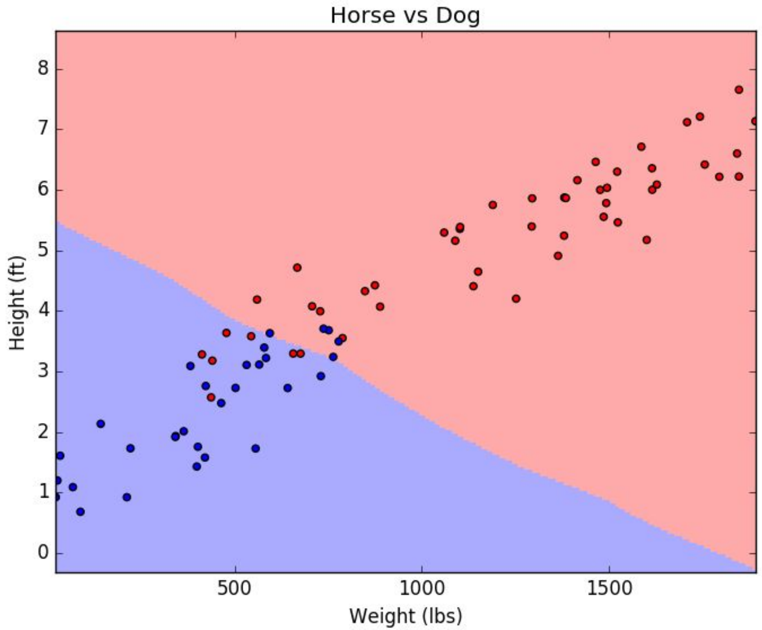
Hyperparameter *k*: the number of nearest neighbors to consider

galvanize

k=10

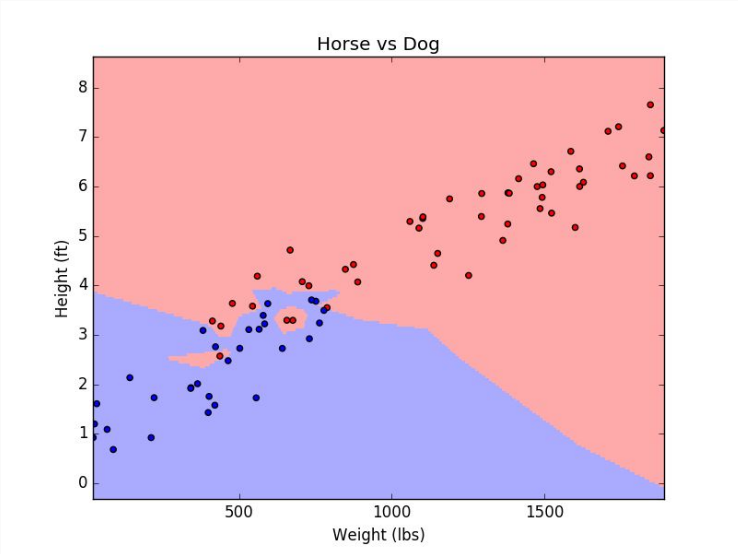


k=50



Which model is overfit?

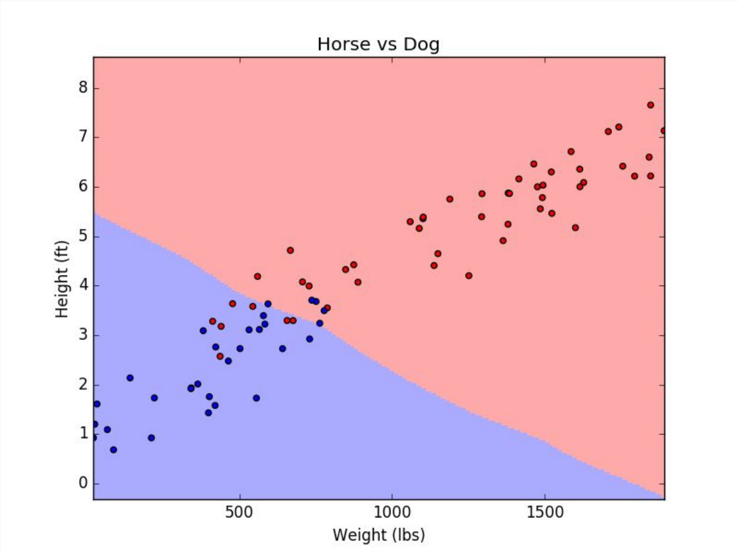
k=1



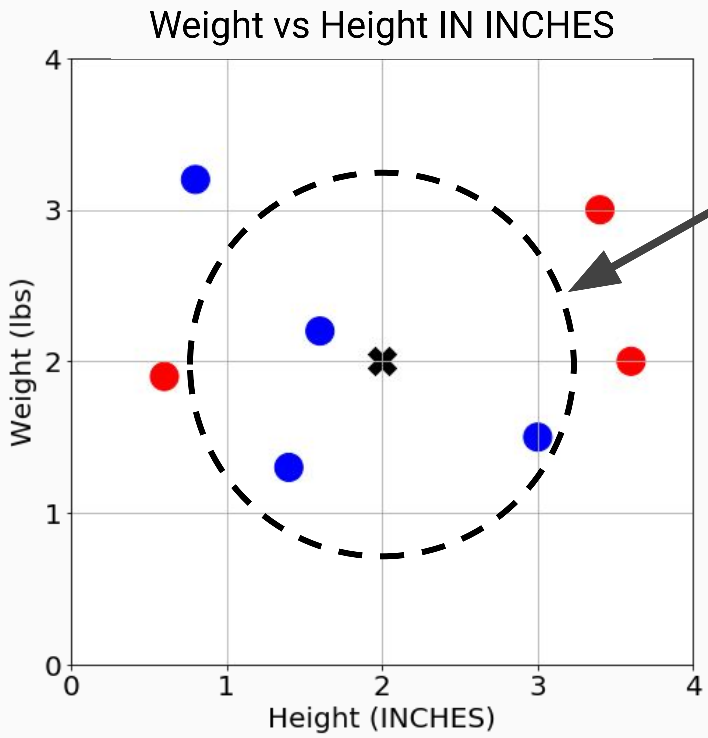
As a <u>general</u> rule, start with:

$$k = \sqrt{n}$$

k=50



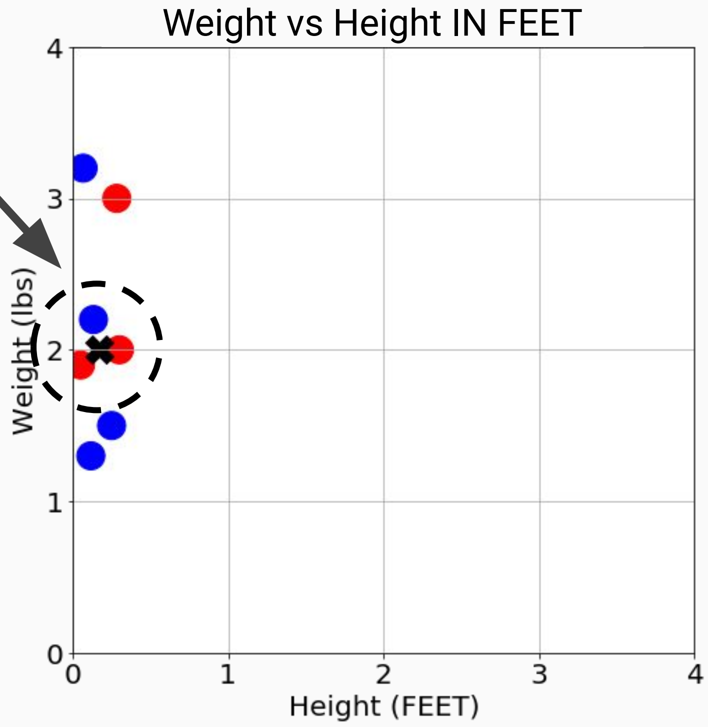
Be careful with the scale of your features!



k = 3

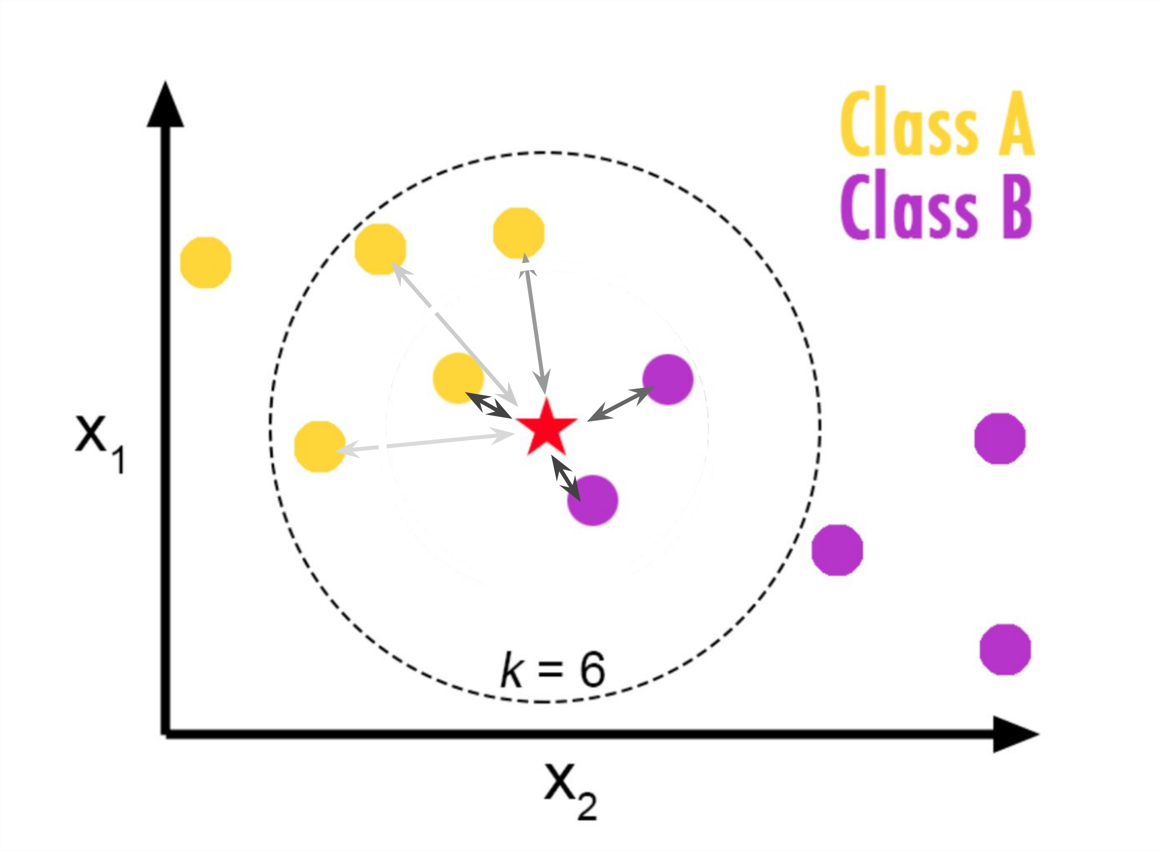
The three “closest neighbors” differ depending on the scale of the feature....

Don’t forget to scale your data!!!!



k = 3

Point weighting (weighted voting)



Let the k nearest points have distances:

$$d_1, d_2, \ldots, d_k$$

The i^{th} point votes with a weight of:

$$\frac{1}{d_i}$$

small distances are weighted more!

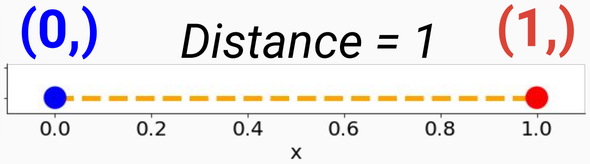
kNN in high dimensions

kNN works pretty well (in *general*) for dimensions < 5
but is problematic when used with high dimensional spaces

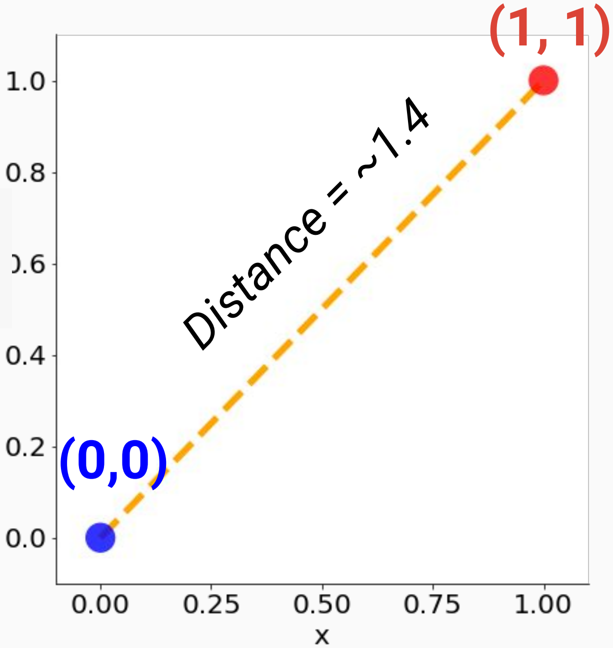
In high dimensions, the nearest neighbors can be very “far away”

The Curse of Dimensionality

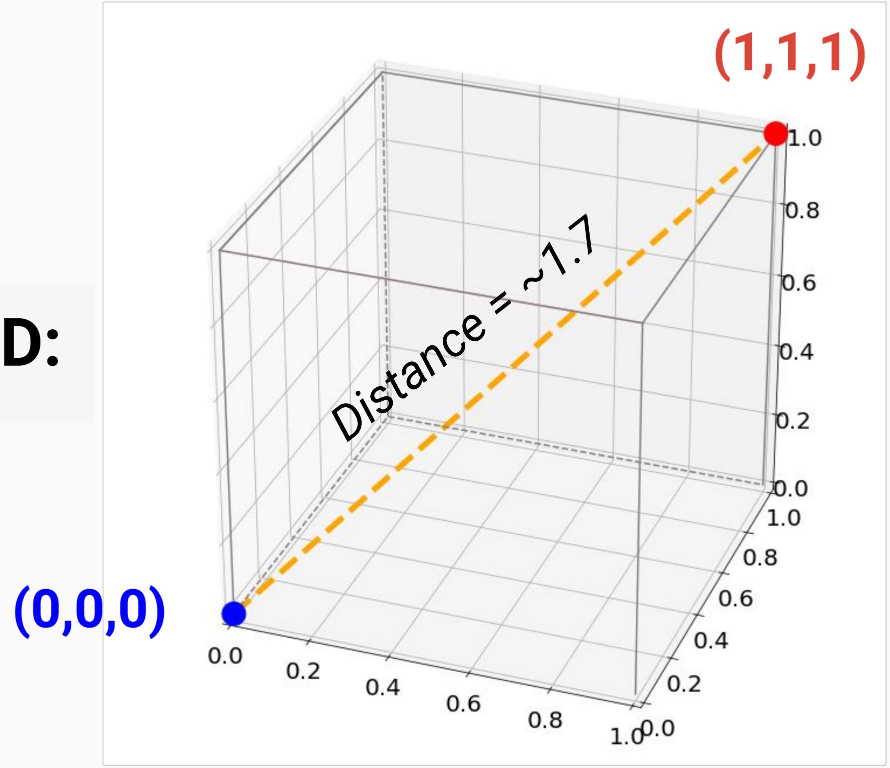
1D:



2D:

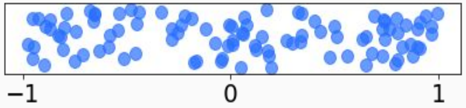


3D:



The Curse of Dimensionality (another perspective)

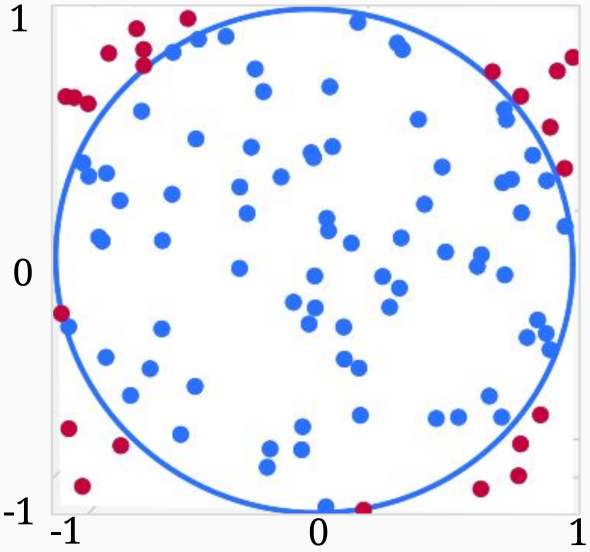
Given 100 (random) sample points....



1D

All 100 pts within 1 unit of center.

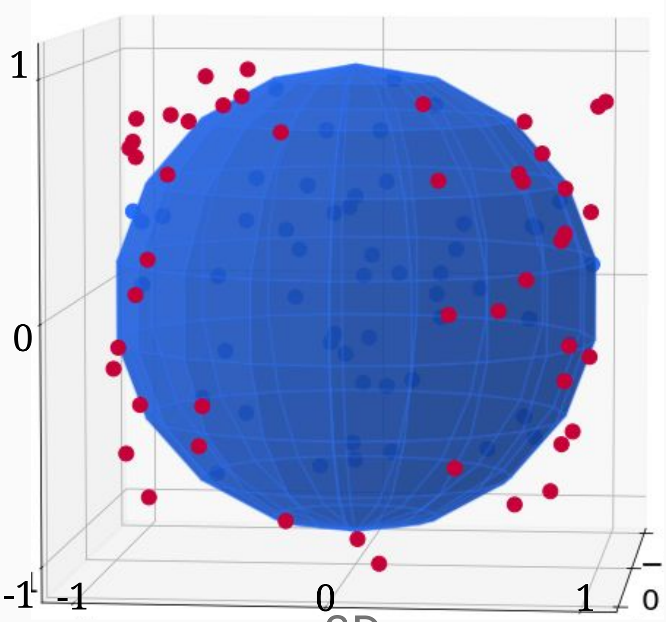
Density = 100



2D

77 pts within 1 unit of center.

Density = 77



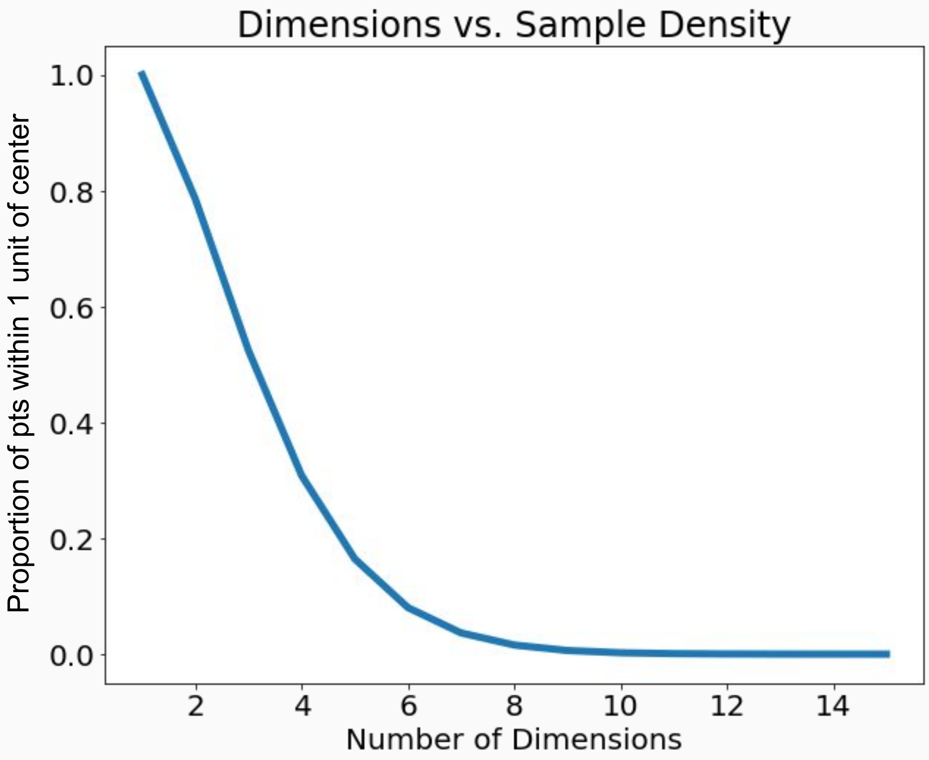
3D

53 pts within 1 unit of center.

Density = 53

The Curse of Dimensionality **(another perspective)**

galvanize



The **more dimensions** you have, the **more data points** you need to maintain density.

General guideline:

- Given *n* data points in d_{orig} dimensions...
- If you want to *increase* the total number of dimensions to d_{new} , you now need:
- $n^{\frac{d_{new}}{d_{orig}}}$ data points to maintain density

The Curse of Dimensionality takeaways

- kNN (or any method that relies on distance metrics) will suffer in high dimensions.
    - Nearest neighbors are “far” away in high dimensions (even for d=10).
- High dimensional data tends to be sparse; it’s easy to overfit sparse data.
    - It takes A LOT OF DATA to make up for increased dimensionality.

Summary: kNN

Pros:

- Super simple
- Training is trivial (store the data)
- Works with any number of classes
- Easy to add more data
- Few hyperparameters:
    - *distance metric*
    - *k*

Cons:

- High prediction cost (especially for large datasets)
- Bad with high dimensions
    - you'll learn dimensionality reduction methods later on!
- Categorical features don't work well

Review: Today's Objectives

- Implement KNN algorithm
- Explain the difference between KNN for regression vs. classification
- Understand KNN hyperparameters
    - How does changing them affect the model?
- Describe the curse of dimensionality

Morning Exercise: Implement a sklearn-style KNN algorithm

Appendix

The Curse of Dimensionality (another perspective)

Don't freak out...

$$\lim_{d\to\infty} \frac{V_{\text{sphere}}(R,d)}{V_{\text{cube}}(R,d)} = \lim_{d\to\infty} \frac{\frac{\pi^{d/2}R^d}{\Gamma(d/2+1)}}{(2R)^d} = \lim_{d\to\infty} \frac{\pi^{d/2}}{2^d\Gamma(d/2+1)} = 0$$

Euler's gamma function…
basically, it's the *factorial* function that can operate on fractional numbers

Factorial overtakes exponentiation in the limit…
e.g.

$$\lim_{x\to\infty} \frac{c^x}{x!} = 0$$

What does this mean?

Parametric vs Non-parametric Models

Parametric models have a <u>fixed</u> number of learned parameters.
- Logistic regression is parametric.
- kNN is non-parametric.

Parametric models are more structured. The added structure often combats the curse of dimensionality... as long as the structure is derived from reasonable assumptions.

Alternate perspective: Parametric models are not distance based, so the curse doesn't apply!