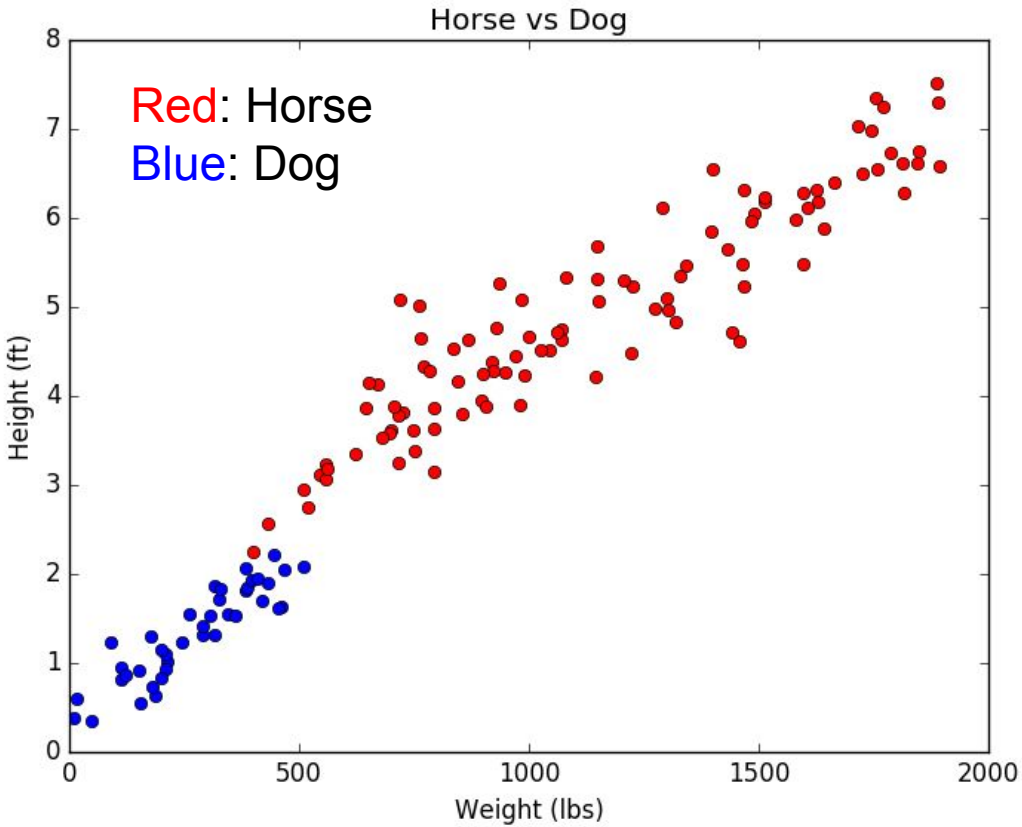
k-Nearest Neighbors
(kNN)

Ryan Henning
(customized by F. Burkholder)

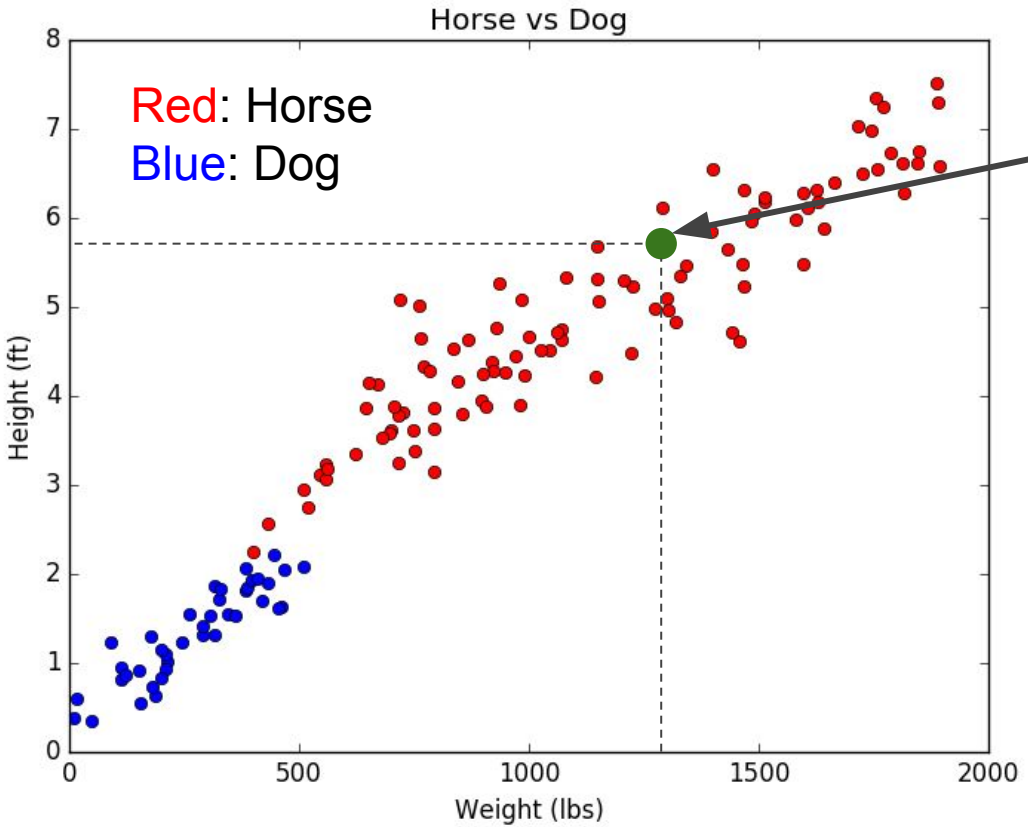
galvanize

- k-Nearest Neighbors
- The Curse of Dimensionality
- Parametric vs Nonparametric Models

(Fake) horse and dog data - height and weight

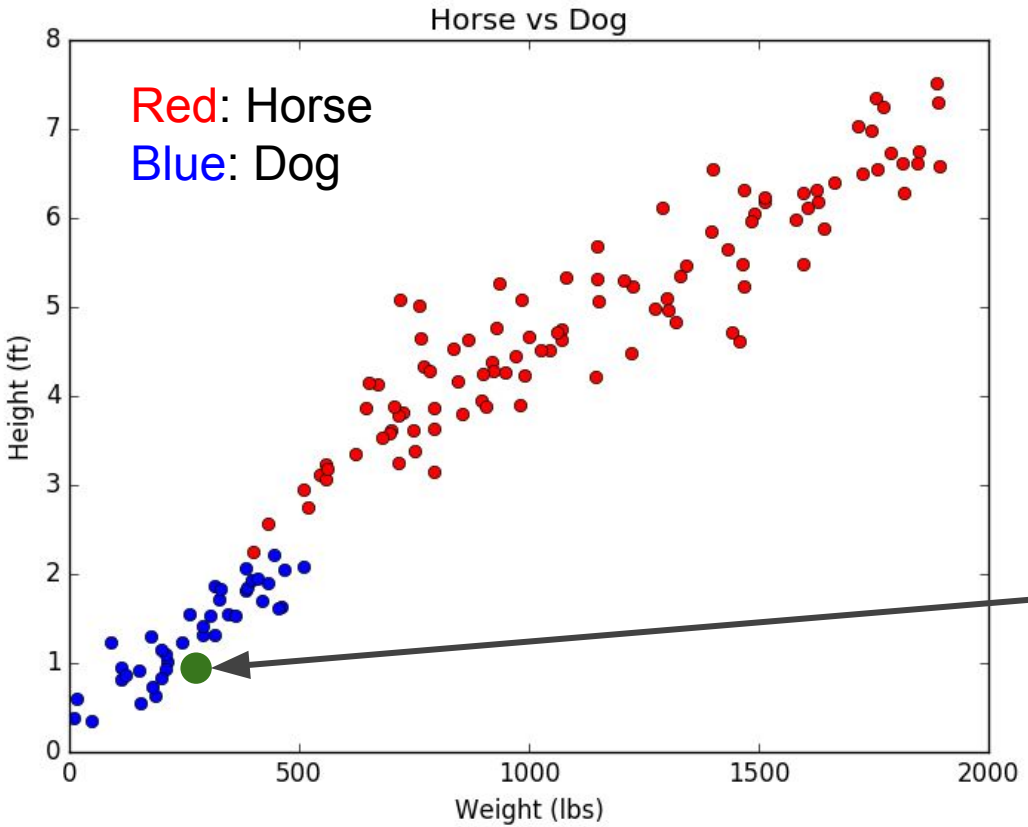


Is it a big dog or a small horse?



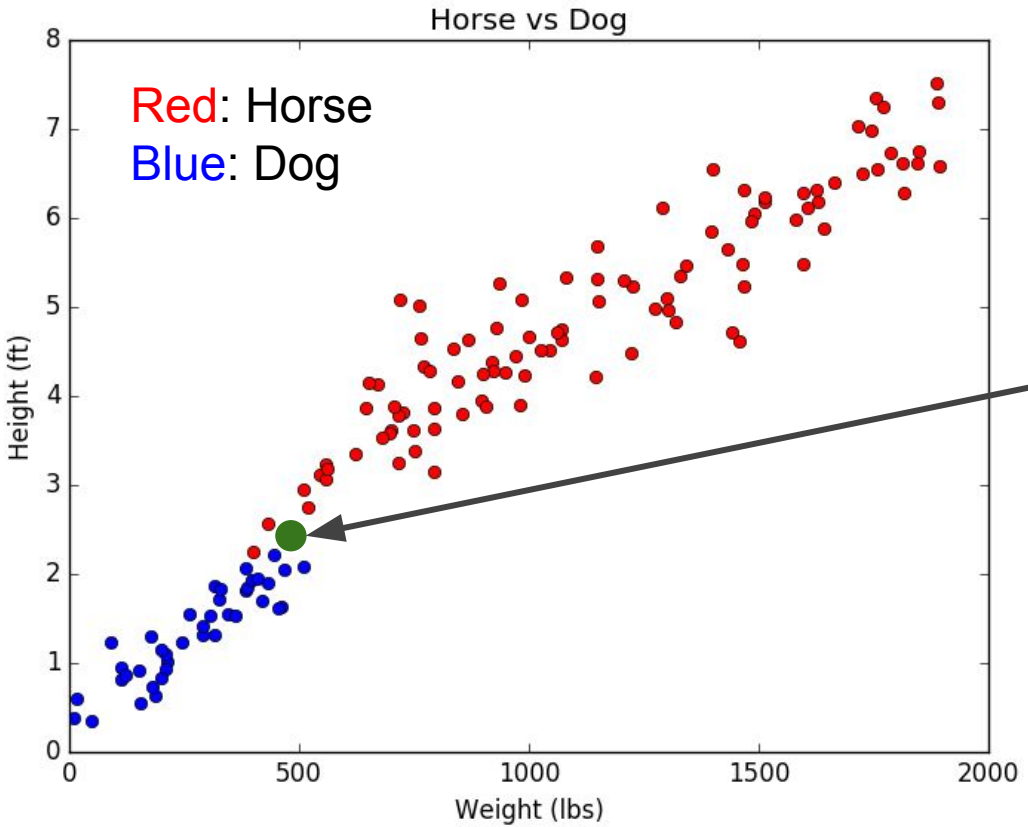
New datapoint:
Is it a horse or a dog?

Is it a big dog or a small horse?



New datapoint:
Is it a horse or a dog?

Is it a big dog or a small horse?



New datapoint:
Is it a horse or a dog?

k-Nearest Neighbors

galvanize

k-nearest neighbors algorithm (k-NN) is a *non-parametric* method used for classification and regression. The input consists of the k closest training examples in the feature space.

> In classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors.
>
> In regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

k-NN is a type of instance-based learning, or lazy learning, where the computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

\- *Wikipedia*

Non-parametric vs parametric

galvanize

Non-parametric models differ from *parametric* models in that the model structure is not specified a priori but is instead determined from data.

In non-parametric models the number and nature of the parameters are flexible and not fixed in advance (they are not parameter-less, e.g. *k* in k-NN)

Non-parametric methods make fewer assumptions, so their applicability is wider than corresponding parametric methods. In particular, they may be applied in situations where less is known about the application in question.

The cost: in cases where a parametric model would be appropriate, non-parametric models have less power. In other words, a larger sample size can be required to draw conclusions with the same degree of confidence.

- *Wikipedia*

galvanize

The k-Nearest Neighbors algorithm:

Training algorithm:

1. Store all the data… that's all.

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.

The k-Nearest Neighbors algorithm:

Training algorithm:

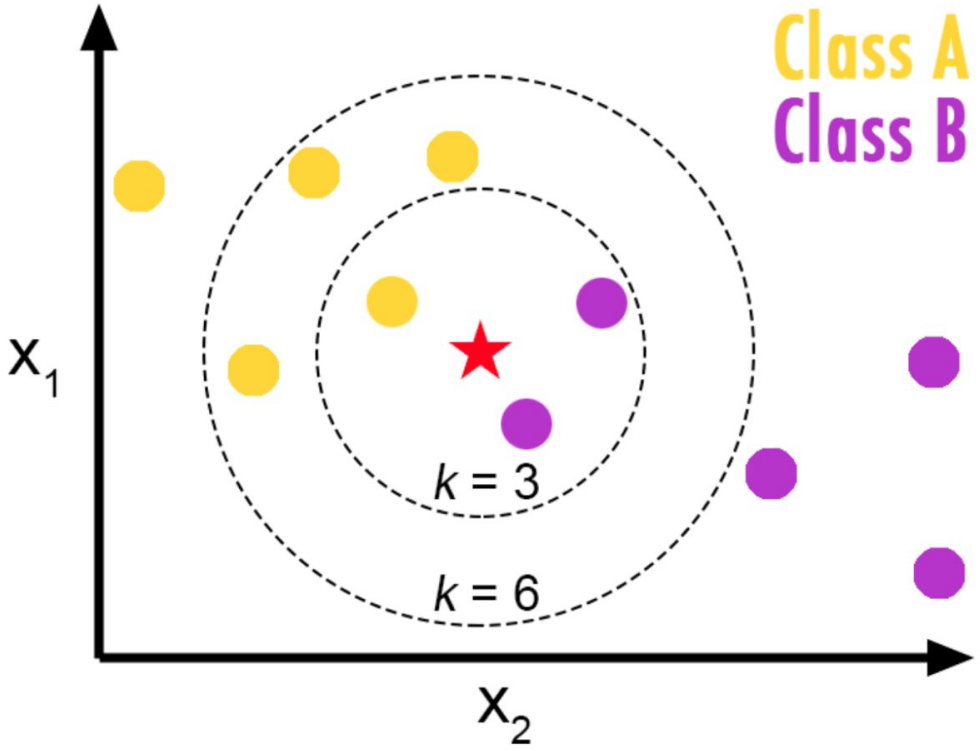
1. Store all the data… that's all.

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.

What is *k*?

kNN: you pick k



What is the prediction for ★ when k=3?

What is the prediction for ★ when k=6?

Demo code

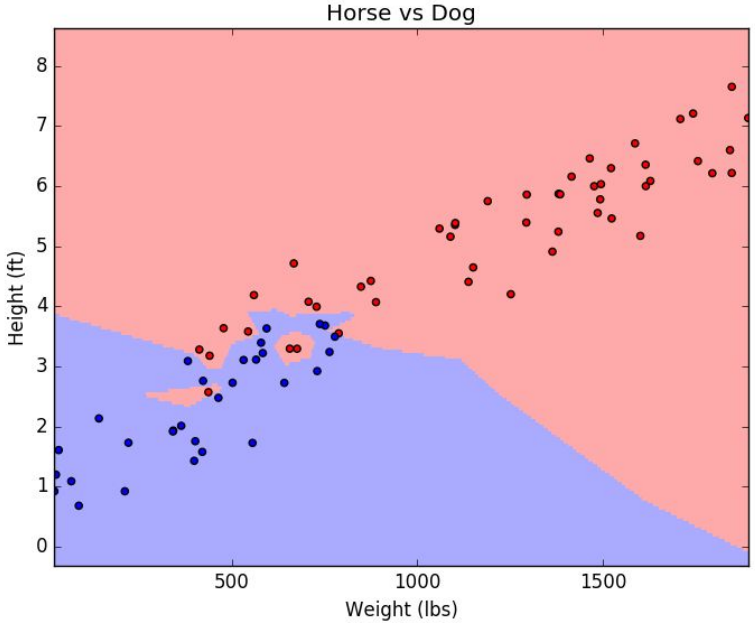
galvanize

`knn_example.ipynb`

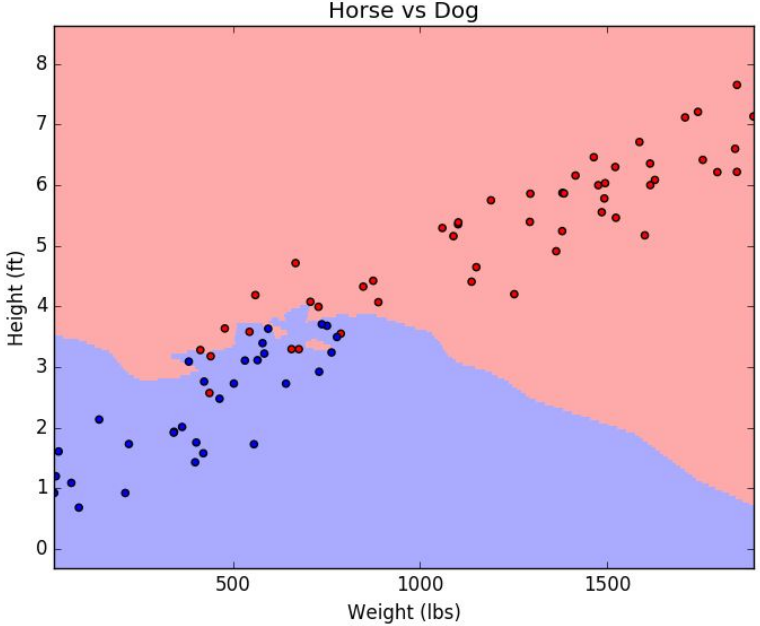
The only hyperparameter… *k*… the number of nearest neighbors to consider

galvanize

k=1



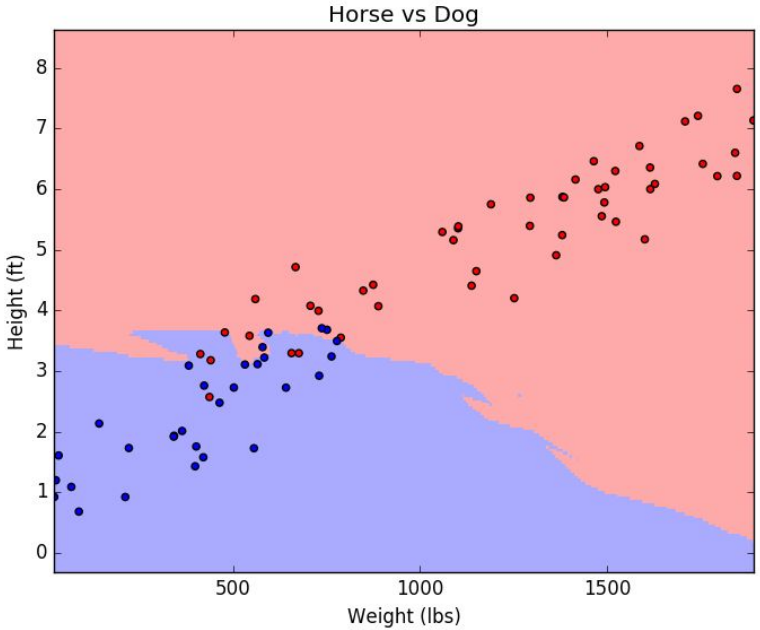
k=5



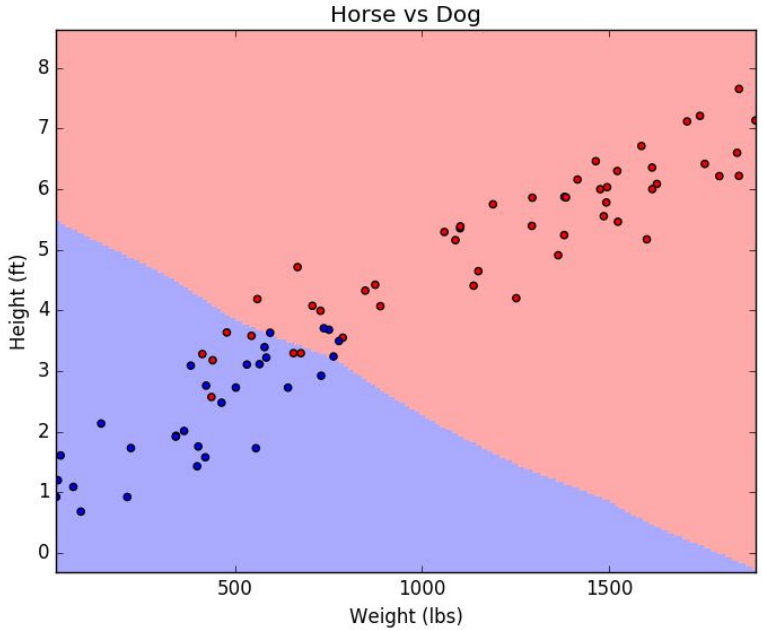
The only hyperparameter… *k*… the number of nearest neighbors to consider

galvanize

k=10

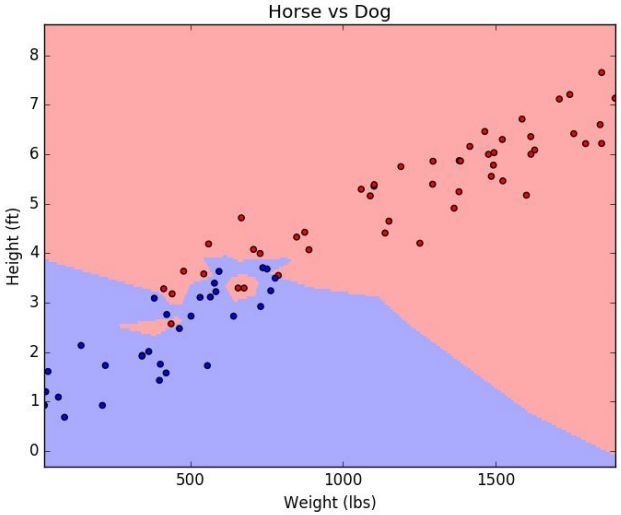


k=50



Which model seems overfit?

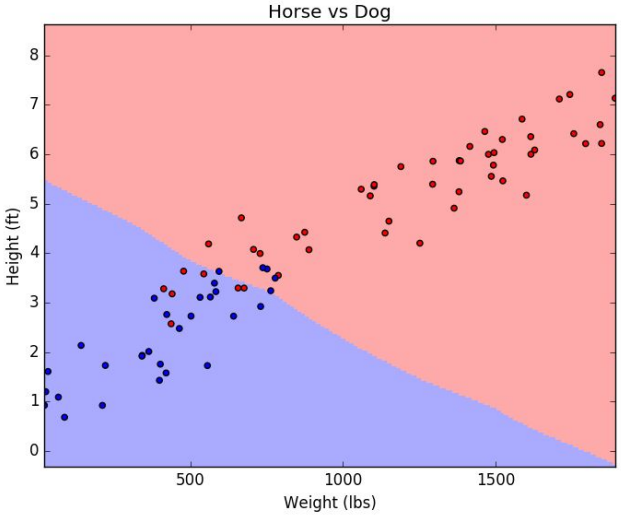
k=1



Btw, as a general rule, start with:

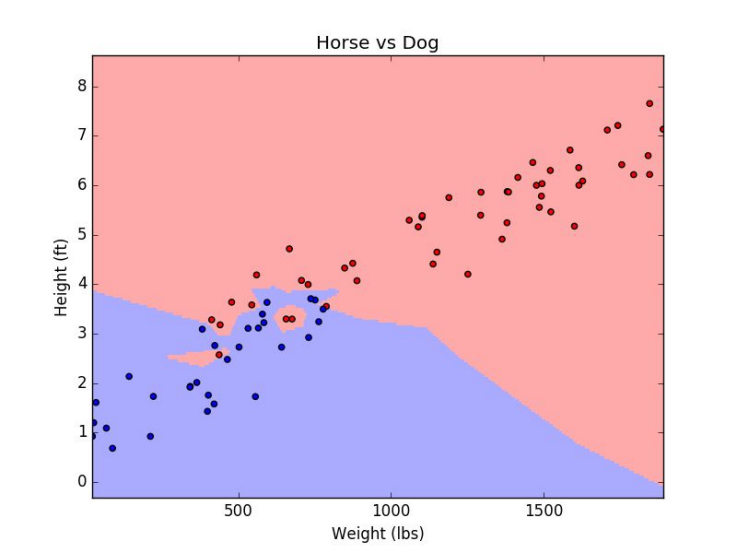
$$k = \sqrt{n}$$

k=50

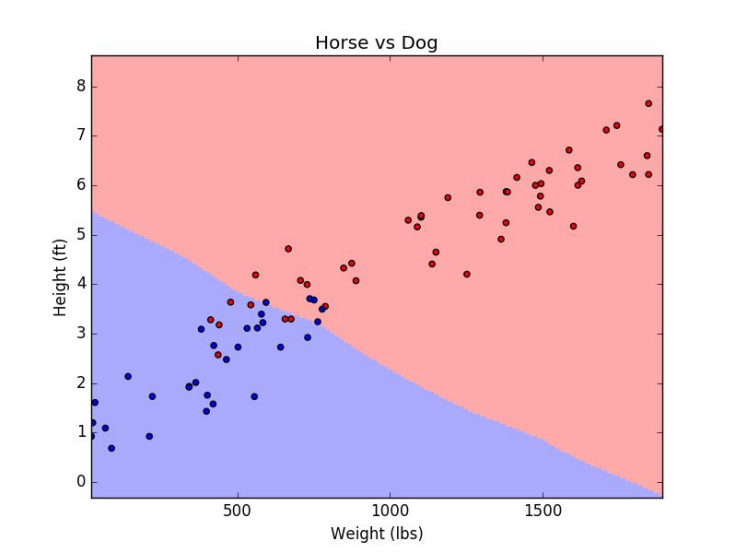


What happens to model variance when *k* increases?

k=1



k=50

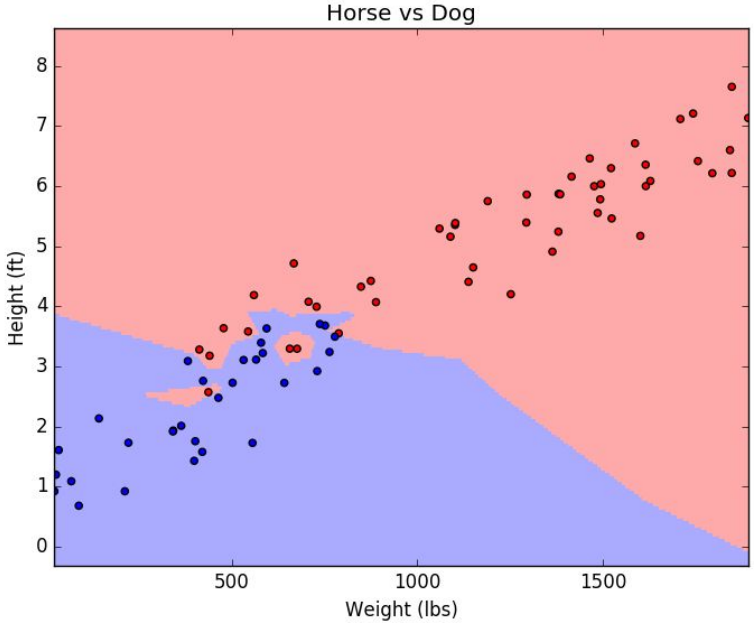


See the high variance?

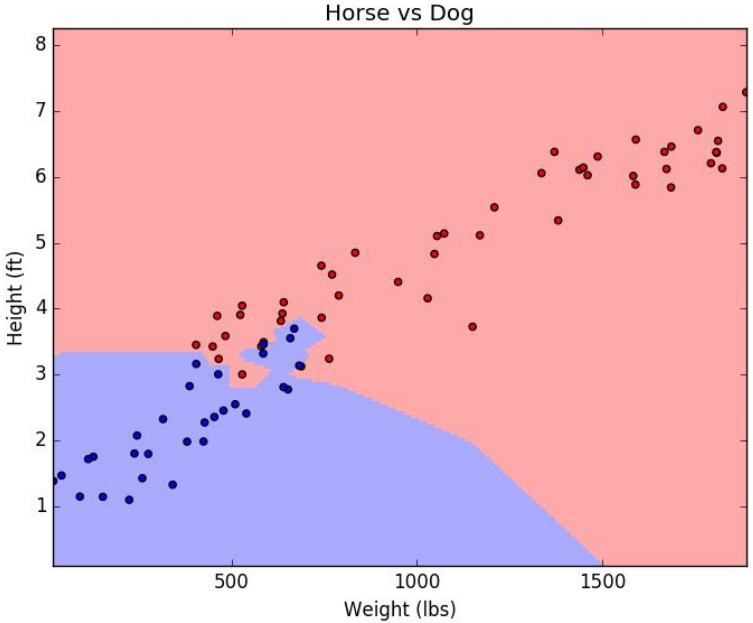
Different training data - different decision boundary, even though data from same pop.

galvanize

k=1



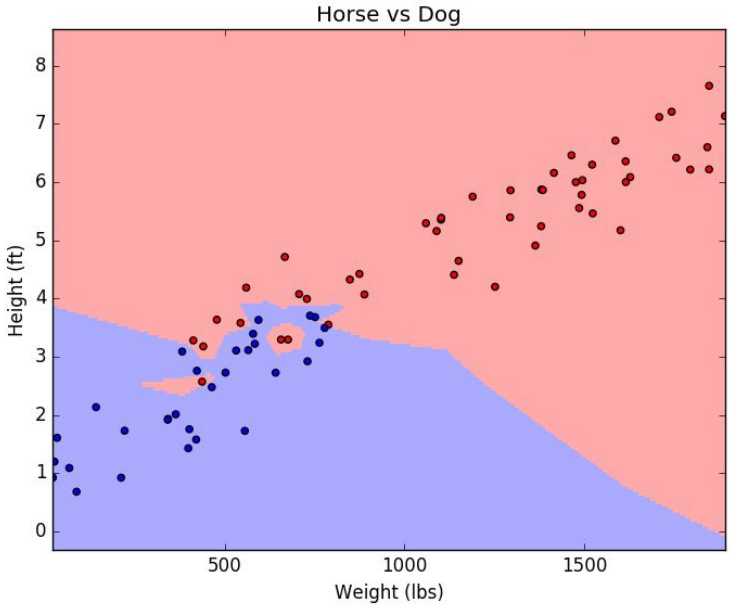
k=1



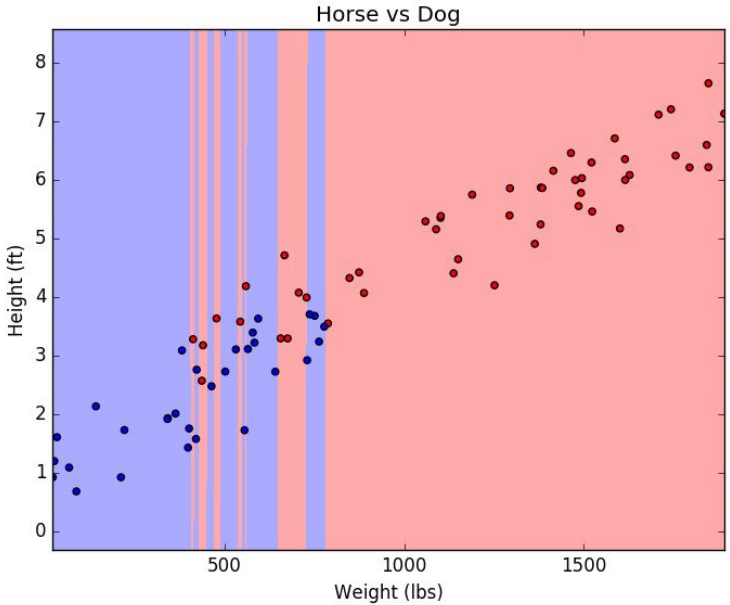
Be careful with the scale of your features!

galvanize

k=1, scaled features



k=1, original-scale features



Speaking of distance… what distance metric are we using anyway?

Distance Metrics

Euclidean Distance (L2):

$$\sum_i (a_i - b_i)^2$$

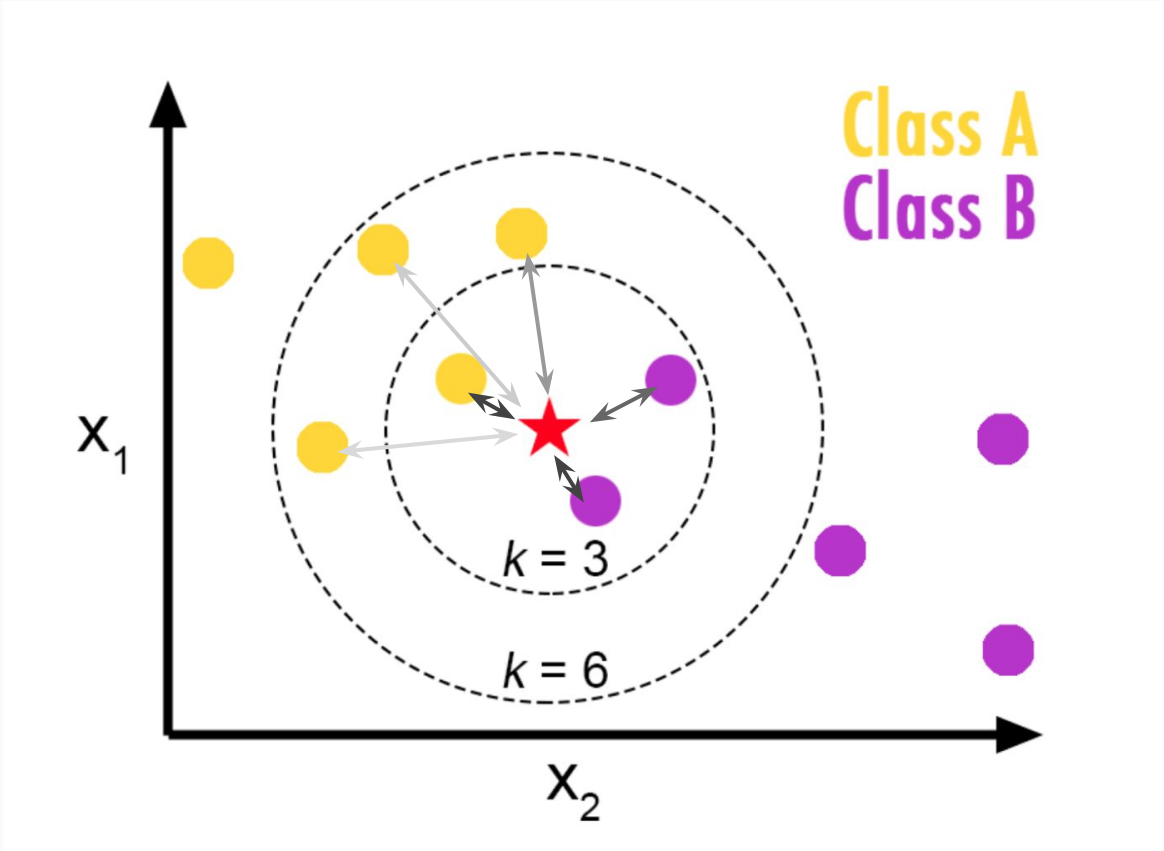
Manhattan Distance (L1):

$$\sum_i |a_i - b_i|$$

Cosine Distance = 1 - Cosine Similarity:

$$1 - \frac{a \cdot b}{||a||||b||}$$

Point weighting (weighted voting)



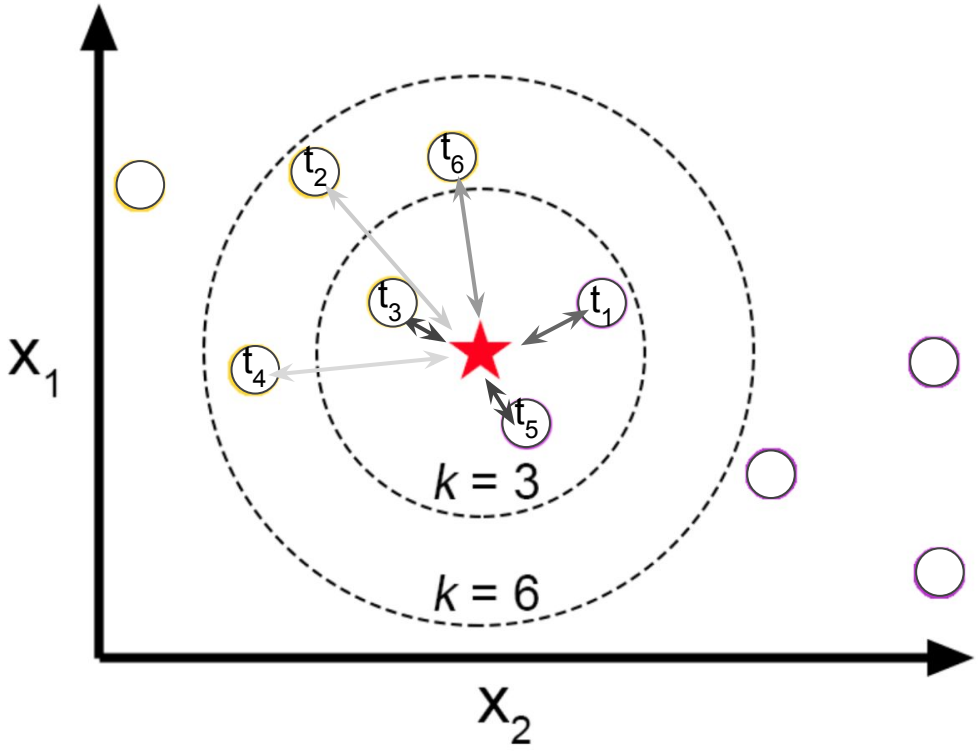
Let the *k* nearest points have distances:

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

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

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

small distances are weighted more!



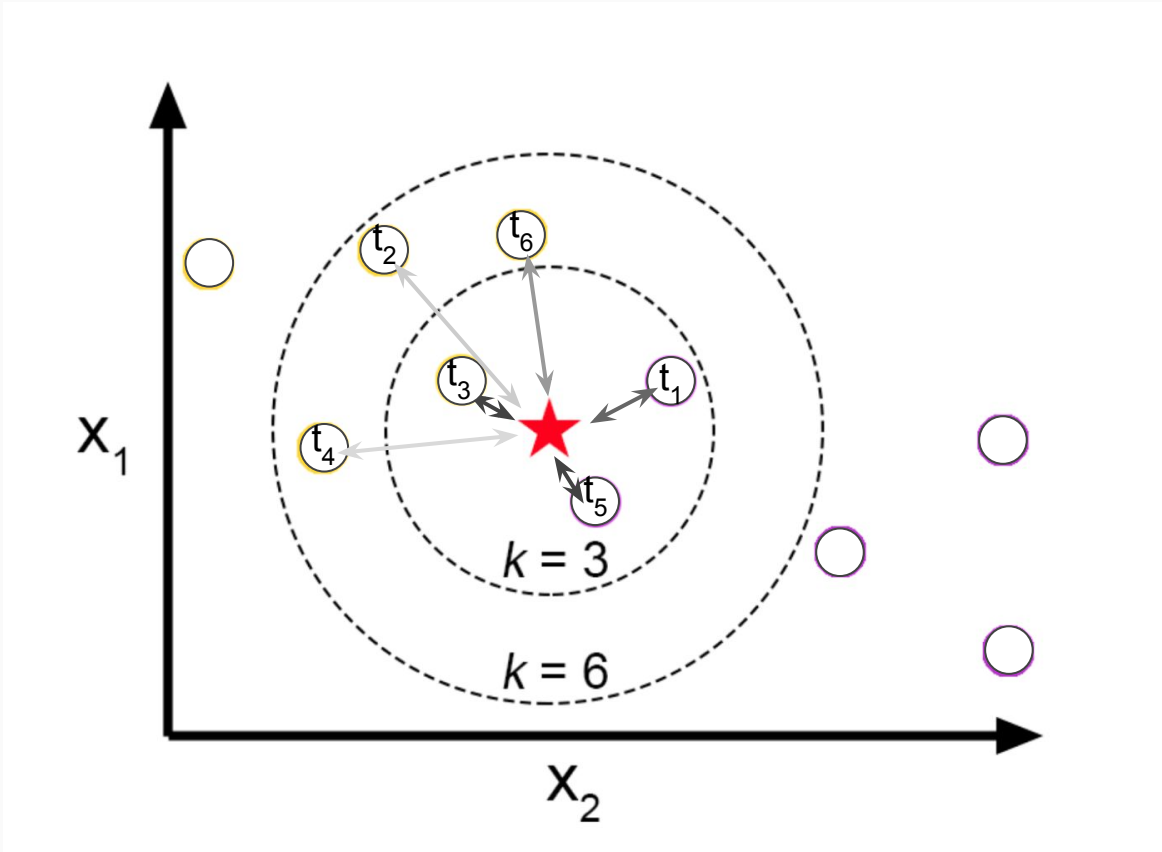
Let the *k* nearest points have distances:

$$d_1, d_2, ..., d_k$$

Let the *k* nearest points have targets:

$$t_1, t_2, ..., t_k$$

How can we do regression with kNN?



Let the *k* nearest points have distances:

$$d_1, d_2, ..., d_k$$

Let the *k* nearest points have targets:

$$t_1, t_2, ..., t_k$$

How can we do regression with kNN?

Predict the mean value of the k neighbors, or predict a weighted average.

kNN in high dimensions...

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

The nearest neighbors can be very **“far” away in high dimensions.** If everything is far away, how do you tell who your nearest neighbors are?

Say you want to use a neighborhood of 10% (i.e. k = 0.1*n)

Let’s see how this looks as we increase the dimensionality…

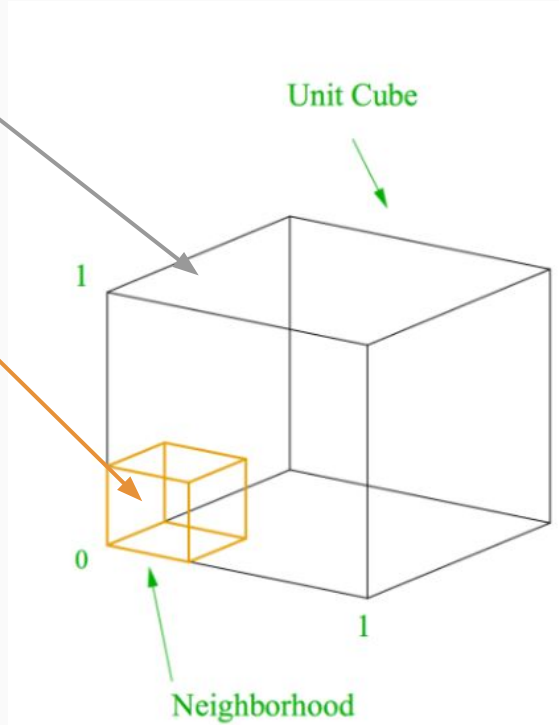
The Curse of Dimensionality **(one perspective)**

galvanize

Say we have a unit (hyper)cube.

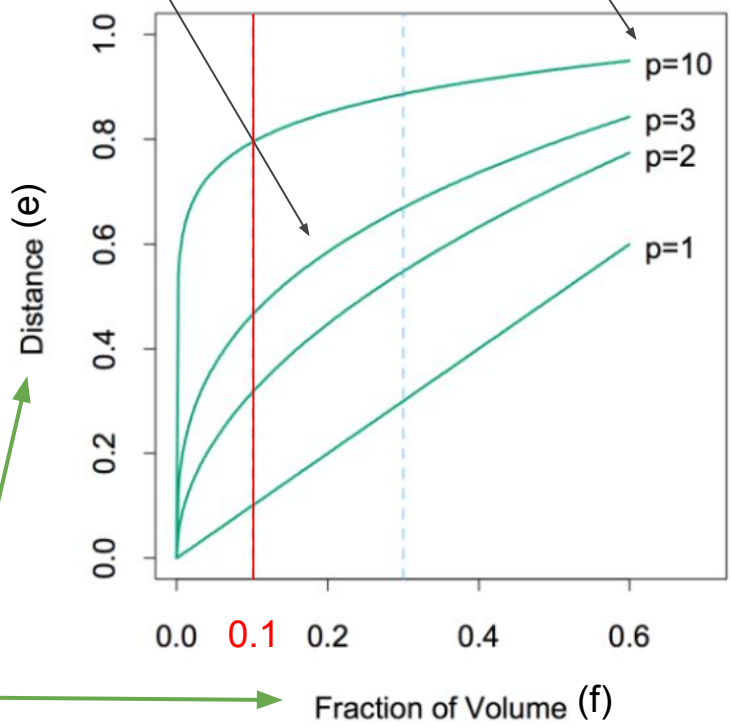
We want to create another (hyper)cube inside the outer cube so that we fill 10% of the outer cube.

How long must the edges of the inner cube be?



$e = f^{1/p}$

p = dimensionality



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

galvanize

Say you have a dataset with 100 samples, each with only one predictor.

But, one predictor doesn't tell you enough, so you collect a new dataset, and this time you measure 10 predictors for each sample.

How many samples do you need in your new (10 predictor) dataset to achieve the same "sample density" as you originally had (in the one-predictor dataset)?

$$N_1^{1/d_1} = N_2^{1/d_2}$$

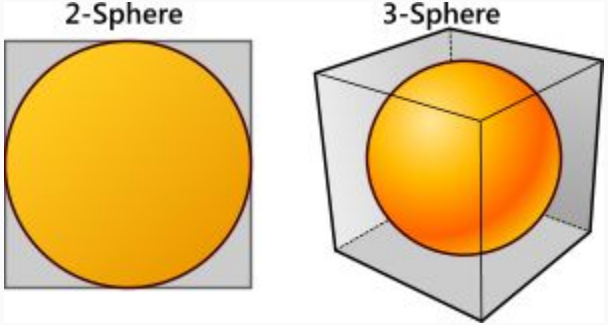
Just 100^10, that not that many… just

100,000,000,000,000,000,000

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

galvanize

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



https://ieatbugsforbreakfast.files.wordpress.com/2011/07/2sphere3sphere.png

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

galvanize

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

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

Factorial overtakes exponentiation in the limit…
e.g.

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

From Wikipedia: "...the high-dimensional unit hypercube can be said to consist almost entirely of the 'corners' of the hypercube, with almost no 'middle'."

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).
- A 10% neighborhood in a high dimensional unit hypercube requires a hypersphere with large radius.
  - Hyperspheres are weird in high dimensions…
  - “They are super-pointy!” (Ryan’s interpretation)...and data most likely isn’t in the points.
- 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.

Parametric vs Non-parametric Models

Parametric models have a fixed number of 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!

Summary: kNN

Pros:

- super-simple
- training is trivial (store the data)
- works with any number of classes
- easy to add more data
- few hyperparameters:
  - *k*
  - *distance metric*

Cons:

- high prediction cost (especially for large datasets)
- high-dims = bad
  - we'll learn dimensionality reduction methods in two weeks!
- categorical features don't work well…