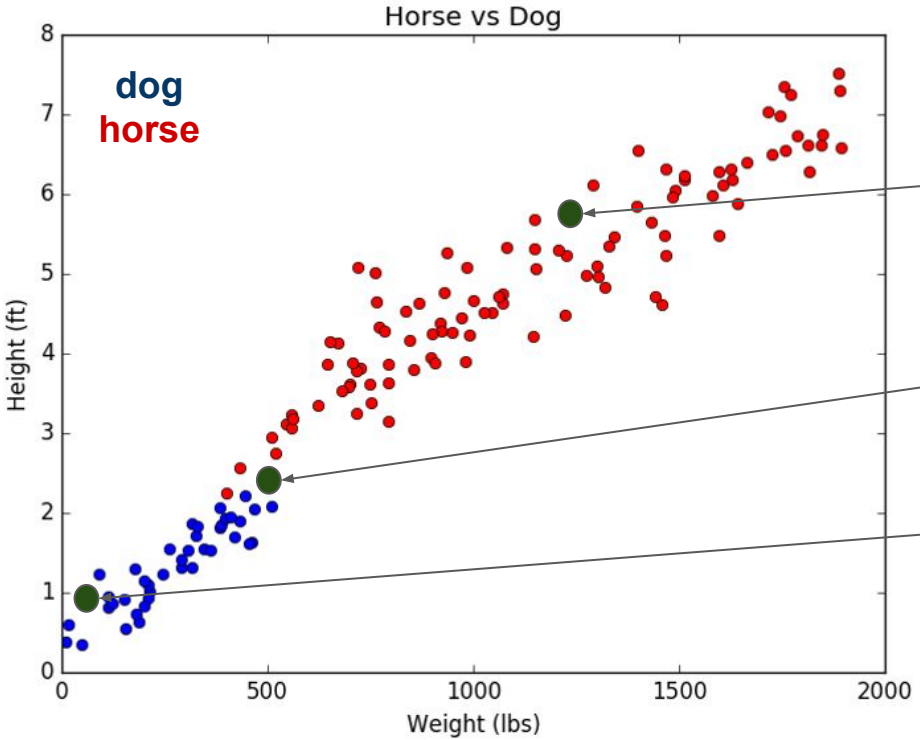
k-Nearest Neighbors (k-NN)

Ivan Corneillet

Is it a big dog or a small horse?



new datapoint to predict:
dog or horse?

new datapoint to predict:
dog or horse?

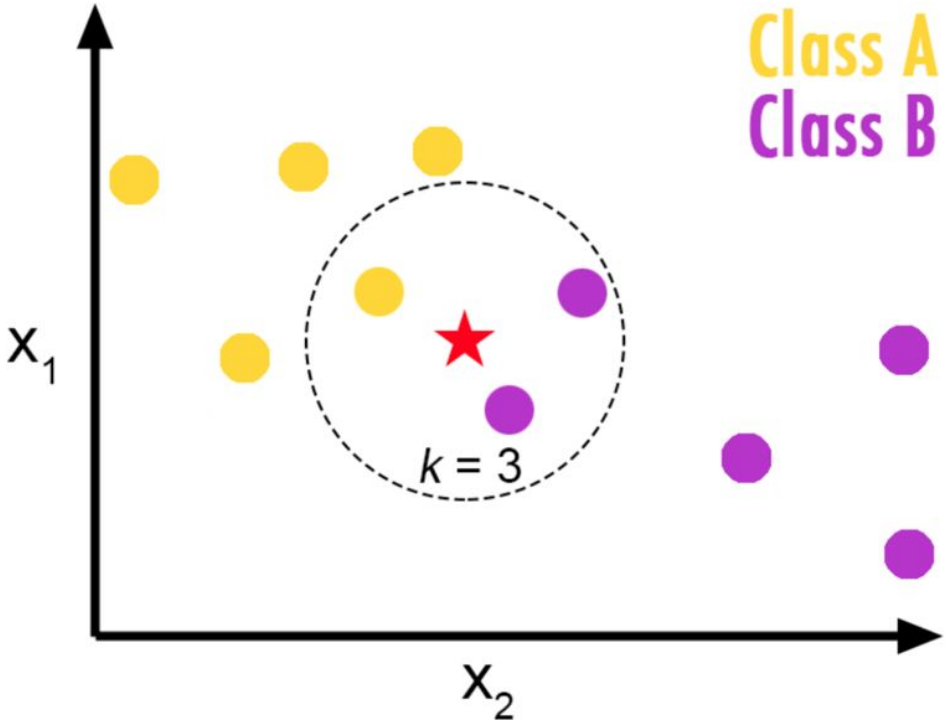
new datapoint to predict:
dog or horse?

k-Nearest Neighbors Algorithm

- training:
  - store/memorize all the training dataset… that's all
- predicting (the class of a new point **x'**):
  - calculate the distance from **x'** to all points **x** from in your training dataset
  - sort the training points **x** by increasing distance (from **x'**)
  - predict the majority label of the **k** closest points

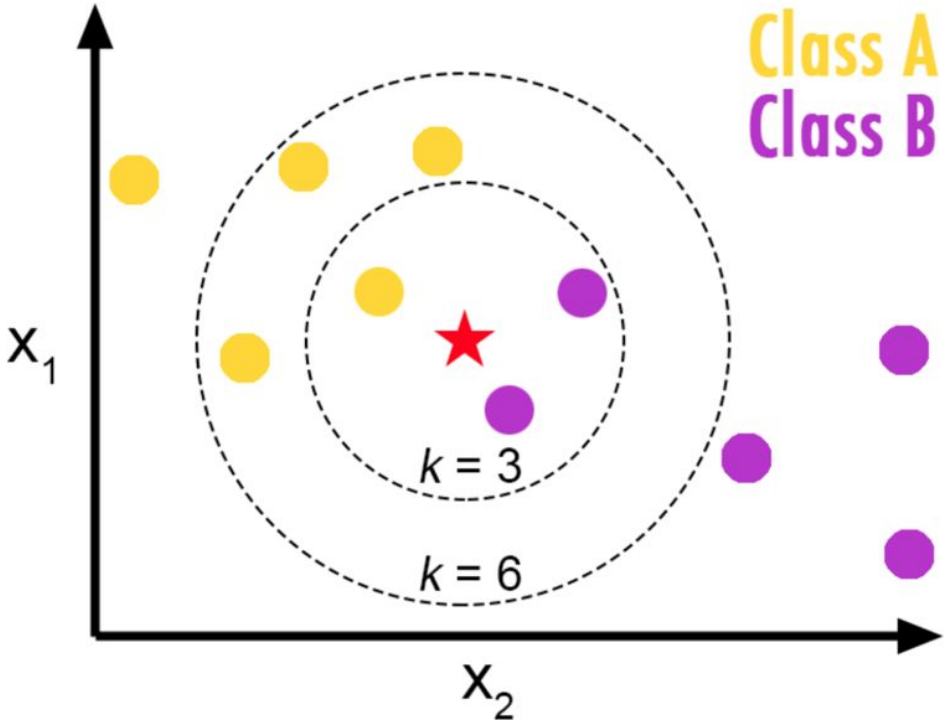
k, the number of neighbors to consider, is a hyper-parameter that **you** need to pick

What value for k?



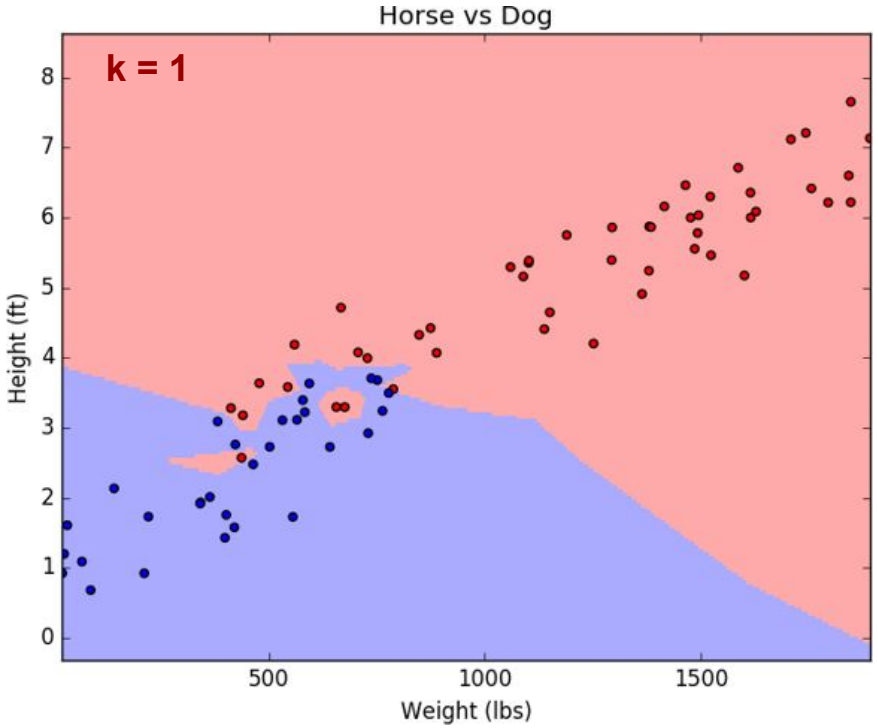
what do you predict for k = 3?

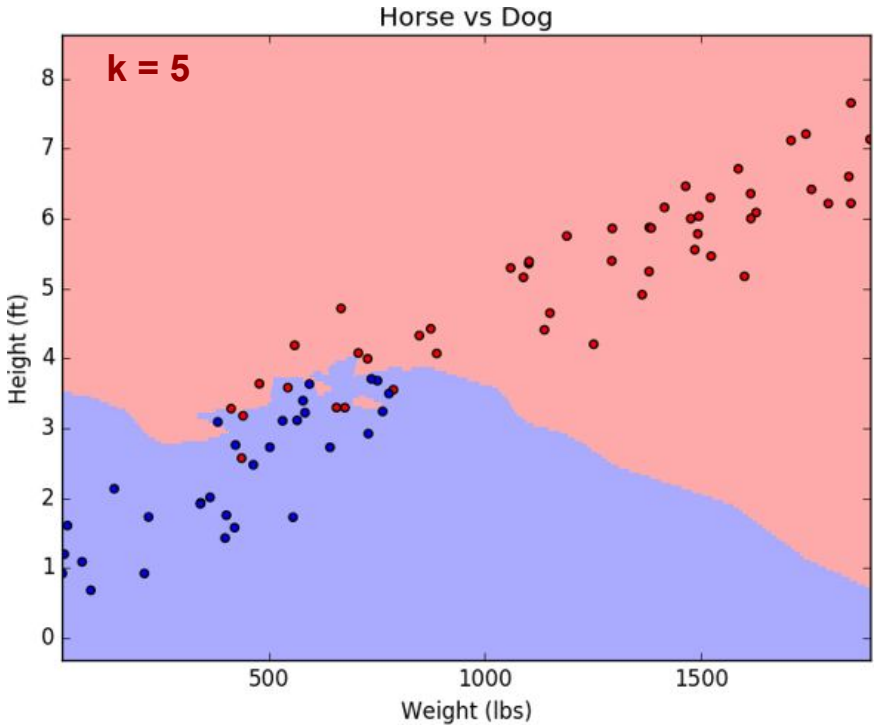
What value for k?



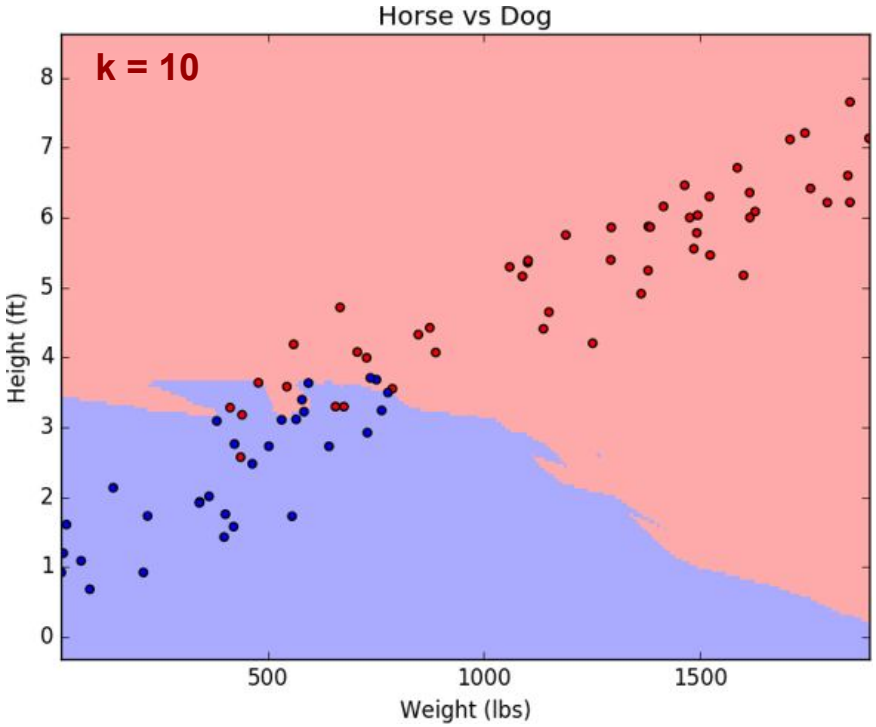
what do you predict for k = 6?

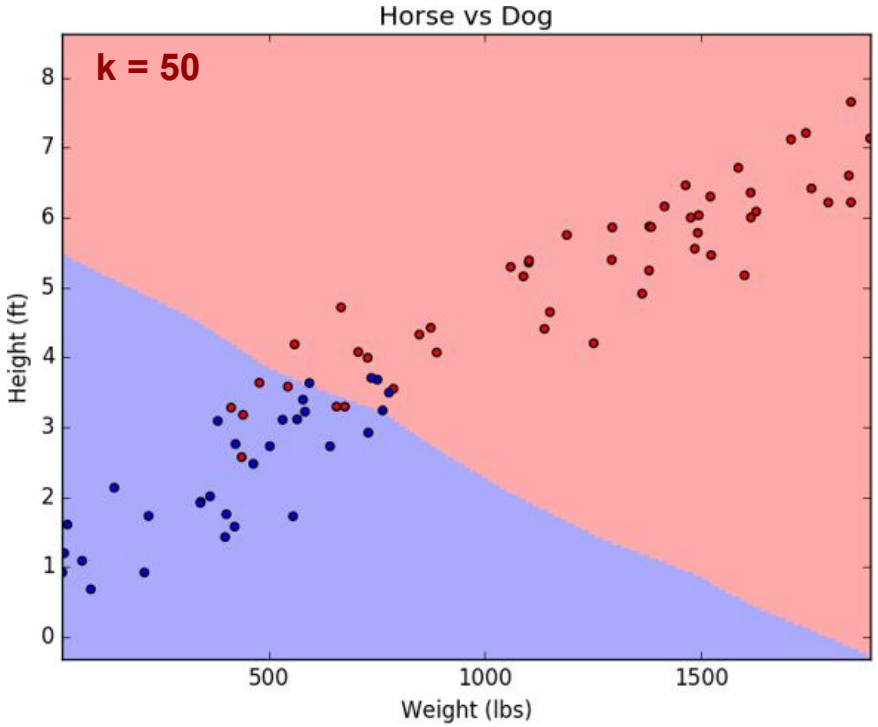
What value for k?



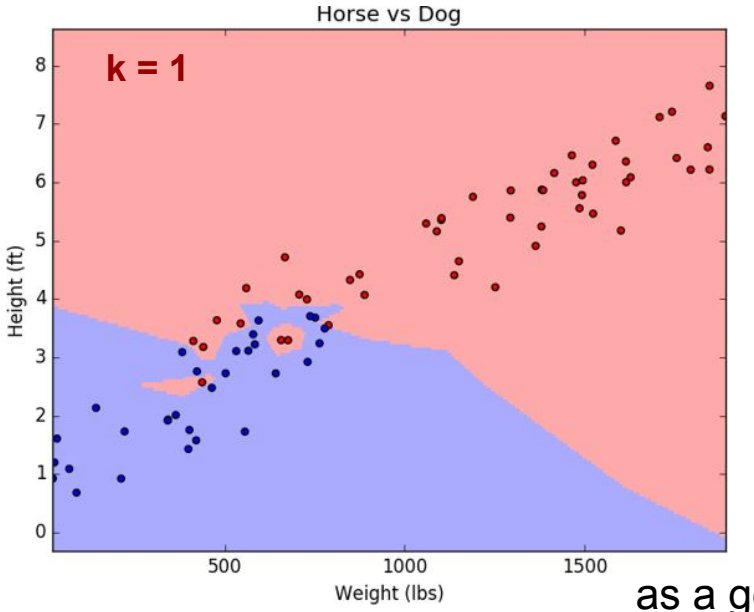


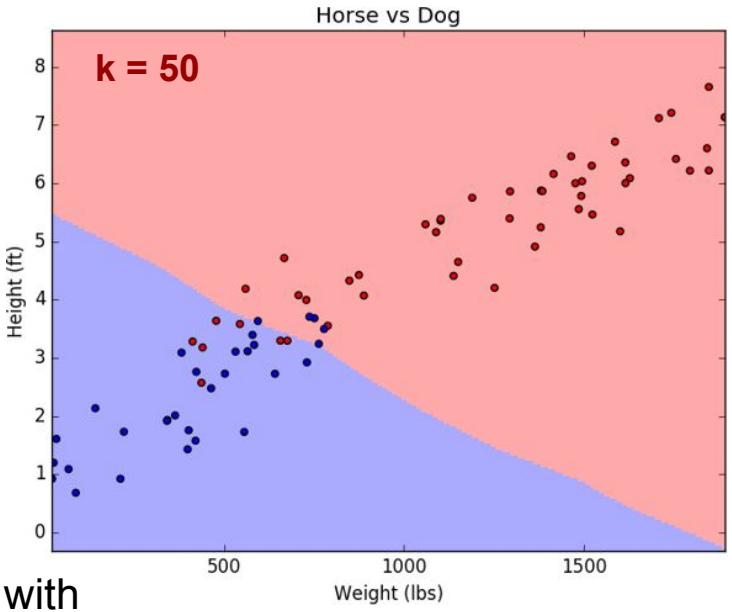
What value for k?





Which Model Overfit?

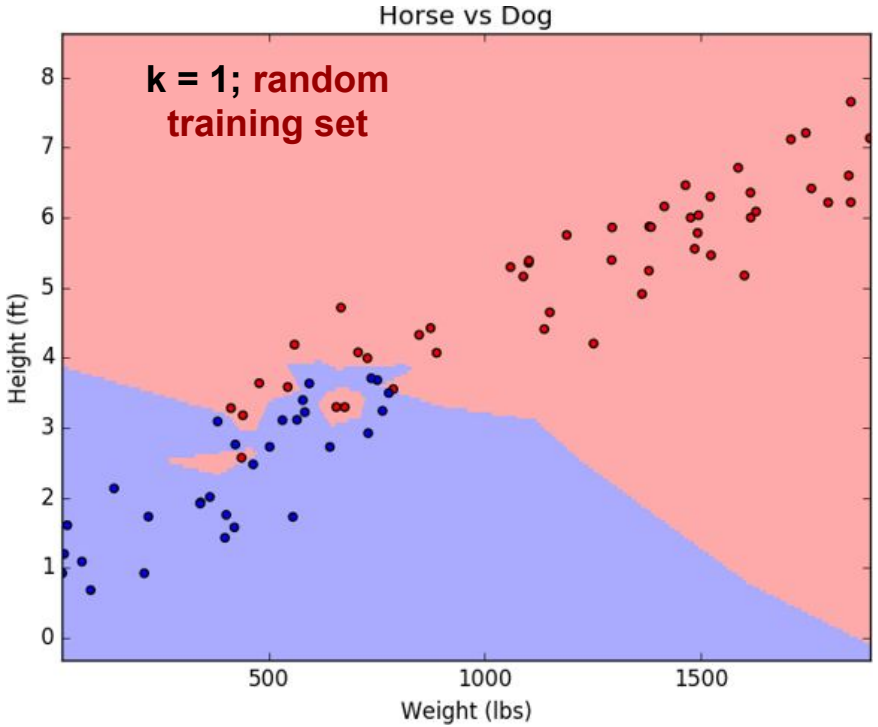


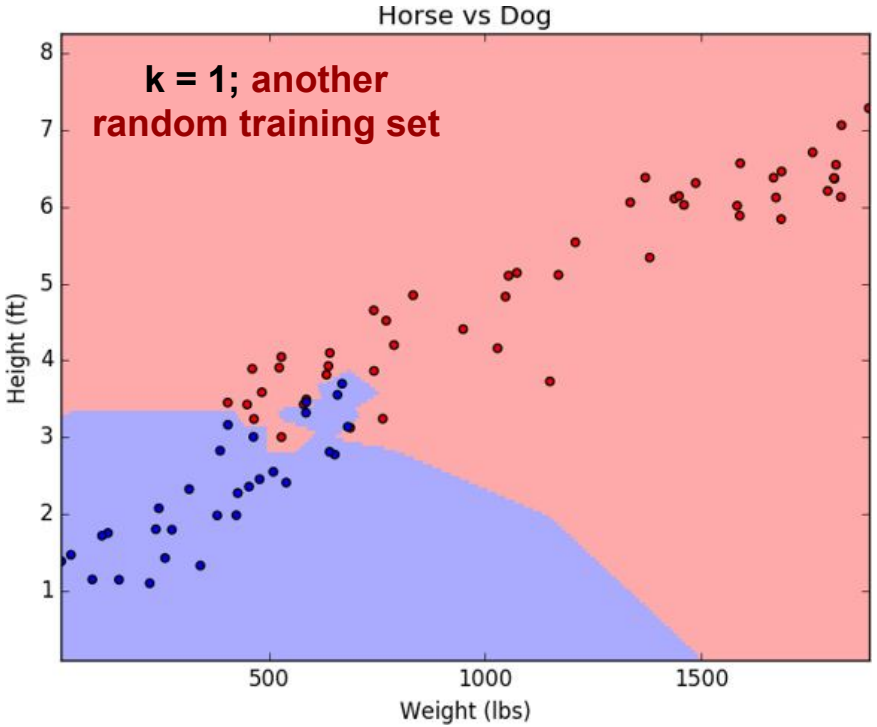


as a general rule, start with

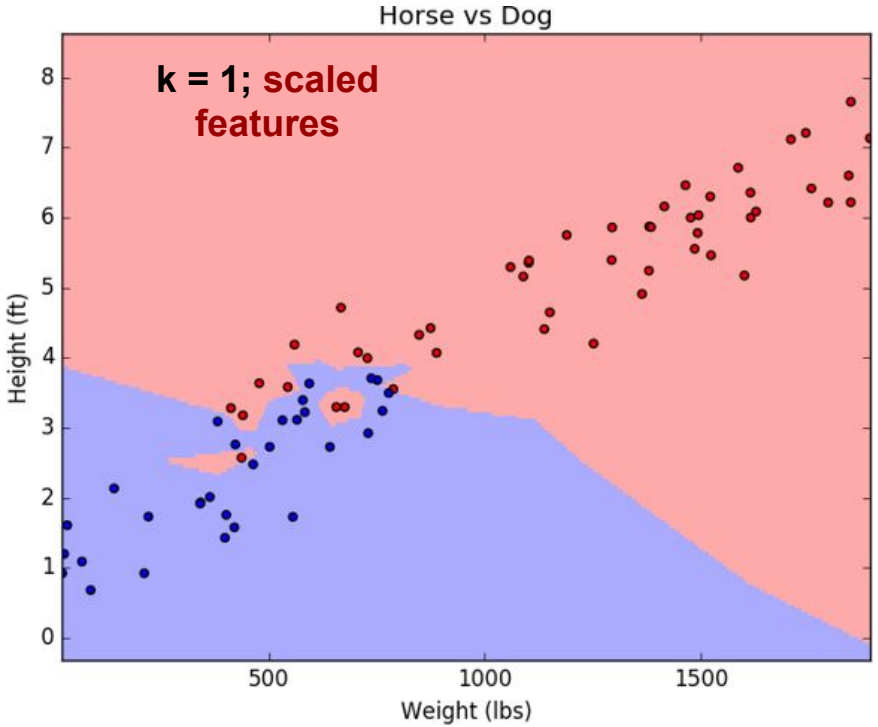
$$k = \sqrt{n}$$

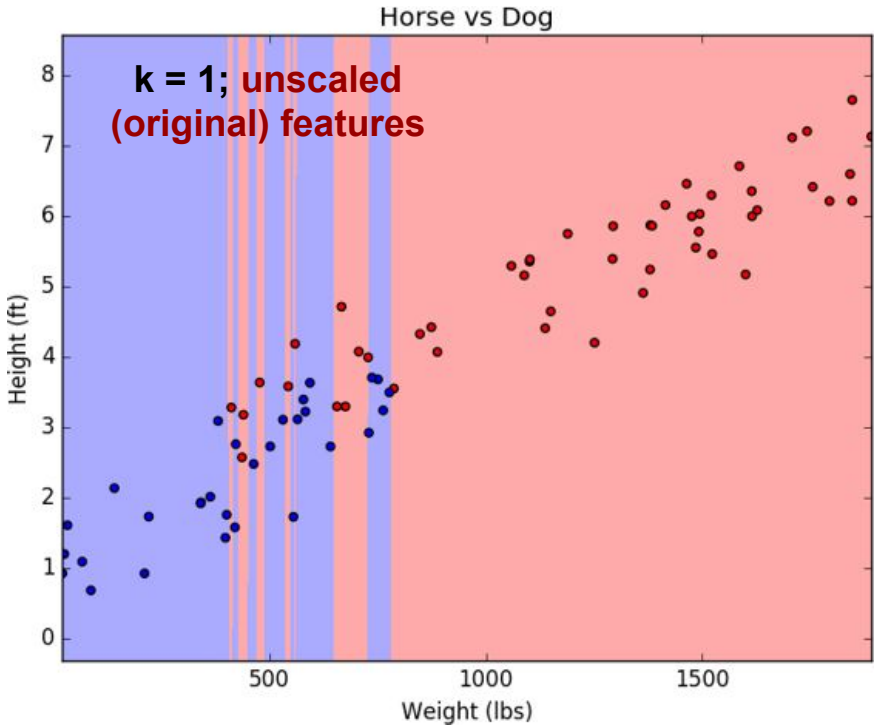
High Variance





Scaling Features





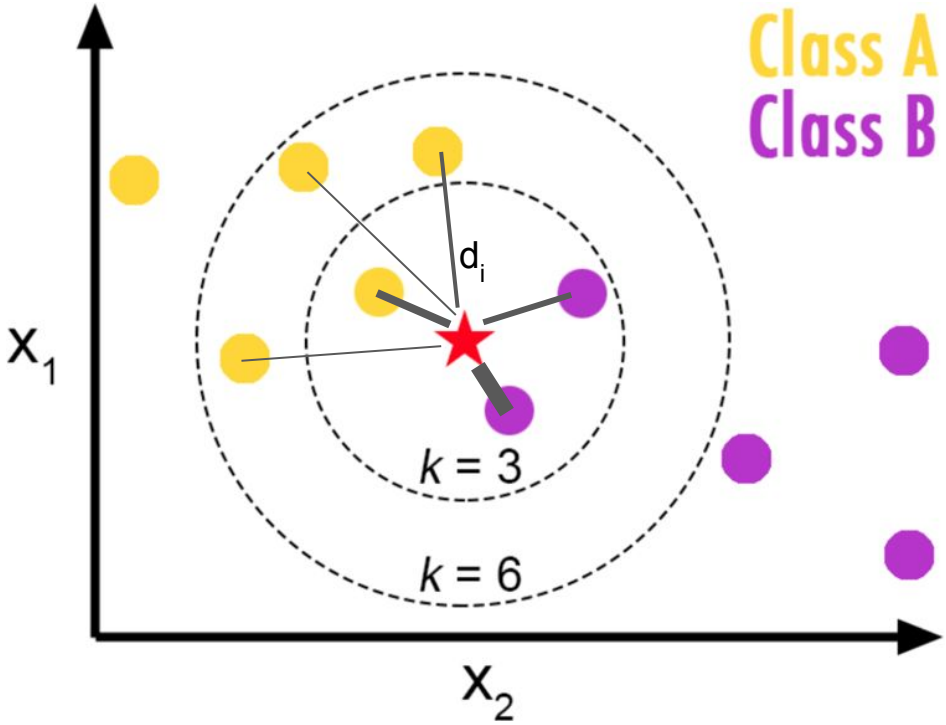
Distance Metrics

$$euclidean\ distance\ (L2) : \sum_i (a_i - b_i)^2$$

$$manhattan\ distance\ (L1) : \sum_i |a_i - b_i|$$

$$cosine\ distance : \ 1 - cosine\ similarity = 1 - \frac{a \cdot b}{\|a\|\ \|b\|}$$

Weighted Voting; Closer Points are Weighted More



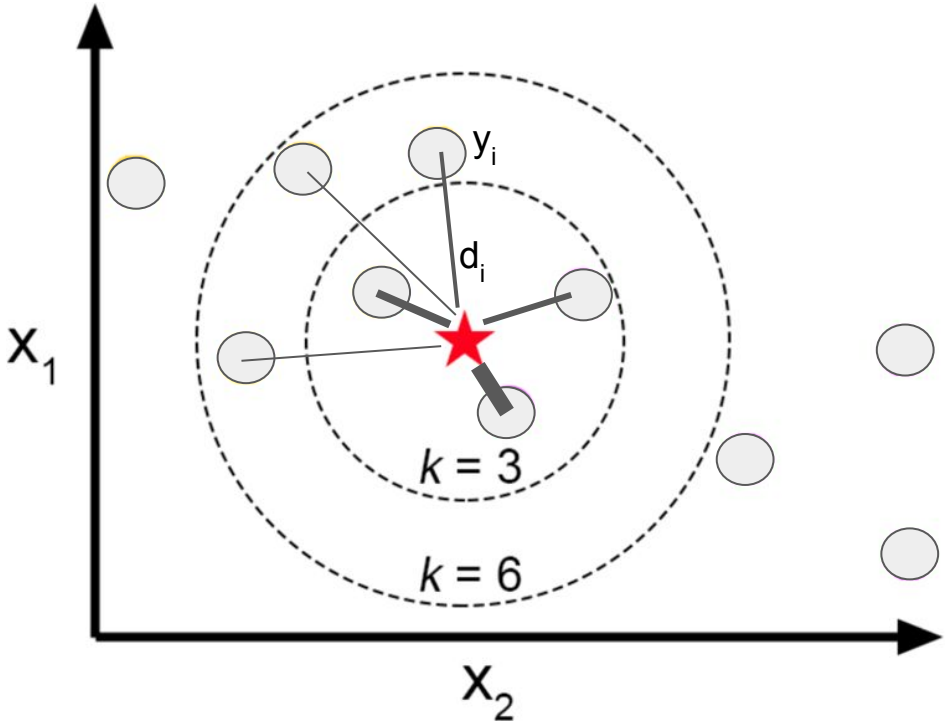
- let the k nearest points i have distances

$$d_i,\ 0 \le i < k$$

- point i votes with weight

$$1 / d_i$$

k-NN for Regression



- let the k nearest points i have outcomes and distances

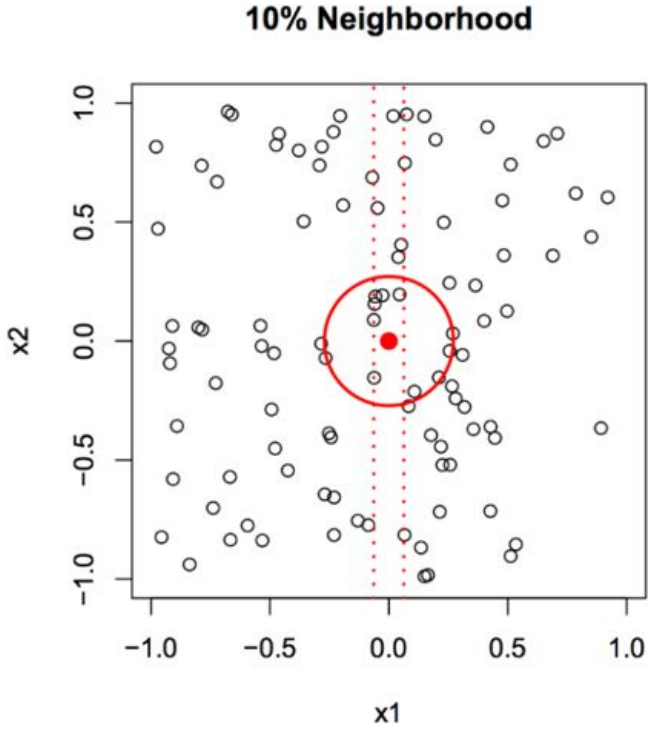
  y_i and d_i, $0 \leq i < k$

- predict the (weighted) mean value of the k neighbors

The Curse of Dimensionality

- k-nn works pretty well (in general) for lower dimensional (d) spaces (d<5) but becomes problematic with higher dimensional spaces, the reason being that “nearest” neighbors become very “far” away in high dimensions
- say you want to use a neighborhood of f = k / n = 10%
- let’s see how the radius behaves as we increase the dimensionality

The Curse of Dimensionality



$$when\ p = 1, r = \frac{f}{2} = .05$$

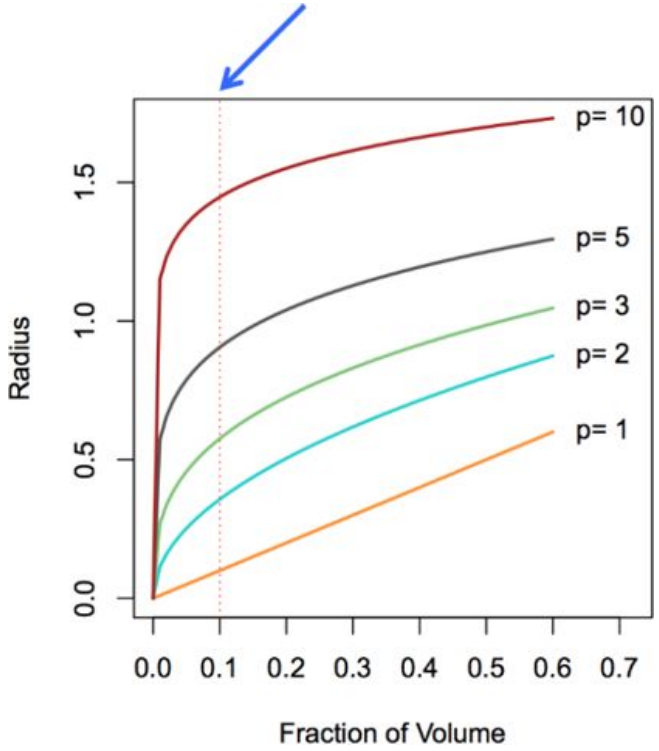
$$when\ p = 2, r = (\frac{f}{\pi})^{1/2} = .18$$

$$when\ p = 3, r = (\frac{f}{\frac{4}{3}\pi})^{1/3} = .28$$

The Curse of Dimensionality

$$for\ p > 3, r = \frac{1}{\pi^{1/2}}(f\,\Gamma(p/2+1)))^{1/p}$$

$$and\ r \to \infty\ as\ p \to \infty$$



Pros and Cons

- pros
  - very simple to use and understand
  - training is fast/trivial (basically memorize the data)
  - can learn a complex function
  - no relationships needed between variables (e.g., linearity)
  - works with any number of classes
  - few hyperparameters

- cons
  - predicting is slow (especially for large datasets)
  - difficulties with imbalanced classes
  - break down with high-dimensionality
    - (but we'll learn dimensionality reduction methods in two weeks!)
  - categorical features don't work well… (how do you define a distance for them?)

(that’s all for this morning)

Decision Trees

Ivan Corneillet

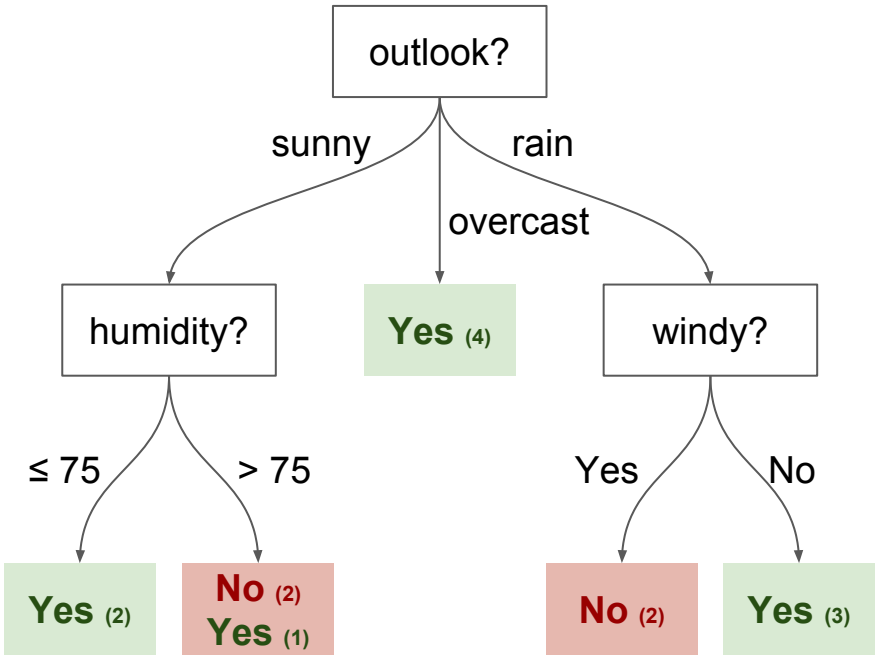
Playing Tennis

- let's say that my buddy and I plan to play tennis this upcoming Saturday and Sunday
  - he flaked on me before :(

- can I use our play history to predict if he is going to flake on me again this weekend?

Temperature	Outlook	Humidity	Windy	Played?
Mild	Sunny	80	No	Yes
Hot	Sunny	75	Yes	**No**
Hot	Overcast	77	No	Yes
Cool	Rain	70	No	Yes
Cool	Overcast	72	Yes	Yes
Mild	Sunny	77	No	**No**
Cool	Sunny	70	No	Yes
Mild	Rain	69	No	Yes
Mild	Sunny	65	Yes	Yes
Mild	Overcast	77	Yes	Yes
Hot	Overcast	74	No	Yes
Mild	Rain	77	Yes	**No**
Cool	Rain	73	Yes	**No**
Mild	Rain	78	No	Yes

A Decision Tree



- saturday
  - cool temperature
  - sunny outlook
  - 70% humidity
  - no wind

  will play

- sunday
  - mild temperature
  - rainy outlook
  - 60% humidity
  - windy

  won't play

Terminology

a decision tree consists of:

- nodes
  - split (test) for the value of a certain attribute
- edges
  - correspond to the outcome of a test and connect a node to the next node
- root
  - the node that performs the first split
- leaves
  - terminal nodes that predict the outcome

Basic Idea in Building Trees: Separating Classes

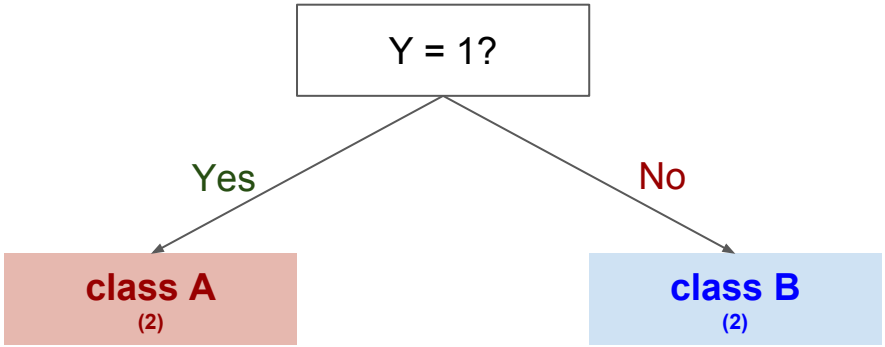
toy example: training set with 3 features (X, Y, and Z) and 2 classes as outcome

X	Y	Z	Class
1	1	1	**A**
1	1	0	**A**
0	0	1	**B**
1	0	0	**B**

how can we distinguish class A from class B?

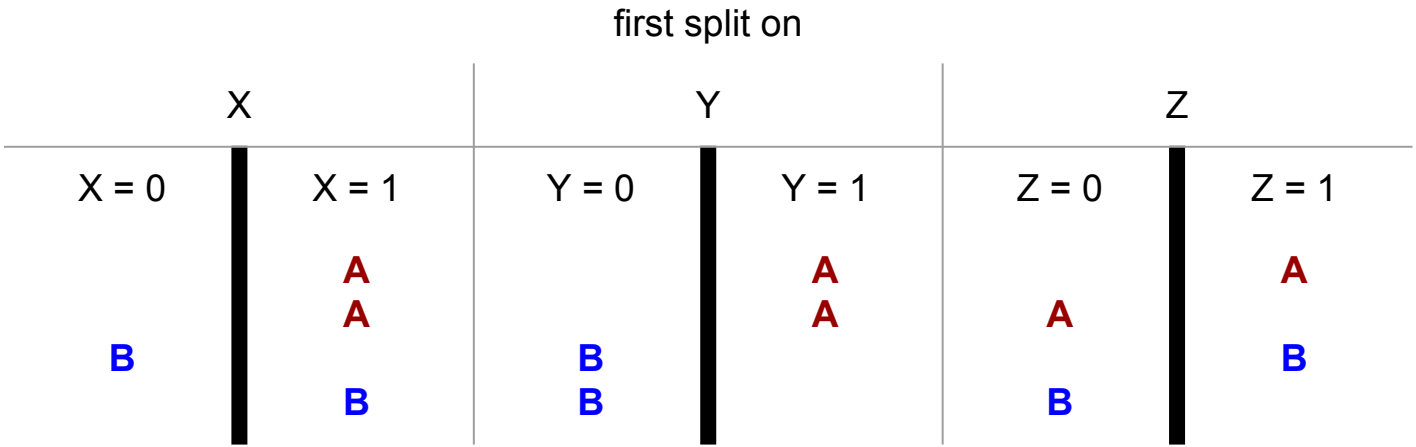
Exploring our Intuition

- splitting on Y gives us a clear separation between classes



- we could also first split on X then on Z but this is less optimal

Formalizing this Intuition



- we need to understand the concept in which a split is “best”
- entropy and information gain gives us a way to quantitatively measure which feature is best to split on on each node of the tree

Entropy and Information Gain

$$Entropy:$$

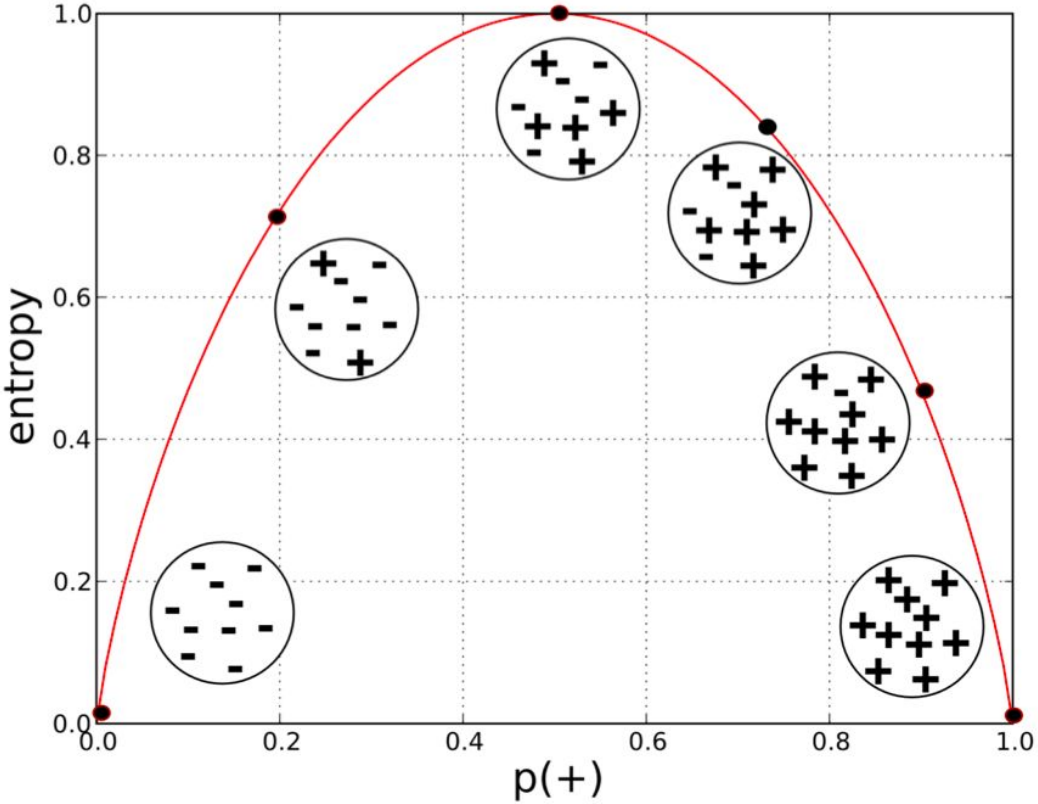
$$H(S) = -\sum_{i} p_i(S)\, log_2\, p_i(S)$$

$$Information\ Gain:$$

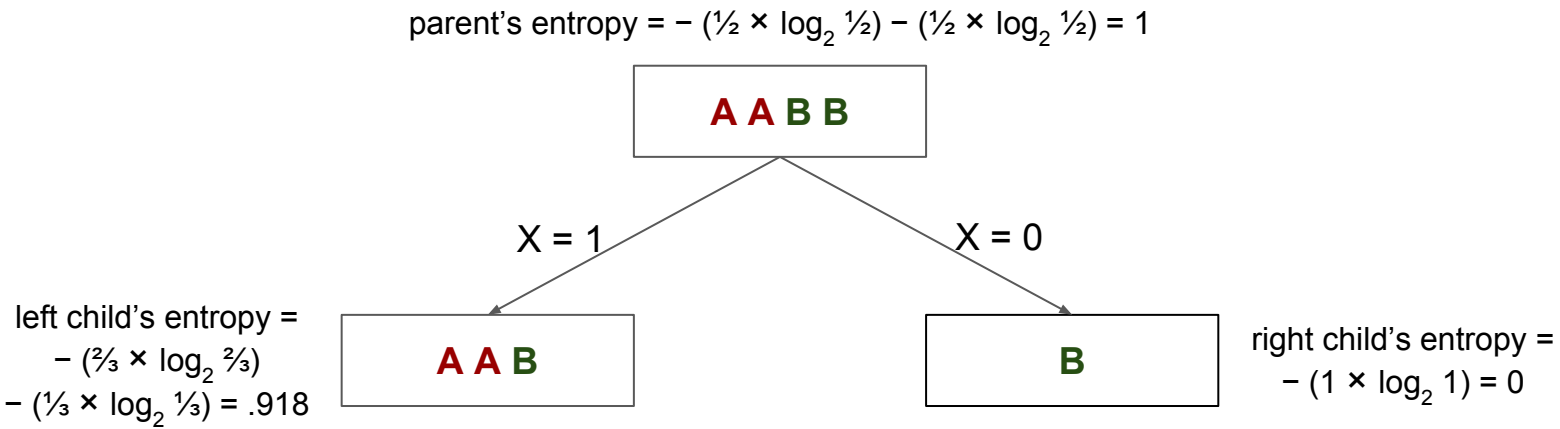
$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{S} H(S_v)$$

- H(S): entropy for a set S; each element of S belongs to a certain class ‘i’
- pi(S): probability of class ‘i’ in the set S
- A: attribute sets can be split on (e.g., outlook, humidity, windy…)

More on Entropy

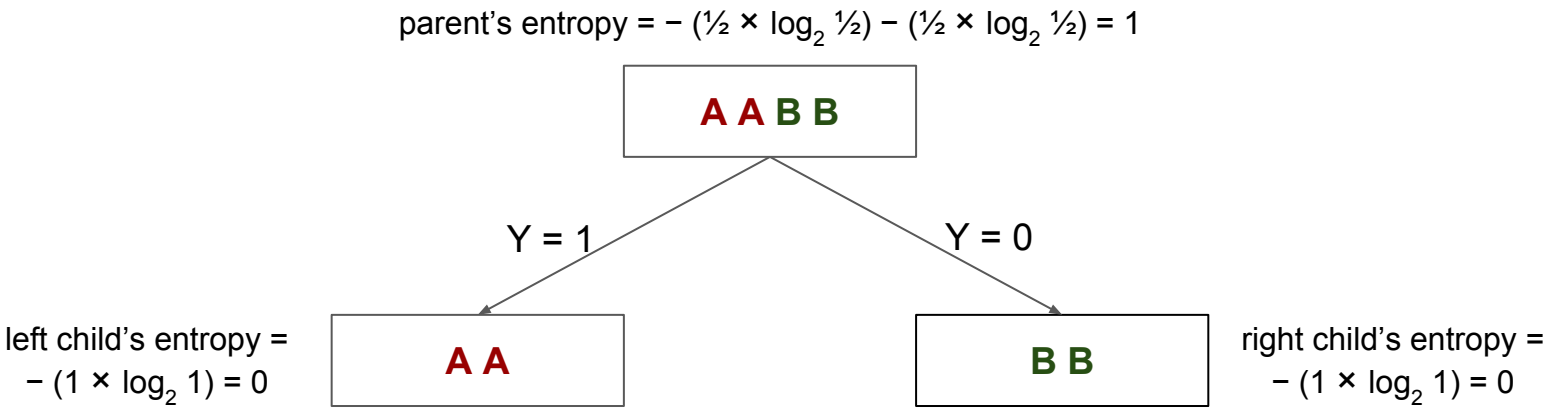


Our Toy Example; Splitting on X



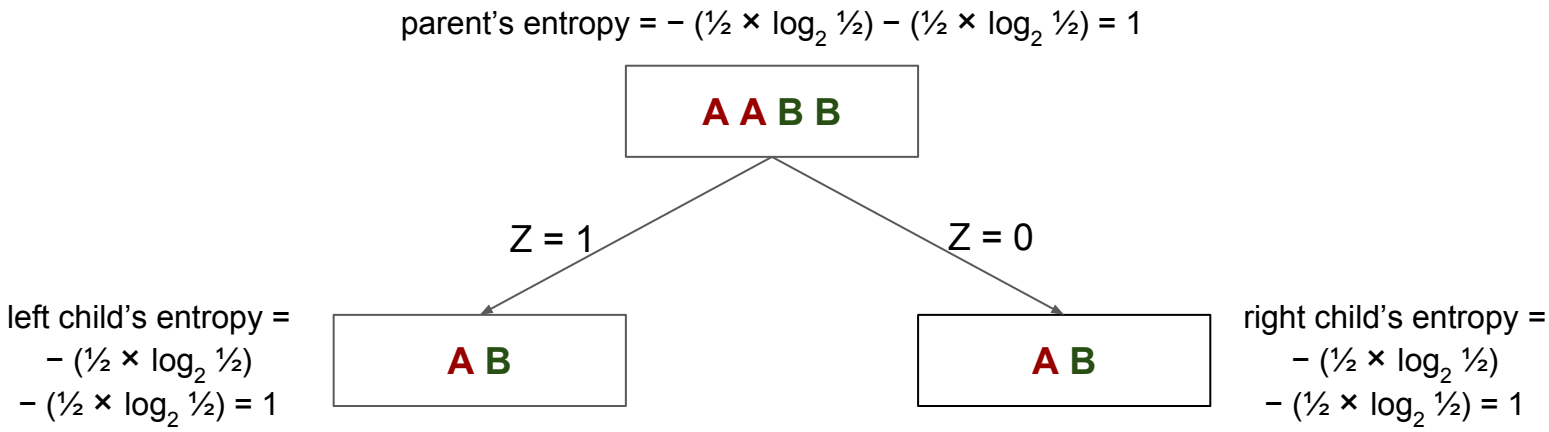
information gain from splitting on X = 1 − (¾ × .918 + ¼ × 0) = .311

Our Toy Example; Splitting on Y



information gain from splitting on X = 1 − (½ × 0 + ½ × 0) = 1, **BEST!**

Our Toy Example; Splitting on Z



information gain from splitting on X = 1 − (½ × 1 + ½ × 1) = 0

Gini Index: Another Splitting Measure

- randomly select an element from S
- randomly label it according the class proportions
- the gini index is the probability it was labeled incorrectly

$$Gini\ Index:$$

$$\sum_i p_i(S)(1 - p_i(S)) = 1 - \sum_i p_i(S)^2$$

Building a Decision Tree

```
function build_decision_tree(dataset):
    if every item in the dataset is in the same class
        or there is no feature left to split the data:
            return a leaf node with the class label
    else:
        create a node
        find the best feature and value to split the data (next slide)
        for each split
            call build_decision_tree(subsetted dataset)
                and add the resulting node as a child of the current
            node
        return node
```

Find the Best Feature and Value to Split the Data

splitting algorithm:

1. calculate the information gains for all possible splits
2. select the split that has the highest information gain

possible splits: consider all binary splits based on a single feature

- categorical feature
  - `variable = value`
    - (or `variable ≠ value`)
- continuous feature
  - `variable ≤ threshold`
    - (or `variable > threshold`)

Regression Decision Trees

- responses are real values
- so we can’t use information gain or gini index
- instead chose the best splits using residual sum of squares (against the mean value of each leaf)
- can also use a combination of decision trees and linear regression on the leaf nodes (model trees)

What could Possibly Go Wrong?

your tree will (correctly) fit EVERY SINGLE observation from the training set

in other words, your decision tree will **OVERFIT**

we prune our trees to address overfitting

Pre-Pruning: Stopping Early when Building the Tree

- min leaf size
  - stop when the number of data points for a leaf gets below a threshold
- max depth
  - stop when the depth of the tree (distance from root to leaf) reaches a threshold
- purity
  - stop when enough of the data points are of the same class
- gain threshold
  - stop when the information gain is not improved significantly

Post-Pruning: Build a Full Tree; then Cut off some Leaves

```
function post_pruning(node):
    if either the left node
        or the right node is not a leaf node (or both):
            call prune the non-leaf node(s)

    if both left and right nodes are leaf nodes:
        calculate error associated with merging two nodes
        calculate error associated without merging two nodes
I       merge the leaf nodes if merging results in lower error
    else:
        (leave them untouched)
```

In Practice

- there are many ways to build trees
- always prune to avoid overfitting
- chose either entropy/information gain or gini index (but classification error is a big no-no)

Decision Trees in sklearn

- doesn’t support missing values
- gini index is default, but you can select entropy instead
- prune with max_depth, min_samples_split, min_samples_leaf or max_leaf_nodes
- does binary splits (you would need to binarize categorical features)

Pros and Cons

- pros
  - low bias (usually)
  - simple to understand and interpret; trees can be visualised!
  - non-parametric and non-linear → can model more complex things
  - computationally easy to predict
  - deals with irrelevant features
  - work with mixed (categorical and continuous features)
- cons
  - high variance (overfit)
  - computationally expensive to train
  - greedy algorithm (locally optimal decisions are made for each split)

(that’s all for today)