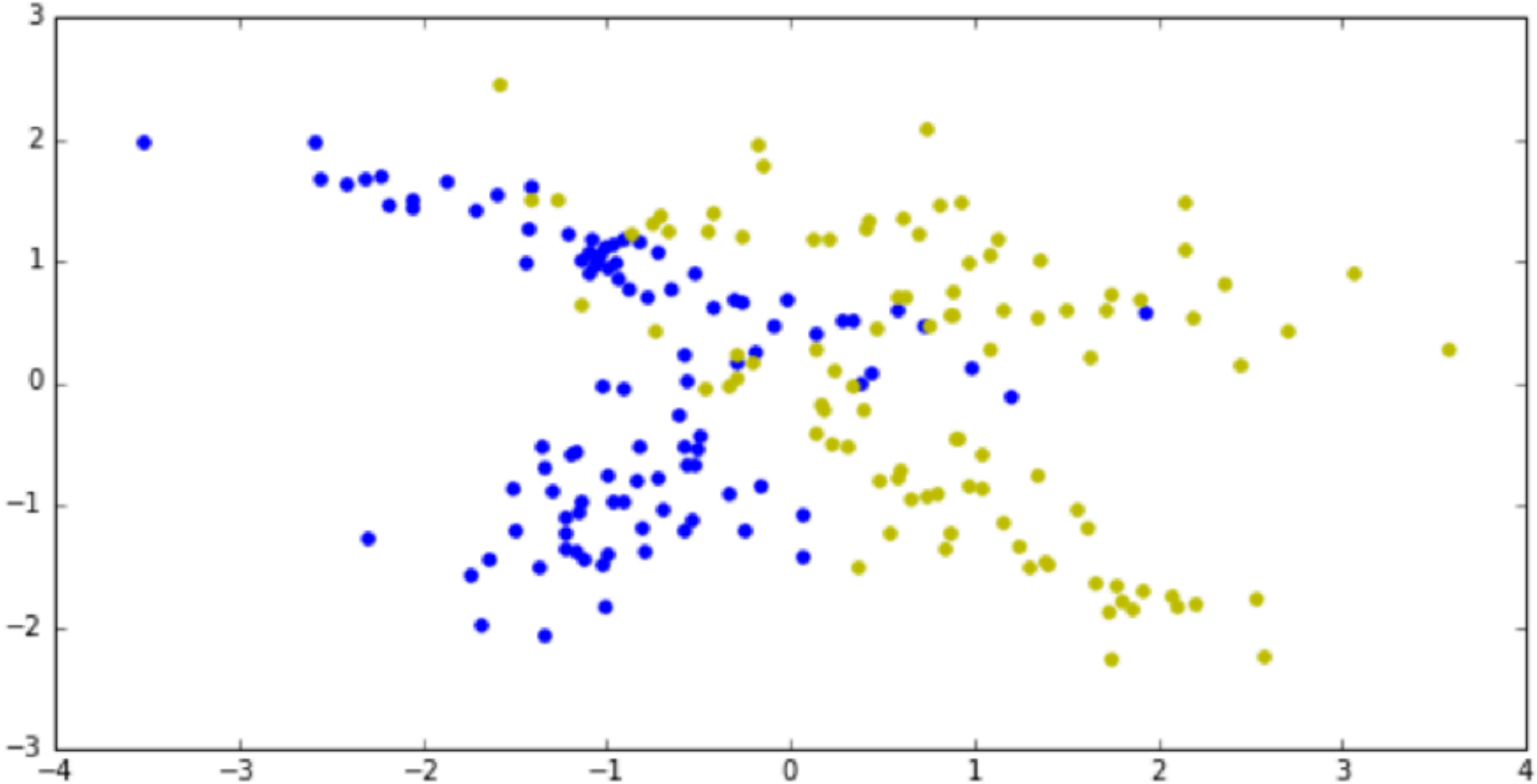
KNN and Decision Trees

Non-parametric learners

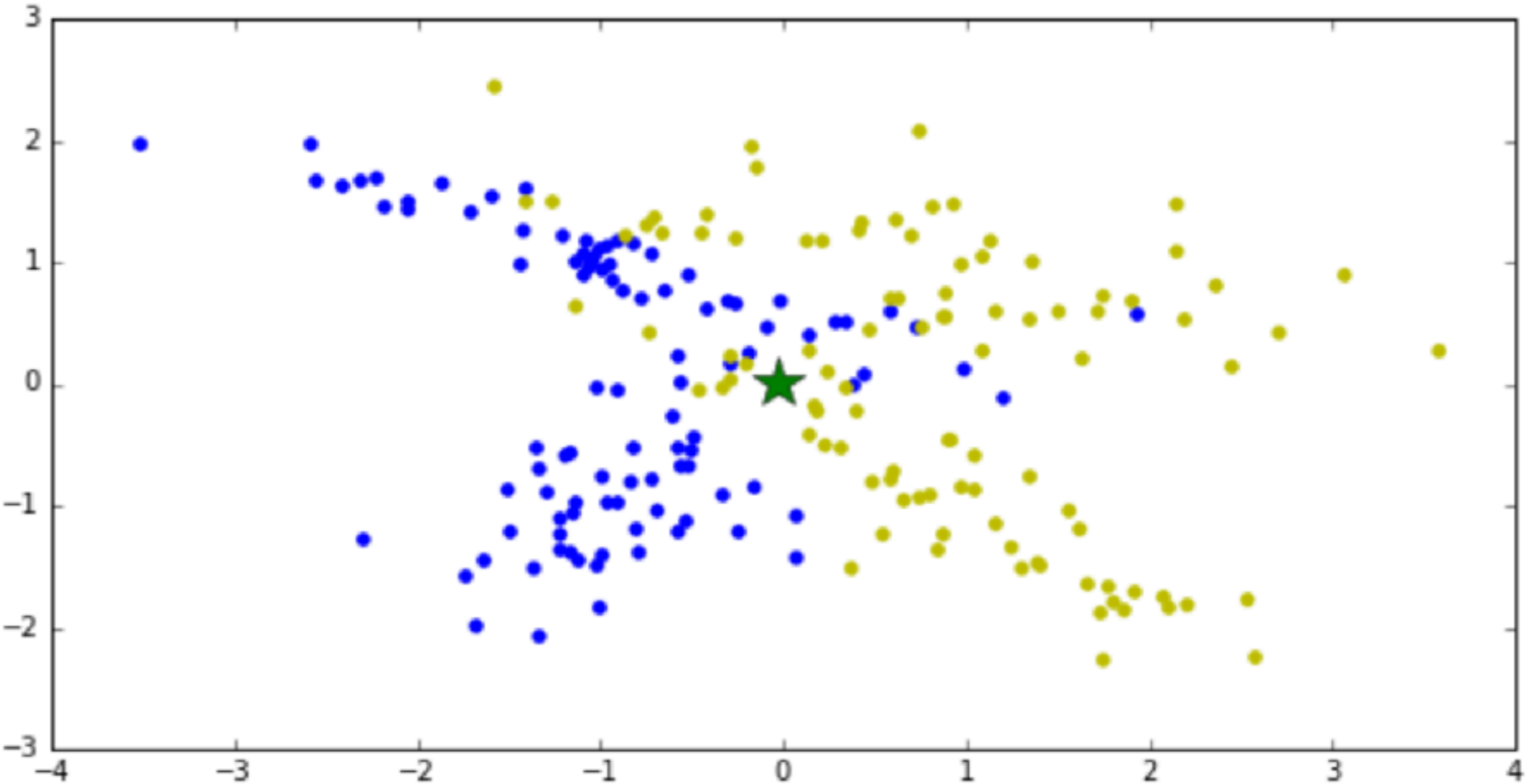
KNN: Goals

- At the end of today's lecture you should:
  - Be able to describe the KNN algorithm
  - Describe the curse of dimensionality
    - Recognize the conditions under which the curse may be problematic
  - Explain the strengths and weaknesses of KNN

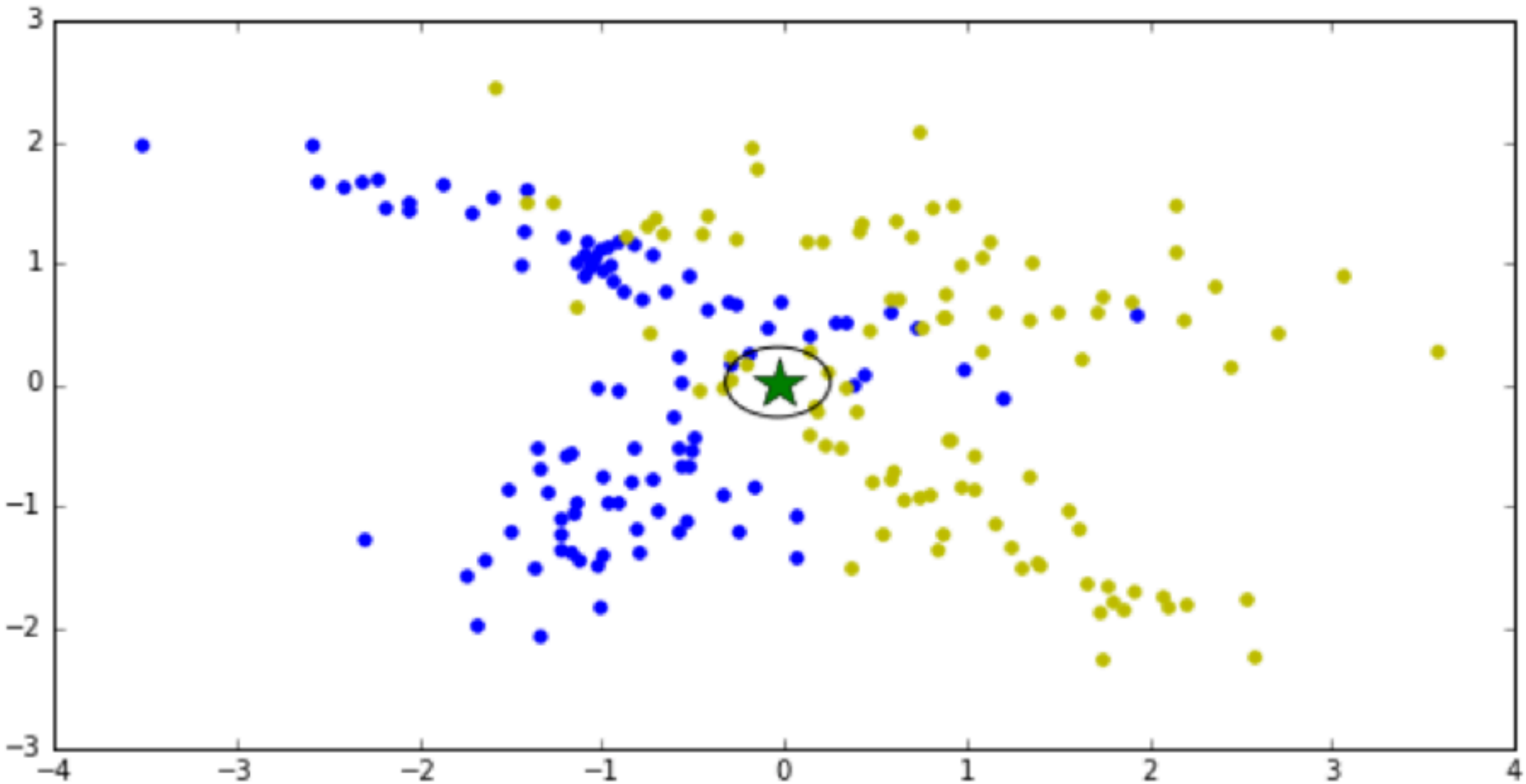
1: data



2: new data point



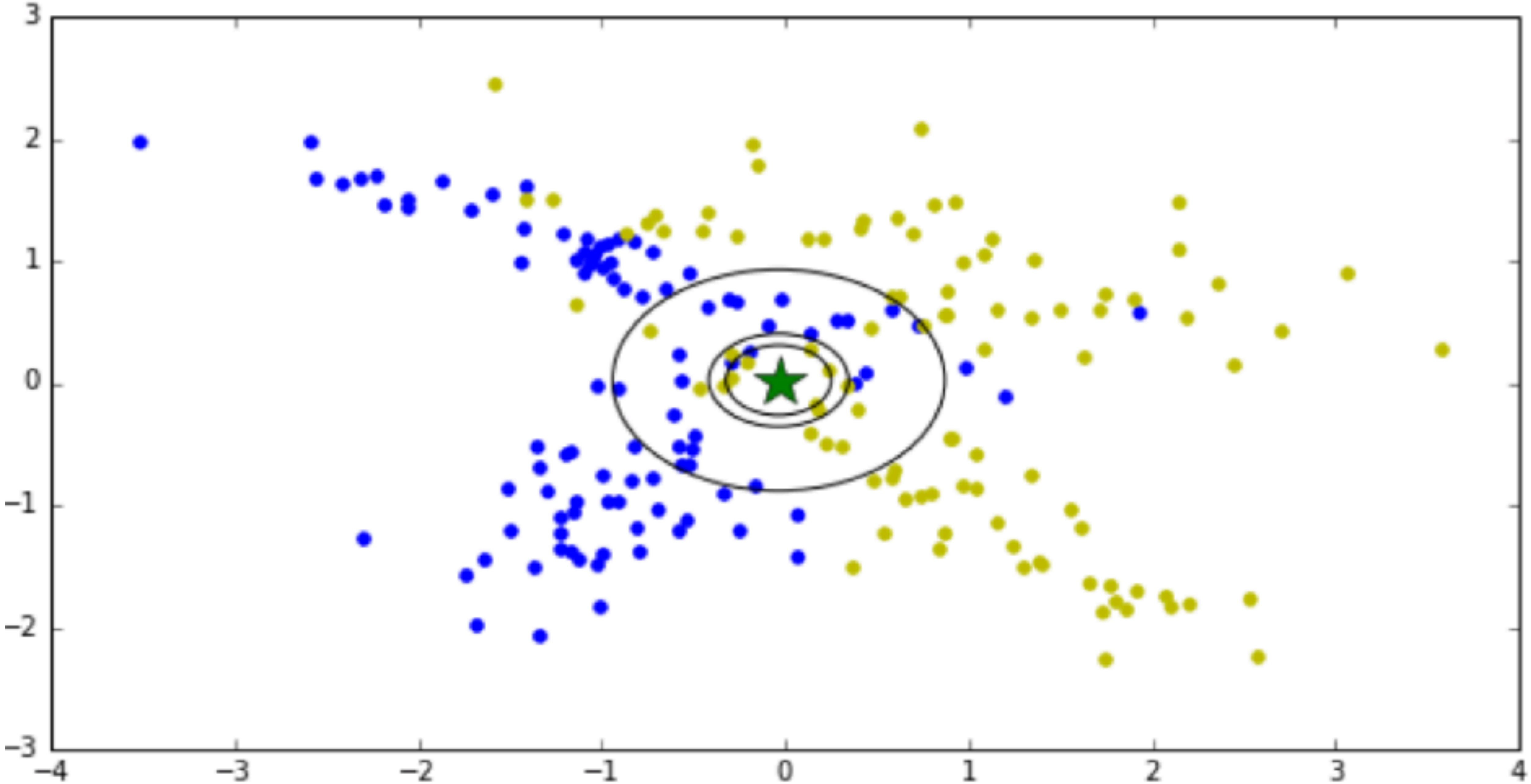
3: neighborhood



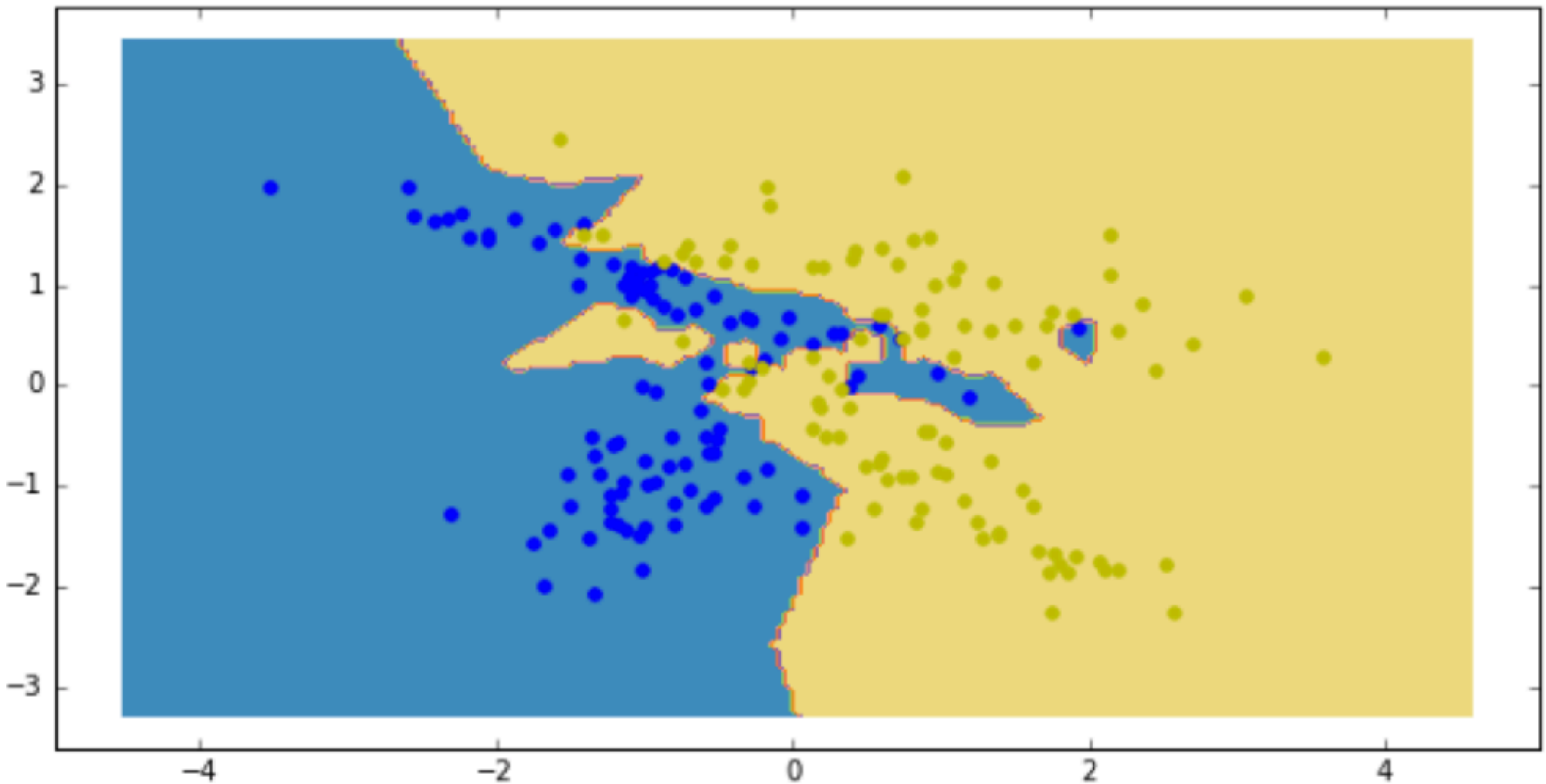
Steps

- Training:
  - 1: Store the data
  - 2: There is no step 2
- Prediction:
  - Find the k nearest points to the new point
  - Output as your prediction the majority (classification) or average (regression) of those k nearest points

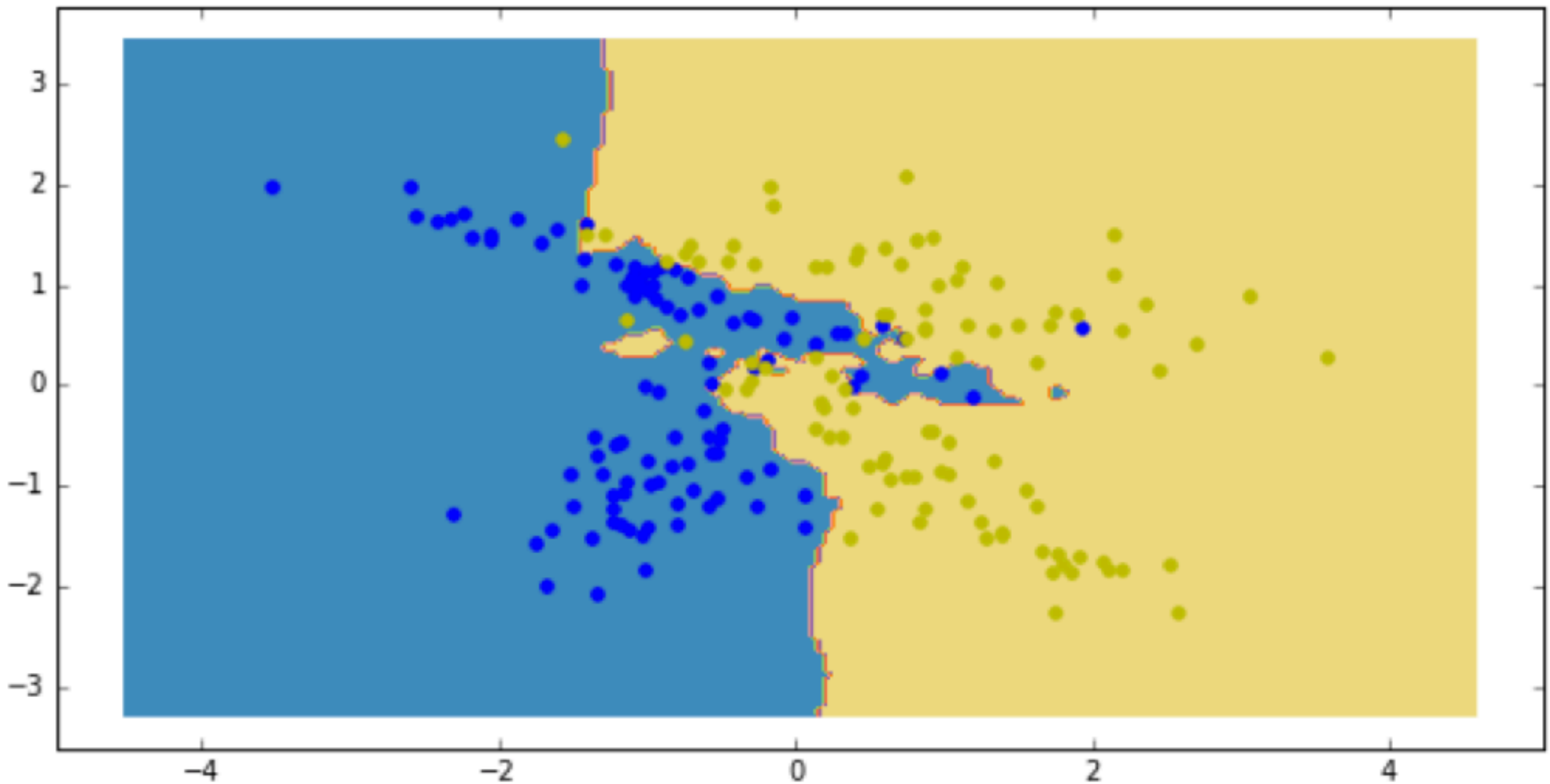
What is k?



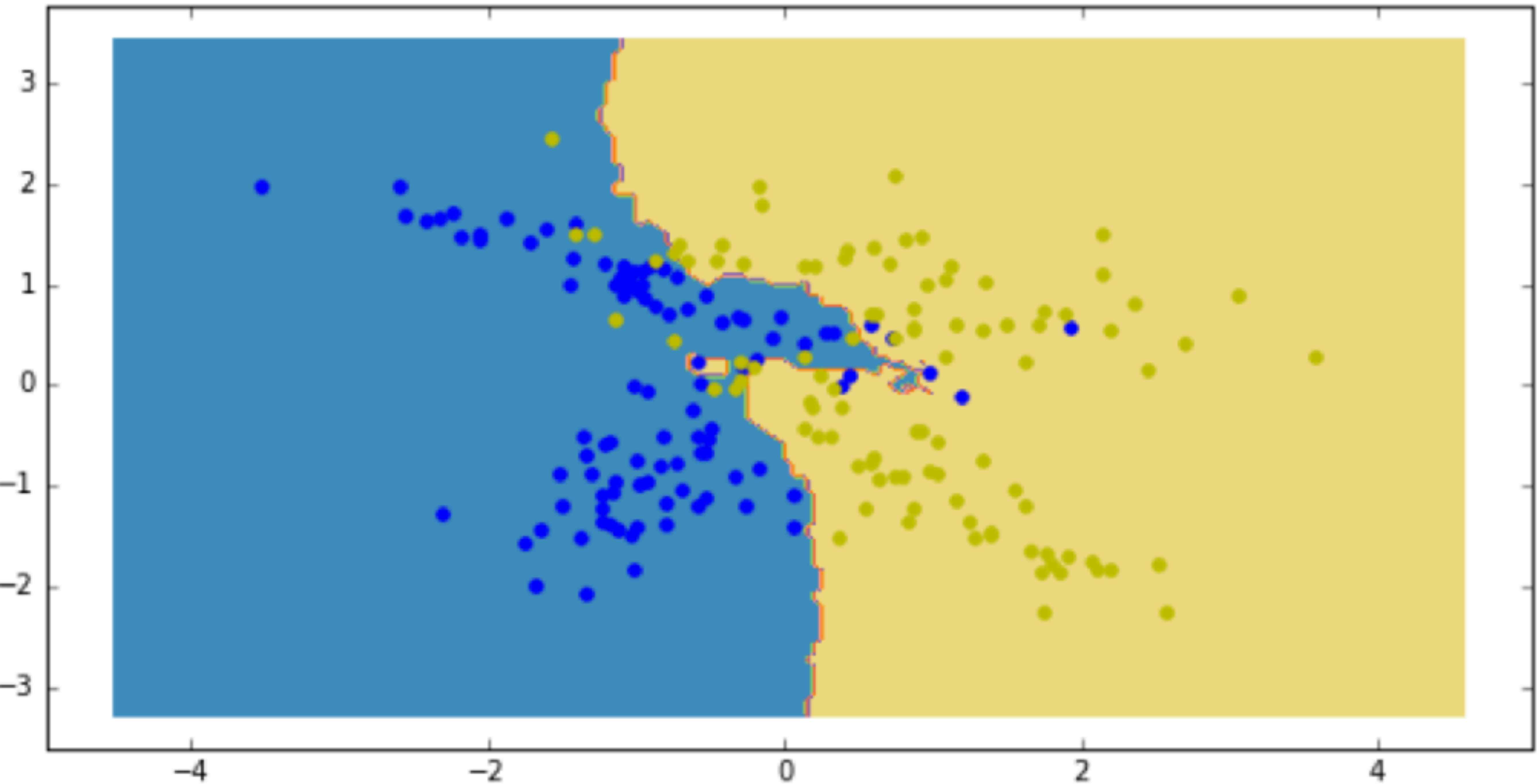
k=1



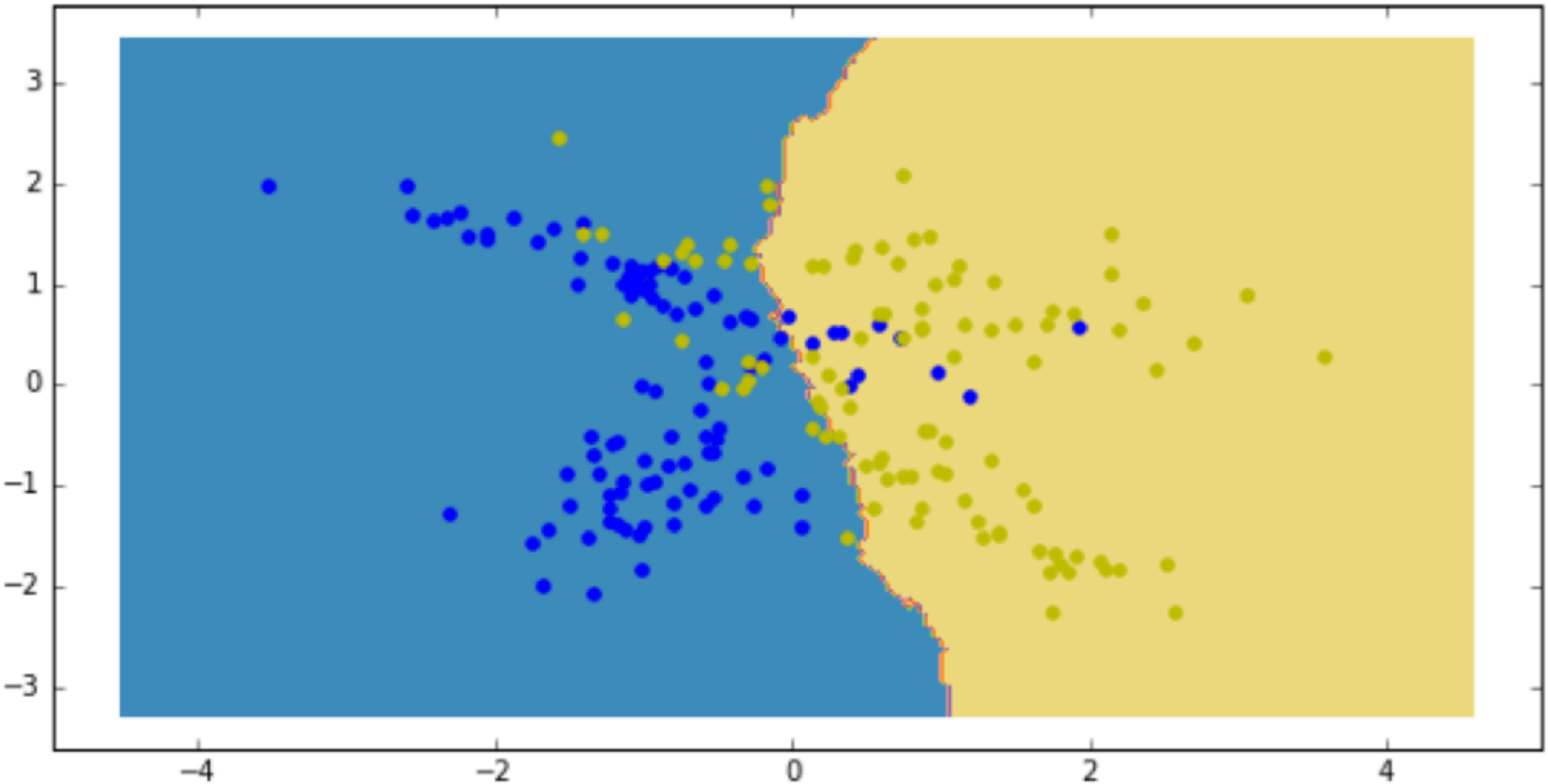
k=3



k=10



k=100



Determining K

- What if $k = 1$?
  - What if we evaluate on our training data? (!)
- What if $k = n$?
- Use Cross Validation
  - Start with $k = \sqrt{n}$

Distance Metrics

- Euclidean
  - (Straight Line, L2)

$$\sqrt{\sum_i (a_i - b_i)^2}$$

- Manhattan
  - (City Blocks, L1)

$$\sum_i |a_i - b_i|$$

- Cosine
  - (Angle)

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

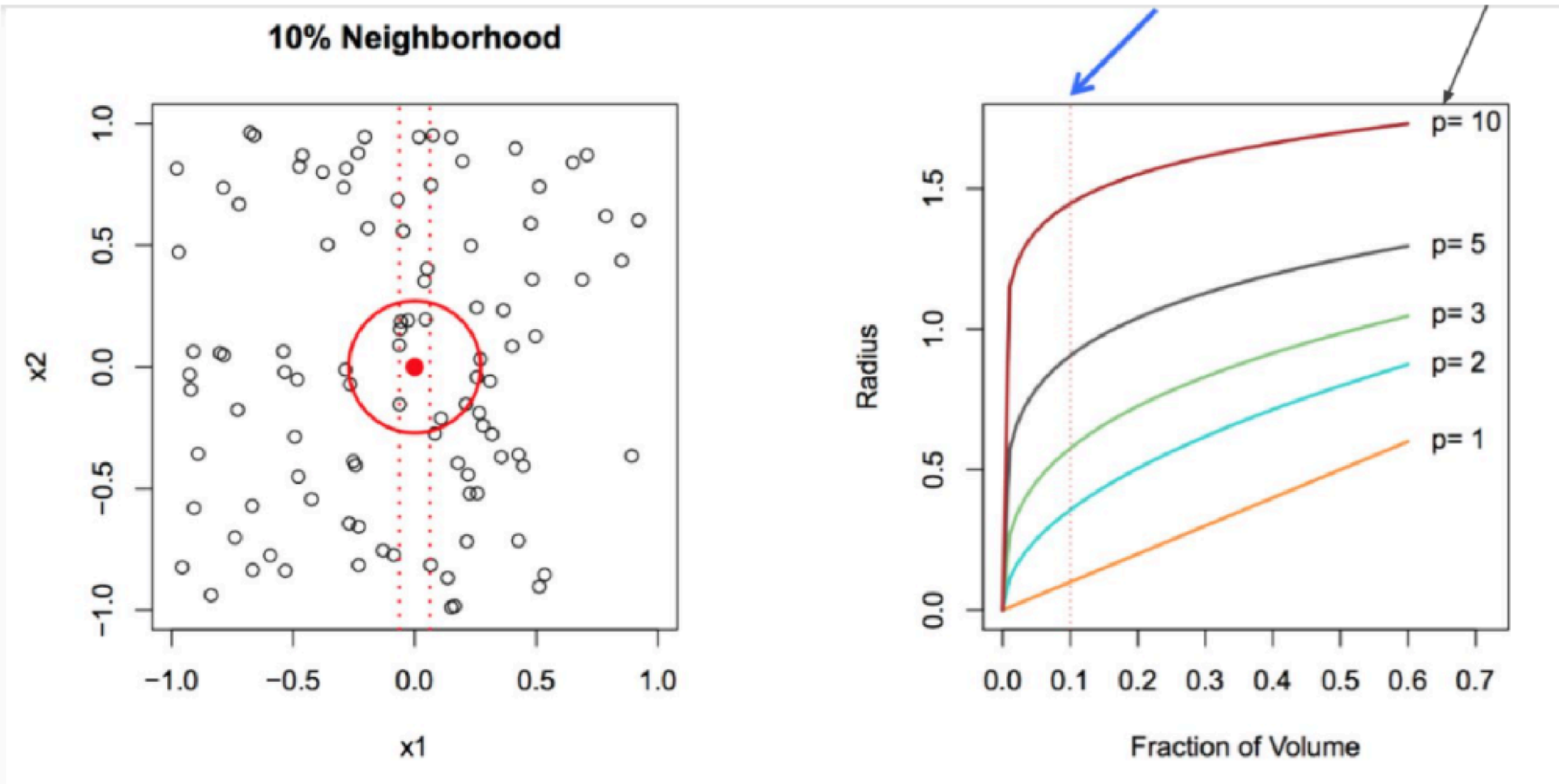
Runtime

- Training
  - Fast, sometimes even trivial (if data already stored as needed)
  - Online updates are easy (just store another data point)
- Prediction
  - Slow
    - IO bound (the cost is reading through data, not calculations)

Curse of Dimensionality

- As dimensionality increases, performance of knn commonly decreases
- *Nearest* neighbors are no longer *nearby* neighbors
- Adding useful features (truly associated with the response) is generally helpful, but noise features increase dimensionality without the upside
- Later: dimensionality reduction

Curse of Dimensionality

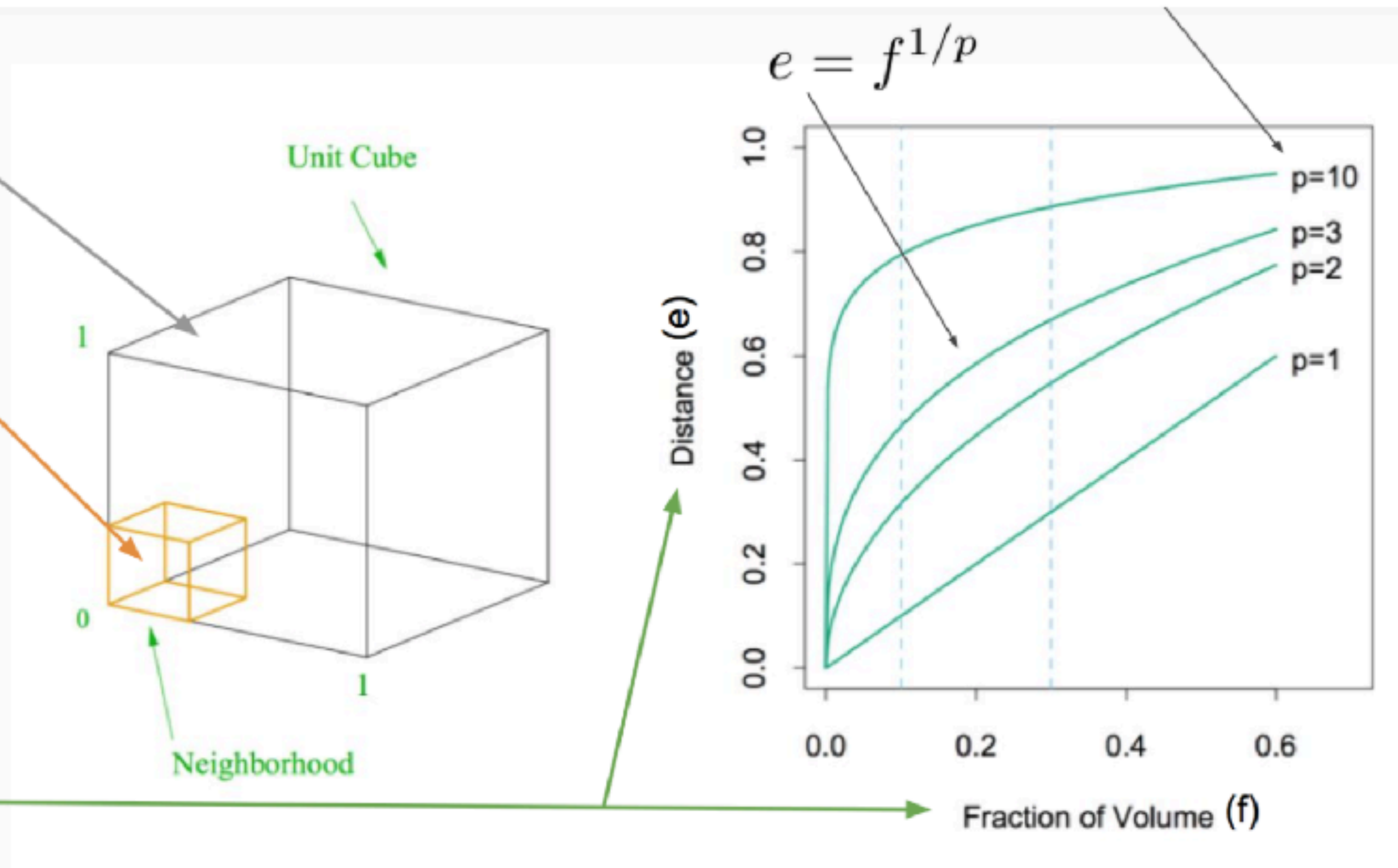


Curse of Dimensionality

Say we have a unit (hyper)cube.

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

How long must the edges of the inner cube be?



Pros and Cons

- Pros
  - Any number of classes is ok
  - Easy to train, store, and update the model
  - Can learn a very complex function
  - No demands on relationships between variables (ie linearity)
  - Few hyperparameters (k, dist metric, …)
- Cons
  - Sloooow to predict
  - Noise can affect results…
    - …particularly irrelevant dimensions
  - Feature interpretation can be tricky
    - Categorical: how to calc distance?
    - Feature scaling

Uses

- Classification
  - Neighbors vote on class
- Regression
  - Neighbors averaged* to give continuous value
    - *or some combo function
- Imputation
  - Replace missing data/values with KNN
- Anomaly Detection
  - Is the nearest neighbor super far away? Maybe new data point is an outlier

KNN Review

- Describe the KNN algorithm
- Describe the curse of dimensionality
    - Recognize the conditions under which the curse may be problematic
- Explain the strengths and weaknesses of KNN

<non-notebook coding review>

Decision Trees: Goals

- Describe a Decision Tree
  - How does one interpret the tree?
  - How does one create the tree?
- Explain Information Gain
- Explain strengths and weaknesses of DTs

Some Data

Temp	Outlook	Humidity	Windy	Played
Hot	Sunny	High	False	No
Hot	Sunny	High	True	No
Hot	Overcast	High	False	Yes
Cool	Rain	Normal	False	Yes
Cool	Overcast	Normal	True	Yes
Mild	Sunny	High	False	No
Cool	Sunny	Normal	False	Yes
Mild	Rain	Normal	False	Yes
Mild	Sunny	Normal	True	Yes
Mild	Overcast	High	True	Yes
Hot	Overcast	Normal	False	Yes
Mild	Rain	High	True	No
Cool	Rain	Normal	True	No
Mild	Rain	High	False	Yes

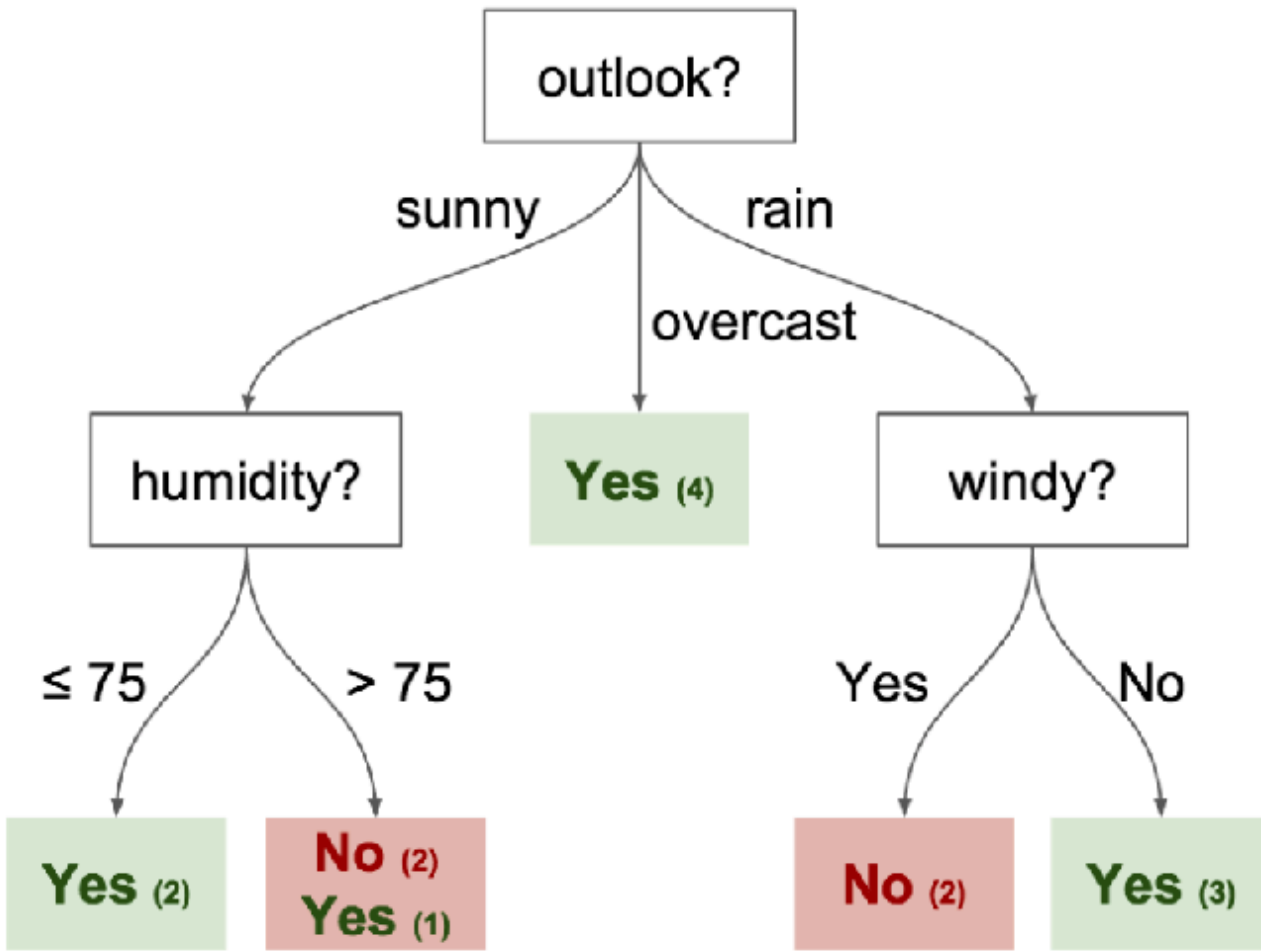
Decision… as if/else

Temp	Outlook	Humidity	Windy	Played
Hot	Sunny	High	False	No
Hot	Sunny	High	True	No
Hot	Overcast	High	False	Yes
Cool	Rain	Normal	False	Yes
Cool	Overcast	Normal	True	Yes
Mild	Sunny	High	False	No
Cool	Sunny	Normal	False	Yes
Mild	Rain	Normal	False	Yes
Mild	Sunny	Normal	True	Yes
Mild	Overcast	High	True	Yes
Hot	Overcast	Normal	False	Yes
Mild	Rain	High	True	No
Cool	Rain	Normal	True	No
Mild	Rain	High	False	Yes

```
def will_play(temp, outlook, humidity,\
              windy):

    if outlook == 'sunny':
        if humidity == 'normal':
            return True
        else: # humidity == 'high'
            return False

    elif outlook == 'overcast':
        return True

    else: # outlook == 'rain'
        if windy == True:
            return False
        else: # windy == False:
            return True
```

! Don't write code like this !

Decision… as 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 tree consists of:
  - Nodes
    - Root
    - Leaves
  - Edges

Building a Decision Tree: Splits

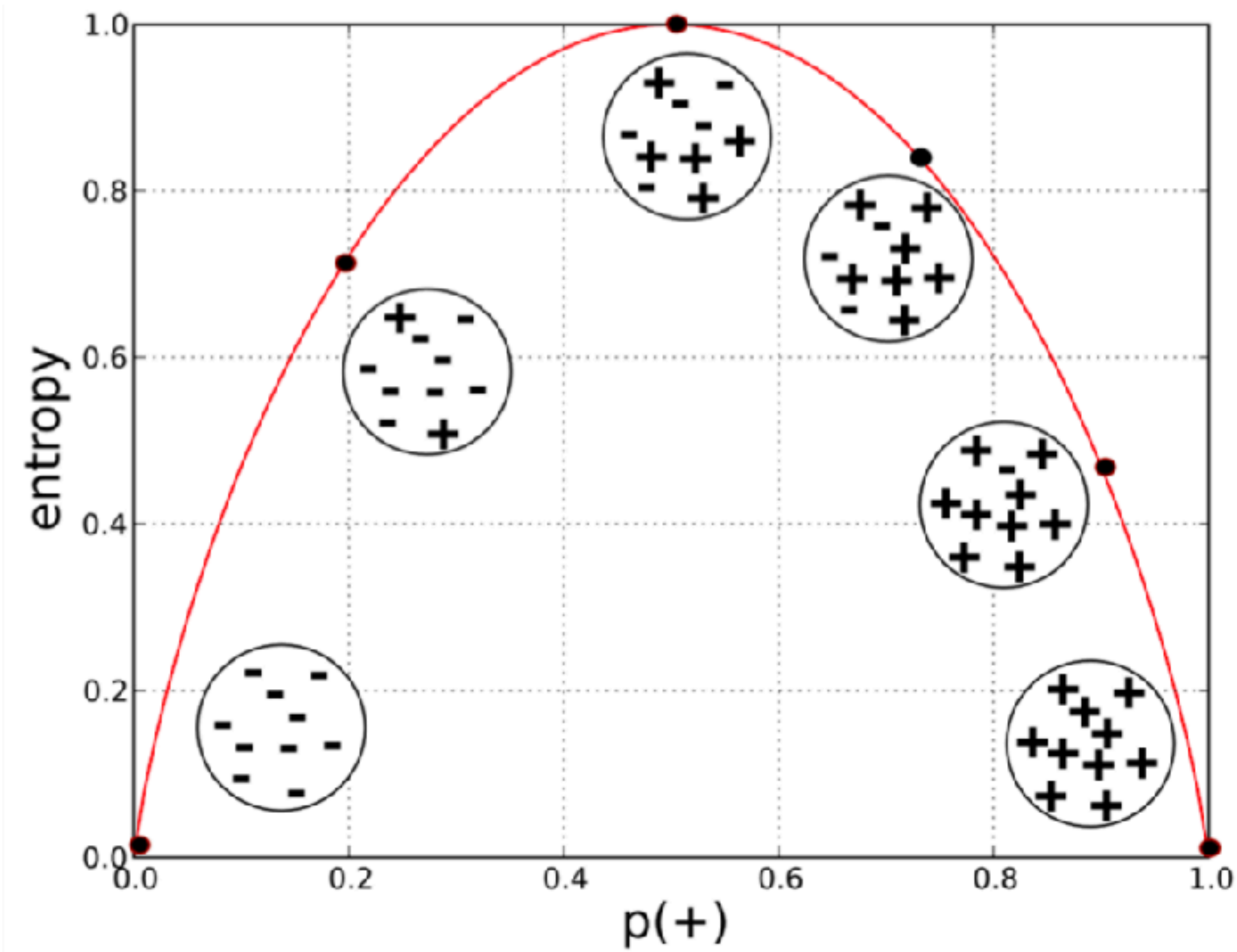
- We're gonna have to make some splits…
- Intuition:
  - Like "20 questions": ask the most useful questions first
  - What's 'useful'?
    - What splits the data well?
      - What's 'well'?

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

Information Gain

- Entropy
  - The entropy of a set is a measure of the amount of disorder
  - If a set has all the same labels, that's pretty ordered, so it has low entropy
  - If a set has a good mix of labels, that's not very ordered, so it has high entropy
- We want to create splits that minimize the entropy in each side of the split
  - If our splits do a good job splitting on the boundary between classes, the splits have more predictive power
- Compare:
  - Measure the entropy of the parent
  - Measure the entropy of the children for proposed split
    - What's the difference?
    - Maximize entropy(parent) - mean(entropy(children))

Entropy



Cross-entropy

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

- A measure of node ‘purity’
- If the class distributions are all close to 0 or 1, this takes on a small value
- If the class distributions are more mixed, this takes on a larger value
- Numerically similar to Gini index

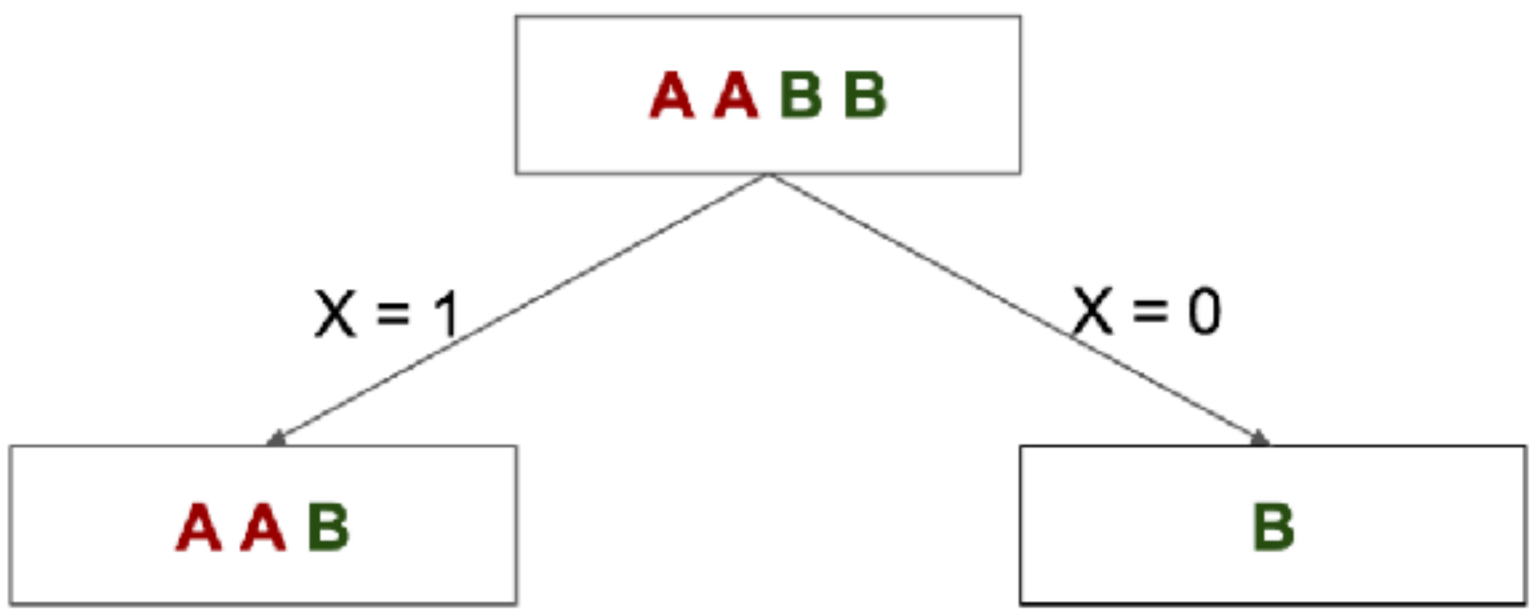
Gini index

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

- A measure of node ‘purity’
- If the class distributions are all close to 0 or 1, this takes on a small value
- If the class distributions are more mixed, this takes on a larger value
- Numerically similar to cross-entropy

Splitting

parent's entropy $= -(\frac{1}{2} \times \log_2 \frac{1}{2}) - (\frac{1}{2} \times \log_2 \frac{1}{2}) = 1$

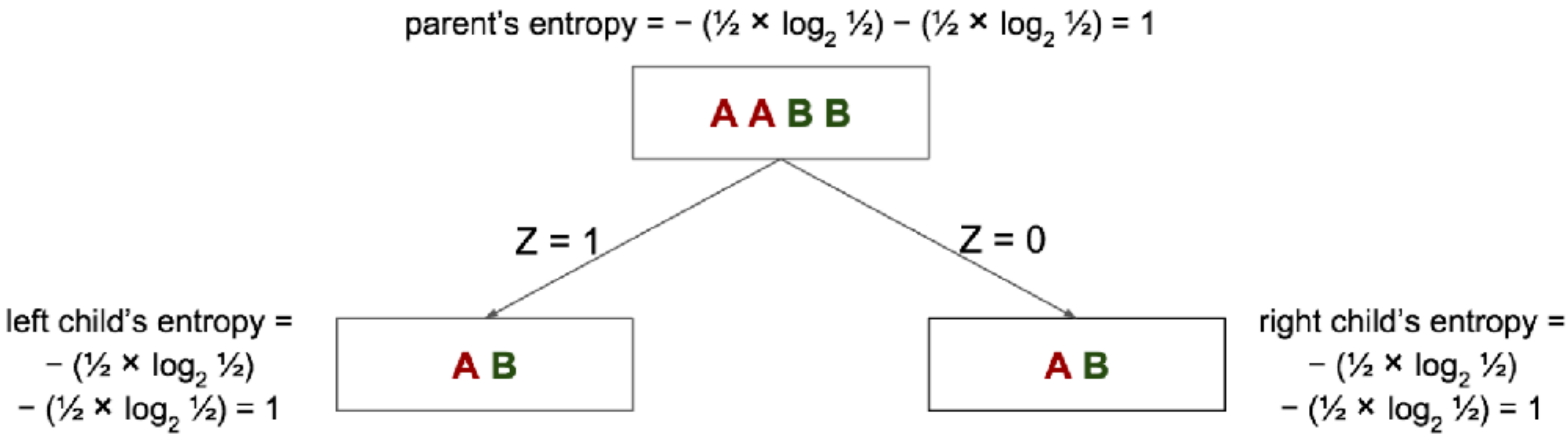


left child's entropy $= -(\frac{2}{3} \times \log_2 \frac{2}{3}) - (\frac{1}{3} \times \log_2 \frac{1}{3}) = .918$

right child's entropy $= -(1 \times \log_2 1) = 0$

information gain from splitting on X $= 1 - (\frac{3}{4} \times .918 + \frac{1}{4} \times 0) = .311$

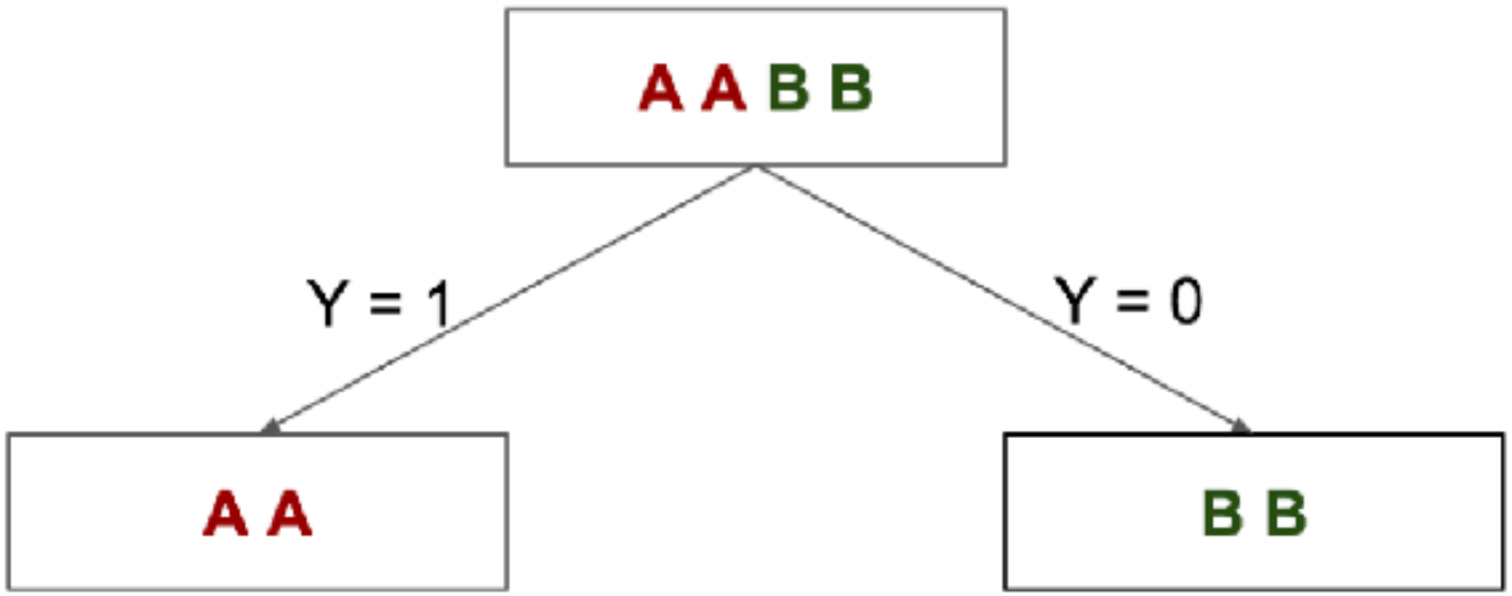
Splitting



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

Splitting

parent's entropy = $-(\frac{1}{2} \times \log_2 \frac{1}{2}) - (\frac{1}{2} \times \log_2 \frac{1}{2}) = 1$



left child's entropy = $-(1 \times \log_2 1) = 0$

right child's entropy = $-(1 \times \log_2 1) = 0$

information gain from splitting on X = $1 - (\frac{1}{2} \times 0 + \frac{1}{2} \times 0) = 1$, **BEST!**

Building a Tree: pseudocode (overall)

```
function BuildTree:
    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:
        find the best feature and value to split the data  # see next slide
        split the dataset
        create a node
        for each split
            call BuildTree and add the result as a child of the node
        return node
```

Building a Tree: pseudocode (finding splits)

splitting algorithm:

1. calculate the information gain for every possible split
2. select the split that has the highest information gain

possible splits:

for categorical features:

variable = value (or !=)

for continuous features:

variable <= threshold (or >)

Pruning

- Decision Trees are high variance
    - They are highly dependent on the training data
    - This makes them prone to overfitting
- We can ease up on the variance by *pruning*
- Prepruning
    - leaf size: stop when there's few data points at a node
    - depth: stop when a tree gets too deep
    - class mix: stop when some percent of data points are the same class
    - error reduction: stop when the information gains are too little
- Postpruning
    - Build a full tree, then cut off some leaves (like, y'know, actual real-life tree pruning)
        - 'cut off' means merge two leaves up to their parent, making that the new leaf

Pros and Cons

- Pros
  - Easily interpretable
  - Can model complex phenomenon/non-linear
    - no assumption that the structure of the model is fixed
    - feature interactions
  - Computationally cheap to predict
  - Can handle irrelevant features, missing values, and outliers well
  - Can handle mixed data (discrete, continuous, categorical)
  - Extensible (future lectures: bagging, random forests, boosting)
- Cons
  - Computationally expensive to train
  - Greedy algorithm (local optima)
  - Very high variance (super easy to overfit)

Practicalities

- Always prune
- Probably going to extend with {bagging, random forests, boosting}
- Gini or cross-entropy are both fine and not that different
- Regression Decision Trees
  - Responses are real values, so we can't use cross-entropy or Gini index
  - Instead, choose the best splits using residual sum of squares (against the mean value of each leaf)
- sklearn
  - doesn't support missing values
  - gini index is default, entropy is an option
  - pruning well supported (max_depth, min_samples_split, min_samples_leaf, max_leaf_nodes)
  - does binary splits (you need to binaries categorical variables)

Decision Trees: Review

- Describe a Decision Tree
  - How does one interpret the tree?
  - How does one create the tree?
- Explain Information Gain
- Explain strengths and weaknesses of DTs