

*November 16, 2017*

---

# CS 361: Probability & Statistics

Classification: decision trees &  
nearest neighbors

---

---

# Normalization

---

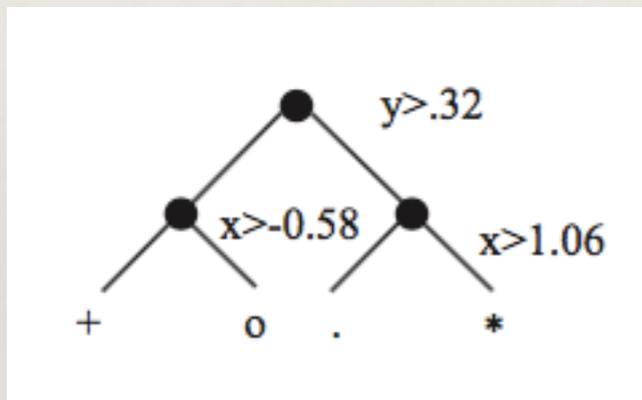
For the classifiers we have learned so far, there is one caveat worth commenting on

It is usually a good idea to normalize your data—make sure each dimension has 0 mean and variance of 1—before feeding it into an SVM or perceptron

Due to the loss function, a feature with a much larger variance than the others will have a larger effect on the resulting classifier than the other features

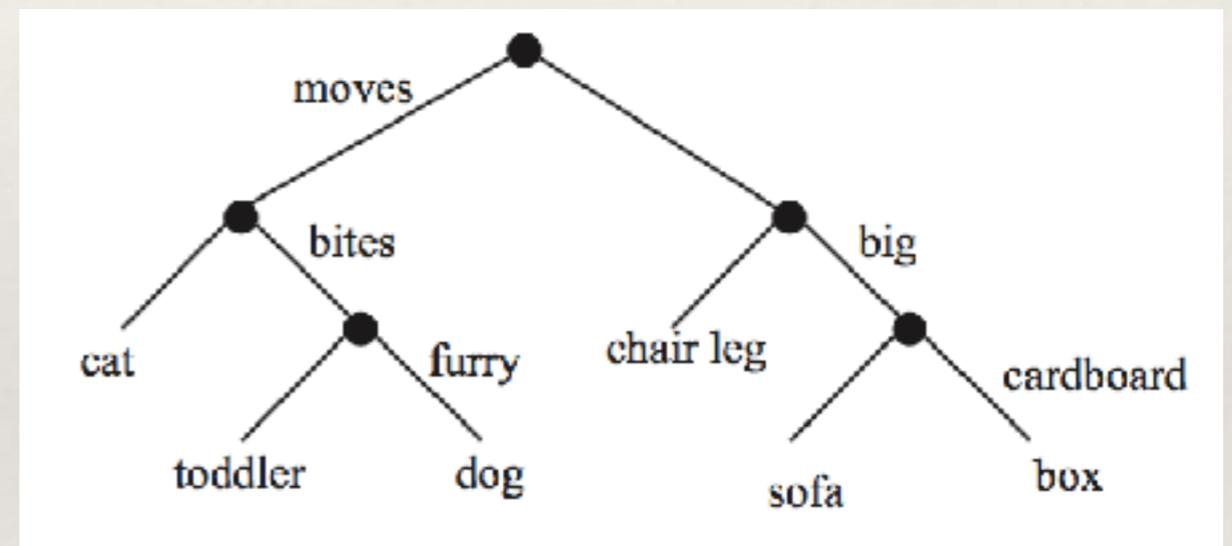
# Decision trees

# Examples



(0, 0)

(2, 1)



---

# Decision trees

---

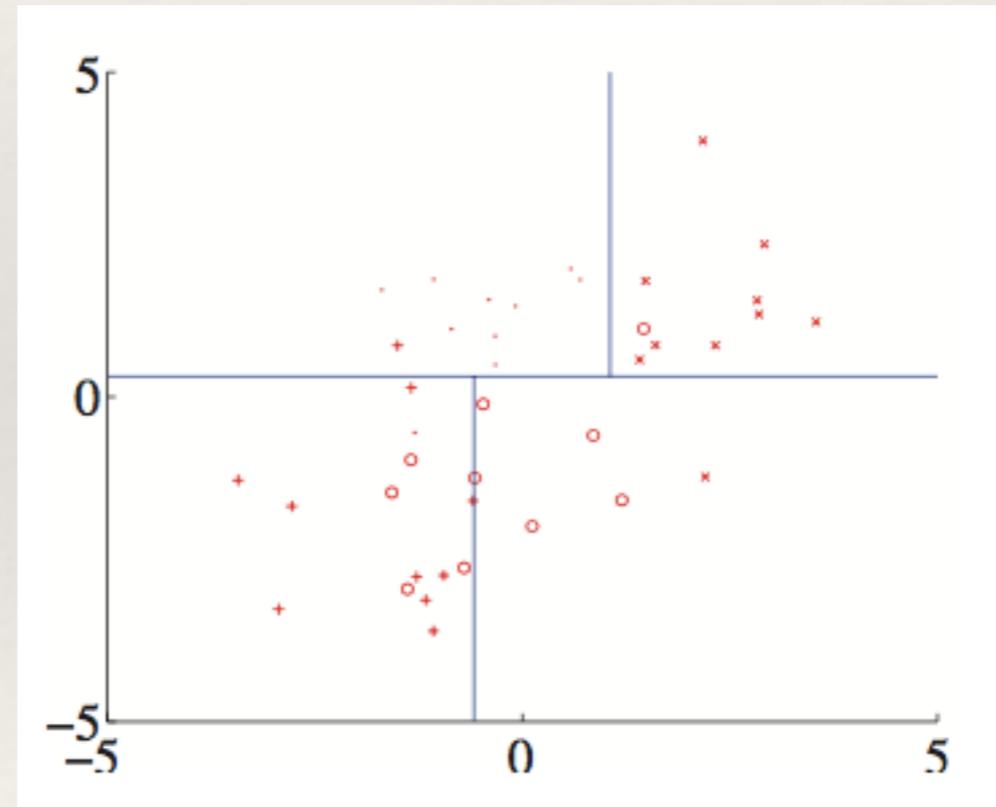
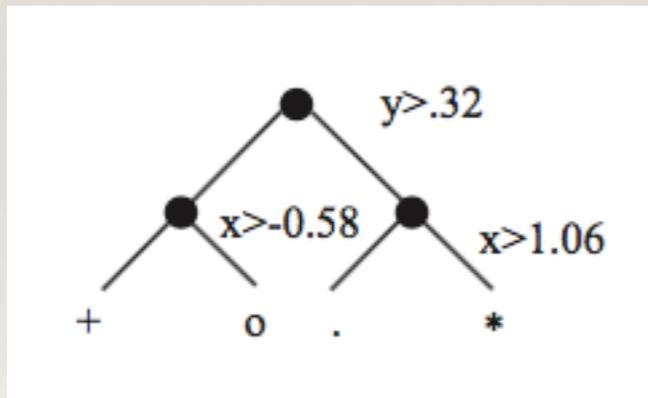
A decision tree is a type of classifier that operates as follows

Each internal node corresponds to a single dimension of the data and a binary **split** for that dimension, i.e. a set of values for the dimension which points to the left child and the remainder of values which point to the right child

To classify a new data item, you start at the root node. Being an internal (non-leaf) node, the root will refer to one of the dimensions of the data item. Depending on the split, the data item will next visit the left or right child

Continue to traverse the tree according to dimensions and splits until reaching a leaf. Each leaf in the tree gives a class label

# The decision surface



---

# How to make a decision tree from training data

---

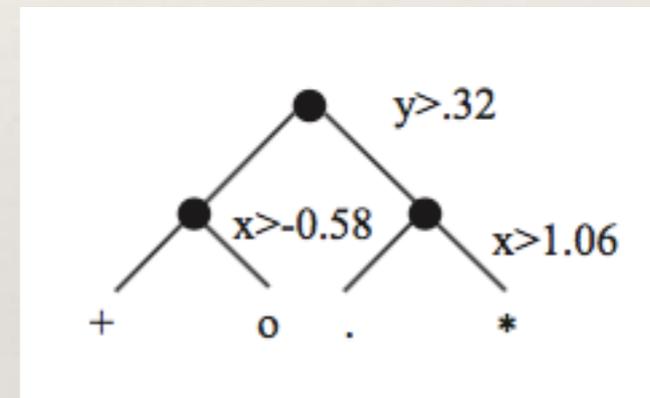
We will recursively build our tree using the training data. We will need some principles in order to

Decide which dimension to split on

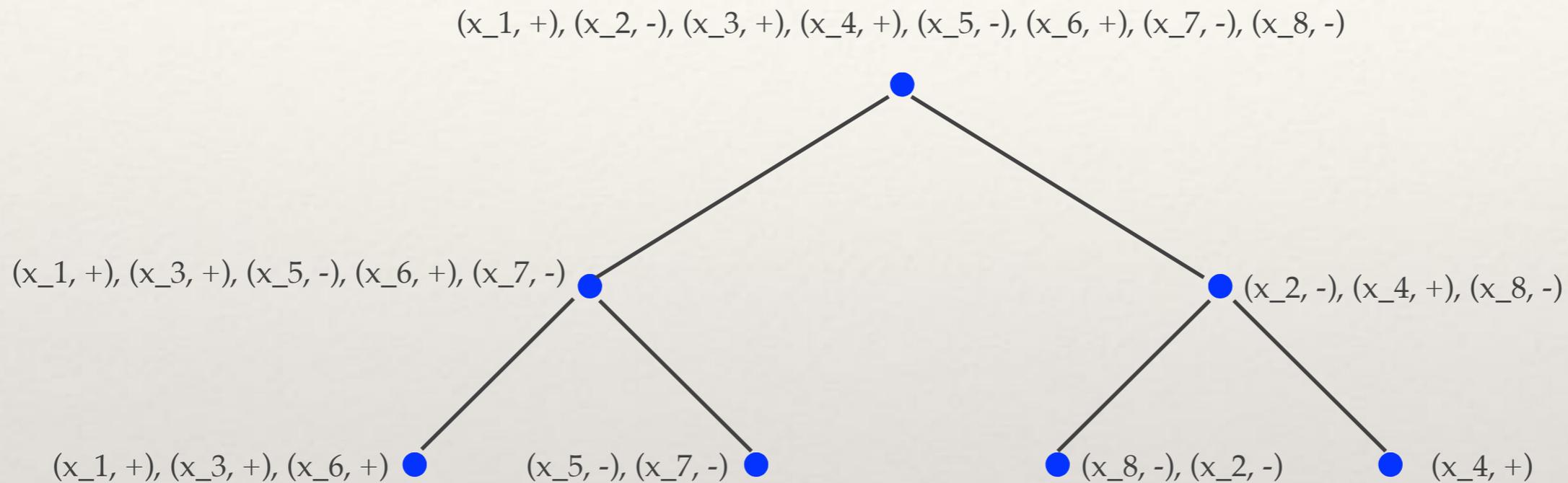
Decide how to split the dimension

Decide how deep to make the tree or equivalently when to end recursion

Decide which labels to put on the leaves



# Recursively splitting the data



---

# What kind of splits

---

We want to split the data in some informative way

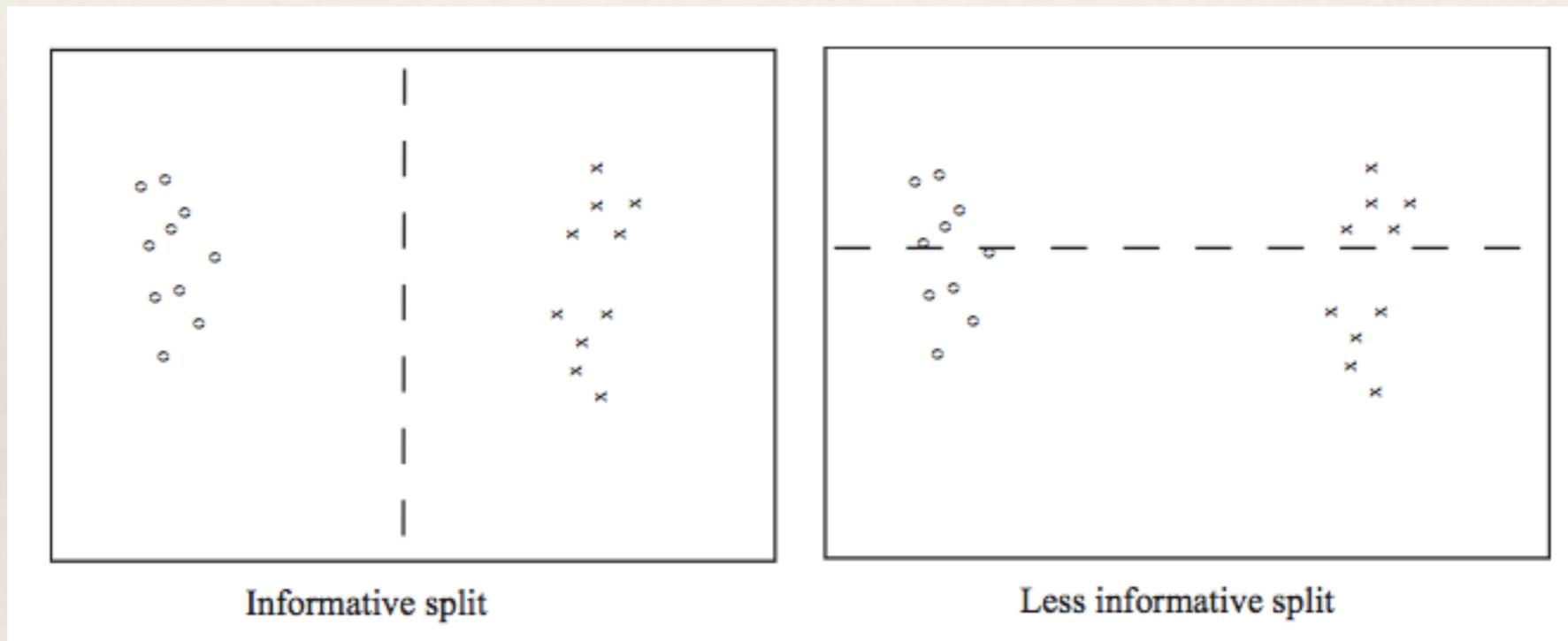
Suppose this was our entire training data set

X1	X2	Class
0	0	-1
0	8.5	-1
0.5	3.25	-1
1.2	-1.5	-1
2.5	5	-1
3.75	1	1
4	1.5	1
4.5	4.5	1
4	3.25	1
5.25	5.5	1

Why is  $X1 > 3$  a great split? Why is  $X2 > 3$  not as good?

# What kind of split

In general we would like it if whatever the “class balance” at the node is (say it starts out 50-50 class 1 and class -1) is more imbalanced in the children



---

# Entropy

---

It turns out that a good way to measure this “class imbalance” feature of a dataset comes to us from communication theory or information theory

Suppose we had four classes: A, B, C, and D

One way to encode those classes, for communication over a network for example, would be A=00, B=01, C=10, D=11. If we did this, it would take on average, indeed in every case, 2 bits to communicate the class label

The idea with entropy is that if we have some class imbalance, say  $P(A) = 1/2$ ,  $P(B) = 1/4$ ,  $P(C) = 1/8$ ,  $P(D) = 1/8$ , maybe there is a better encoding we can use

---

# Entropy

---

If we had classes with frequencies given by  $P(A) = 1/2$ ,  $P(B) = 1/4$ ,  $P(C) = 1/8$ ,  $P(D) = 1/8$

Then if we used the following encoding scheme

A	0
B	10
C	110
D	111

The expected number of bits to communicate a class label is given by

$$(1/2)(1) + (1/4)(2) + (1/8)(3) + (1/8)(3)$$

Which is 1.75 bits

---

# Entropy

---

The **entropy** of a probability distribution is the smallest number of bits on average you would need to identify an item sampled from a distribution. It can be thought of as measuring the degree of uncertainty of the distribution. We can use this to evaluate the splits we are considering for our data

This quantity is large when the probability distribution is close to uniform and small when one class is very likely while the others are unlikely. The entropy is denoted with an  $H$  and is given by the following formula

$$-\sum_i p(i) \log_2 p(i)$$

Where the  $p(i)$  are the probability of class  $i$  which in this case is just the relative frequency of the class

---

# Example

---

Calculate the entropy for this dataset

X1	X2	Class
0	0	-1
0	8.5	-1
0.5	3.25	-1
1.2	-1.5	-1
2.5	5	-1
3.75	1	1
4	1.5	1
4.5	4.5	1
4	3.25	1
5.25	5.5	1

$$-\sum_i p(i) \log_2 p(i)$$

We get

$$-5/10 \log(5/10) - 5/10 \log(5/10) = 1$$

---

# Split evaluation

---

Let  $H(\mathcal{P})$  be the entropy of all the training data. In order to evaluate a potential split at the root node of the tree, we want to compute the difference between this entropy and the average entropy remaining in the left and right children nodes, which we will call  $H(\mathcal{P}_l)$  and  $H(\mathcal{P}_r)$  respectively.

The average entropy after the split between the left and right subtrees will be

$$\frac{N(\mathcal{P}_l)}{N(\mathcal{P})}H(\mathcal{P}_l) + \frac{N(\mathcal{P}_r)}{N(\mathcal{P})}H(\mathcal{P}_r)$$

Where  $N(\mathcal{P}_l)$  is, for instance, the number of items that wind up in the left subtree

---

# Information gain

---

This difference between the entropy or average number of bits needed before and after a split is called the **information gain** of the split. We would like to choose the split with the largest information gain

$$I(\mathcal{P}_l, \mathcal{P}_r; \mathcal{P}) = H(\mathcal{P}) - \left( \frac{N(\mathcal{P}_l)}{N(\mathcal{P})} H(\mathcal{P}_l) + \frac{N(\mathcal{P}_r)}{N(\mathcal{P})} H(\mathcal{P}_r) \right)$$

---

# Example

---

For the training set below, evaluate the information gain of the split  $x_2 > 1$

X1	X2	Class
0	0	-1
0	8.5	-1
0.5	3.25	-1
1.2	-1.5	-1
2.5	5	-1
3.75	1	1
4	1.5	1
4.5	4.5	1
4	3.25	1
5.25	5.5	1

From our prior example we know that  $H(P) = 1$

Let's say that data items {2, 3, 5, 7, 8, 9, 10} go to the right subtree and items {1, 4, 6} go to the left

---

# Example

---

Let's say that data items {2, 3, 5, 7, 8, 9, 10} go to the right and items {1, 4, 6} go to the left

X1	X2	Class
0	0	-1
0	8.5	-1
0.5	3.25	-1
1.2	-1.5	-1
2.5	5	-1
3.75	1	1
4	1.5	1
4.5	4.5	1
4	3.25	1
5.25	5.5	1

$$-\sum_i p(i) \log_2 p(i)$$

What is  $H(P_r)$ ?

$$-3/7 \log(3/7) - 4/7 \log(4/7) = 0.985$$

---

# Example

---

Let's say that data items {2, 3, 5, 7, 8, 9, 10} go to the right and items {1, 4, 6} go to the left

X1	X2	Class
0	0	-1
0	8.5	-1
0.5	3.25	-1
1.2	-1.5	-1
2.5	5	-1
3.75	1	1
4	1.5	1
4.5	4.5	1
4	3.25	1
5.25	5.5	1

$$-\sum_i p(i) \log_2 p(i)$$

What is  $H(P_1)$ ?

$$-2/3 \log(2/3) - 1/3 \log(1/3) = 0.918$$

---

# Example

---

$$I(\mathcal{P}_l, \mathcal{P}_r; \mathcal{P}) = H(\mathcal{P}) - \left( \frac{N(\mathcal{P}_l)}{N(\mathcal{P})} H(\mathcal{P}_l) + \frac{N(\mathcal{P}_r)}{N(\mathcal{P})} H(\mathcal{P}_r) \right)$$

So we had  $H(\mathcal{P}) = 1$ ,  $H(\mathcal{P}_l) = 0.918$ ,  $H(\mathcal{P}_r) = 0.985$ ,  $N(\mathcal{P}_l) = 3$ ,  $N(\mathcal{P}_r) = 7$ ,  $N(\mathcal{P}) = 10$

Giving an information gain of 0.0351

---

# Choosing a split in practice

---

In practice the way we will choose which feature to split on will be by looking at a random subset of the features. For each node in the tree, we will choose  $m$  of the dimensions of the data at random and then evaluate each as a candidate for the split for that node

We will choose  $m = \text{square root of } d$

Then for each of our  $m$  features we will compute the best split for that feature. And we will choose to split on the feature that gives the greatest information gain

---

# Evaluating an ordinal feature

---

If the feature we are trying to evaluate is continuous or otherwise ordinal we can sort the data according to that feature and N-1 potential splits by evaluating the information gain of splitting on the halfway point between each consecutive pair of points, i.e.

$$\frac{x_i + x_{i+1}}{2}$$

For example if our data in a given dimension  $x^{(j)}$  was {1, 3, 6, 8} we would evaluate the information gain for  $x^{(j)} > 2$ ,  $x^{(j)} > 4.5$ , and  $x^{(j)} > 7$

---

# Evaluating a categorical feature

---

If we have a categorical feature, one trick we can do is look at each potential value for the feature and just flip a coin to say whether data with that value goes left or right. In order to give the feature a good evaluation, we will want to repeat this some number of times and keep the split that gives the best information gain

For example if we had some dimension that was categorical with possible values {rain, sunny, windy, cloudy, snow}. We would flip coins for each value and maybe say that data items with {rain, snow} go left and items with {sunny, windy, cloudy} go right

We would then evaluate the information gain if we did that split and repeat the process a few times in order to find a good split for the categorical variable in question

---

# Tree depth

---

So when do we stop splitting?

If all of the data in a node has the same class label, there is no need to continue splitting. So we could adopt the strategy of continuing to split until each item in the training set is unambiguously classified by the tree.

Alternatively, we can specify a maximum depth for the tree ahead of time. We can use a validation set in order to test out various depths to see which one seems to work best

---

# Prediction

---

If we do happen to stop creating nodes after a depth threshold we will have a situation in which the leaves of the tree may not be unanimous

If this is the case we will do the obvious thing of outputting the class label that has a majority in the leaf, flipping a coin in the case of ties

---

# Forests

---

The approach to constructing a tree we have just outlined won't find the "best possible" tree for the data. At each node we have only looked at a random subset of the features. If we looked at every feature we would very often get a decision tree that predicts the training data with higher accuracy

It turns out that decision trees can fit training data really well (even data that's not linearly separable). So well in fact that overfitting can become a problem and generalization to unseen test data can suffer

It also turns out that a very effective strategy in both theory and practice is to train a number of slightly different decent classifiers and combine them together to make one really good classifier

---

# Forests

---

A decision forest is when we train many distinct decision trees on the same training dataset, using randomization to get different trees

To make a prediction with a decision forest, we present the data item to all of the trees and predict the class label that gets the most votes

An alternative and even more effective way to predict is to give each tree not just one vote. In particular, if a tree reaches a leaf with  $p$  training data instances that vote for its majority class, then that tree will cast  $p$  votes for that class instead of just one

# Nearest neighbors

---

# Closeness and labels

---

We've made the argument a few times so far that if two points are close to one another, we expect them to have the same label much of the time

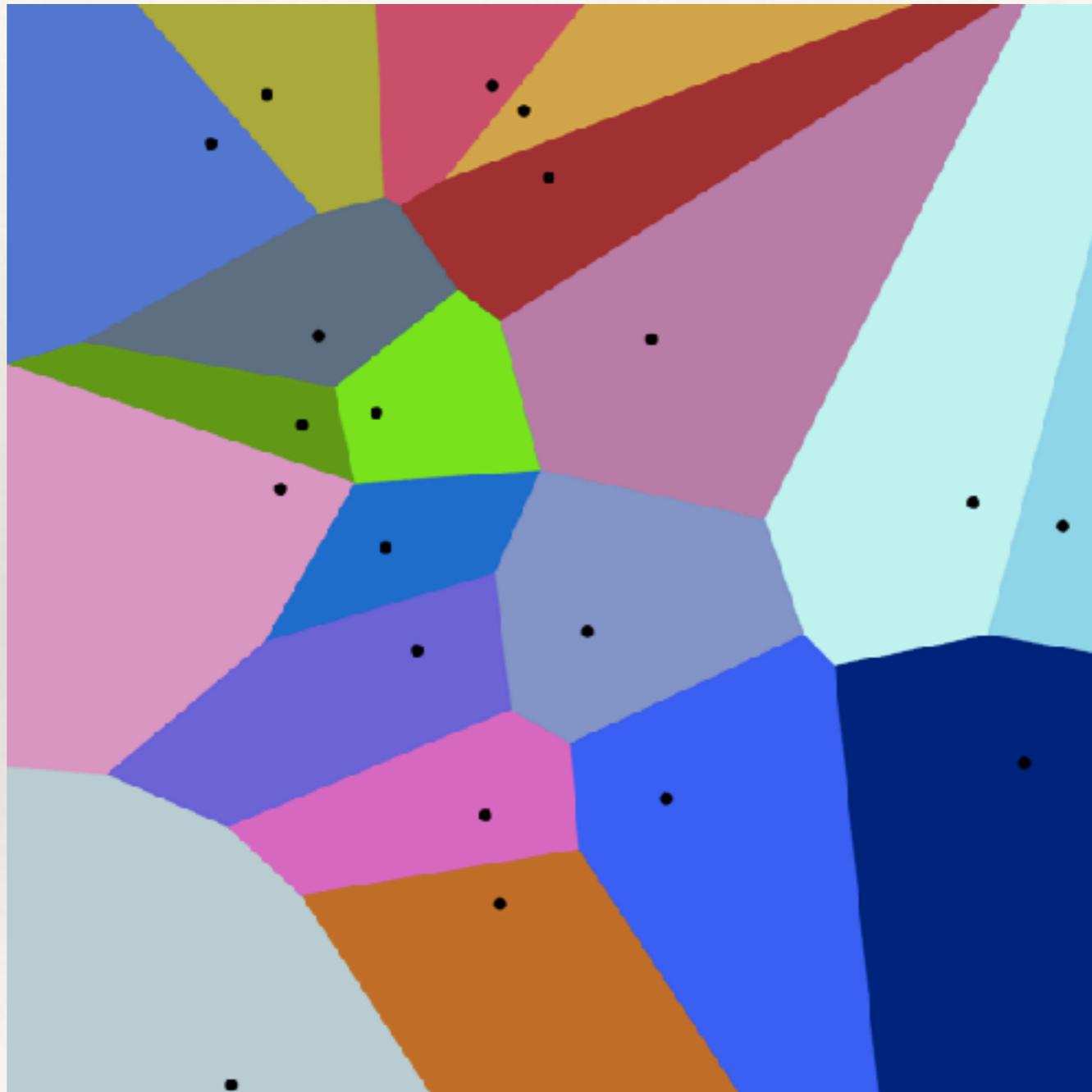
Of course, we could have two points that are close and on opposite sides of a decision boundary, but we expect that for most pairs of points that are near one another, their labels will be the same

Our prediction function for this classifier will find the closest point in the training set to the query point and predict the label associated with that item of training data

---

# The decision surface

---



---

# k-nearest neighbors, etc.

---

Rather than just use one neighbor for the classification, we might use  $k$ . This is the most common implementation of nearest neighbors and is called k-nearest neighbors

Furthermore, we might only output a label if at least  $l$  of these  $k$  can agree on a common label. This is maybe less common and is called a  $(k,l)$ -nearest neighbor classifier

---

# Training and other practical issues

---

“Training” then for this classifier is just any pre-processing we do that makes looking up near neighbors easier

Of course, one of the things we are going to need to make this work at all is a notion of “distance” between points. There are many potential choices we could use depending on the nature of our data and our assumptions about it

If our dimensions have wildly different scales, this might be problematic. We may wish to standardize our data in each dimension prior to inserting it into the search data structure