Classification

Chris Amato
Northeastern University

Some images and slides are used from: Rob Platt, CS188 UC Berkeley, AIMA, Kevin Murphy

Supervised learning

Given: *Training set* $\{(x_i, y_i) \mid i = 1 \ldots N\}$, given a labeled set of input-output pairs $D = \{(x_i, y_i)\}_i$

Find: A good approximation to $f : X \rightarrow Y$ Function approximation

Examples: what are X and Y ?

Spam Detection – Map email to {Spam, Not Spam} Binary Classification

Digit recognition – Map pixels to {0,1,2,3,4,5,6,7,8,9} Multiclass Classification

Stock Prediction – Map new, historic prices, etc. to (the real numbers) Regression

Supervised learning

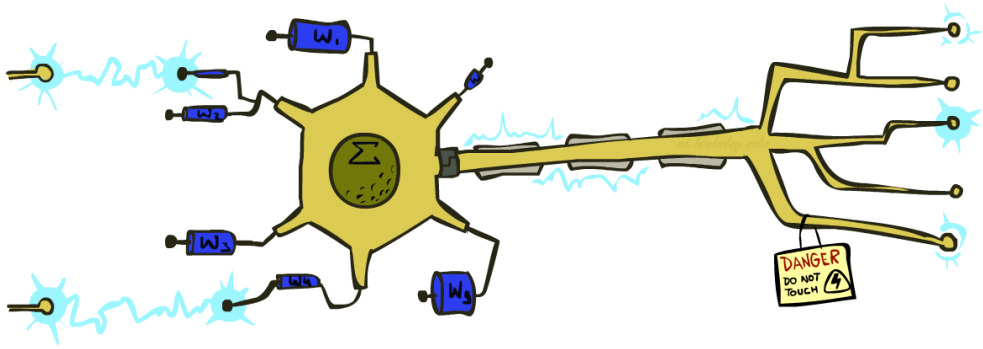
Goal: make predictions on novel inputs, meaning ones that we have not seen before (this is called *generalization*)

Formalize this problem as approximating a function: $f(x)=y$

The leaning problem is then to use *function approximation* to discover: $\hat{f}(x) = \hat{y}$

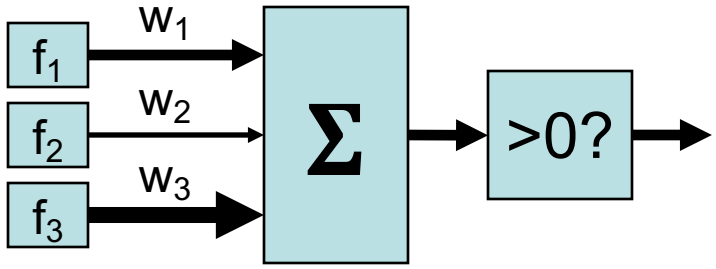
Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output +1
  - Negative, output -1

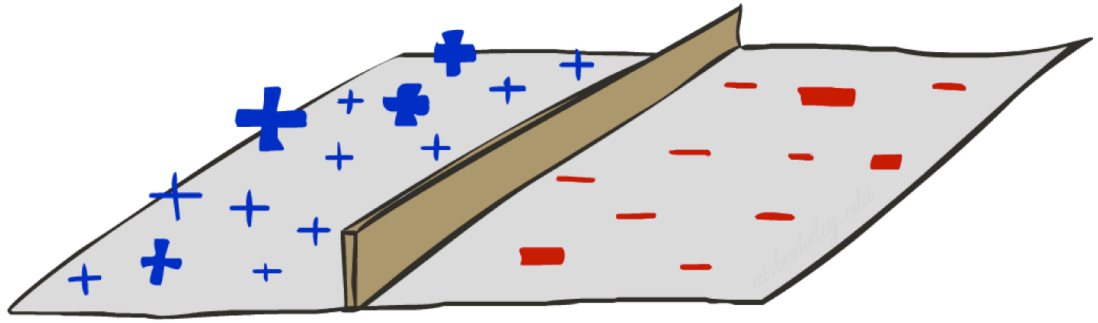


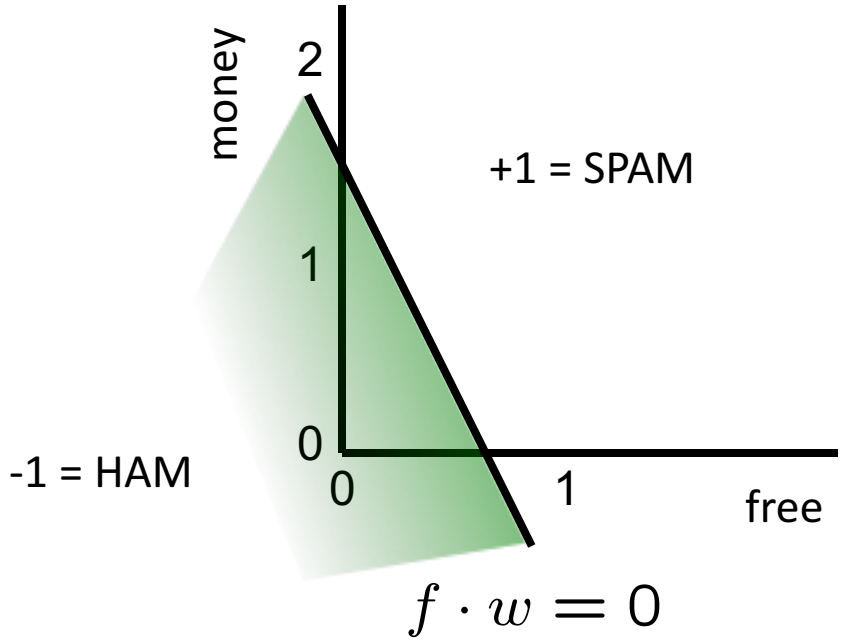
Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - Decision boundary:
    - One side corresponds to Y=+1
    - Other corresponds to Y=-1

w

```
BIAS   : -3
free   :  4
money  :  2
...
```





$f \cdot w = 0$

Learning: Multiclass Perceptron

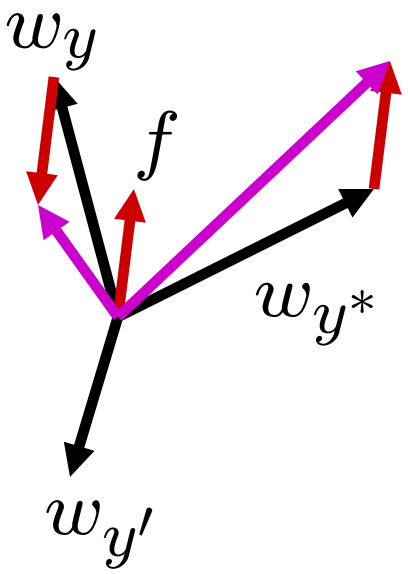
- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

$$y = \arg\max_y \; w_y \cdot f(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

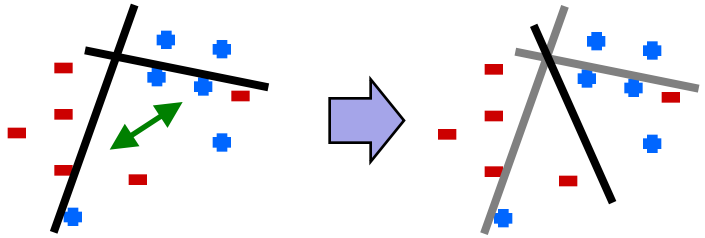
$$w_y = w_y - f(x)$$

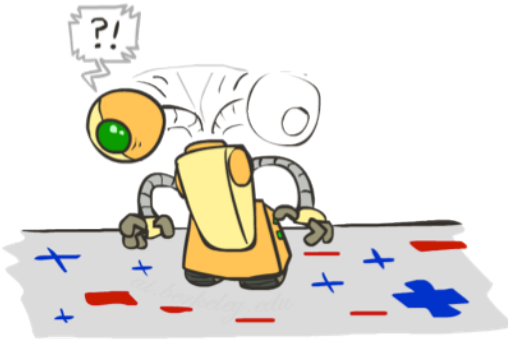
$$w_{y^*} = w_{y^*} + f(x)$$



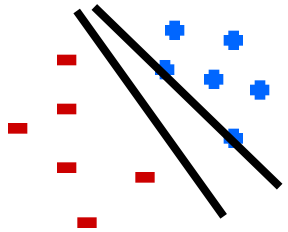
Problems with the Perceptron

- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)

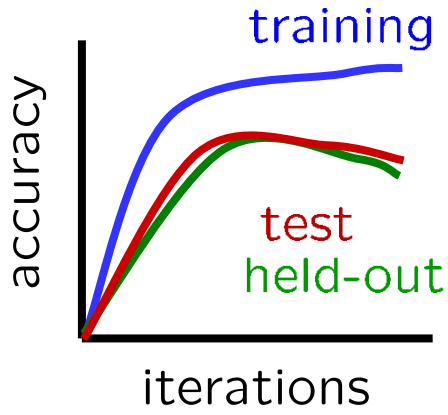


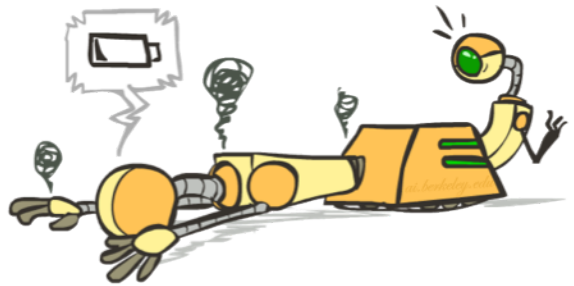


- Mediocre generalization: finds a "barely" separating solution



- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting





Fixing the Perceptron

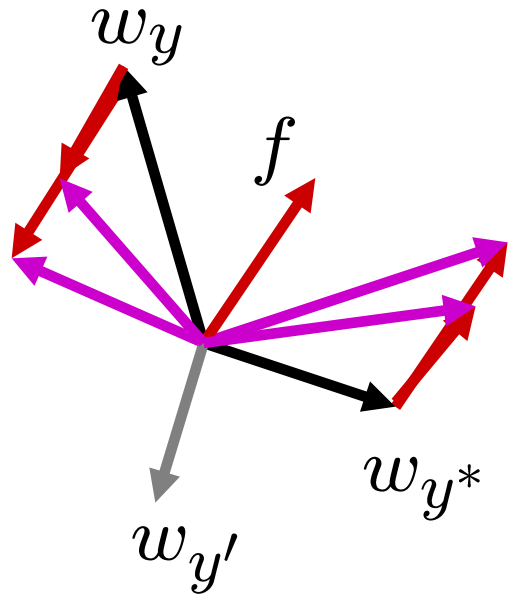
- Idea: adjust the weight update to mitigate these effects

- MIRA*: choose an update size that fixes the current mistake…
- … but, minimizes the change to w

$$\min_w \frac{1}{2} \sum_y ||w_y - w'_y||^2$$

$$w_{y^*} \cdot f(x) \geq w_y \cdot f(x) + 1$$

- The +1 helps to generalize



Guessed y instead of y^* on example x with features $f(x)$

$$w_y = w'_y - \tau f(x)$$
$$w_{y^*} = w'_{y^*} + \tau f(x)$$

* Margin Infused Relaxed Algorithm

Minimum Correcting Update

$$\min_w \frac{1}{2}\sum_y ||w_y - w'_y||^2$$

$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$

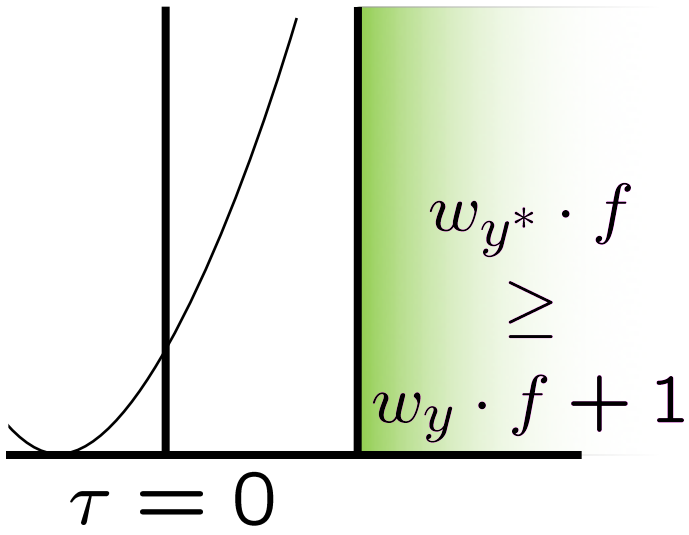
$$\min_\tau ||\tau f||^2$$

$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$

$$(w'_{y^*} + \tau f) \cdot f = (w'_y - \tau f) \cdot f + 1$$

$$\tau = \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}$$

$$w_y = w'_y - \tau f(x)$$
$$w_{y^*} = w'_{y^*} + \tau f(x)$$



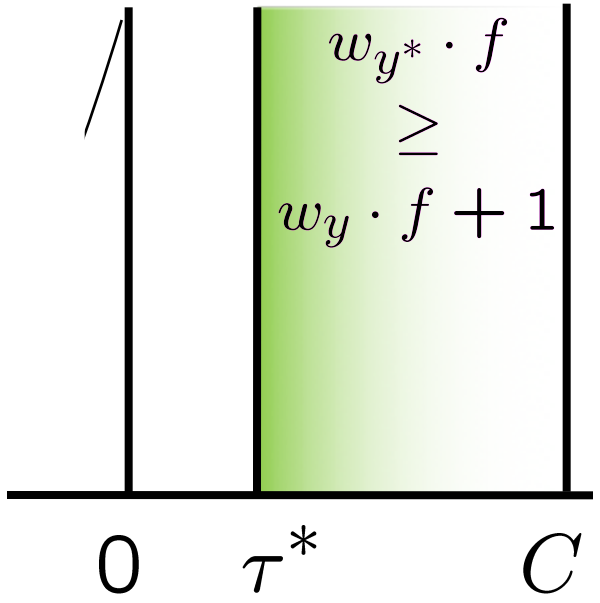
min not τ=0, or would not have made an error, so min will be where equality holds

Maximum Step Size

- In practice, it's also bad to make updates that are too large
  - Example may be labeled incorrectly
  - You may not have enough features
  - Solution: cap the maximum possible value of τ with some constant C

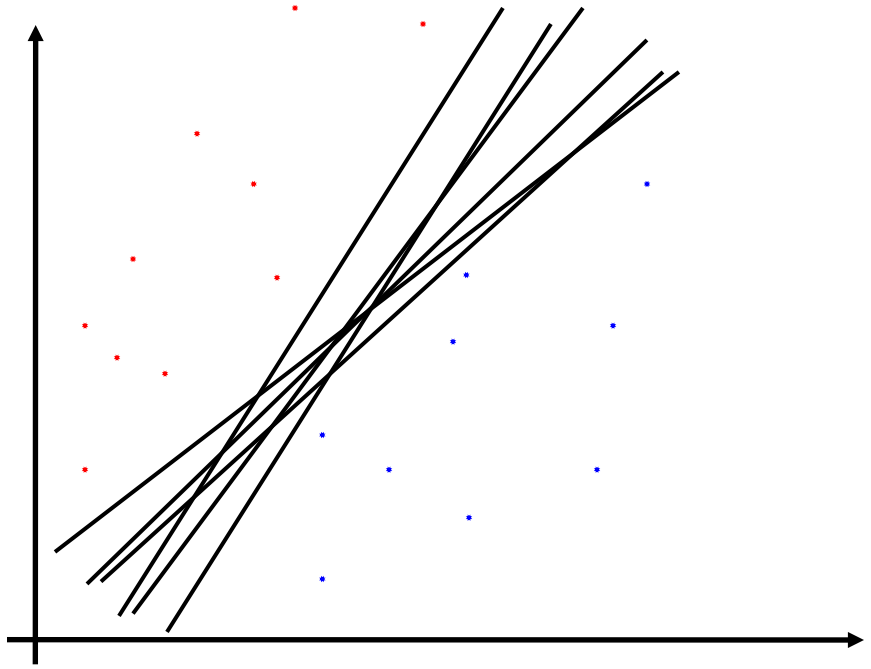
$$\tau^* = \min\left(\frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}, C\right)$$

  - Corresponds to an optimization that assumes non-separable data
  - Usually converges faster than perceptron
  - Usually better, especially on noisy data



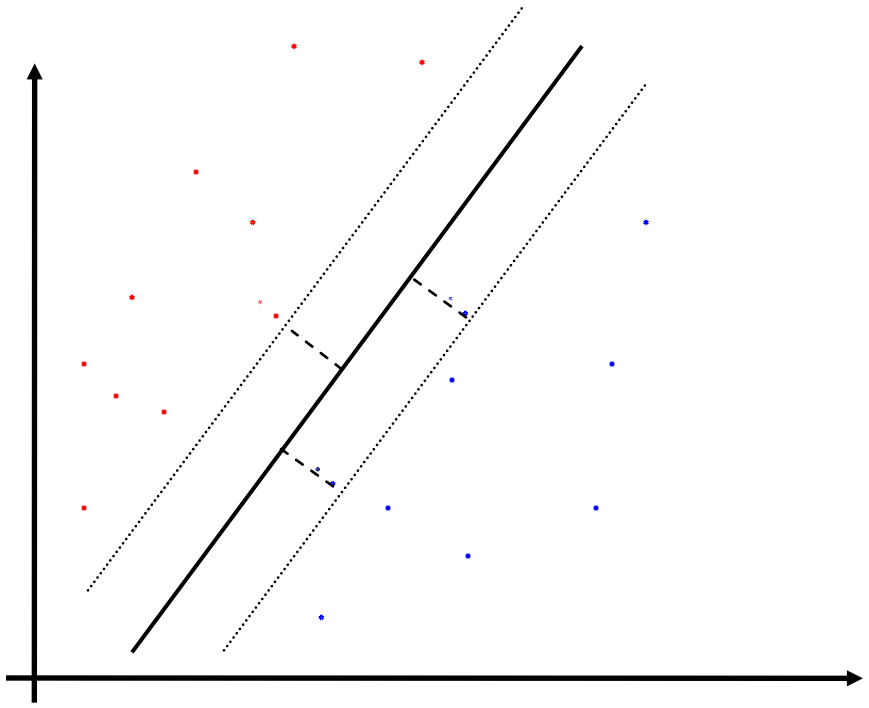
Linear Separators

- Which of these linear separators is optimal?



Support Vector Machines

- Maximizing the margin: good according to intuition, theory, practice
- Only support vectors matter; other training examples are ignorable
- Support vector machines (SVMs) find the separator with max margin
- Basically, SVMs are MIRA where you optimize over all examples at once



MIRA

$$\min_{w} \frac{1}{2}||w - w'||^2$$

$$w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

SVM

$$\min_{w} \frac{1}{2}||w||^2$$

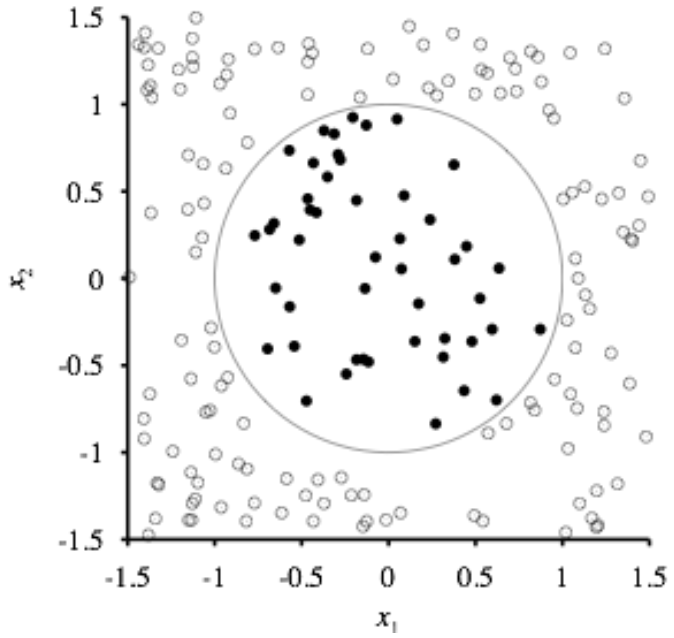
$$\forall i, y \; w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

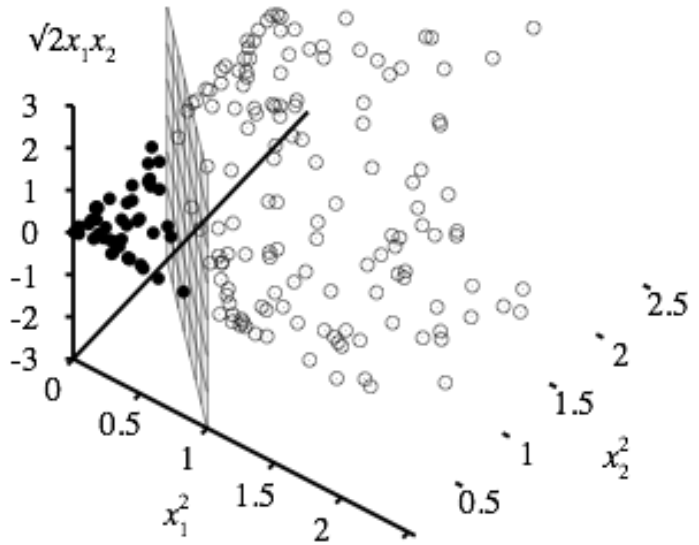
Kernels

What if the data is not linearly separable?

Consider features $F(\mathrm{x})$: $f_1 = x_1^2 \quad f_2 = x_2^2 \quad f_3 = \sqrt{2}x_1x_2$

If data is mapped to a sufficiently high-dimensional space, it is likely to become linearly separable





Kernels

Using kernels replaces x with $\mathrm{F}(\mathrm{x})$ (like typical feature expansion)

But for SVMs, this means replacing $\mathrm{x}_j \cdot \mathrm{x}_k$ with $F(\mathrm{x}_j) \cdot F(\mathrm{x}_k)$

Using *kernels functions* lets us not calculate $F(\mathrm{x}_j)$ and replace the dot product with $K(\mathrm{x}_j,\mathrm{x}_k)$

In our example, $F(\mathrm{x}_j) \cdot F(\mathrm{x}_k)=K(\mathrm{x}_j,\mathrm{x}_k)=(\mathrm{x}_j \cdot \mathrm{x}_k)^2$

Note the dot product has the original dimensionality

The separator is linear in the high-dimensional space, but non-linear in the low-dimensional space

Classification: Comparison so far

- Naïve Bayes
  - Builds a model training data
  - Gives prediction probabilities
  - Strong assumptions about feature independence
  - One pass through data (counting)

- Perceptrons / MIRA:
  - Makes less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data (prediction)
  - Often more accurate

Decision Trees

A *decision tree* represents a function that takes input as a vector of attributes and returns a single output value (which could be discrete or continuous)

General idea: make a decision by using a sequence of tests

Each internal node in the tree is a test on some input variables (e.g, features or attributes)

Each leaf is a return value

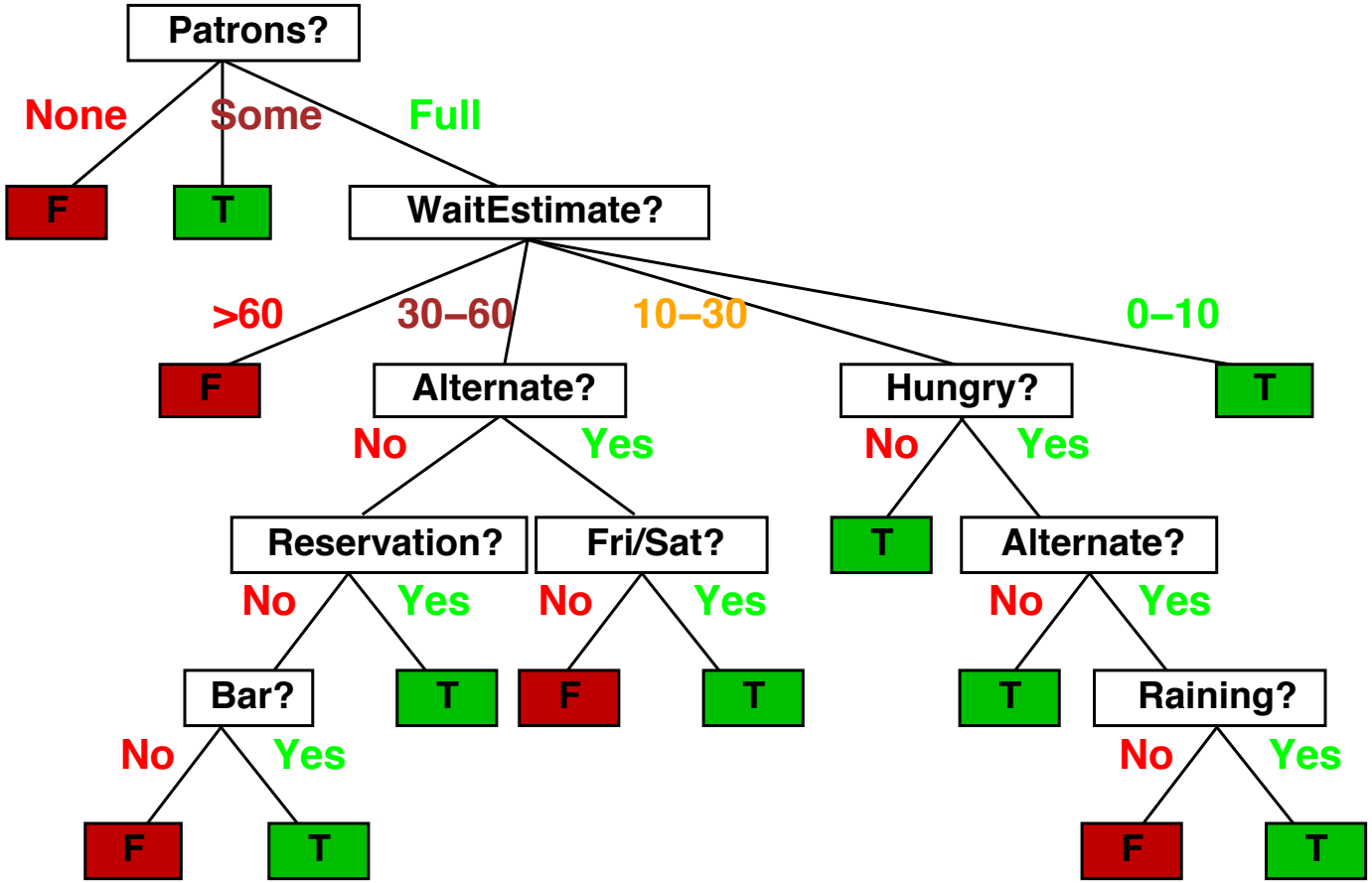
Decision tree example

Deciding to wait for a table at a restaurant (binary classification task)

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

Decision tree example

One solution (Stuart Russell's) with Wait=T

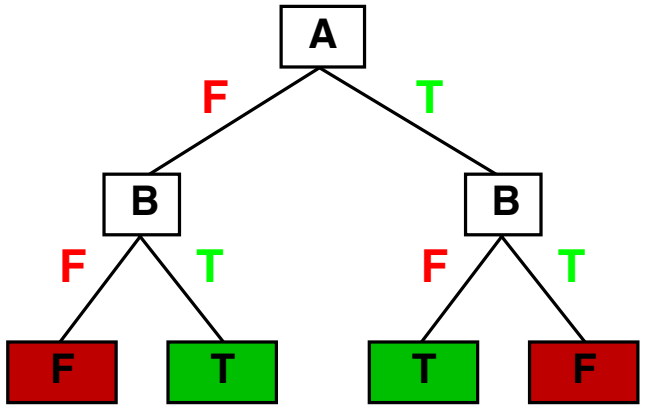


Decision tree expressiveness

Decision trees can express any function of the input attributes

E.g., for Boolean functions, truth table row → path to leaf (ands along path, ors over paths):

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Trivially, there is a consistent decision tree for any training set w/ one path to leaf for each example (unless f nondeterministic in x) but it won't generalize well

Prefer to find more *compact* decision trees

Hypothesis spaces

How many distinct decision trees with *n* Boolean attributes?

Hypothesis spaces

How many distinct decision trees with n Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows

Hypothesis spaces

How many distinct decision trees with n Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows = 2^{2^n}

E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

Hypothesis spaces

More expressive hypothesis space

– increases chance that target function can be expressed (yay!)

– increases number of hypotheses consistent w/ training set (boo!)

$\Rightarrow$ may get worse predictions

Can’t solve this problem optimally!

Decision tree learning

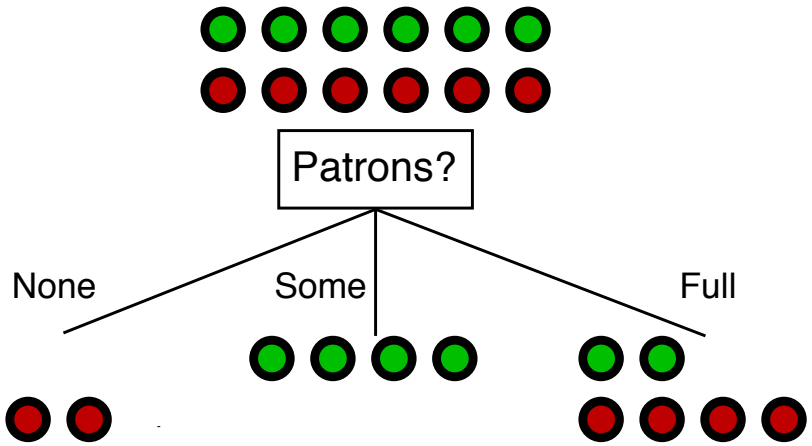
Aim: find a small tree consistent with the training examples

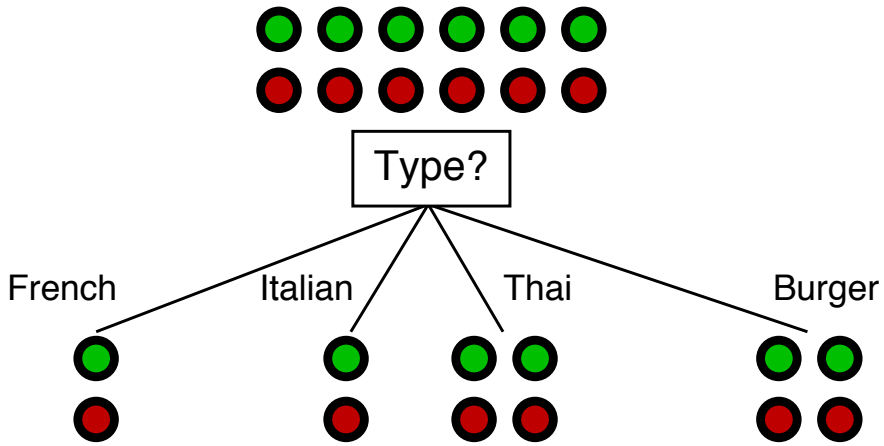
Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

Choosing an Attribute

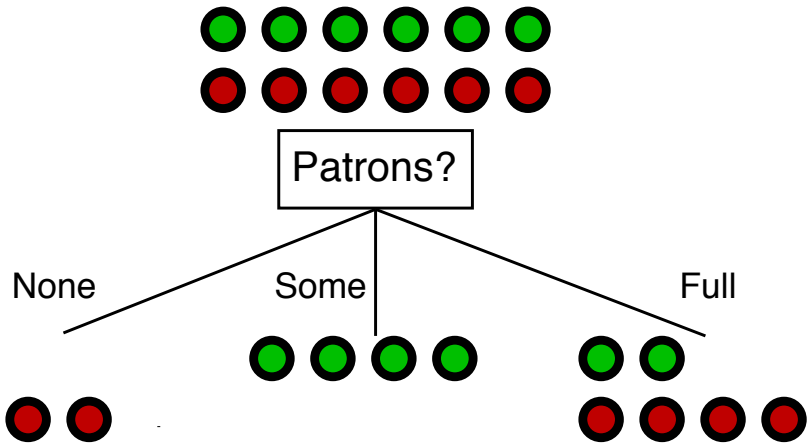
Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”

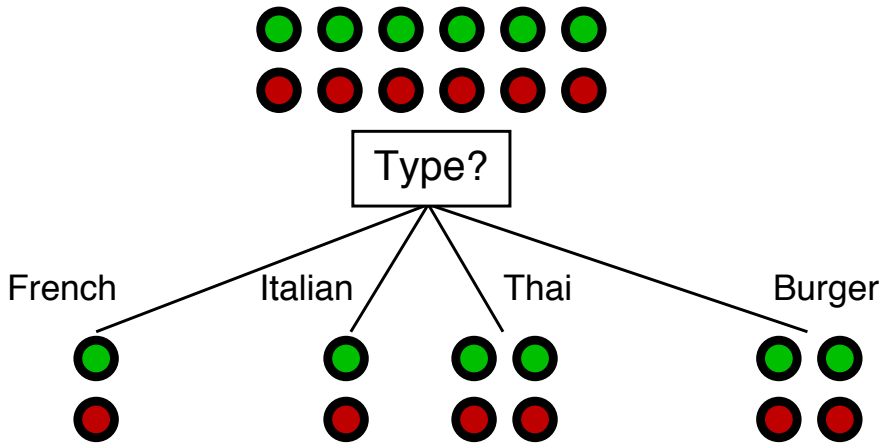




Choosing an Attribute

Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”





Patrons? is a better choice—gives *information* about the classification

Information

Use information theory (in particular, *entropy* or *information gain*) to choose attributes

The more clueless I am about the answer initially, the more information is contained in the answer

Scale: 1 bit = answer to Boolean question with prior ⟨0.5, 0.5⟩

Information in an answer when prior is $\langle P_1, \ldots, P_n \rangle$ is

$$H(\langle P_1, \ldots, P_n \rangle) = -\sum_i P_i \log_2 P_i$$

Information

An attribute splits the examples E into subsets E_i, each of which (we hope) needs less information to complete the classification

Let E_i have p_i positive and n_i negative examples

$\Rightarrow \mathrm{H}(\langle p_i/(p_i+n_i), n_i/(p_i+n_i)\rangle)$ bits needed to classify a new example

$\Rightarrow$ *expected* number of bits per example over all branches is

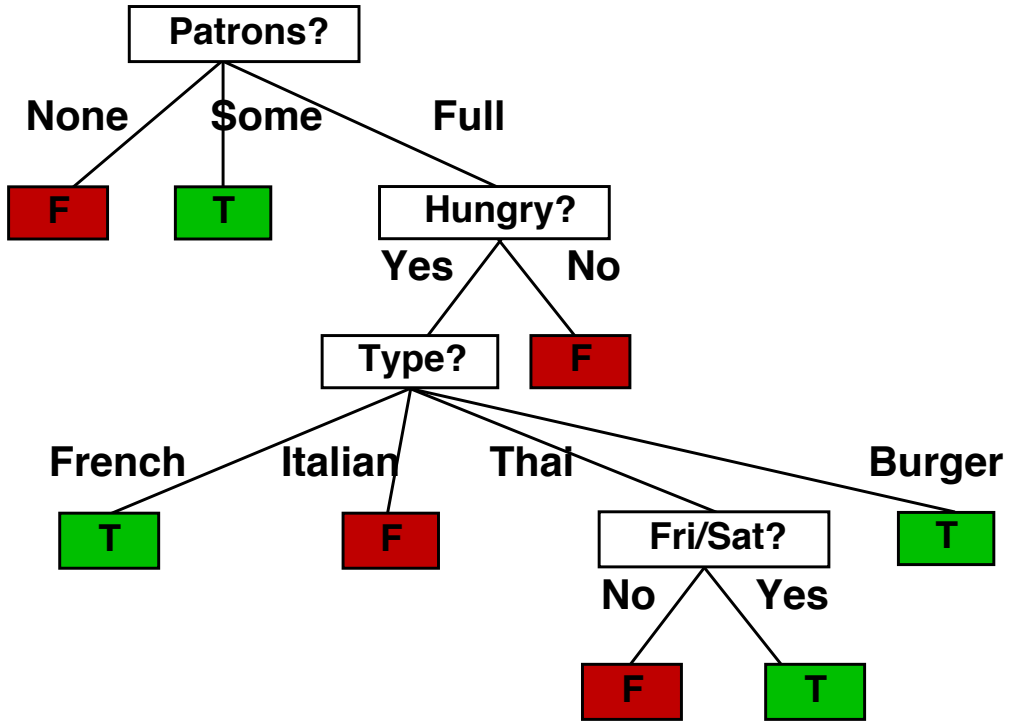
$$\sum_i (p_i + n_i)/(p+n) H(\langle p_i/(p_i+n_i), n_i/(p_i+n_i)\rangle)$$

For Patrons?, this is 0.459 bits, for Type this is (still) 1 bit

$\Rightarrow$ choose the attribute that minimizes the remaining information needed

Example

Decision tree learned from the 12 examples:



Substantially simpler than “true” tree—a more complex hypothesis isn’t justified by small amount of data

Regression trees

Can split on continuous features

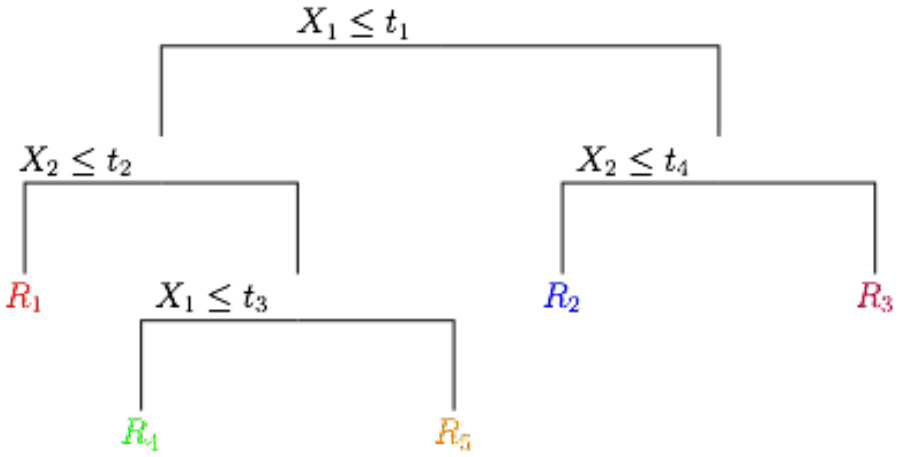
For regression, can generate a *regression tree* with functions of attributes at leaves

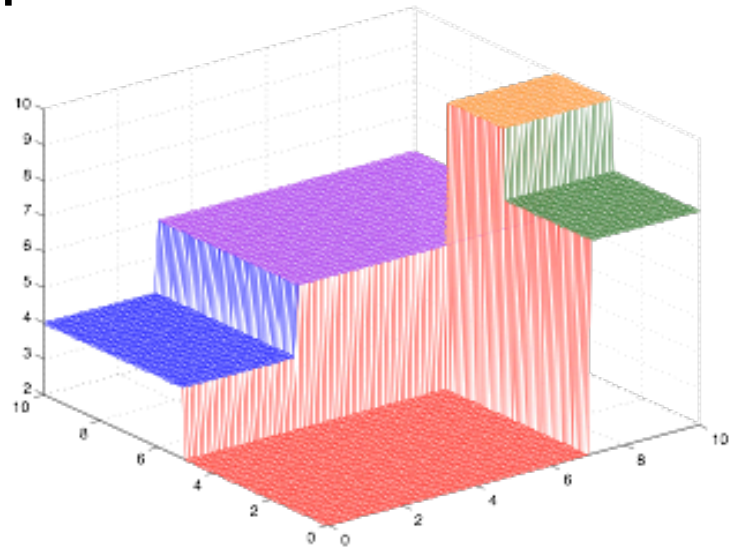
Regression trees

One method is to use *axis parallel splits* to partition the space based on each variable

Can represent function as $f(\mathbf{x}) = \mathbb{E}[y \mid x] = \sum_{m=1}^{M} w_m \mathbb{I}(\mathbf{x} \in R_m) = \sum_{m=1}^{M} w_m \phi(\mathbf{x}, \mathbf{v}_m)$

where R_m is the mth region w_m is the mean response for the region and v_m encodes the choice of variables and split





Generalization and overfitting

There may be many extraneous features (e.g., rolling a die when considering color, weather, size, etc.)

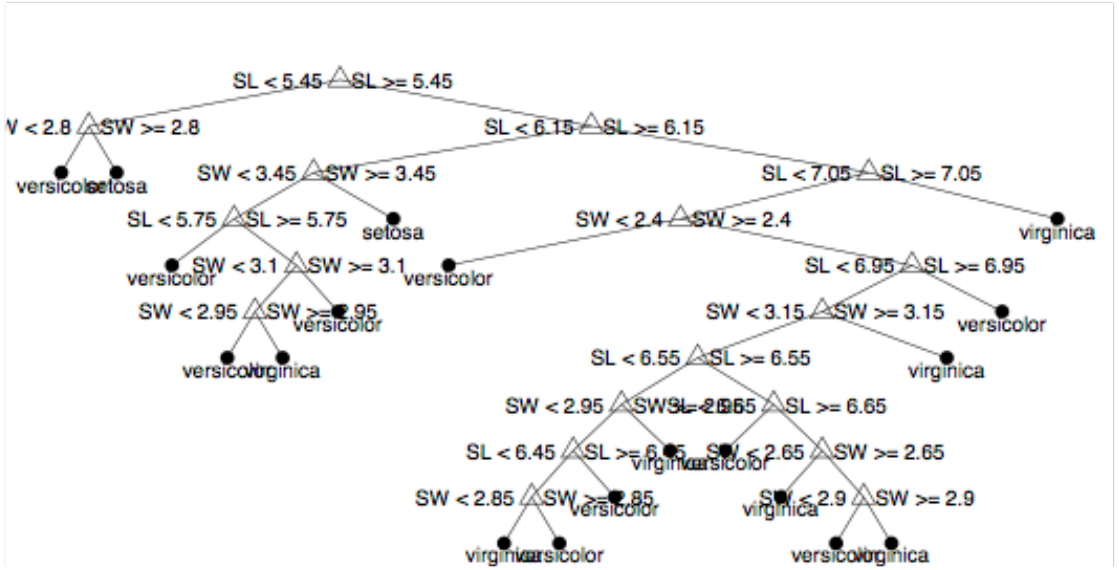
Decision tree algorithms can overfit in this case

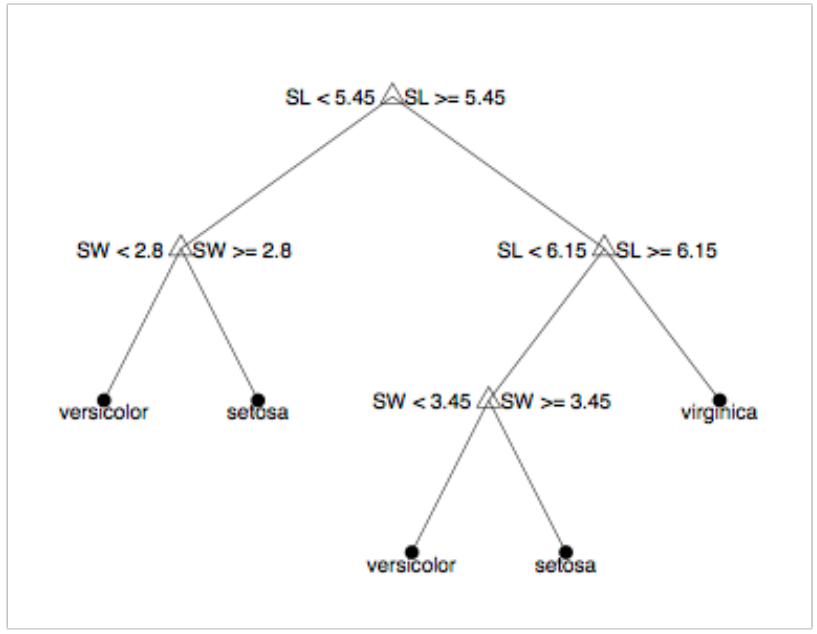
Use *decision tree pruning* to reduce overfitting

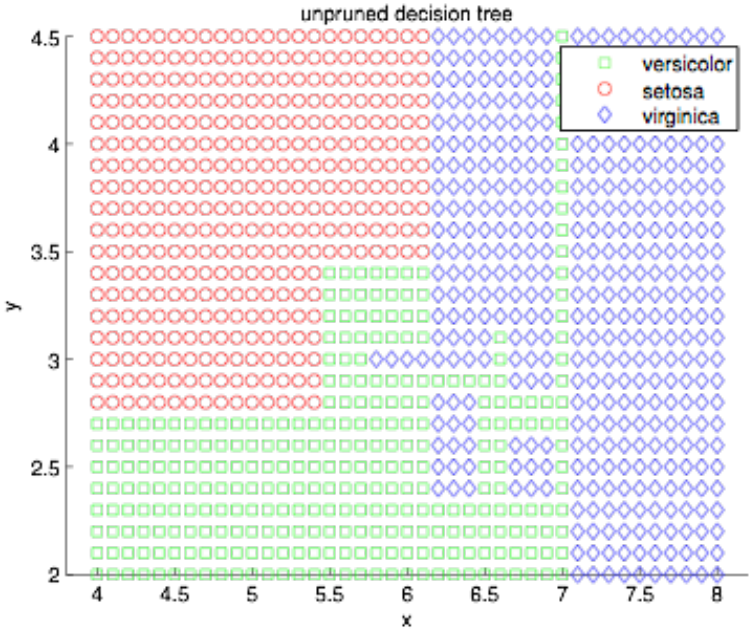
Why not just stop early?

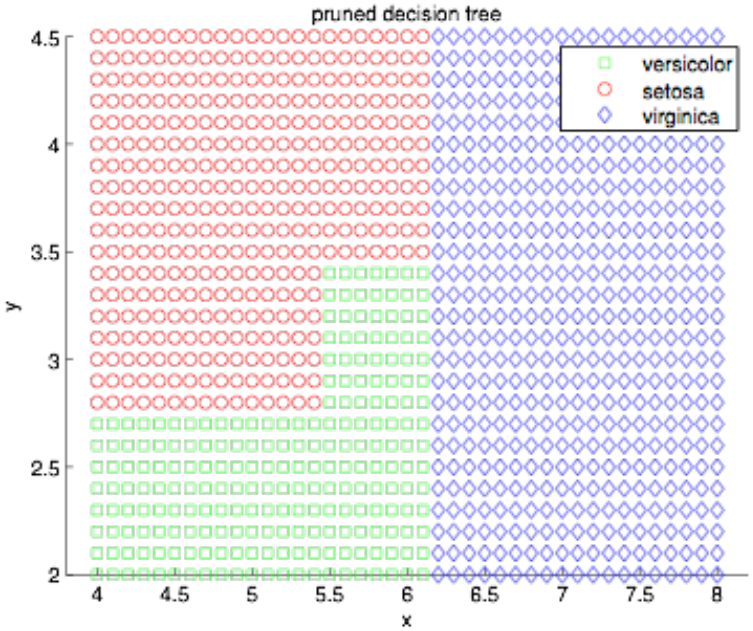
Idea: generate tree and then examine nodes from leaves to root and check with irrelevance (using information gain)

Decision tree pruning









Decision tree summary

Decision trees can be easy for humans to understand (unlike representations like neural nets)

Can mix continuous and discrete variables

Scale well to large datasets

Not as accurate as many other approaches

Tree structure can change drastically with small input changes (unstable or high variance)

Bias and Variance

Ideally, methods would have low bias and low variance, but this is difficult to attain

Bias

The *bias* of an estimator

$$\text{bias}(\hat{\theta}(\cdot)) = \mathbb{E}_{P(D|\theta*)}\left[\hat{\theta}(D) - \theta*\right]$$

I.e., the difference between the expected and the true value

An *unbiased estimator* has a bias of 0

Bias-Variance Tradeoff

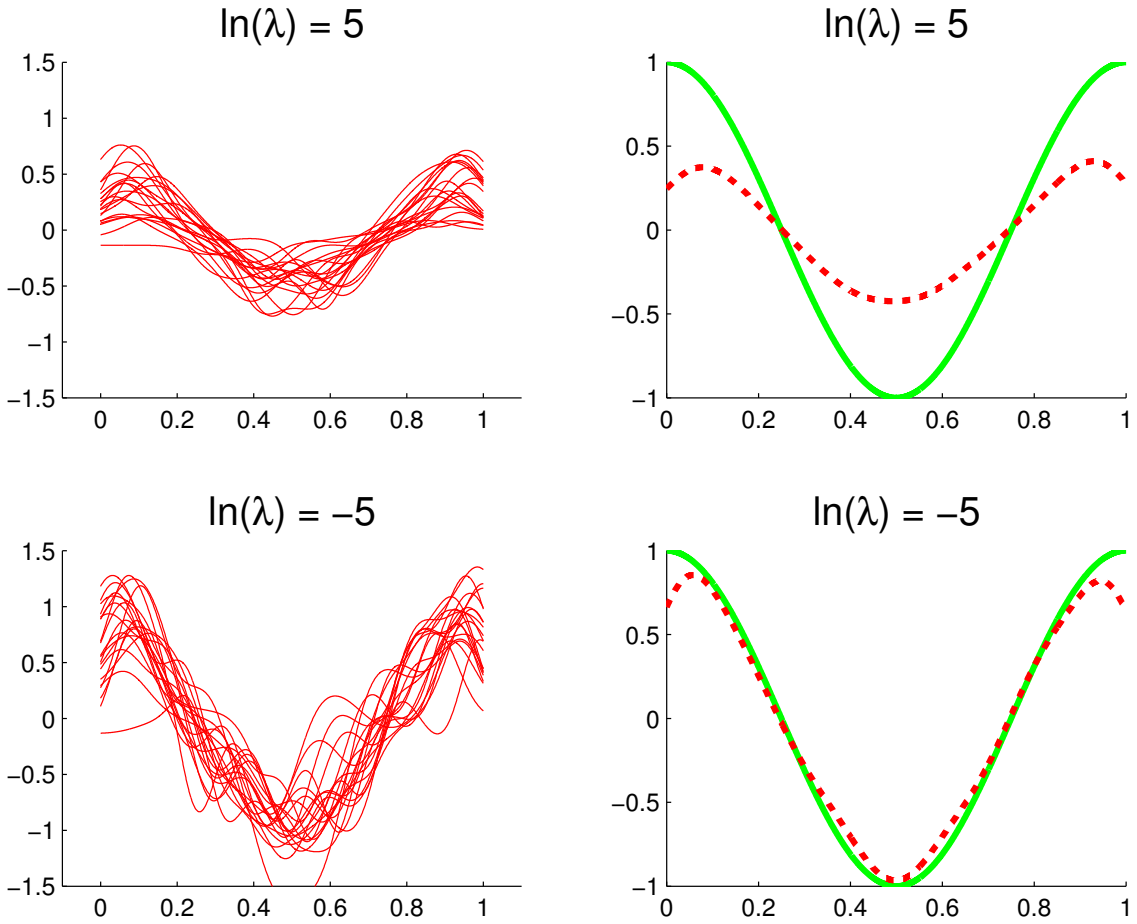
Can calculate the MSE of an estimate:

$$\begin{aligned}
\mathbb{E}_{P(D|\theta^*)}\left[\hat{\theta}-\theta^*\right]^2 &= \mathbb{E}\left[[(\hat{\theta}-\bar{\theta})+(\bar{\theta}-\theta^*)]^2\right] \\
&= \mathbb{E}\left[(\hat{\theta}-\bar{\theta})^2\right]+2(\bar{\theta}-\theta^*)\mathbb{E}\left[\hat{\theta}-\bar{\theta}\right]+\ (\bar{\theta}-\theta^*)^2 \\
&= \mathbb{E}\left[(\hat{\theta}-\bar{\theta})^2\right]+(\bar{\theta}-\theta^*)^2
\end{aligned}$$

So, MSE= variance + bias^2

For MSE, but often important to consider when choosing/designing methods!

Bias-Variance Tradeoff



Ridge regression: true function is solid green

left=20 fits, right=average fit

top=strong regularization, bottom=weak regularization

Ensemble learning

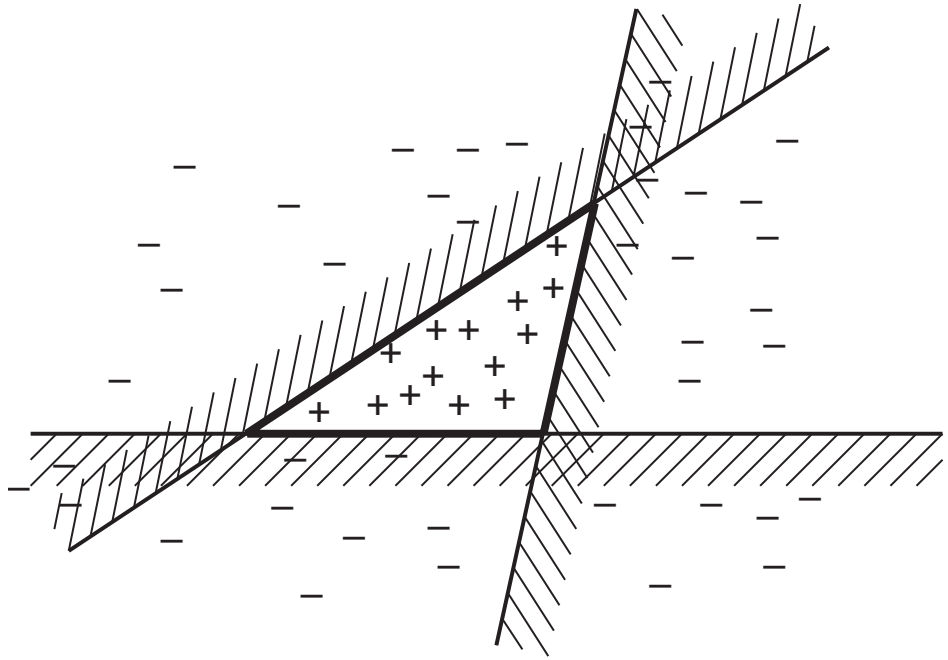
Ensemble learning considers a collection (or ensemble) of hypotheses and combine their predictions

Example: consider an ensemble of $K=5$ classifiers (hypotheses). If we use majority voting, at least 3 would have be incorrect to misclassify an example. If each is independent(?) and has an error of p, the probability that many are wrong becomes very small

Ensemble learning

Ensembles also expand the hypothesis space

Combine 3 linear threshold hypotheses and classify when all 3 are positive



Bagging

Train *M* different trees on different subsets of the data (chosen randomly with replacement)

Compute

For regression: $f(\mathbf{x}) = \sum_{m=1}^{M} \frac{1}{M} f_m(\mathbf{x})$

For classification: voting

This is called *bagging* (bootstrap aggregating)

This reduces the variance and overfitting, but often results in many similar trees (many are highly correlated)

Random forests

Similar to bagging, but reduce the correlation between trees

Randomly chose a subset of *variables and data*

Again, use averaging or voting over trees

Relatively simple to implement, but can be slow to train

Often works very well and is widely used (e.g., Microsoft kinect)

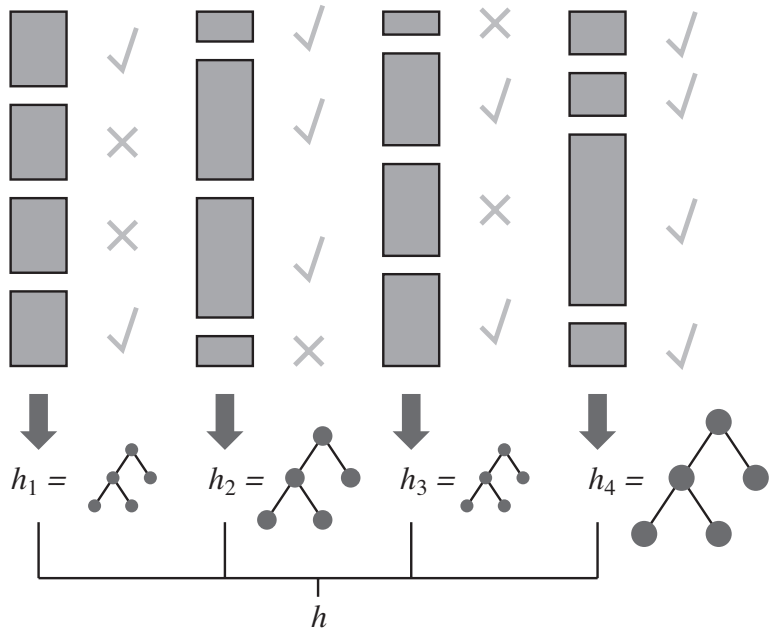
Boosting

Use a *weighted training set* where each example has a weight, $w_j \geq 0$

Boosting is an ensemble learning method that uses a weighted training set

- Starts with $w_j = 1$
- Generates a hypothesis, h_1
- Increase weights on misclassified examples, decrease on correct ones
- Generate next hypothesis and continue for *K* steps
- Use the *K* hypotheses to classify by weighted majority (based on how well they perform on the training set)

Boosting



Boosting methods can use *weak learning algorithms*, which perform only slightly better than random guessing

If the input learning algorithm is a weak learning algorithm, boosting can boost the accuracy of the original algorithm by using the ensemble

AdaBoost can classify the training data perfectly with a large enough K

AdaBoost

Can use decision stumps (decision trees with only a root node) as weak learners

For the restaurant example: