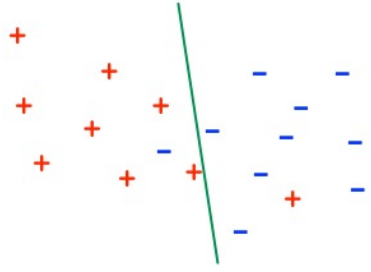
Kernel methods I
Basis expansion

Topics we'll cover

1. Two deviations from linear separability
2. Learning quadratic boundaries using basis expansion

Deviations from linear separability

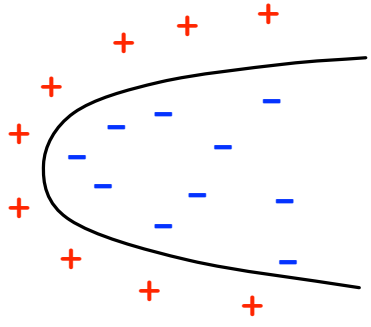
Noise



Find a separator that minimizes a convex loss function related to the number of mistakes.

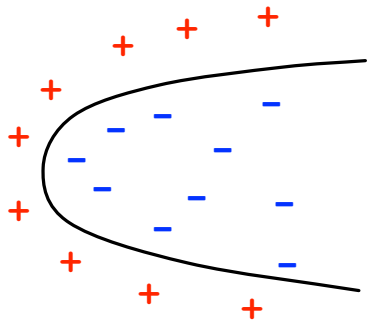
e.g. SVM, logistic regression.

Systematic deviation



What to do with this?

Adding new features



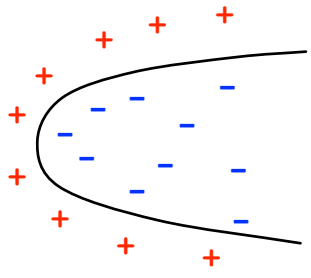
Actual boundary is something like $x_1 = x_2^2 + 5$.

- This is quadratic in $x = (x_1, x_2)$
- But it is linear in $\Phi(x) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

Basis expansion: embed data in a higher-dimensional feature space.
Then we can use a linear classifier!

Basis expansion for quadratic boundaries

How to deal with a **quadratic** boundary?



Idea: augment the regular features $x = (x_1, x_2, \ldots, x_d)$ with

$$x_1^2,\ x_2^2,\ \ldots,\ x_d^2$$
$$x_1x_2,\ x_1x_3,\ \ldots,\ x_{d-1}x_d$$

Enhanced data vectors of the form:

$$\Phi(x) = (x_1, \ldots, x_d, x_1^2, \ldots, x_d^2, x_1x_2, \ldots, x_{d-1}x_d)$$

Quick question

Suppose $x = (x_1, x_2, x_3)$. What is the dimension of $\Phi(x)$?

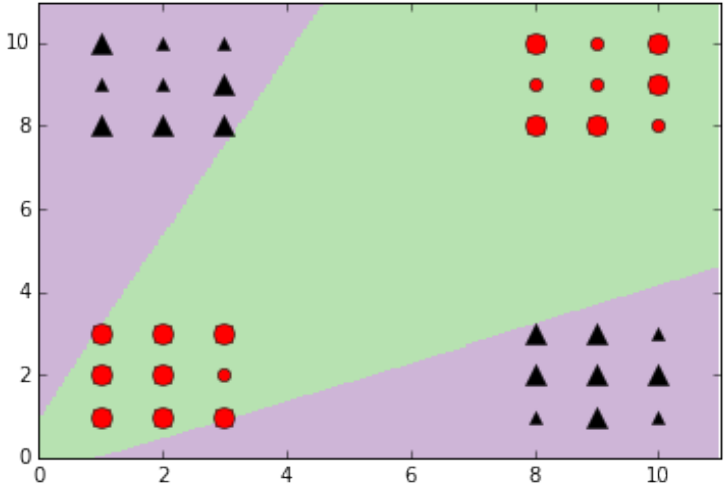
Suppose $x = (x_1, \ldots, x_d)$. What is the dimension of $\Phi(x)$?

Perceptron revisited

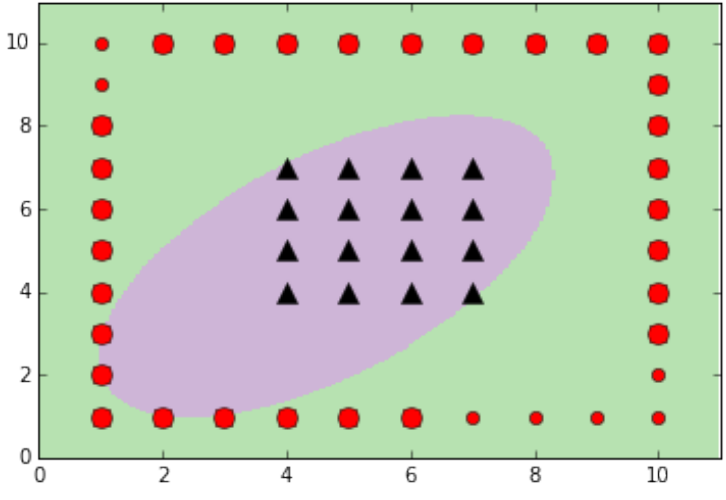
Learning in the higher-dimensional feature space:

- $w = 0$ and $b = 0$
- while some $y(w \cdot \Phi(x) + b) \leq 0$:
  - $w = w + y\,\Phi(x)$
  - $b = b + y$

Perceptron with basis expansion: examples



Perceptron with basis expansion: examples

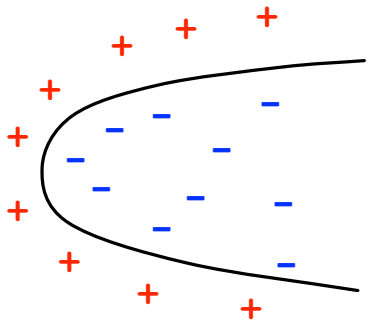


Kernel methods II
The kernel trick

Topics we'll cover

1. The kernel trick for quadratic boundaries
2. The kernel Perceptron

Adding new features



Actual boundary is something like $x_1 = x_2^2 + 5$.

- This is quadratic in $x = (x_1, x_2)$
- But it is linear in $\Phi(x) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

Basis expansion: embed data in a higher-dimensional feature space.
Then we can use a linear classifier!

Perceptron with basis expansion

Learning in the higher-dimensional feature space:

- $w = 0$ and $b = 0$
- while some $y(w \cdot \Phi(x) + b) \leq 0$:
  - $w = w + y\,\Phi(x)$
  - $b = b + y$

Problem: number of features has now increased dramatically.
For MNIST, with quadratic boundary: from 784 to 308504.

The kernel trick: implement this without ever writing down a vector in the higher-dimensional space!

The kernel trick

1. w is always a linear combination of the $\Phi(x^{(i)})$.

$$w = \sum_{j=1}^{n} \alpha_j y^{(j)} \Phi(x^{(j)})$$

   Represent w in **dual** form: $\alpha = (\alpha_1, \ldots, \alpha_n)$.

2. Compute $w \cdot \Phi(x)$ using the dual representation.

$$w \cdot \Phi(x) = \sum_{j=1}^{n} \alpha_j y^{(j)} (\Phi(x^{(j)}) \cdot \Phi(x))$$

3. Compute $\Phi(x) \cdot \Phi(z)$ without ever writing out $\Phi(x)$ or $\Phi(z)$.

- $w = 0$ and $b = 0$
- while some $y^{(i)}(w \cdot \Phi(x^{(i)}) + b) \leq 0$:
  - $w = w + y^{(i)} \Phi(x^{(i)})$
  - $b = b + y^{(i)}$

Computing dot products

First, in 2-d.
Suppose $x = (x_1, x_2)$ and $\Phi(x) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$.

Actually, tweak a little: $\Phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)$

What is $\Phi(x) \cdot \Phi(z)$?

Computing dot products

Suppose $x = (x_1, x_2, \ldots, x_d)$ and

$$\Phi(x) = (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{d-1}x_d)$$

$$\begin{aligned}
\Phi(x) \cdot \Phi(z) &= (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{d-1}x_d) \cdot \\
&\quad (1, \sqrt{2}z_1, \ldots, \sqrt{2}z_d, z_1^2, \ldots, z_d^2, \sqrt{2}z_1z_2, \ldots, \sqrt{2}z_{d-1}z_d) \\
&= 1 + 2\sum_i x_iz_i + \sum_i x_i^2z_i^2 + 2\sum_{i \neq j} x_ix_jz_iz_j \\
&= (1 + x_1z_1 + \cdots + x_dz_d)^2 \quad = \quad (1 + x \cdot z)^2
\end{aligned}$$

For MNIST:
We are computing dot products in 308504-dimensional space.
But it takes time proportional to 784, the original dimension!

Kernel Perceptron

Learning from data $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathcal{X} \times \{-1, 1\}$

Primal form:

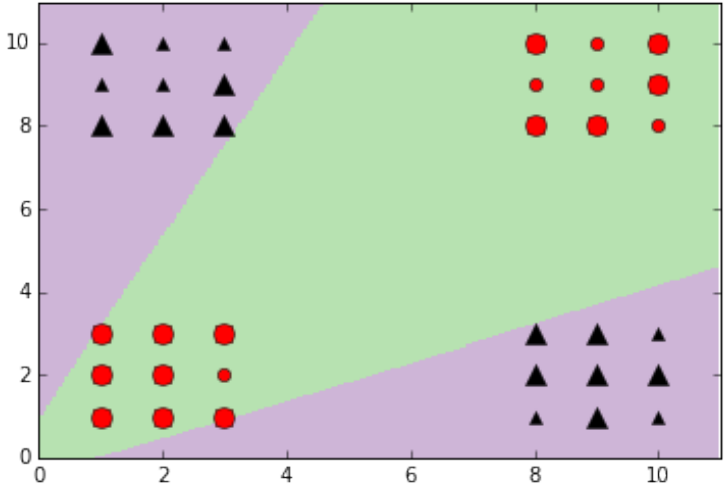
- $w = 0$ and $b = 0$
- while there is some i with $y^{(i)}(w \cdot \Phi(x^{(i)}) + b) \leq 0$:
  - $w = w + y^{(i)}\, \Phi(x^{(i)})$
  - $b = b + y^{(i)}$

Dual form: $w = \sum_j \alpha_j y^{(j)} \Phi(x^{(j)})$, where $\alpha \in \mathbb{R}^n$

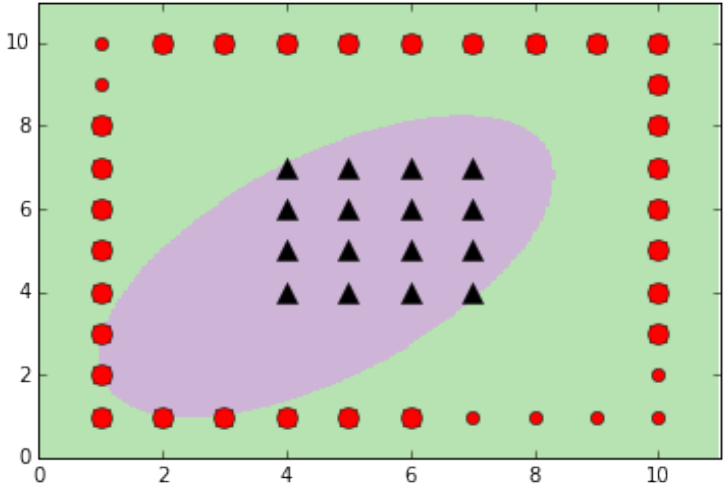
- $\alpha = 0$ and $b = 0$
- while some i has $y^{(i)} \left(\sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x^{(i)}) + b \right) \leq 0$:
  - $\alpha_i = \alpha_i + 1$
  - $b = b + y^{(i)}$

To classify a new point x: sign $\left(\sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x) + b \right)$.

Kernel Perceptron: examples



Kernel Perceptron: examples

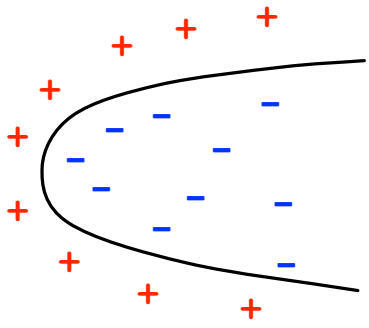


Kernel methods III
Kernel SVM

Topics we'll cover

1. Kernel SVM
2. Polynomial decision boundaries

Step 1: basis expansion



Actual boundary is something like $x_1 = x_2^2 + 5$.

- This is quadratic in $x = (x_1, x_2)$
- But it is linear in $\Phi(x) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)$

Basis expansion: embed data in a higher-dimensional feature space.
Then we can use a linear classifier!

Perceptron with basis expansion

Learning from data $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathcal{X} \times \{-1, 1\}$

Primal form of the Perceptron:

- $w = 0$ and $b = 0$
- while there is some i with $y^{(i)}(w \cdot \Phi(x^{(i)}) + b) \leq 0$:
  - $w = w + y^{(i)}\, \Phi(x^{(i)})$
  - $b = b + y^{(i)}$

Problem: w and $\Phi(x)$ can be very high-dimensional.

Solution: work in the dual space, writing

$$w = \sum_j \alpha_j y^{(j)} \Phi(x^{(j)})$$

Step 2: the kernel trick

Dual form of the Perceptron:

- $\alpha = 0$ and $b = 0$
- while some i has $y^{(i)} \left(\sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x^{(i)}) + b \right) \leq 0$:
  - $\alpha_i = \alpha_i + 1$
  - $b = b + y^{(i)}$

Classify a new point x: sign $\left(\sum_j \alpha_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x) + b \right)$.

Does this work with SVMs?

$$\begin{aligned} &(\text{PRIMAL}) && \min_{w\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} && \|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ &\text{s.t.:} && y^{(i)}(w\cdot x^{(i)} + b) \geq 1-\xi_i && \text{for all } i = 1,2,\ldots,n \\ & && \xi \geq 0 \end{aligned}$$

$$\begin{aligned} &(\text{DUAL}) && \max_{\alpha\in\mathbb{R}^n} && \sum_{i=1}^{n}\alpha_i - \sum_{i,j=1}^{n}\alpha_i\alpha_j y^{(i)}y^{(j)}(x^{(i)}\cdot x^{(j)}) \\ & && \text{s.t.:} && \sum_{i=1}^{n}\alpha_i y^{(i)} = 0 \\ & && && 0 \leq \alpha_i \leq C \end{aligned}$$

Solution: $w = \sum_i \alpha_i y^{(i)} x^{(i)}$.

Kernel SVM

1. **Basis expansion.** Mapping $x \mapsto \Phi(x)$.
2. **Learning.** Solve the dual problem:

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} (\Phi(x^{(i)}) \cdot \Phi(x^{(j)}))$$

$$\text{s.t.:} \sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

$$0 \leq \alpha_i \leq C$$

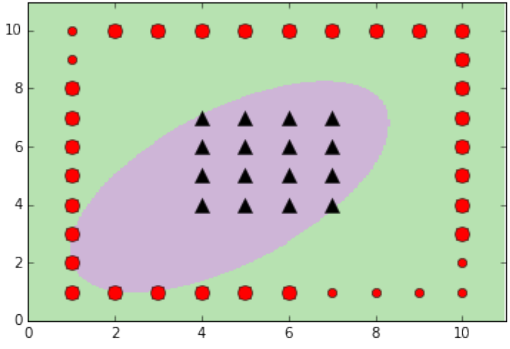
   This yields $w = \sum_i \alpha_i y^{(i)} \Phi(x^{(i)})$. Offset b also follows.

3. **Classification.** Given a new point x, classify as

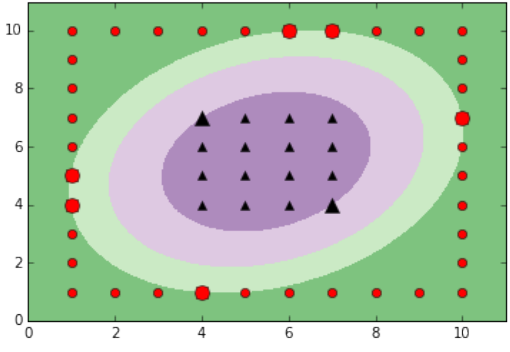
$$\text{sign} \left(\sum_i \alpha_i y^{(i)} (\Phi(x^{(i)}) \cdot \Phi(x)) + b \right).$$

Kernel Perceptron vs. Kernel SVM: examples

Perceptron:

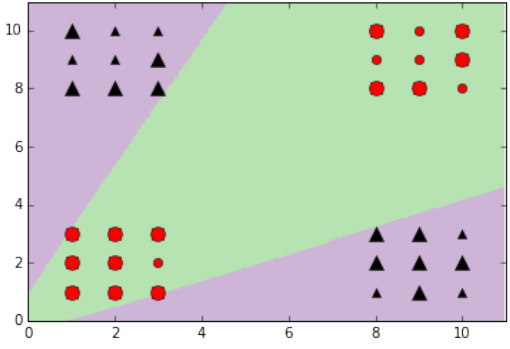


SVM:

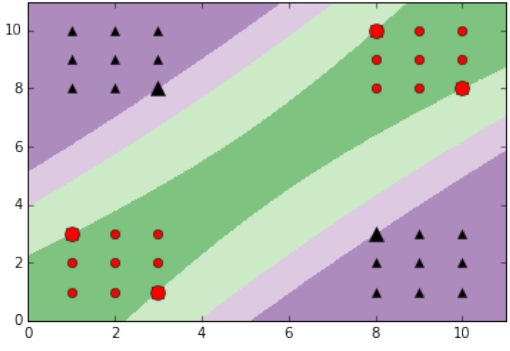


Kernel Perceptron vs. Kernel SVM: examples

Perceptron:

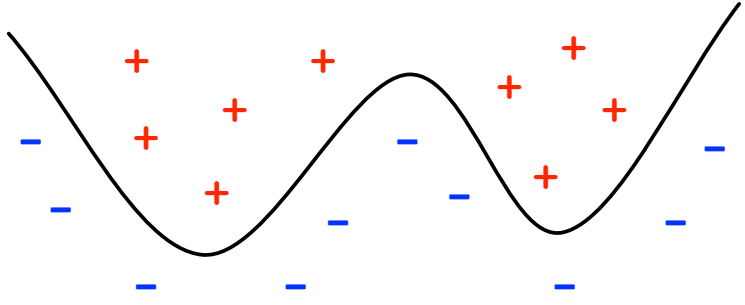


SVM:



Polynomial decision boundaries

When the decision surface is a polynomial of order p:



- Let $\Phi(x)$ consist of all terms of order $\leq p$, such as $x_1 x_2^2 x_3^{p-3}$. (How many such terms are there, roughly?)
- Same trick works: $\Phi(x) \cdot \Phi(z) = (1 + x \cdot z)^p$.

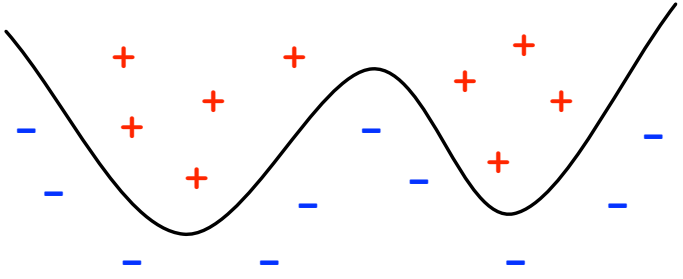
Kernel methods IV
The kernel function

Topics we'll cover

1. The kernel function
2. The RBF kernel

Basis expansion

Suppose we want a decision boundary that is a polynomial of order p:



Add new features to data vectors x:

- Let $\Phi(x)$ consist of all terms of order $\leq p$, such as $x_1 x_2^2 x_3^{p-3}$.
- Degree-p polynomial in $x \Leftrightarrow$ linear in $\Phi(x)$.
- $\Phi(x) \cdot \Phi(z) = (1 + x \cdot z)^p$.

Kernel SVM

1. **Basis expansion.** Mapping $x \mapsto \Phi(x)$.
2. **Learning.** Solve the dual problem:

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} (\Phi(x^{(i)}) \cdot \Phi(x^{(j)}))$$

$$\text{s.t.:} \sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

$$0 \leq \alpha_i \leq C$$

   This yields $\alpha = (\alpha_1, \ldots, \alpha_n)$. Offset b also follows.

3. **Classification.** Given a new point x, classify as

$$\text{sign} \left(\sum_i \alpha_i y^{(i)} (\Phi(x^{(i)}) \cdot \Phi(x)) + b \right).$$

Kernel SVM, revisited

1. **Kernel function.** Define a similarity function $k(x, z)$.
2. **Learning.** Solve the dual problem:

$$\max_{\alpha \in \mathbb{R}^n} \quad \sum_{i=1}^{n} \alpha_i - \sum_{i,j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} k(x^{(i)}, x^{(j)})$$

$$\text{s.t.:} \quad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

$$0 \leq \alpha_i \leq C$$

   This yields α. Offset b also follows.

3. **Classification.** Given a new point x, classify as

$$\text{sign}\left(\sum_i \alpha_i y^{(i)} k(x^{(i)}, x) + b\right).$$

The kernel function

We never explicitly construct the embedding $\Phi(x)$.

- What we actually use is the **kernel function** $k(x, z) = \Phi(x) \cdot \Phi(z)$.
- Can think of $k(x, z)$ as a **measure of similarity** between x and z.
- Rewrite learning algorithm and final classifier in terms of k.

Kernel Perceptron:

- $\alpha = 0$ and $b = 0$
- while some i has $y^{(i)} \left(\sum_j \alpha_j y^{(j)} k(x^{(j)}, x^{(i)}) + b \right) \leq 0$:
  - $\alpha_i = \alpha_i + 1$
  - $b = b + y^{(i)}$

To classify a new point x: sign $\left(\sum_j \alpha_j y^{(j)} k(x^{(j)}, x) + b \right)$.

Choosing the kernel function

The final classifier is a **similarity-weighted vote**,

$$F(x) = \alpha_1 y^{(1)} k(x^{(1)}, x) + \cdots + \alpha_n y^{(n)} k(x^{(n)}, x)$$

(plus an offset term, b).

Can we choose k to be **any** similarity function?

- Not quite: need $k(x, z) = \Phi(x) \cdot \Phi(z)$ for *some* embedding Φ.
- **Mercer's condition**: same as requiring that for any finite set of points $x^{(1)}, \ldots, x^{(m)}$, the $m \times m$ similarity matrix K given by

  $$K_{ij} = k(x^{(i)}, x^{(j)})$$

  is positive semidefinite.

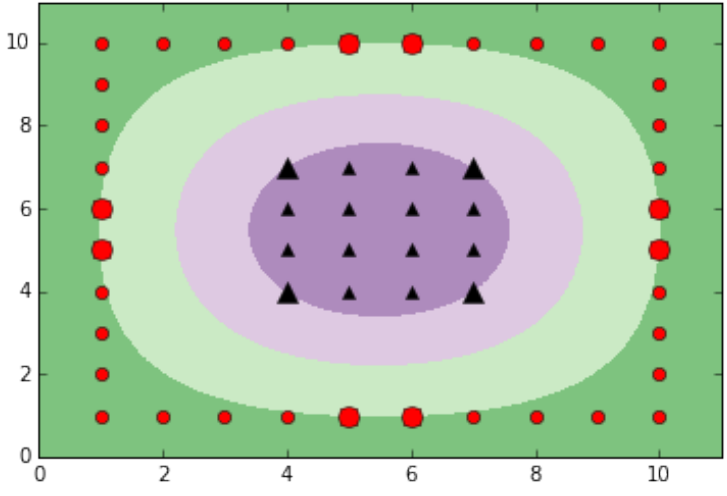
The RBF kernel

A popular similarity function: the **Gaussian kernel** or **RBF kernel**

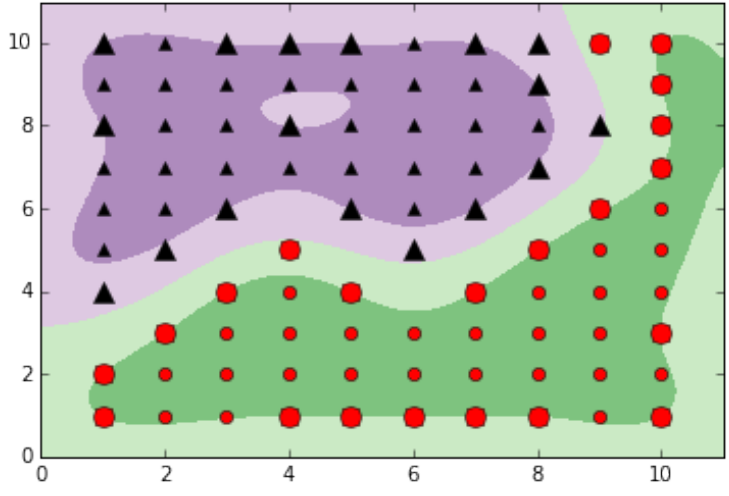
$$k(x, z) = e^{-\|x-z\|^2/s^2},$$

where s is an adjustable scale parameter.

RBF kernel: examples



RBF kernel: examples



The scale parameter

Recall prediction function: $F(x) = \alpha_1 y^{(1)} k(x^{(1)}, x) + \cdots + \alpha_n y^{(n)} k(x^{(n)}, x)$.

For the RBF kernel, $k(x, z) = e^{-\|x-z\|^2/s^2}$,

1. How does this function behave as $s \uparrow \infty$?
2. How does this function behave as $s \downarrow 0$?
3. As we get more data, should we increase or decrease s?

Combining classifiers

Choosing a classifier

So many choices:

- Nearest neighbor
- Different generative models
- Linear predictors with different loss functions
- Different kernels
- Neural nets
- etc.

Can one **combine** them?
And get a classifier that is better than any of them individually?

Combining simple classifiers

1. No one classifier is going to be the final product.
   So why not keep the individual components simple?

2. How to train each constituent classifier?
   On the full training set?

3. The full (combined) models may get enormous.
   Is this bad for generalization?

Decision trees

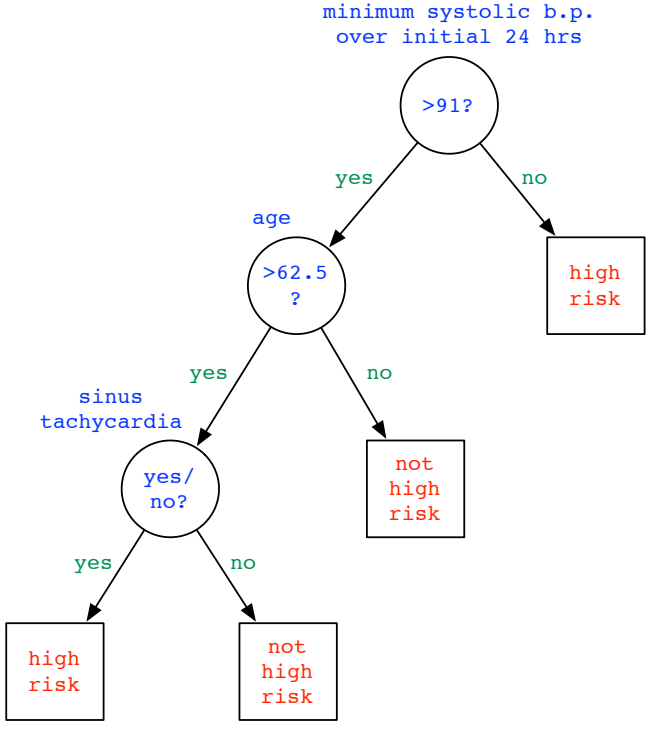
Topics we'll cover

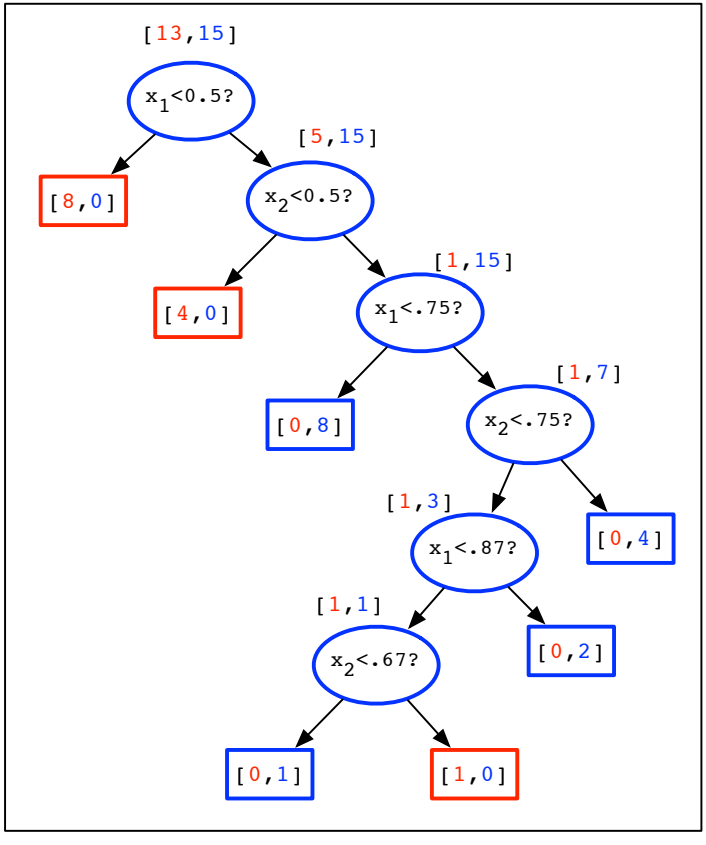
1. The form of a decision tree classifier
2. A top-down learning algorithm
3. Overfitting

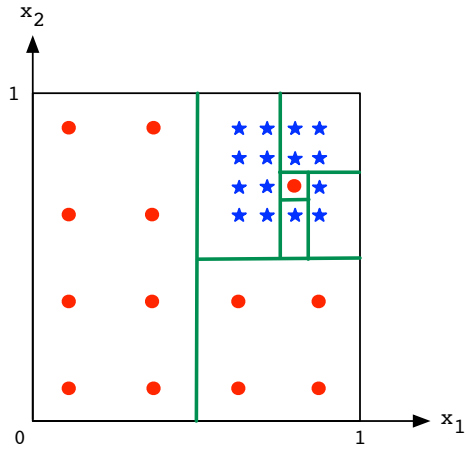
Decision trees

UCSD Medical Center (1970s):
identify patients at risk of dying
within 30 days after heart attack.

Data set:
215 patients.
37 (=20%) died.
19 features.




[13,15]
x1<0.5?
[8,0]
[5,15]
x2<0.5?
[4,0]
[1,15]
x1<.75?
[0,8]
[1,7]
x2<.75?
[1,3]
x1<.87?
[0,4]
[1,1]
x2<.67?
[0,2]
[0,1]
[1,0]


x2
1
0
1
x1

Building a decision tree: summary

Greedy algorithm: build tree top-down.

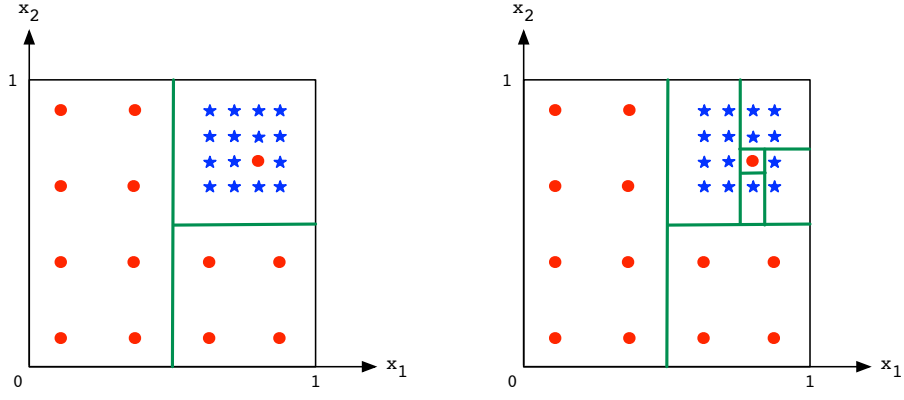
- Start with a single node containing all data points
- Repeat:
  - Look at all current leaves and all possible splits
  - Choose the split that most reduces **uncertainty in prediction**. Several ways to quantify this: Gini index, entropy, etc.

When to stop?

- When each leaf is pure?
- When the tree is already pretty big?
- When each leaf has uncertainty below some threshold?

Overfitting?

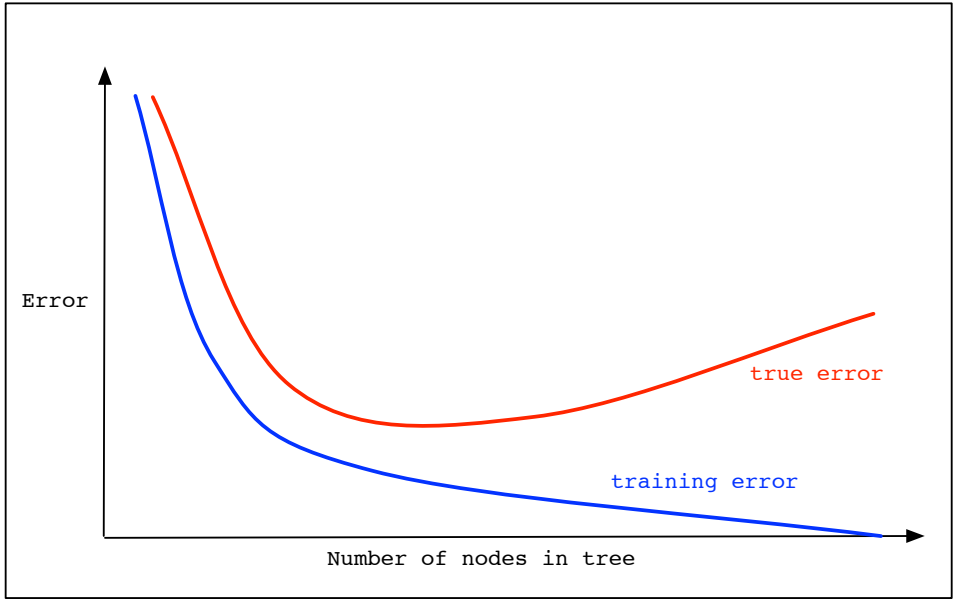
Go back a few steps...



Final partition does better on training data, but is more complex.
That one point might have been an outlier anyway.

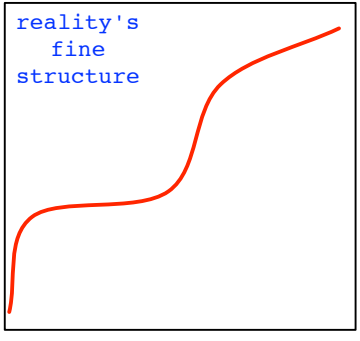
We have probably ended up **overfitting** the data.

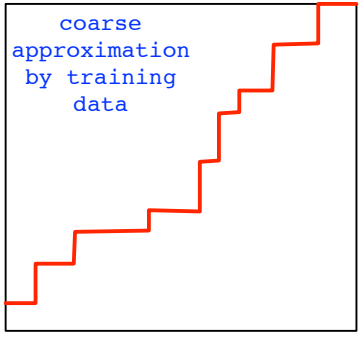
Overfitting: picture



Overfitting: perspective

- The training data reflects an underlying reality, so it helps us.
- But it also has chance structure of its own – we must avoid modeling this.





Decision tree properties

A flexible and expressive family of classifiers:

- Can accommodate any type of data: numeric or categorical
- Can accommodate any number of classes
- Can fit any data set

But this also means that there is serious danger of overfitting.

Common strategies:

- Stop when leaves are pure enough
- Stop when tree reaches a certain size
- Grow tree, then **prune** with a validation set

Boosting

Topics we'll cover

1. Weak learners
2. The AdaBoost learning algorithm

Weak learners

It is often easy to come up with a **weak classifier**, one that is marginally better than random guessing:

$$\Pr(h(X) \neq Y) \leq \frac{1}{2} - \epsilon$$

A learning algorithm that can consistently generate such classifiers is called a **weak learner**.

Is it possible to systematically boost the quality of a weak learner?

The blueprint for boosting

Given: data set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$.

- Initially give all points equal weight.
- Repeat for $t = 1, 2, \ldots$:
  - Feed weighted data set to the weak learner, get back a weak classifier h_t
  - Reweight data to put more emphasis on points that h_t gets wrong
- Combine all these h_t's linearly

AdaBoost

Data set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$, labels $y^{(i)} \in \{-1, +1\}$.

1. Initialize $D_1(i) = 1/n$ for all $i = 1, 2, \ldots, n$
2. For $t = 1, 2, \ldots, T$:
   - Give D_t to weak learner, get back some $h_t : \mathcal{X} \to [-1, 1]$
   - Compute h_t's margin of correctness:

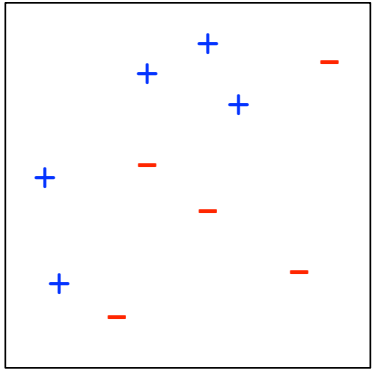
$$r_t = \sum_{i=1}^{n} D_t(i) y^{(i)} h_t(x^{(i)}) \in [-1, 1]$$

$$\alpha_t = \frac{1}{2} \ln \frac{1 + r_t}{1 - r_t}$$

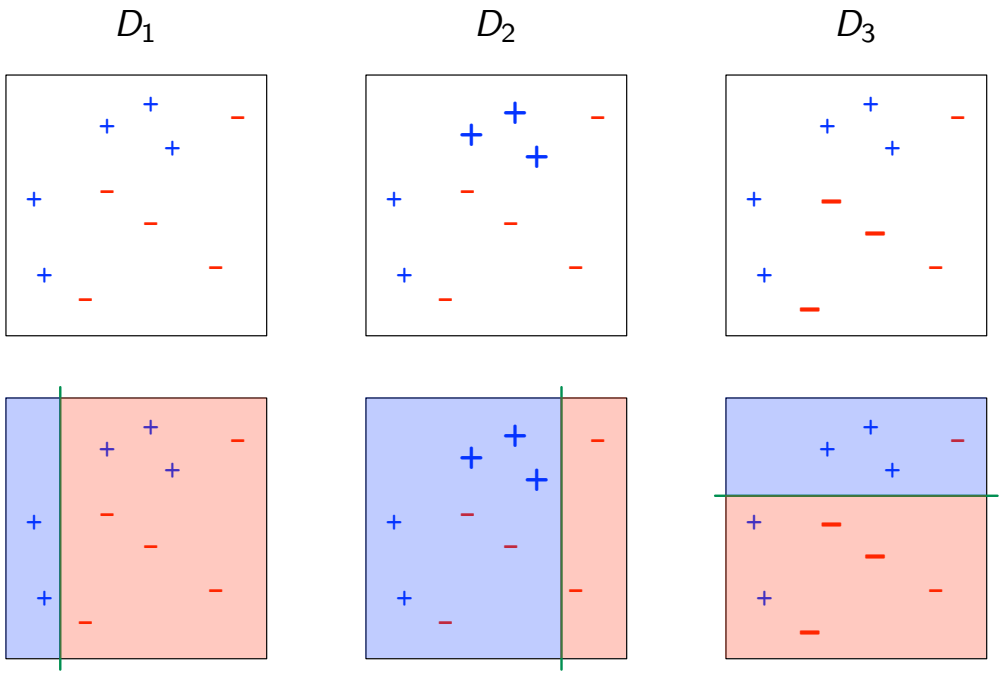
   - Update weights: $D_{t+1}(i) \propto D_t(i) \exp\left(-\alpha_t y^{(i)} h_t(x^{(i)})\right)$
3. Final classifier: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Example (Freund-Schapire)

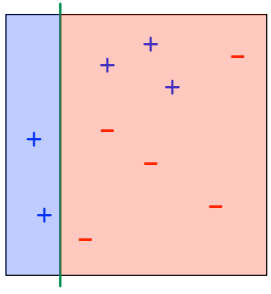
Training set:



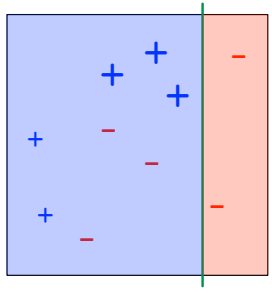
Use “decision stumps” (single-feature thresholds) as weak classifiers



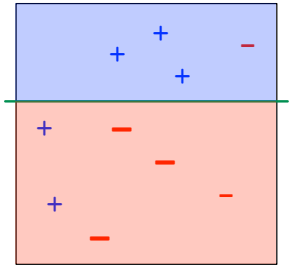
$r_1 = 0.40, \alpha_1 = 0.42$ $r_2 = 0.58, \alpha_2 = 0.65$ $r_3 = 0.72, \alpha_3 = 0.92$



h_1
$\alpha_1 = 0.42$



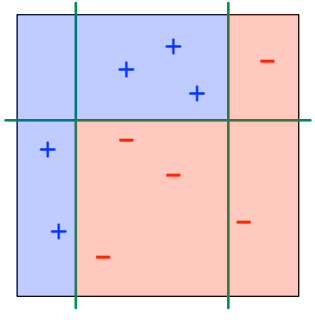
h_2
$\alpha_2 = 0.65$



h_3
$\alpha_3 = 0.92$

Final classifier:

$\text{sign}\left(0.42h_1(x) + 0.65h_2(x) + 0.92h_3(x)\right)$



The surprising power of weak learning

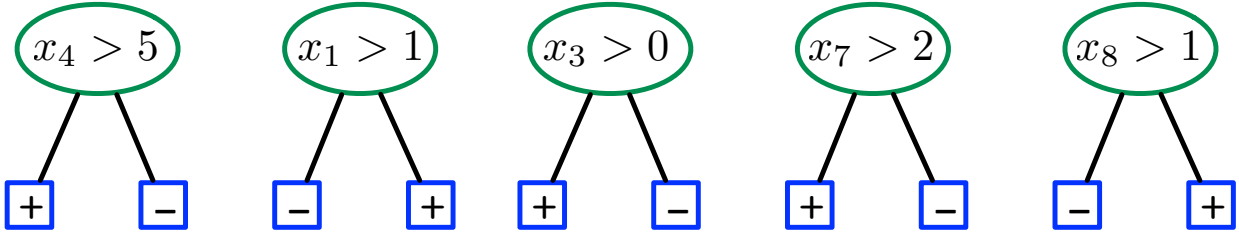
Suppose that on each round t, the weak learner returns a rule h_t whose error on the time-t weighted data distribution is $\leq 1/2 - \gamma$.

Then, after T rounds, the training error of the combined rule

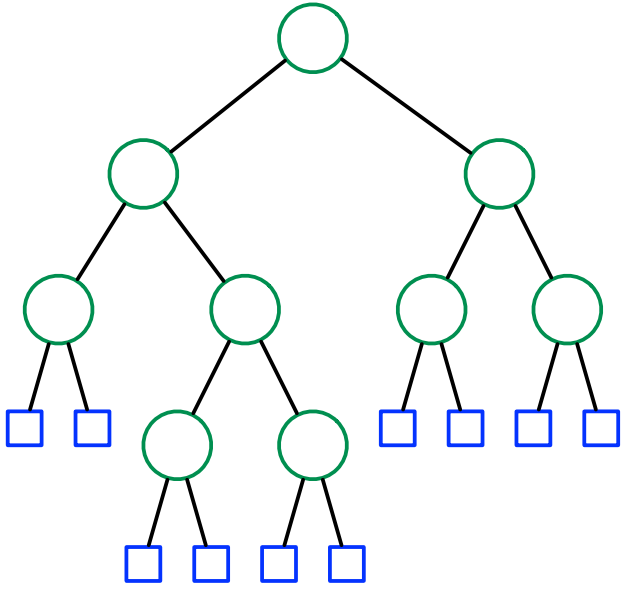
$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

is at most $e^{-\gamma^2 T/2}$.

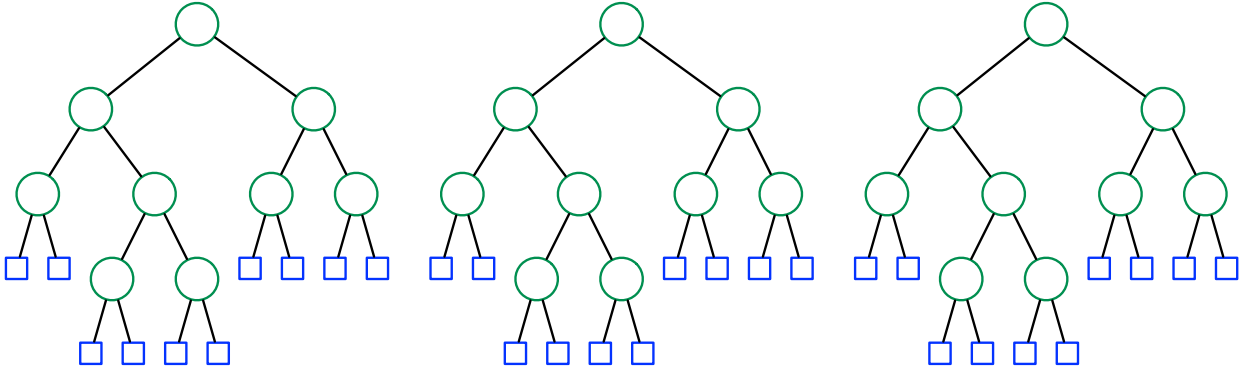
Boosting decision stumps and trees



Boosting decision stumps and trees



Boosting decision stumps and trees

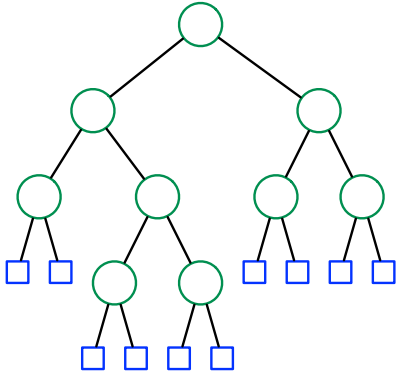


Random forests

Topics we'll cover

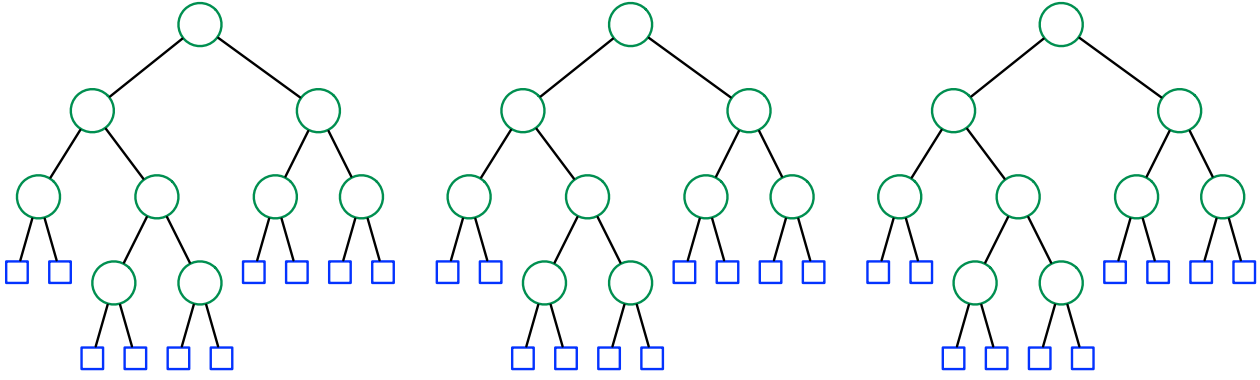
1. Ensembles of tree classifiers
2. The random forest construction
3. An illustrative experiment

From tree to forest



- Decision tree. Starts overfitting beyond a point.

From tree to forest



- Boosted decision trees. Learning is sequential, slow.

Random forests

Given a data set S of n labeled points:

- For $t = 1$ to T:
  - Choose n' points randomly, with replacement, from S.
  - Fit a decision tree h_t to these points.
    - At each node restrict to one of k features chosen at random.

Example settings:

- $n' = n$
- $k = \sqrt{d}$ for d-dimensional data

Final predictor: majority vote of $h_1, \ldots, h_T$.

An ecological prediction problem: "covertype" data

Predict forest type:

- Spruce-fir
- Lodgepole pine
- 5 other classes

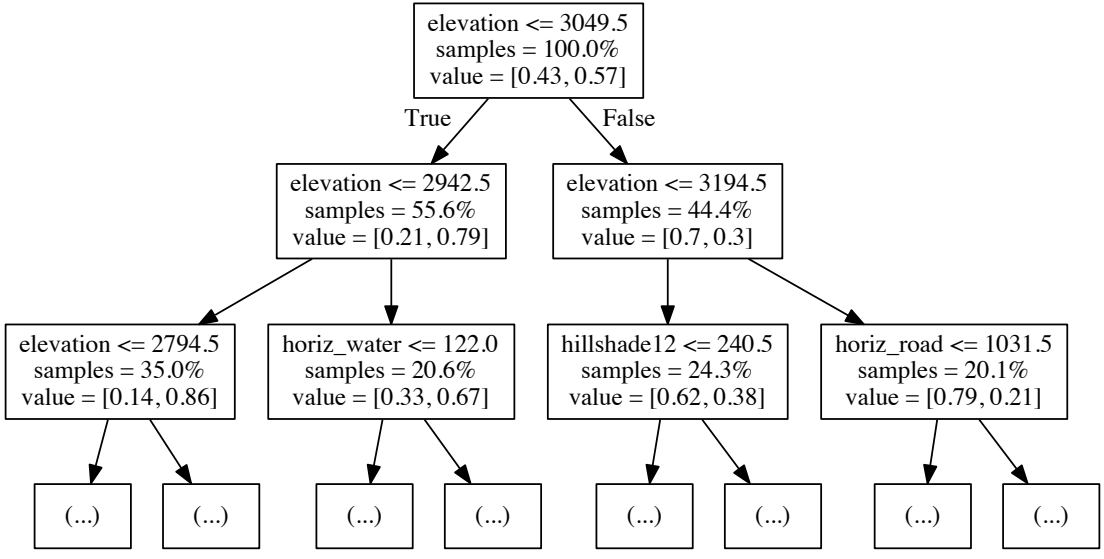
54 cartographic/geological features:

- Elevation, slope, amount of shade, ...
- Distance to water, road, ...
- Soil type

Data set details:

- 49,514 training points
- 445,627 test points

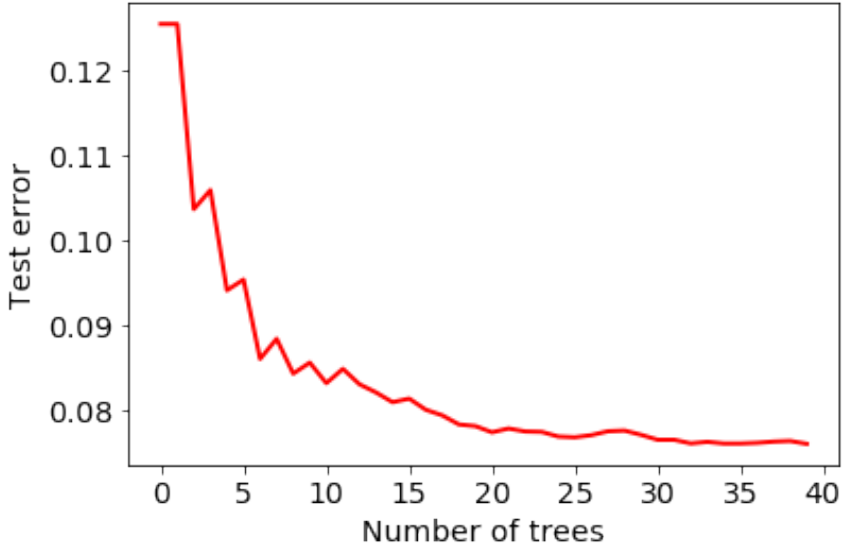
Decision tree



Depth 20: training error 1%, test error 12.6%

Boosted decision trees

Trees of depth 20.



Random forest

Recall:

- Decision tree: depth 20, test error 12.6%
- Boosted decision trees, 10 trees, depth 20: test error 8.7%

Random forest setting: 10 trees, 50% features dropped, depth 40.

- Each individual tree has test error 15% to 17%
- Forest test error: 8.8%