# Machine Learning Algorithms for Classification

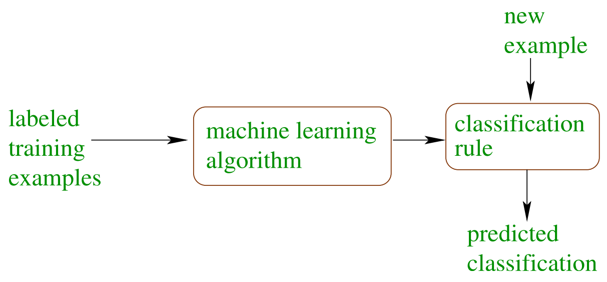
Rob Schapire

## Why Use Machine Learning?

- advantages:
  - often much more accurate than human-crafted rules (since data driven)
  - humans often incapable of expressing what they know (e.g., rules of English, or how to recognize letters), but can easily classify examples
  - don't need a human expert or programmer
  - flexible — can apply to any learning task
  - cheap — can use in applications requiring many classifiers (e.g., one per customer, one per product, one per web page, ...)
- disadvantages
  - need a lot of labeled data
  - error prone — usually impossible to get perfect accuracy

## Machine Learning

- studies how to automatically learn to make accurate predictions based on past observations
- classification problems:
  - classify examples into given set of categories



## Machine Learning Algorithms

- this talk:
  - decision trees
  - boosting
  - support-vector machines
- others not covered:
  - neural networks
  - nearest neighbor algorithms
  - Naive Bayes
  - bagging
  - ⋮

## Examples of Classification Problems

- text categorization
  - e.g.: spam filtering
  - e.g.: categorize news articles by topic
- fraud detection
- optical character recognition
- natural-language processing
  - e.g.: part-of-speech tagging
  - e.g.: spoken language understanding
- market segmentation
  - e.g.: predict if customer will respond to promotion
  - e.g.: predict if customer will switch to competitor
- medical diagnosis
- ⋮

## Decision Trees

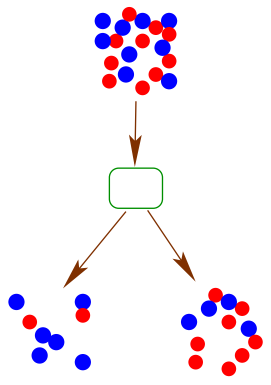
## Example: Good versus Evil

- problem: identify people as good or bad from their appearance

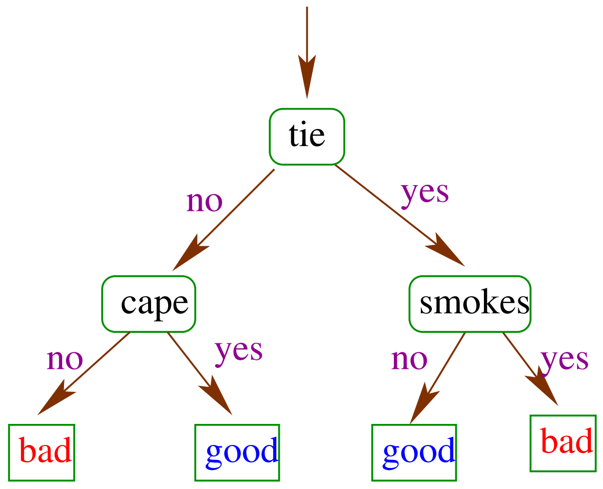
| | sex | mask | cape | tie | ears | smokes | class |
|---|---|---|---|---|---|---|---|
| | | | training data | | | | |
| batman | male | yes | yes | no | yes | no | Good |
| robin | male | yes | yes | no | no | no | Good |
| alfred | male | no | no | yes | no | no | Good |
| penguin | male | no | no | yes | no | yes | Bad |
| catwoman | female | yes | no | no | yes | no | Bad |
| joker | male | no | no | no | no | no | Bad |
| | | | test data | | | | |
| batgirl | female | yes | yes | no | yes | no | ?? |
| riddler | male | yes | no | no | no | no | ?? |

## Choosing the Splitting Rule

- choose rule that leads to greatest increase in “purity”:



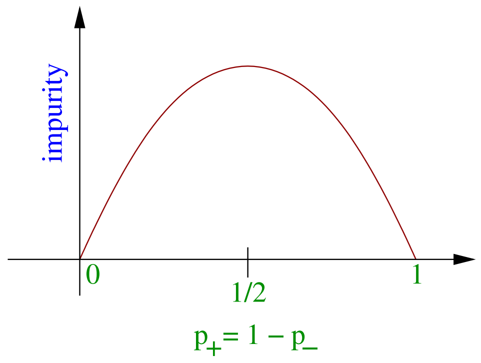
## Example (cont.)



## Choosing the Splitting Rule (cont.)

- (im)purity measures:
  - entropy: $-p_+ \ln p_+ - p_- \ln p_-$
  - Gini index: $p_+ p_-$

  where $p_+$ / $p_-$ = fraction of positive / negative examples



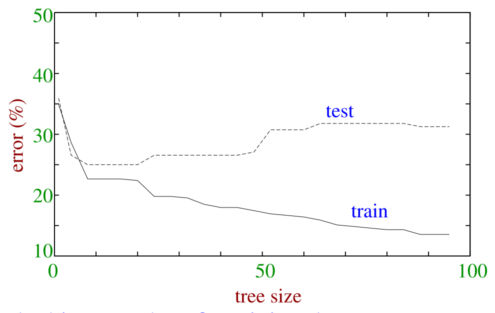
## How to Build Decision Trees

- choose rule to split on
- divide data using splitting rule into disjoint subsets
- repeat recursively for each subset
- stop when leaves are (almost) “pure”

## Kinds of Error Rates

- training error = fraction of training examples misclassified
- test error = fraction of test examples misclassified
- generalization error = probability of misclassifying new random example

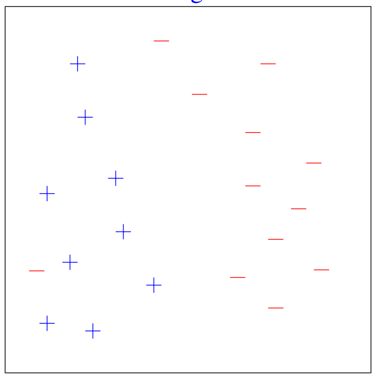
## Tree Size versus Accuracy



- trees must be big enough to fit training data (so that "true" patterns are fully captured)
- BUT: trees that are too big may overfit (capture noise or spurious patterns in the data)
- significant problem: can't tell best tree size from training error

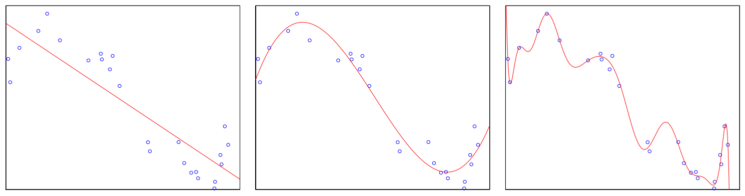
## Example

Training data:



## Overfitting Example

- fitting points with a polynomial



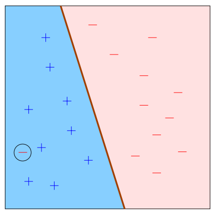
underfit (degree = 1)

ideal fit (degree = 3)

overfit (degree = 20)

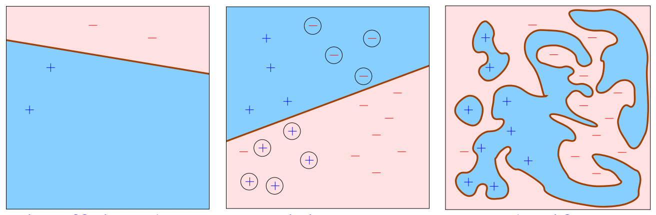
## Good and Bad Classifiers

Good:



sufficient data
low training error
simple classifier

Bad:



insufficient data

training error too high

classifier too complex

## Building an Accurate Classifier

- for good test peformance, need:
  - enough training examples
  - good performance on training set
  - classifier that is not too "complex" ("Occam's razor")
    - measure "complexity" by:
      - · number bits needed to write down
      - · number of parameters
      - · VC-dimension

## Theory

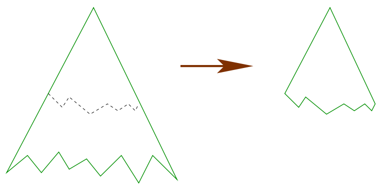
- can prove:

$$(\text{generalization error}) \leq (\text{training error}) + \tilde{O}\left(\sqrt{\frac{d}{m}}\right)$$

with high probability
  - $d$ = VC-dimension
  - $m$ = number training examples

## Controlling Tree Size

- typical approach: build very large tree that fully fits training data, then prune back



- pruning strategies:
  - grow on just part of training data, then find pruning with minimum error on held out part
  - find pruning that minimizes

$$(\text{training error}) + \text{constant} \cdot (\text{tree size})$$

## Example: Spam Filtering

- problem: filter out spam (junk email)
- gather large collection of examples of spam and non-spam:

| | | |
|---|---|---|
| From: yoav@att.com | Rob, can you review a paper... | non-spam |
| From: xa412@hotmail.com | Earn money without working!!!! ... | spam |
| ⋮ | ⋮ | ⋮ |

- main observation:
  - easy to find "rules of thumb" that are "often" correct
    - *If* 'buy now' *occurs in message, then predict* 'spam'
  - hard to find single rule that is very highly accurate

## Decision Trees

- best known:
  - C4.5 (Quinlan)
  - CART (Breiman, Friedman, Olshen & Stone)
- very fast to train and evaluate
- relatively easy to interpret
- but: accuracy often not state-of-the-art

## The Boosting Approach

- devise computer program for deriving rough rules of thumb
- apply procedure to subset of emails
- obtain rule of thumb
- apply to 2nd subset of emails
- obtain 2nd rule of thumb
- repeat $T$ times

## Boosting

## Details

- how to choose examples on each round?
  - concentrate on "hardest" examples (those most often misclassified by previous rules of thumb)
- how to combine rules of thumb into single prediction rule?
  - take (weighted) majority vote of rules of thumb

## Boosting

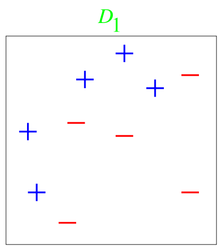
- boosting = general method of converting rough rules of thumb into highly accurate prediction rule
- technically:
  - assume given "weak" learning algorithm that can consistently find classifiers ("rules of thumb") at least slightly better than random, say, accuracy $\geq 55\%$
  - given sufficient data, a boosting algorithm can provably construct single classifier with very high accuracy, say, 99%

## AdaBoost

- given training examples $(x_i, y_i)$ where $y_i \in \{-1, +1\}$
- initialize $D_1$ = uniform distribution on training examples
- for $t = 1, \ldots, T$:
  - train weak classifier ("rule of thumb") $h_t$ on $D_t$
  - choose $\alpha_t > 0$
  - compute new distribution $D_{t+1}$:
    - for each example $i$:

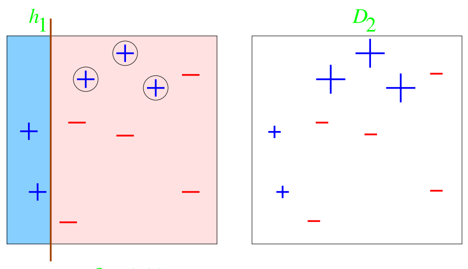
      multiply $D_t(x_i)$ by $\begin{cases} e^{-\alpha_t} & (< 1) \text{ if } y_i = h_t(x_i) \\ e^{\alpha_t} & (> 1) \text{ if } y_i \neq h_t(x_i) \end{cases}$
    - renormalize
- output final classifier $H_{\text{final}}(x) = \text{sign}\left(\sum_t \alpha_t h_t(x)\right)$

## Toy Example



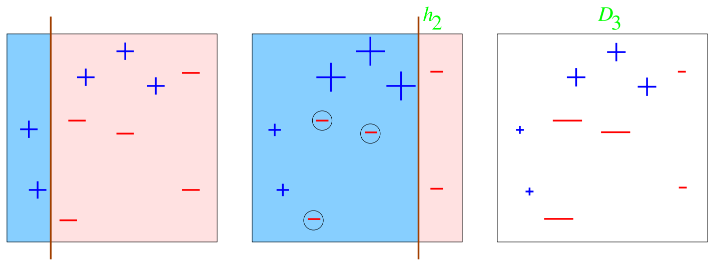
weak classifiers = vertical or horizontal half-planes

## Round 1



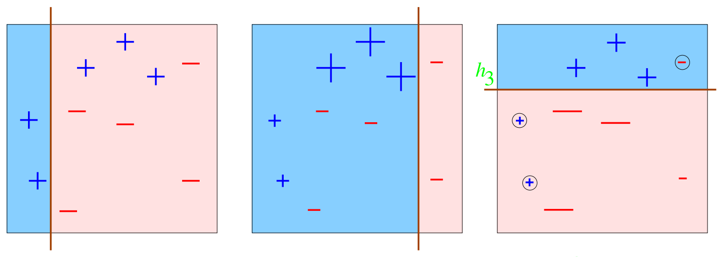
$\varepsilon_1 = 0.30$
$\alpha_1 = 0.42$

## Round 2



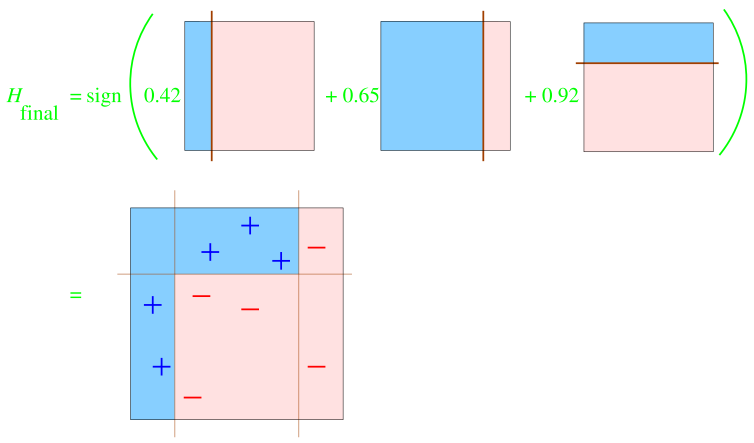
$\varepsilon_2 = 0.21$
$\alpha_2 = 0.65$

## Round 3



$\varepsilon_3 = 0.14$
$\alpha_3 = 0.92$

## Final Classifier

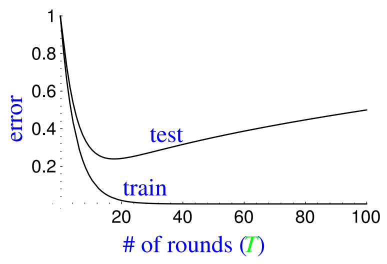


## Theory: Training Error

- weak learning assumption: each weak classifier at least slightly better than random
  - i.e., (error of $h_t$ on $D_t$) $\le 1/2 - \gamma$ for some $\gamma > 0$
- given this assumption, can prove:

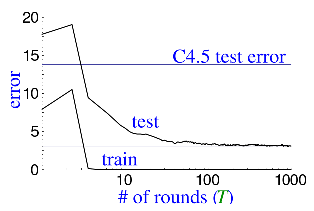
$$\text{training error}(H_{\text{final}}) \le e^{-2\gamma^2 T}$$

## How Will Test Error Behave? (A First Guess)



- expect:
  - training error to continue to drop (or reach zero)
  - test error to increase when $H_{\text{final}}$ becomes "too complex" (overfitting)

## Actual Typical Run



(boosting C4.5 on "letter" dataset)

- test error does not increase, even after 1000 rounds
  - (total size > 2,000,000 nodes)
- test error continues to drop even after training error is zero!

| | # rounds | | |
|---|---|---|---|
| | 5 | 100 | 1000 |
| train error | 0.0 | 0.0 | 0.0 |
| test error | 8.4 | 3.3 | 3.1 |

## The Margins Explanation

- key idea:
  - training error only measures whether classifications are right or wrong
  - should also consider confidence of classifications
- recall: $H_{\text{final}}$ is weighted majority vote of weak classifiers
- measure confidence by margin = strength of the vote
- empirical evidence and mathematical proof that:
  - large margins $\Rightarrow$ better generalization error (regardless of number of rounds)
  - boosting tends to increase margins of training examples (given weak learning assumption)

## Boosting

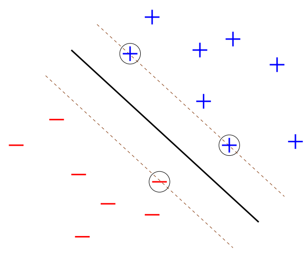
- fast (but not quite as fast as other methods)
- simple and easy to program
- flexible: can combine with any learning algorithm, e.g.
  - C4.5
  - very simple rules of thumb
- provable guarantees
- state-of-the-art accuracy
- tends not to overfit (but occasionally does)
- many applications

## Support-Vector Machines

## Finding the Maximum Margin Hyperplane

- examples $\mathbf{x}_i, y_i$ where $y_i \in \{-1, +1\}$
- find hyperplane $\mathbf{w} \cdot \mathbf{x} = 0$ with $\| \mathbf{w} \| = 1$
- margin $= y(\mathbf{w} \cdot \mathbf{x})$
- maximize: $\gamma$
  subject to: $y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq \gamma$ and $\| \mathbf{w} \| = 1$
- set $\mathbf{w} \leftarrow \mathbf{w}/\gamma \Rightarrow \gamma = 1/ \| \mathbf{w} \|$
- minimize $\frac{1}{2} \| \mathbf{w} \|^2$
  subject to: $y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1$

## Geometry of SVM's



- given linearly separable data
- margin = distance to separating hyperplane
- choose hyperplane that maximizes minimum margin
- intuitively:
  - want to separate $+$'s from $-$'s as much as possible
  - margin = measure of confidence

## Convex Dual

- form Lagrangian, set $\partial / \partial \mathbf{w} = 0$
- get quadratic program:
- maximize $\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$
  subject to: $\alpha_i \geq 0$
- $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
- $\alpha_i =$ Lagrange multiplier
  $> 0 \Leftrightarrow$ support vector
- key points:
  - optimal $\mathbf{w}$ is linear combination of support vectors
  - dependence on $\mathbf{x}_i$'s only through inner products
  - maximization problem is convex with no local maxima

## Theoretical Justification

- let $\gamma =$ minimum margin
  $R =$ radius of enclosing sphere
- then

$$\text{VC-dim} \leq \left( \frac{R}{\gamma} \right)^2$$

  - so larger margins $\Rightarrow$ lower "complexity"
  - independent of number of dimensions
- in contrast, unconstrained hyperplanes in $\mathbb{R}^n$ have

$$\text{VC-dim} = (\text{\# parameters}) = n + 1$$

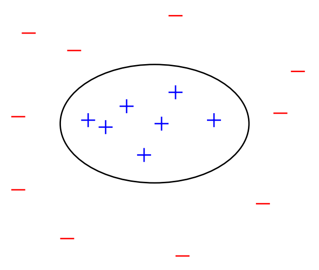
## What If Not Linearly Separable?

- answer #1: penalize each point by distance from margin $1$, i.e., minimize:

$$\frac{1}{2} \| \mathbf{w} \|^2 + \text{constant} \cdot \sum_i \max\{0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i)\}$$

- answer #2: map into higher dimensional space in which data becomes linearly separable

## Example



- not linearly separable
- map $\mathbf{x} = (x_1, x_2) \mapsto \Phi(\mathbf{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$
- hyperplane in mapped space has form
$$a + bx_1 + cx_2 + dx_1x_2 + ex_1^2 + fx_2^2 = 0$$
  $=$ conic in original space
- linearly separable in mapped space

## Kernels

- kernel = function $K$ for computing
$$K(\mathbf{x}, \mathbf{z}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{z})$$
- permits efficient computation of SVM's in very high dimensions
- $K$ can be any symmetric, positive semi-definite function (Mercer's theorem)
- some kernels:
  - polynomials
  - Gaussian $\exp\left(- \|\mathbf{x} - \mathbf{z}\|^2 / 2\sigma\right)$
  - defined over structures (trees, strings, sequences, etc.)
- evaluation:
$$\mathbf{w} \cdot \Phi(\mathbf{x}) = \Sigma\, \alpha_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) = \Sigma\, \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})$$
  - time depends on # support vectors

## Higher Dimensions Don't (Necessarily) Hurt

- may project to very high dimensional space
- statistically, may not hurt since VC-dimension independent of number of dimensions ($(R/\gamma)^2$)
- computationally, only need to be able to compute inner products
$$\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z})$$
  - sometimes can do very efficiently using kernels

## SVM's versus boosting

- both are large-margin classifiers
(although with slightly different definitions of margin)
- both work in very high dimensional spaces
(in boosting, dimensions correspond to weak classifiers)
- but different tricks are used:
  - SVM's use kernel trick
  - boosting relies on weak learner to select one dimension (i.e., weak classifier) to add to combined classifier

## Example (cont.)

- modify $\Phi$ slightly:
$$\Phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$$
- then
$$\begin{aligned} \Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) &= 1 + 2x_1z_1 + 2x_2z_2 + 2x_1x_2z_1z_2 + x_1^2z_1^2 + x_2^2 + z_2^2 \\ &= (1 + x_1z_1 + x_2z_2)^2 \\ &= (1 + \mathbf{x} \cdot \mathbf{z})^2 \end{aligned}$$
- in general, for polynomial of degree $d$, use $(1 + \mathbf{x} \cdot \mathbf{z})^d$
- very efficient, even though finding hyperplane in $O(n^d)$ dimensions

## SVM's

- fast algorithms now available, but not so simple to program
(but good packages available)
- state-of-the-art accuracy
- power and flexibility from kernels
- theoretical justification
- many applications

**Further reading on machine learning in general:**

Luc Devroye, Lázló Györfi and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, 1996.

Richard O. Duda, Peter E. Hart and David G. Stork. *Pattern Classification (2nd ed.)*. Wiley, 2000.

Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The Elements of Statistical Learning : Data Mining, Inference, and Prediction*. Springer, 2001.

Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

**Decision trees:**

Leo Breiman, Jerome H. Friedman, Richard A. Olshen and Charles J. Stone. *Classification and Regression Trees*. Wadsworth & Brooks, 1984.

J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

**Boosting:**

Robert E. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002. Available from: www.research.att.com/$\sim$schapire/boost.html.

Many more papers, tutorials, etc. available at www.boosting.org.

**Support-vector machines:**

Nello Cristianni and John Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000. See www.support-vector.net.

Many more papers, tutorials, etc. available at www.kernel-machines.org.