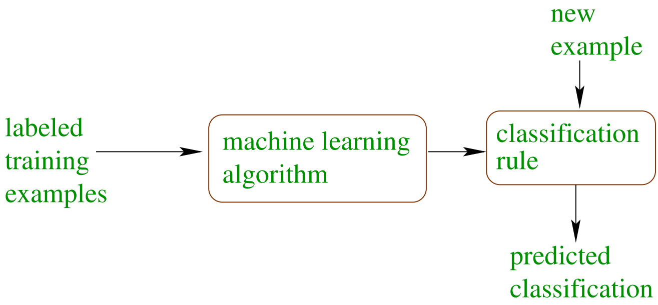
Decision Trees, Boosting, Random Forests

Rob Schapire
Microsoft Research (NYC)

Machine Learning

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



Examples of Classification Problems

- text categorization (e.g., spam filtering)
- topic classification of web pages, news articles, etc.
- fraud/abuse detection
- machine vision (e.g., face detection)
- natural-language processing
  (e.g., part-of-speech tagging)
- scientific applications
  (e.g., classify proteins according to their function)

  $\vdots$

Characteristics of Modern Machine Learning

- primary goal: highly accurate predictions on test data
  - uncovering underlying "truth" usually only secondary
- methods should be general purpose, fully automatic and "off-the-shelf"
  - however, in practice, incorporation of prior, human knowledge can be crucial
- rich interplay between theory and practice
- emphasis on methods that can handle large datasets

This Lecture

- machine learning algorithms for classification:
  - decision trees
  - boosting
  - random forests
- along the way...
  - fundamental conditions for successful learning

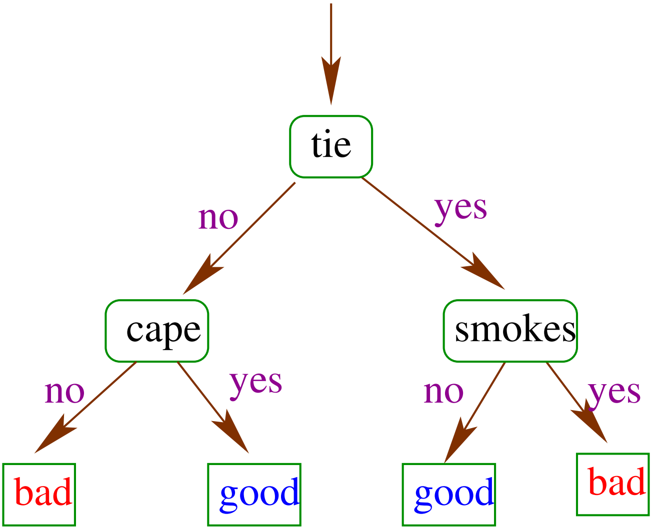
Decision Trees

Example: Good versus Evil

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

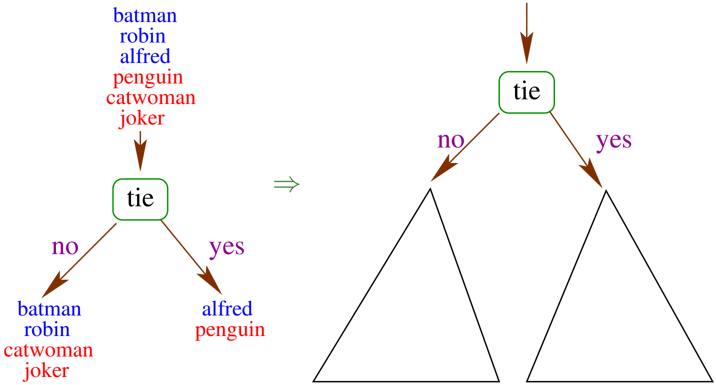
	features / attributes / dimensions						*class / label*
	sex	mask	cape	tie	ears	smokes	
	training data						
batman	male	yes	yes	no	yes	no	Good
robin	male	yes	yes	no	no	no	Good
alfred	male	no	no	yes	no	no	Good
penguin	male	no	no	yes	no	yes	Bad
catwoman	female	yes	no	no	yes	no	Bad
joker	male	no	no	no	no	no	Bad
	test data						
batgirl	female	yes	yes	no	yes	no	??
riddler	male	yes	no	no	no	no	??

A Decision Tree Classifier



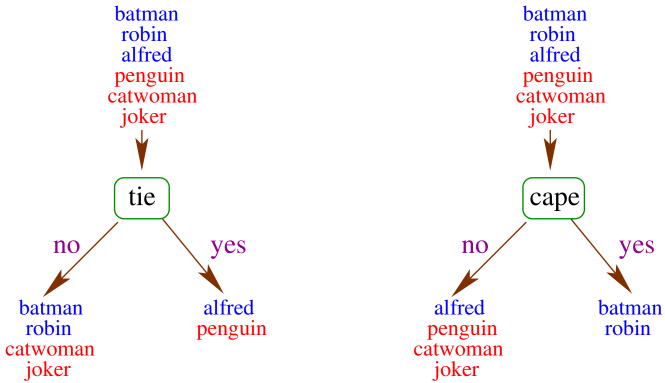
How to Build Decision Trees

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



How to Choose the Splitting Rule

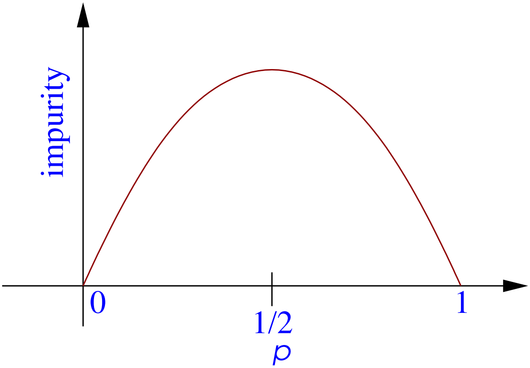
- key problem: choosing best rule to split on:



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

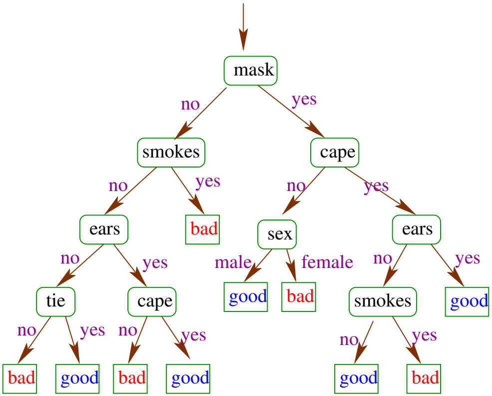
How to Measure Purity

- want (im)purity function to look like this:
  (p = fraction of positive examples)



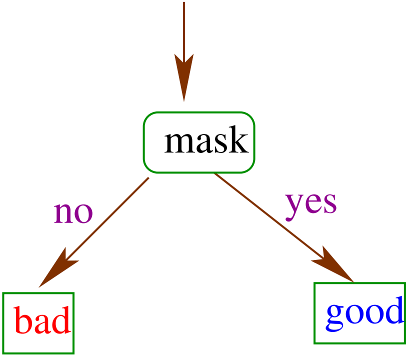
- commonly used impurity measures:
  - entropy: $-p \ln p - (1-p)\ln(1-p)$
  - Gini index: $p(1-p)$

A Possible Classifier



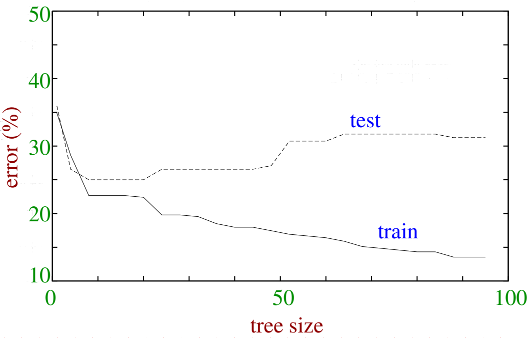
- perfectly classifies training data
- BUT: intuitively, overly complex

Another Possible Classifier



- overly simple
- doesn't even fit available data

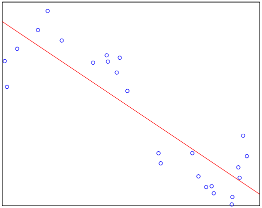
Tree Size versus Accuracy



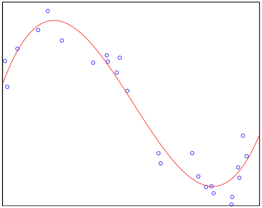
- trees must be big enough to fit training data (so that "true" patterns are fully captured)
- BUT: trees that are too big may overfit (capture noise or spurious patterns in the data)
- significant problem: can't tell best tree size from training error

Overfitting Example

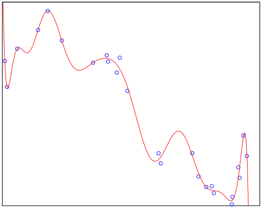
- fitting points with a polynomial



underfit
(degree $= 1$)



ideal fit
(degree $= 3$)



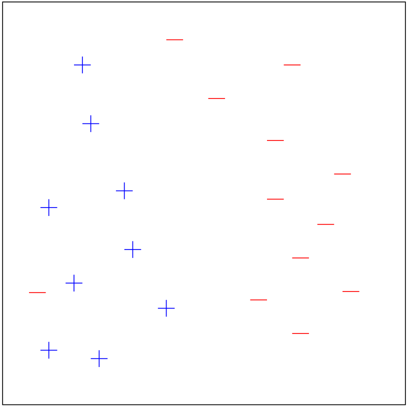
overfit
(degree $= 20$)

Building an Accurate Classifier

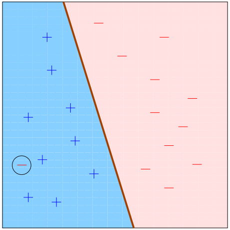
- for good test peformance, need:
  - enough training examples
  - good performance on training set
  - classifier that is not too "complex" ("Occam's razor")
- classifiers should be "as simple as possible, but no simpler"
- "simplicity" closely related to prior expectations
- measure "complexity" by:
  - number bits needed to write down
  - number of parameters
  - VC-dimension

Example

Training data:

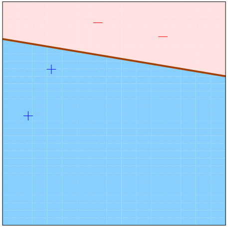


Good and Bad Classifiers

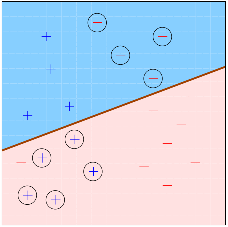
Good: 

sufficient data
low training error
simple classifier

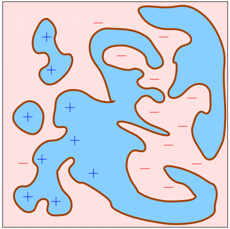
Bad:



insufficient data



training error too high



classifier too complex

Theory

- can prove:

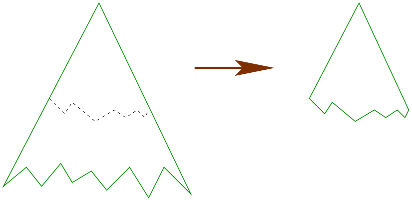
$$(\text{generalization error}) \leq (\text{training error}) + \tilde{O}\left(\sqrt{\frac{d}{m}}\right)$$

  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

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

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

Boosting

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
⋮	⋮	⋮

- goal: have computer learn from examples to distinguish spam from non-spam
- main observation:
  - easy to find "rules of thumb" that are "often" correct
    - *If 'viagra' occurs in message, then predict 'spam'*
  - hard to find single rule that is very highly accurate

The Boosting Approach

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

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%

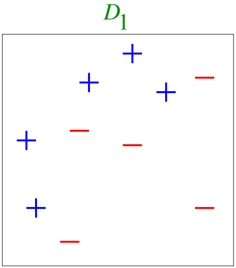
AdaBoost

[with Freund]

- 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 $\alpha_t =$ [some formula] > 0
  - compute new distribution D_{t+1}:
    - for each example i:

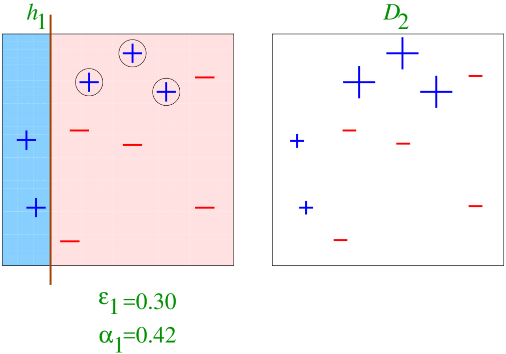
      multiply $D_t(x_i)$ by $\begin{cases} e^{-\alpha_t} & (< 1) \quad \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & (> 1) \quad \text{if } y_i \neq h_t(x_i) \end{cases}$
    - renormalize
- output final classifier $H_{\text{final}}(x) = \operatorname{sign}\left(\sum_t \alpha_t h_t(x)\right)$

Toy Example

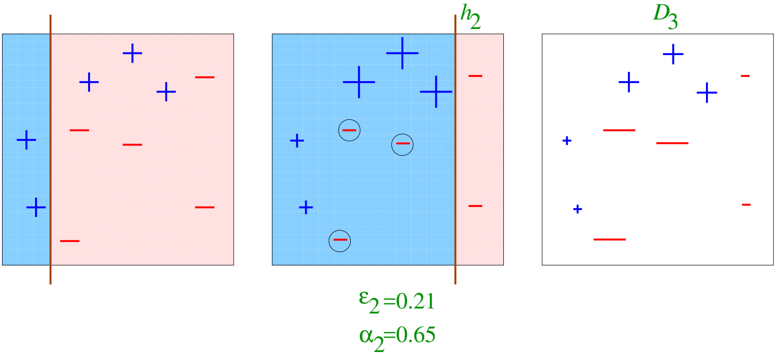


weak classifiers = vertical or horizontal half-planes

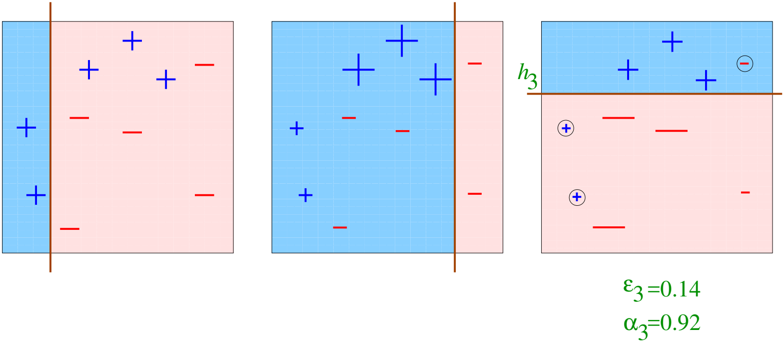
Round 1



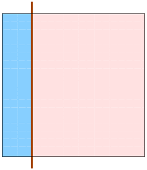
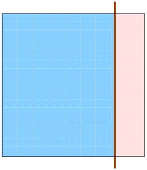
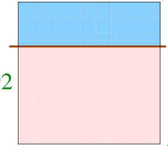
Round 2

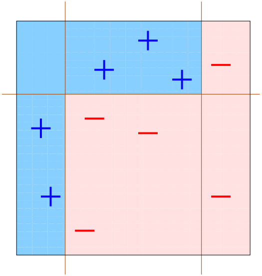


Round 3



Final Classifier

$H_{\text{final}} = \text{sign}\Big(0.42$  $+ 0.65$  $+ 0.92$  $\Big)$

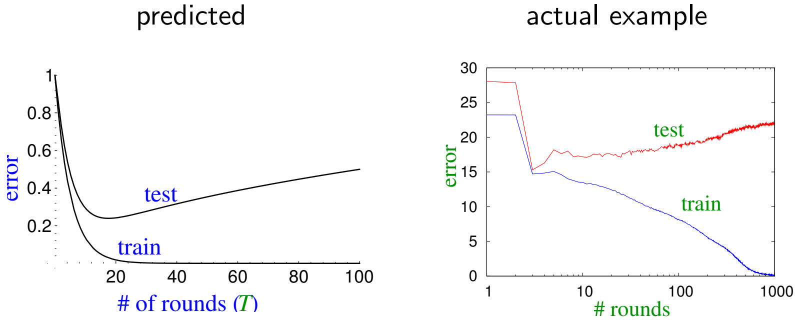
$=$ 

Theory: Training Error

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

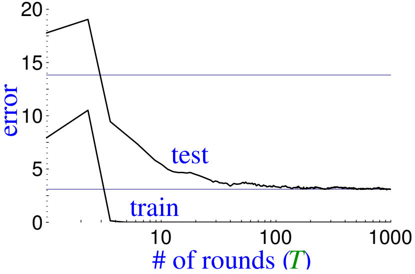
$$\text{training error}(H_{\text{final}}) \leq e^{-2\gamma^2 T}$$

Predicted Behavior



- complexity increases with # rounds ⇒ overfitting
- can happen
- but often doesn't...

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

- Occam's razor wrongly predicts "simpler" rule is better

The Margins Explanation

[with Freund, Bartlett & Lee]

- key idea:
  - training error only measures whether classifications are right or wrong
  - should also consider confidence of classifications
- recall: H_{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)

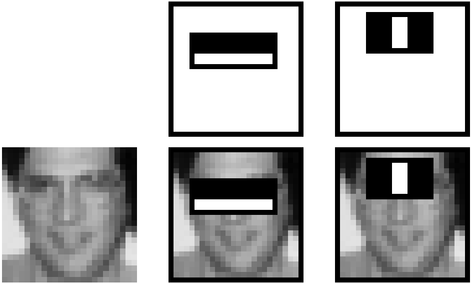
Decision Trees as a Weak Learner

- what to use for weak learning algorithm?
- option #1: use any off-the-shelf learning algorithm
- decision trees work especially well with boosting
- e.g.: [Caruana & Niculescu-Mizil] compared many learning algorithms, datasets, evaluation metrics
  - boosted decision trees best overall
  - close competitors all based on "ensembles" of decision trees
- option #2: design special-purpose weak learner...

Application: Detecting Faces

[Viola & Jones]

- problem: find faces in photograph or movie
- weak classifiers: detect light/dark rectangles in image



- many clever tricks to make extremely fast and accurate

AdaBoost and Loss Minimization

- how to use boosting for learning problems other than classification?
- to answer: first understand in terms of loss minimization
  - loss = function measuring fit to data
  - learning problems often associated with particular loss
- helpful to understand because:
  - clarifies goal of algorithm and useful in proving convergence properties
  - decoupling of algorithm from its objective means:
    - faster algorithms possible for same objective
    - same algorithm may generalize for new learning challenges

Exponential Loss

- AdaBoost is greedy procedure for minimizing exponential loss

$$\frac{1}{m}\sum_i \exp(-y_i F(x_i))$$

where

$$F(x) = \sum_t \alpha_t h_t(x)$$

- why exponential loss?
  - intuitively, strongly favors $F(x_i)$ to have same sign as y_i
  - upper bound ("surrogate") for training error
    - smooth and convex (but very loose)
- minimize using:
  - coordinate descent [Breiman]
  - functional gradient descent / gradient boosting...

    [Mason, Baxter, Bartlett, Frean][Friedman]

Functional Gradient Descent / Gradient Boosting

[Mason, Baxter, Bartlett, Frean][Friedman]

- want to minimize

$$\mathcal{L}(F) = \mathcal{L}(F(x_1), \ldots, F(x_m)) = \sum_i \exp(-y_i F(x_i))$$

- say have current estimate F and want to improve
- to do gradient descent, would like update

$$F \leftarrow F - \alpha \nabla_F \mathcal{L}(F)$$

- but update restricted in class of weak classifiers

$$F \leftarrow F + \alpha h_t$$

- so choose h_t "closest" to $-\nabla_F \mathcal{L}(F)$

Gradient Boosting (cont.)

- for exponential loss, equivalent to AdaBoost
- applied to other losses, get generalized algorithm for other learning problems
  - e.g.: for regression (predict real-valued labels): use square loss $(y - F(x))^2$
- overall algorithm is same:
  - repeatedly find classifiers/predictors (e.g. decision trees) on weighted versions of dataset
  - combine with voting or averaging
- main change: how weights on training examples are computed

Boosting

- fast
- simple and easy to program
- flexible — can combine with any learning algorithm
- provable guarantees
- often state-of-the-art accuracy
- tends not to overfit (but occasionally does)
- generalizes to other learning problems

Random Forests

Ensembles of Decision Trees

- boosting decision trees yields ensemble (“forest”) of trees
- basic idea:
  - build many trees
  - combine
- other tree ensemble methods:
  - bagging and random forests [Breiman]
  - use explicit randomness in how trees trained to encourage diversity

Random Forests

[Breiman]

- given m training examples
- repeat
  - build tree as follows:
    - train on "bootstrap" sample – m uniformly random examples from training set selected with replacement
    - only use k randomly chosen features
    - grow to maximum size (no pruning)
- combine trees by voting / averaging
- idea: balance
  - fit to training data
  - "diversity"

Random Forests (cont.)

- overall performance comparable to boosted decision trees
- does not overfit
- easy to parallelize
- can get "automatic" accuracy estimates using "out-of-bag" samples ($\approx 1/3$ of data not used to train each tree)
- less general than boosting since specialized to trees

Summary

- central issues in machine learning:
  - avoidance of overfitting
  - balance between simplicity and fit to data
- machine learning algorithms:
  - decision trees
  - boosting (and gradient boosting)
  - random forests
- boosted decision trees and random forests:
  - simple, fast, general purpose
  - often state-of-the art

Further reading:

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.

Robert E. Schapire and Yoav Freund. *Boosting: Foundations and algorithms*, MIT Press, 2012.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

Other works cited:

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Leo Breiman. Prediction games and arcing classifiers. *Neural Computation*, 11(7):1493–1517, 1999.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 161–168, 2006.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, October 2001.

Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Frean. Functional gradient techniques for combining hypotheses. In *Advances in Large Margin Classifiers*, pages 221–246. MIT Press, 2000.

Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, October 1998.

Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 511–518, 2001.