Trees, Bagging, Random Forests and Boosting

- Classification Trees
- Bagging: Averaging Trees
- Random Forests: Cleverer Averaging of Trees
- Boosting: Cleverest Averaging of Trees

Methods for improving the performance of weak learners such as Trees. Classification trees are adaptive and robust, but do not generalize well. The techniques discussed here enhance their performance considerably.

Two-class Classification

- Observations are classified into two or more classes, coded by a response variable Y taking values 1, 2, $\ldots, K$.

- We have a feature vector $X = (X_1, X_2, \ldots, X_p)$, and we hope to build a classification rule $C(X)$ to assign a class label to an individual with feature X.

- We have a sample of pairs (y_i, x_i), $i = 1, \ldots, N$. Note that each of the x_i are vectors $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$.

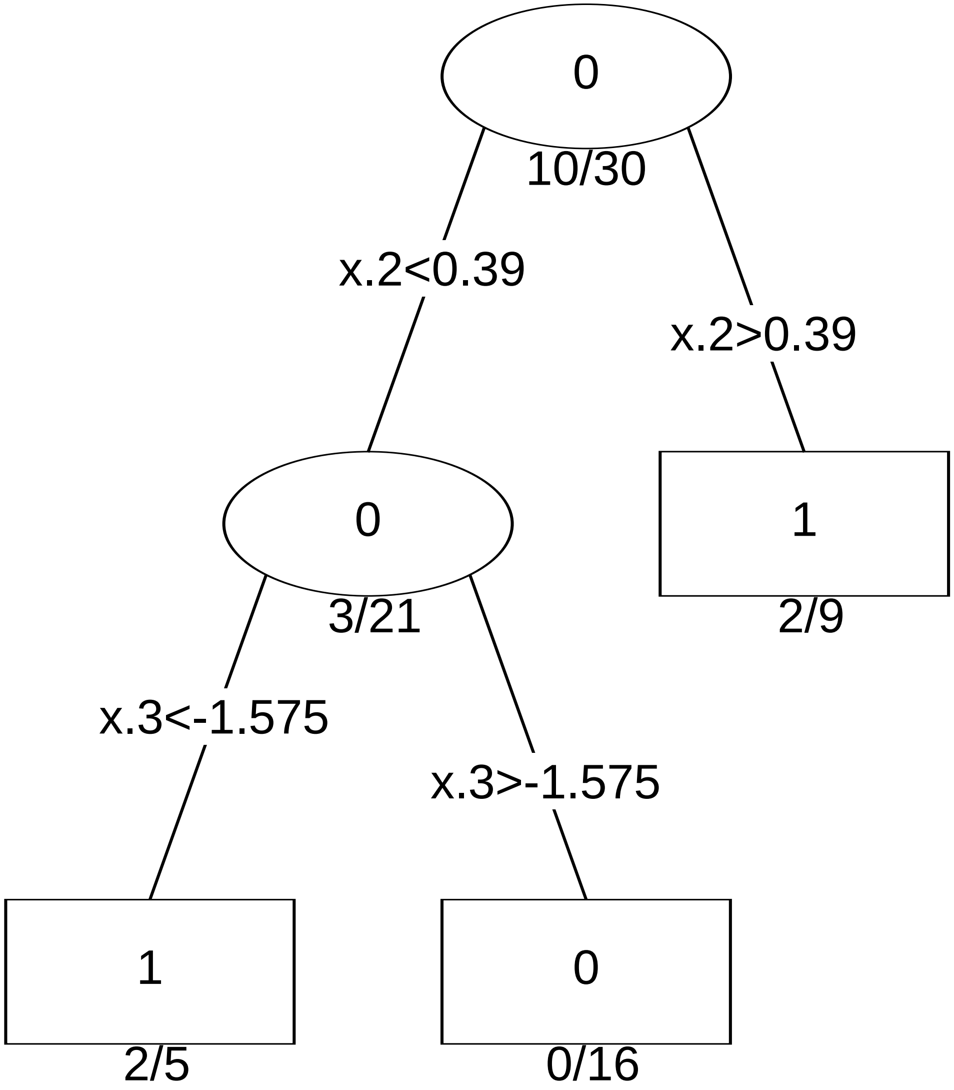
- Example: Y indicates whether an email is spam or not. X represents the relative frequency of a subset of specially chosen words in the email message.

- The technology described here estimates $C(X)$ directly, or via the probability function $P(C = k|X)$.

Classification Trees

- Represented by a series of binary splits.
- Each internal node represents a value query on one of the variables — e.g. "Is $X_3 > 0.4$". If the answer is "Yes", go right, else go left.
- The terminal nodes are the decision nodes. Typically each terminal node is dominated by one of the classes.
- The tree is grown using training data, by recursive splitting.
- The tree is often pruned to an optimal size, evaluated by cross-validation.
- New observations are classified by passing their X down to a terminal node of the tree, and then using majority vote.

Classification Tree



Properties of Trees

- ✔ Can handle huge datasets
- ✔ Can handle mixed predictors—quantitative and qualitative
- ✔ Easily ignore redundant variables
- ✔ Handle missing data elegantly
- ✔ Small trees are easy to interpret
- ✘ large trees are hard to interpret
- ✘ Often prediction performance is poor

Example: Predicting e-mail spam

- data from 4601 email messages
- goal: predict whether an email message is spam (junk email) or good.
- input features: relative frequencies in a message of 57 of the most commonly occurring words and punctuation marks in all the training the email messages.
- for this problem not all errors are equal; we want to avoid filtering out good email, while letting spam get through is not desirable but less serious in its consequences.
- we coded `spam` as 1 and `email` as 0.
- A system like this would be trained for each user separately (e.g. their word lists would be different)

Predictors

- 48 quantitative predictors—the percentage of words in the email that match a given word. Examples include `business`, `address`, `internet`, `free`, and `george`. The idea was that these could be customized for individual users.
- 6 quantitative predictors—the percentage of characters in the email that match a given character. The characters are `ch;`, `ch(`, `ch[`, `ch!`, `ch$`, and `ch#`.
- The average length of uninterrupted sequences of capital letters: `CAPAVE`.
- The length of the longest uninterrupted sequence of capital letters: `CAPMAX`.
- The sum of the length of uninterrupted sequences of capital letters: `CAPTOT`.

Details

- A test set of size 1536 was randomly chosen, leaving 3065 observations in the training set.
- A full tree was grown on the training set, with splitting continuing until a minimum bucket size of 5 was reached.
- This bushy tree was pruned back using cost-complexity pruning, and the tree size was chosen by 10-fold cross-validation.
- We then compute the test error and ROC curve on the test data.

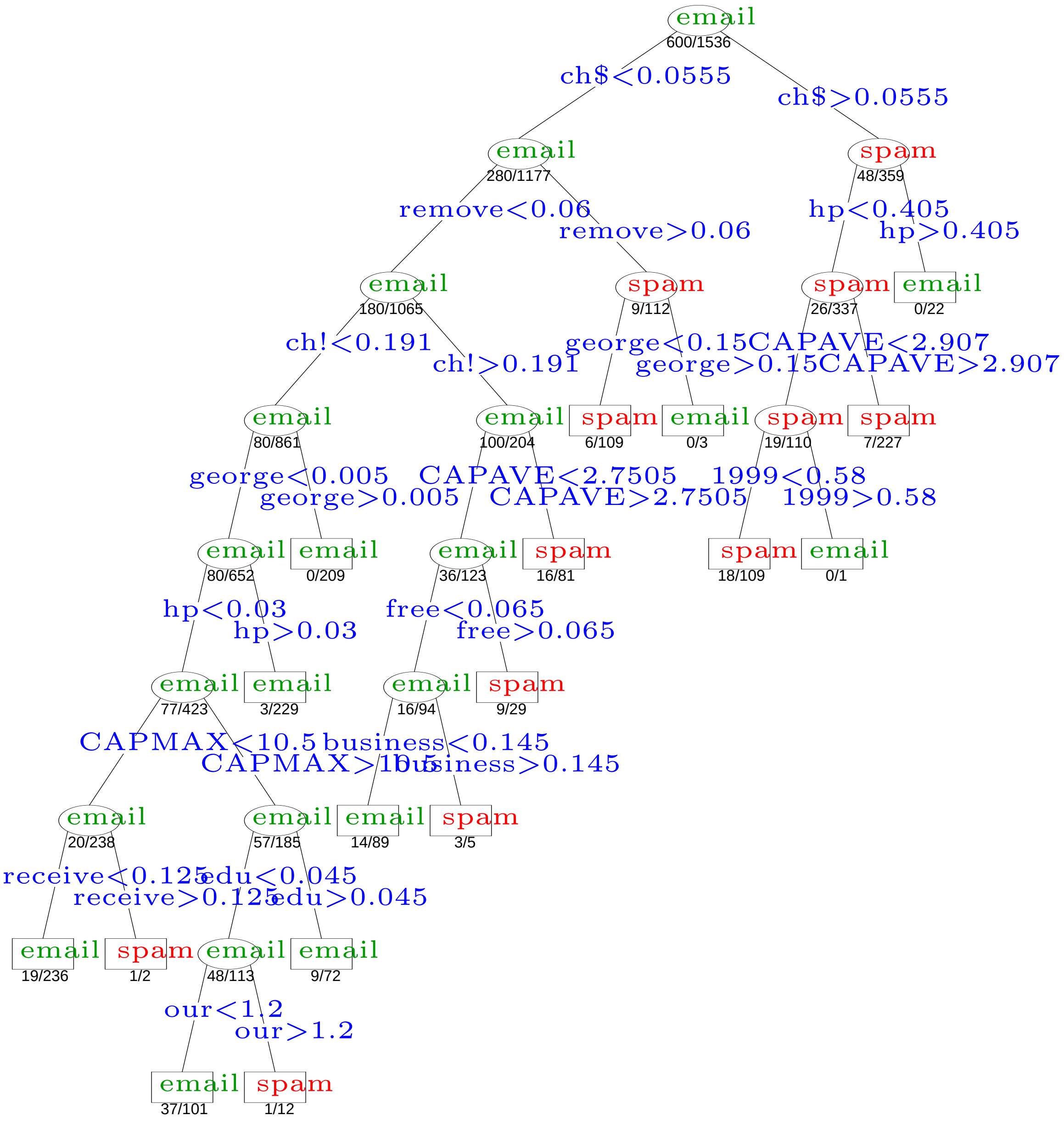
Some important features

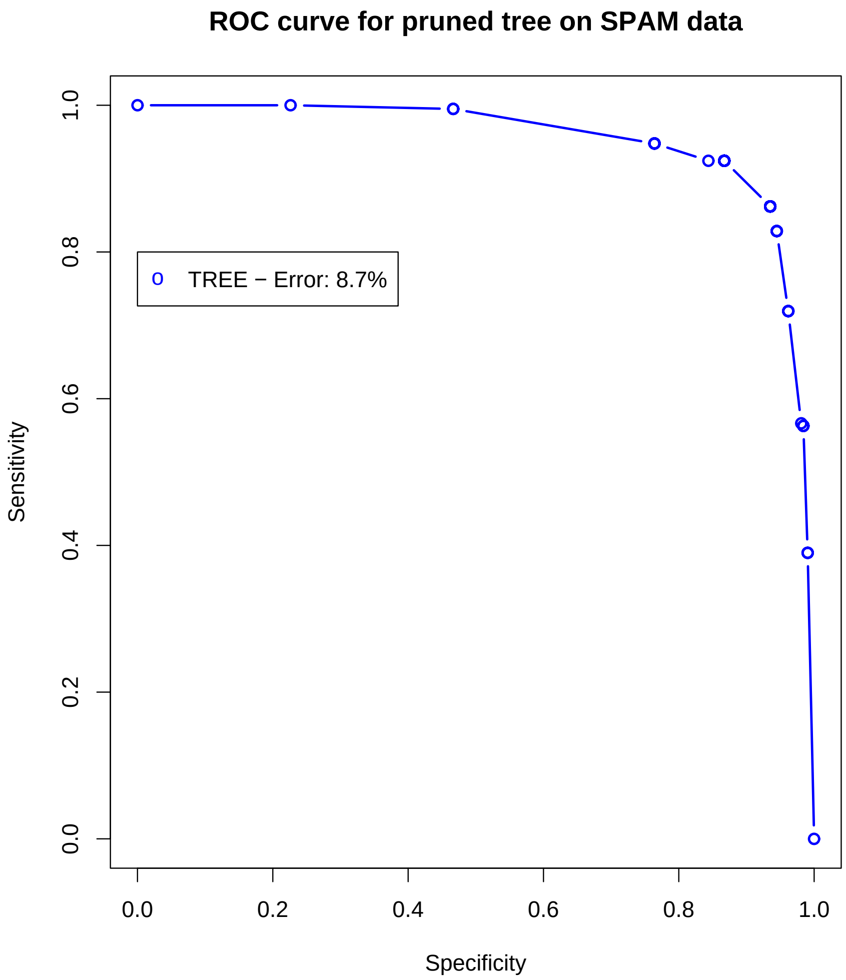
39% of the training data were spam.

Average percentage of words or characters in an email message equal to the indicated word or character. We have chosen the words and characters showing the largest difference between spam and email.

	george	you	your	hp	free	hpl
spam	0.00	2.26	1.38	0.02	0.52	0.01
email	1.27	1.27	0.44	0.90	0.07	0.43

	!	our	re	edu	remove
spam	0.51	0.51	0.13	0.01	0.28
email	0.11	0.18	0.42	0.29	0.01


email
600/1536
ch$<0.0555
ch$>0.0555
email
280/1177
spam
48/359
remove<0.06
remove>0.06
hp<0.405
hp>0.405
email
180/1065
spam
9/112
spam
26/337
email
0/22
ch!<0.191
ch!>0.191
george<0.15
george>0.15
CAPAVE<2.907
CAPAVE>2.907
email
80/861
email
100/204
spam
6/109
email
0/3
spam
19/110
spam
7/227
george<0.005
george>0.005
CAPAVE<2.7505
CAPAVE>2.7505
1999<0.58
1999>0.58
email
80/652
email
0/209
email
36/123
spam
16/81
spam
18/109
email
0/1
hp<0.03
hp>0.03
free<0.065
free>0.065
email
77/423
email
3/229
email
16/94
spam
9/29
CAPMAX<10.5
CAPMAX>10.5
business<0.145
business>0.145
email
20/238
email
57/185
email
14/89
spam
3/5
receive<0.125
receive>0.125
edu<0.045
edu>0.045
email
19/236
spam
1/2
email
48/113
email
9/72
our<1.2
our>1.2
email
37/101
spam
1/12



SPAM Data

Overall error rate on test data: 8.7%.

ROC curve obtained by varying the threshold c_0 of the classifier:

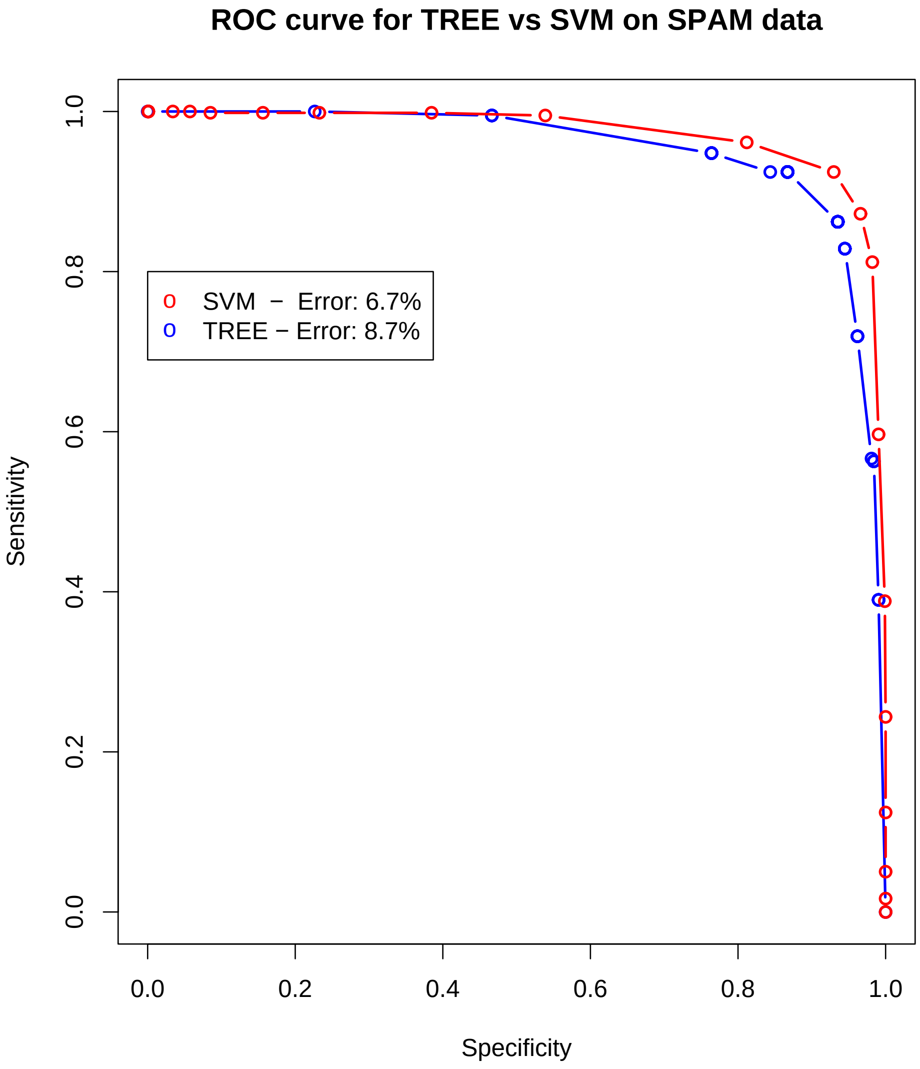
$C(X) = +1$ if $\hat{P}(+1|X) > c_0$.

Sensitivity: proportion of true spam identified

Specificity: proportion of true email identified.

We may want specificity to be high, and suffer some spam:

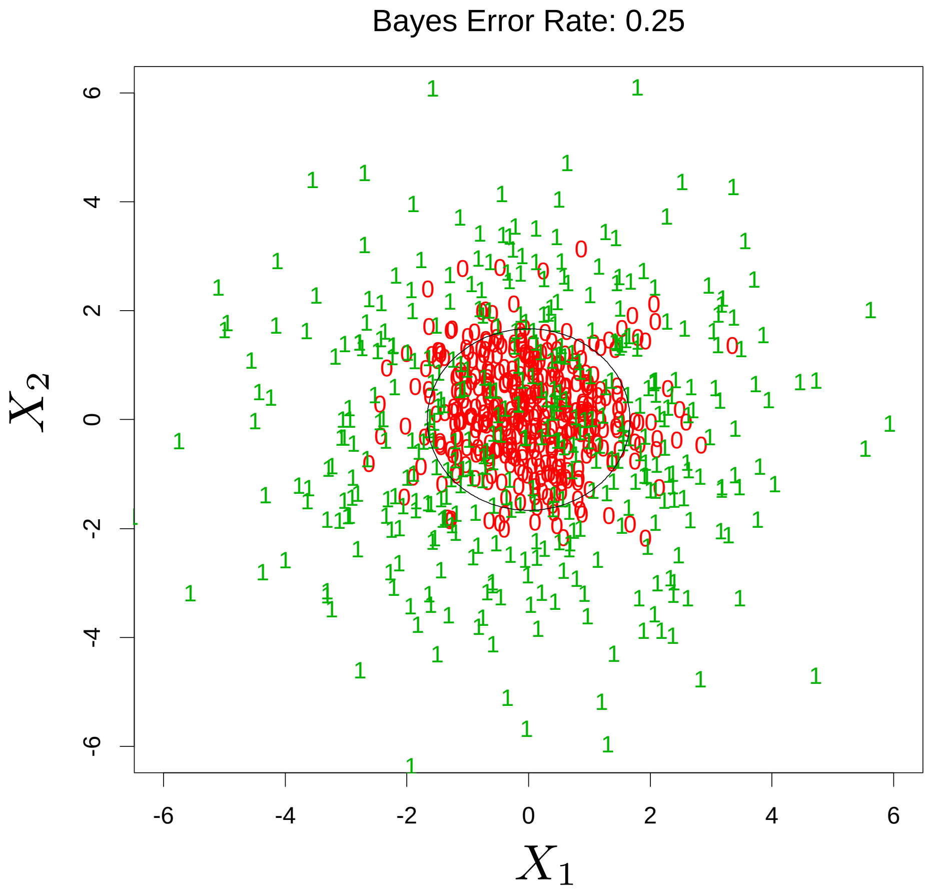
$\text{Specificity} : 95\% \implies \text{Sensitivity} : 79\%$



TREE vs SVM

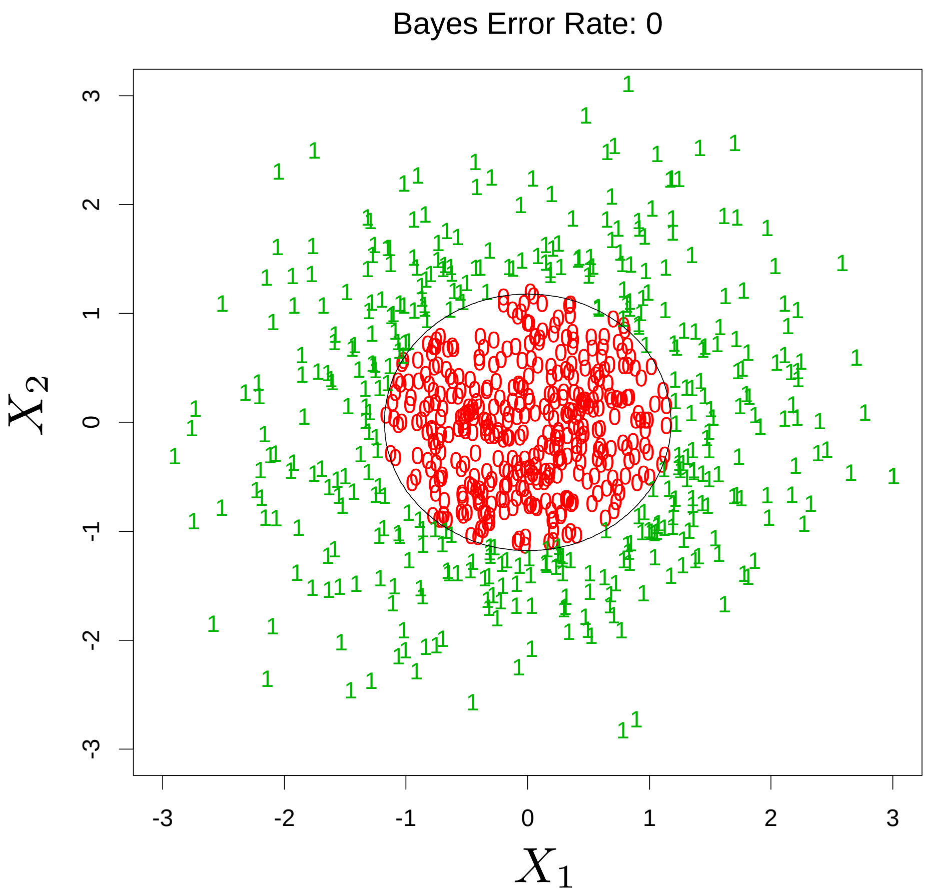
Comparing ROC curves on the test data is a good way to compare classifiers. SVM dominates TREE here.

Toy Classification Problem


Bayes Error Rate: 0.25

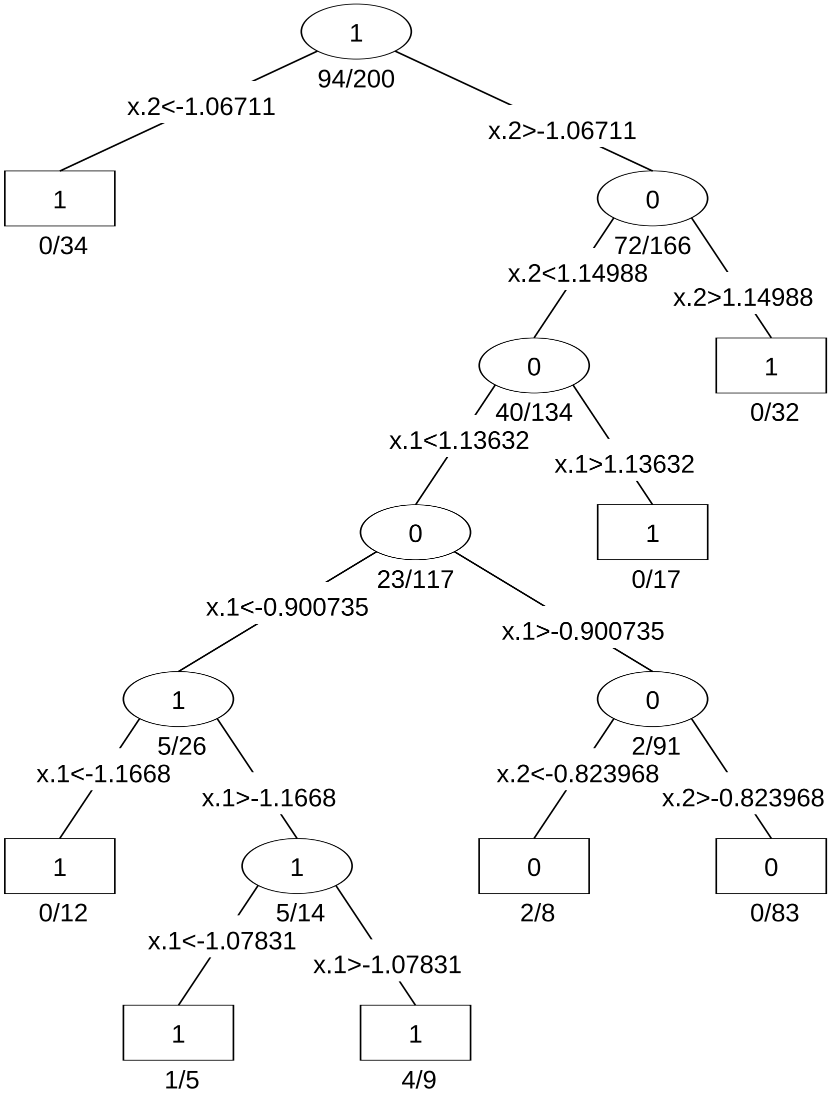
- Data X and Y, with Y taking values $+1$ or -1.
- Here $X = (X_1, X_2)$
- The black boundary is the Bayes Decision Boundary - the best one can do.
- Goal: Given N training pairs (X_i, Y_i) produce a classifier $\hat{C}(X) \in \{-1, 1\}$
- Also estimate the probability of the class labels $P(Y = +1|X)$.

Toy Example - No Noise

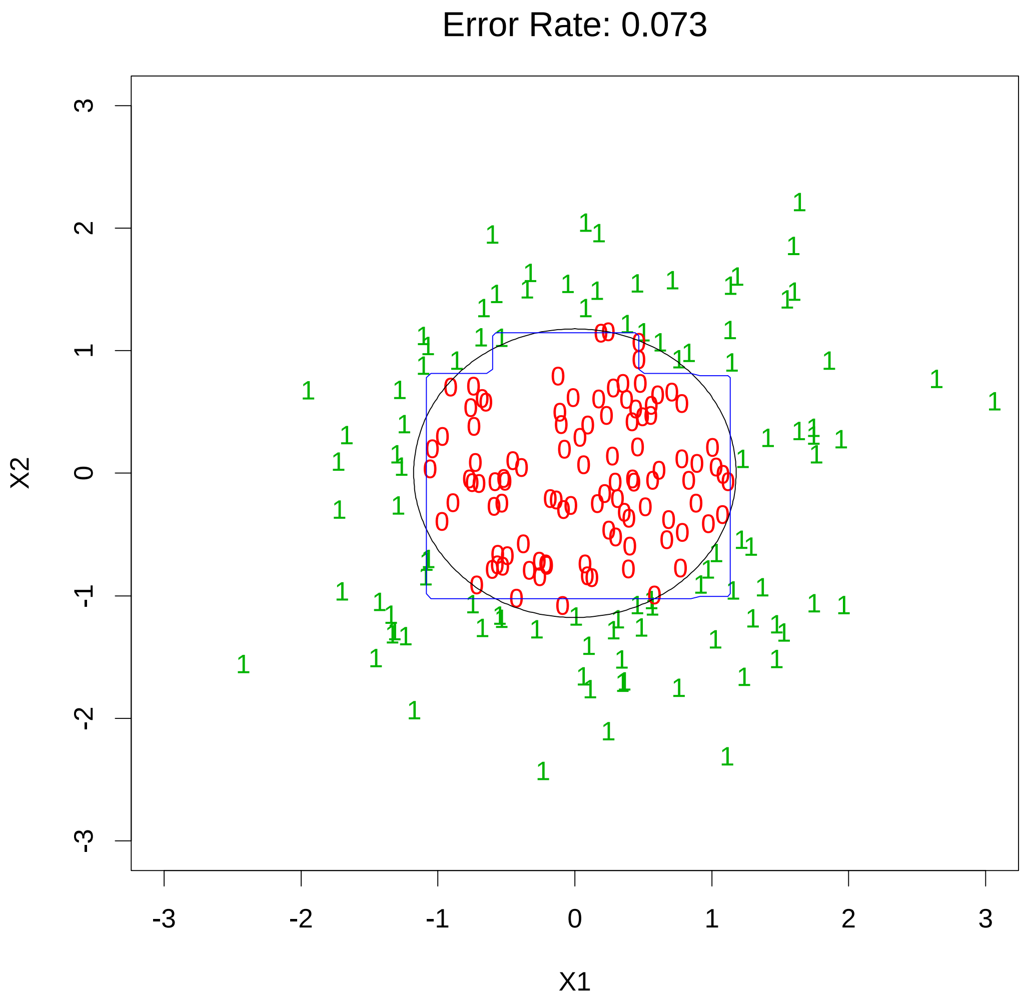


- Deterministic problem; noise comes from sampling distribution of X.
- Use a training sample of size 200.
- Here Bayes Error is 0%.

Classification Tree



Decision Boundary: Tree



When the nested spheres are in 10-dimensions, Classification Trees produces a rather noisy and inaccurate rule $\hat{C}(X)$, with error rates around 30%.

Model Averaging

Classification trees can be simple, but often produce noisy (bushy) or weak (stunted) classifiers.

- Bagging (Breiman, 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by majority vote.
- Boosting (Freund & Shapire, 1996): Fit many large or small trees to reweighted versions of the training data. Classify by weighted majority vote.
- Random Forests (Breiman 1999): Fancier version of bagging.

In general Boosting $\succ$ Random Forests $\succ$ Bagging $\succ$ Single Tree.

Bagging

Bagging or bootstrap aggregation averages a given procedure over many samples, to reduce its variance — a poor man's Bayes. See

pp 246.

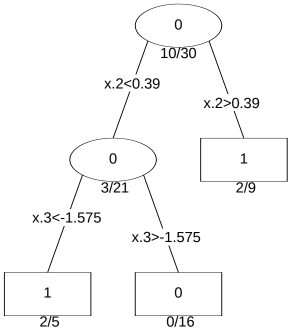
Suppose $C(\mathcal{S}, x)$ is a classifier, such as a tree, based on our training data $\mathcal{S}$, producing a predicted class label at input point x.

To bag C, we draw bootstrap samples $\mathcal{S}^{*1}, \ldots \mathcal{S}^{*B}$ each of size N with replacement from the training data. Then

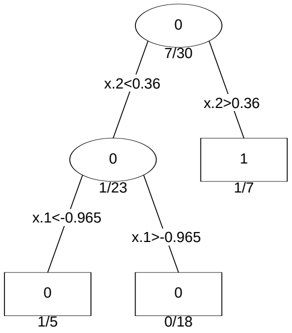
$$\hat{C}_{bag}(x) = \text{Majority Vote } \{C(\mathcal{S}^{*b}, x)\}_{b=1}^{B}.$$

Bagging can dramatically reduce the variance of unstable procedures (like trees), leading to improved prediction. However any simple structure in C (e.g a tree) is lost.

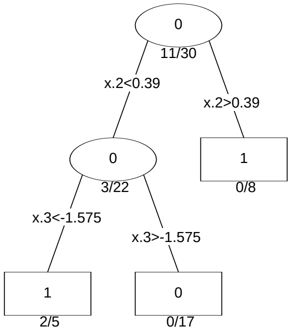
Original Tree



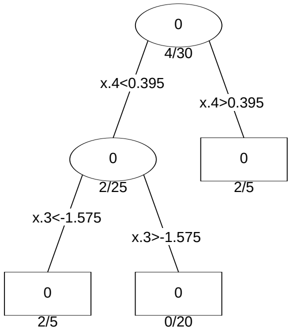
Bootstrap Tree 1



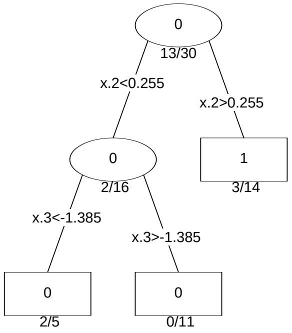
Bootstrap Tree 2



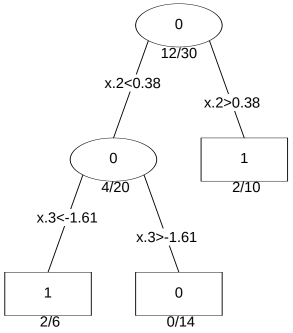
Bootstrap Tree 3



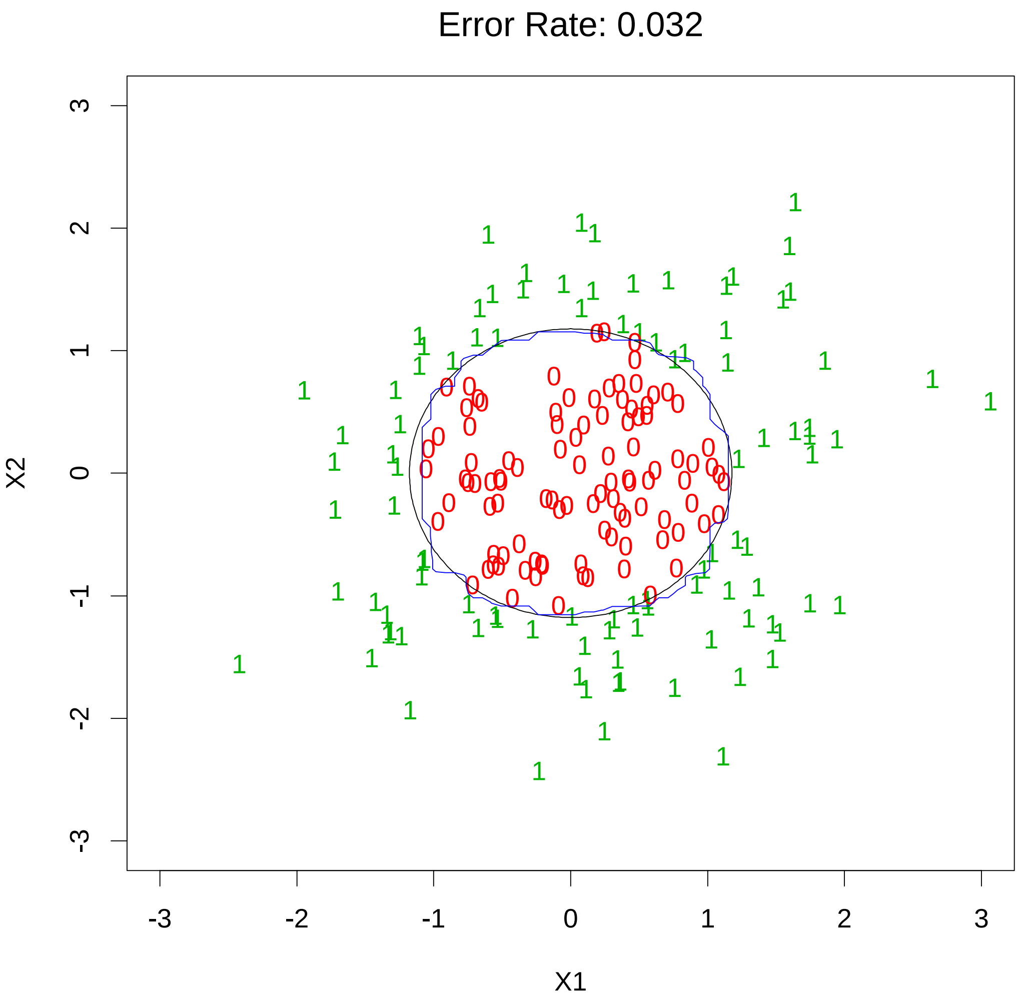
Bootstrap Tree 4



Bootstrap Tree 5



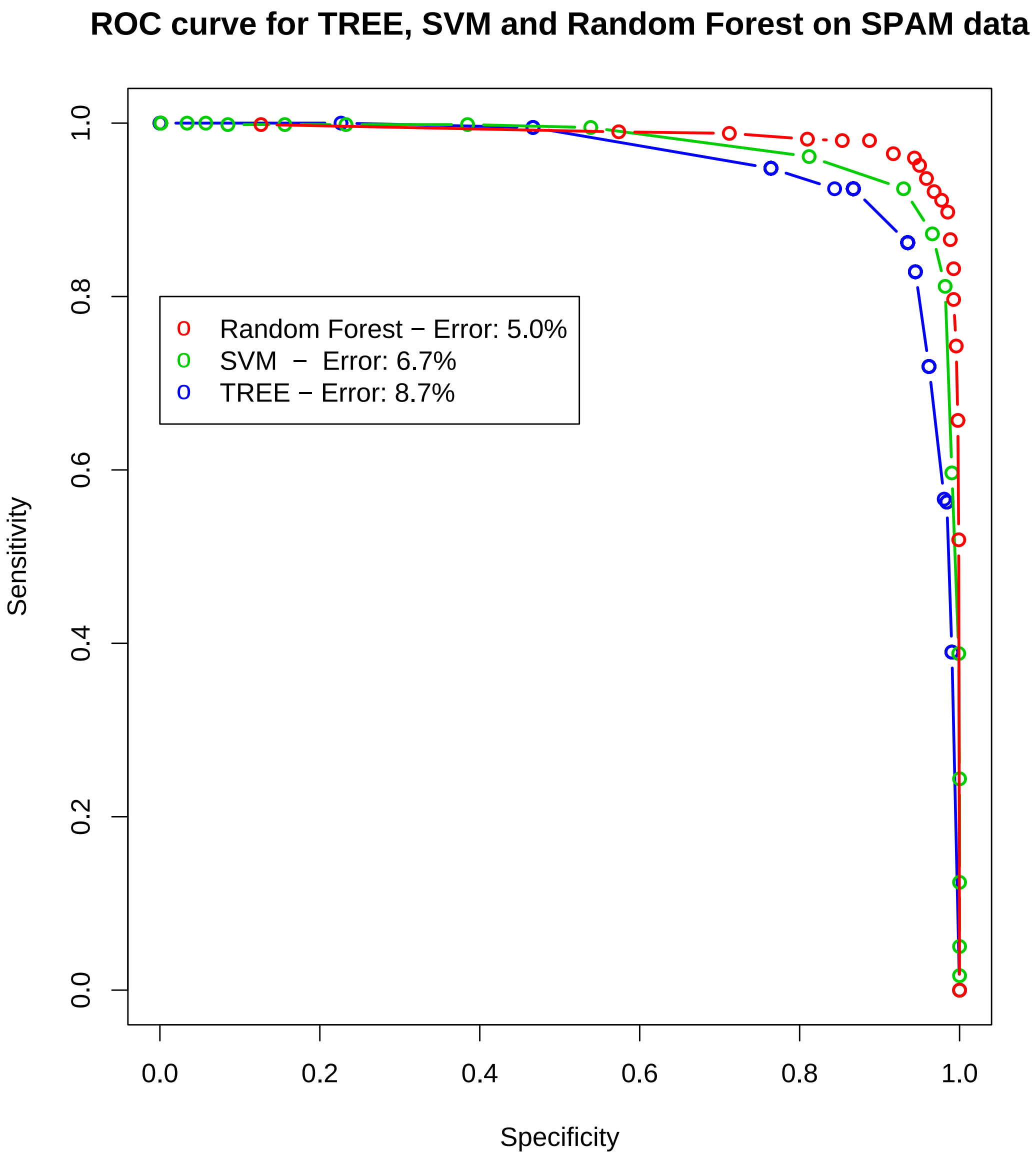
Decision Boundary: Bagging



Bagging averages many trees, and produces smoother decision boundaries.

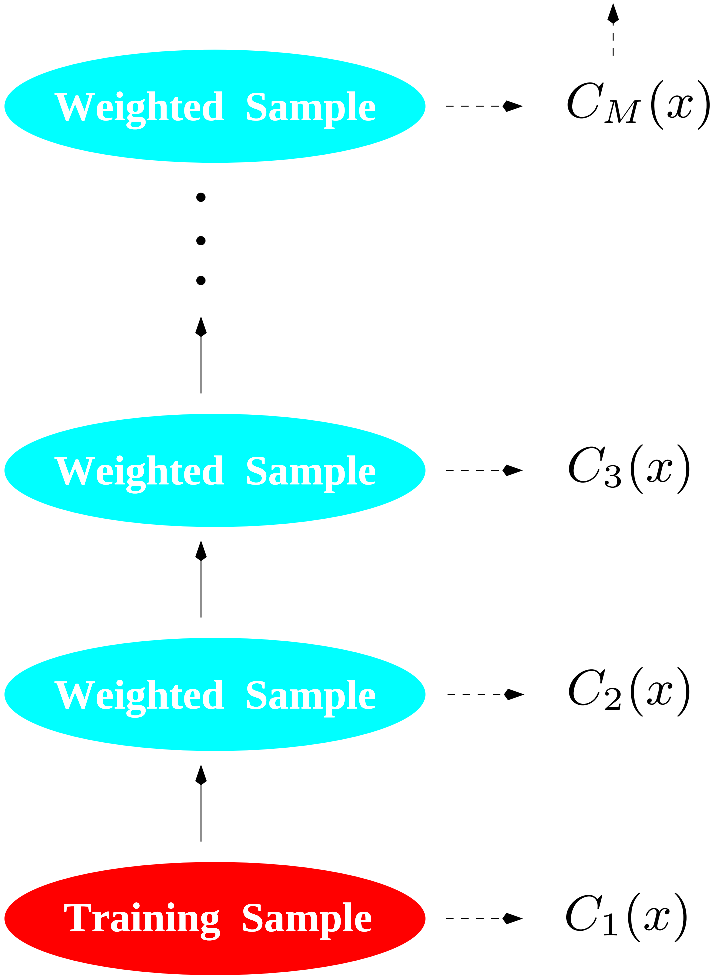
Random forests

- refinement of bagged trees; quite popular
- at each tree split, a random sample of m features is drawn, and only those m features are considered for splitting. Typically $m = \sqrt{p}$ or $\log_2 p$, where p is the number of features
- For each tree grown on a bootstrap sample, the error rate for observations left out of the bootstrap sample is monitored. This is called the "out-of-bag" error rate.
- random forests tries to improve on bagging by "de-correlating" the trees. Each tree has the same expectation.


ROC curve for TREE, SVM and Random Forest on SPAM data

TREE, SVM and RF

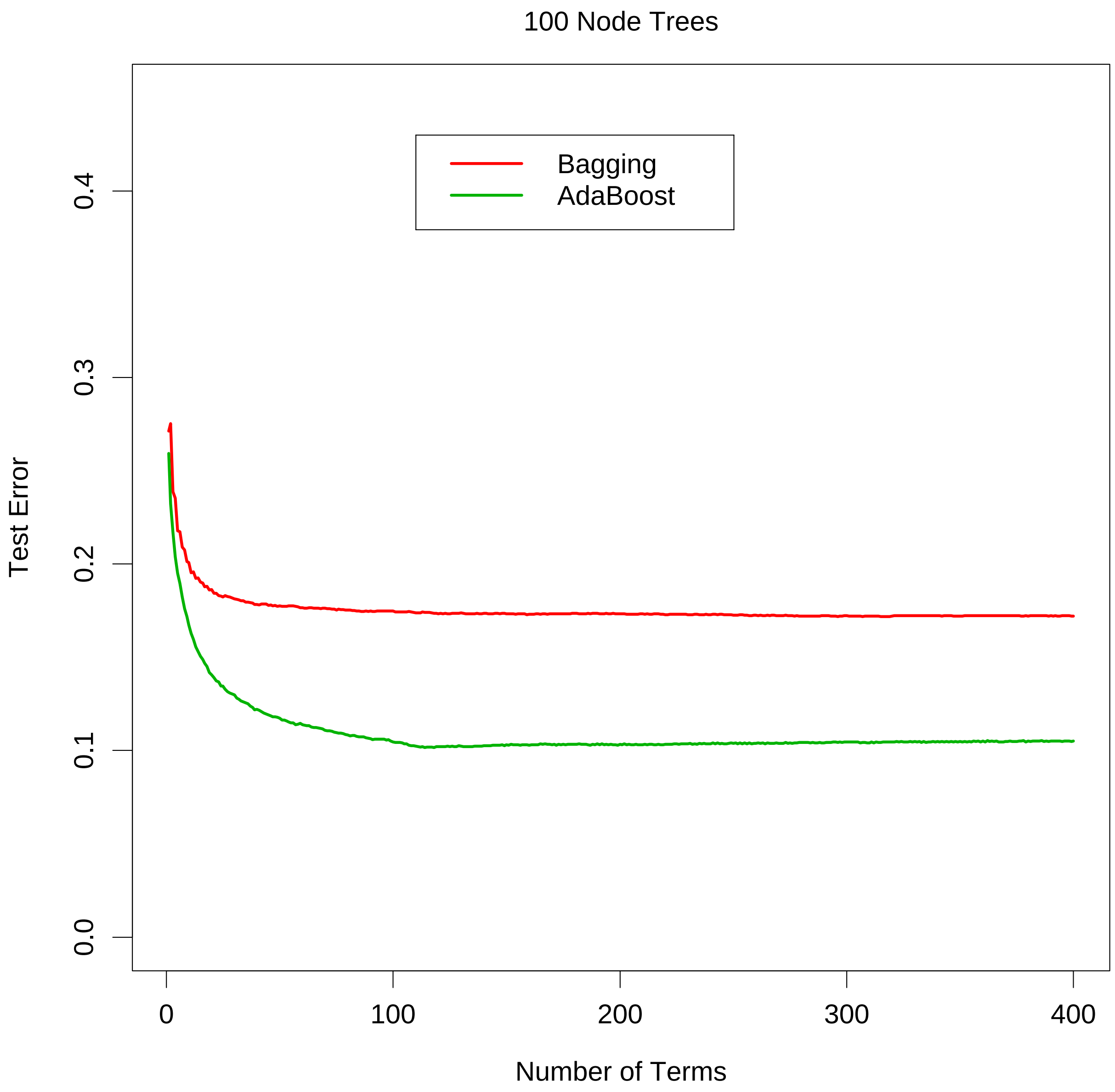
Random Forest dominates both other methods on the SPAM data — 5.0% error. Used 500 trees with default settings for `random Forest` package in `R`.



Boosting

- Average many trees, each grown to re-weighted versions of the training data.
- Final Classifier is weighted average of classifiers:

$$C(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m C_m(x)\right]$$

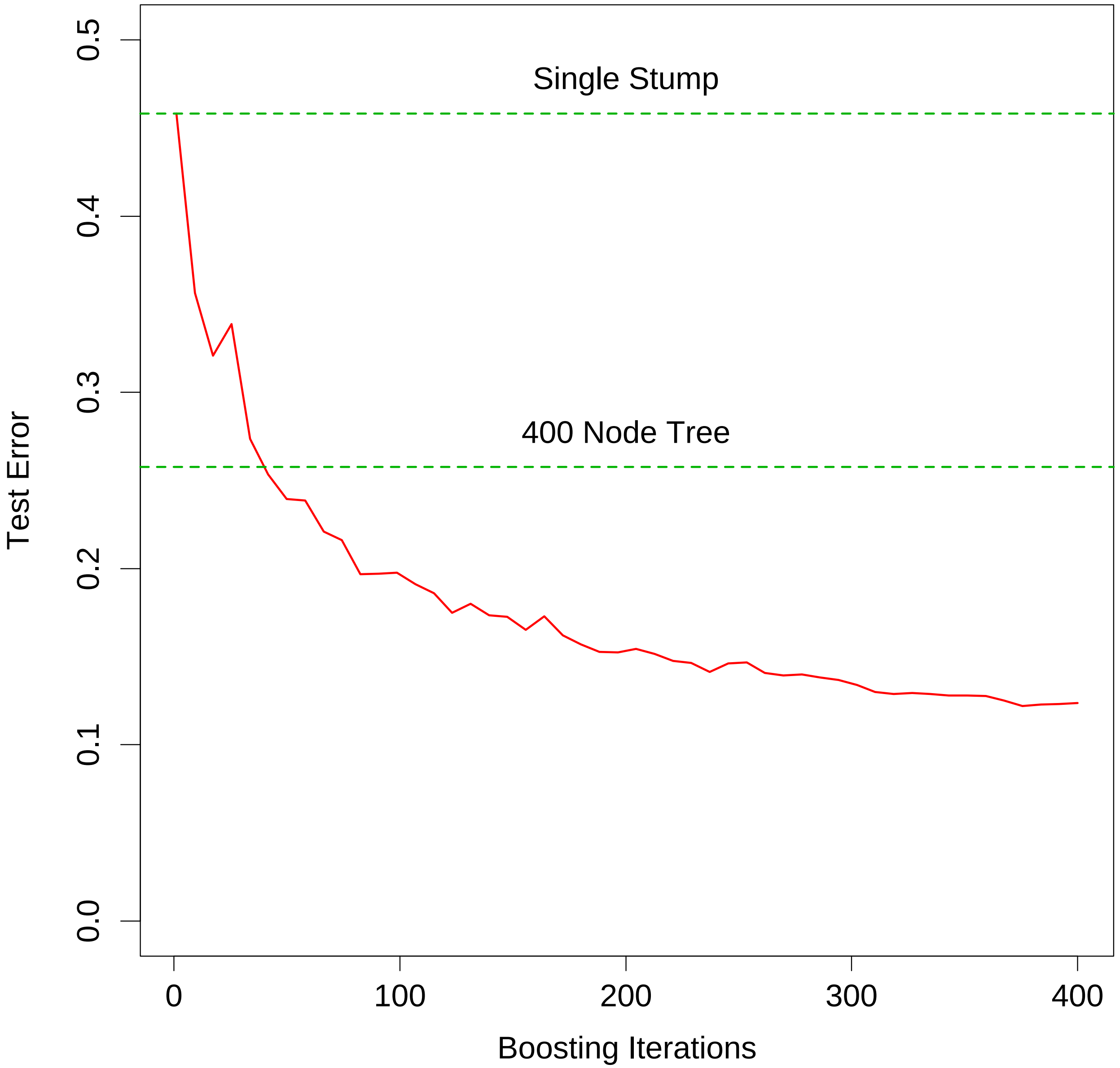


Boosting vs Bagging

- 2000 points from Nested Spheres in R^{10}
- Bayes error rate is 0%.
- Trees are grown best first without pruning.
- Leftmost term is a single tree.

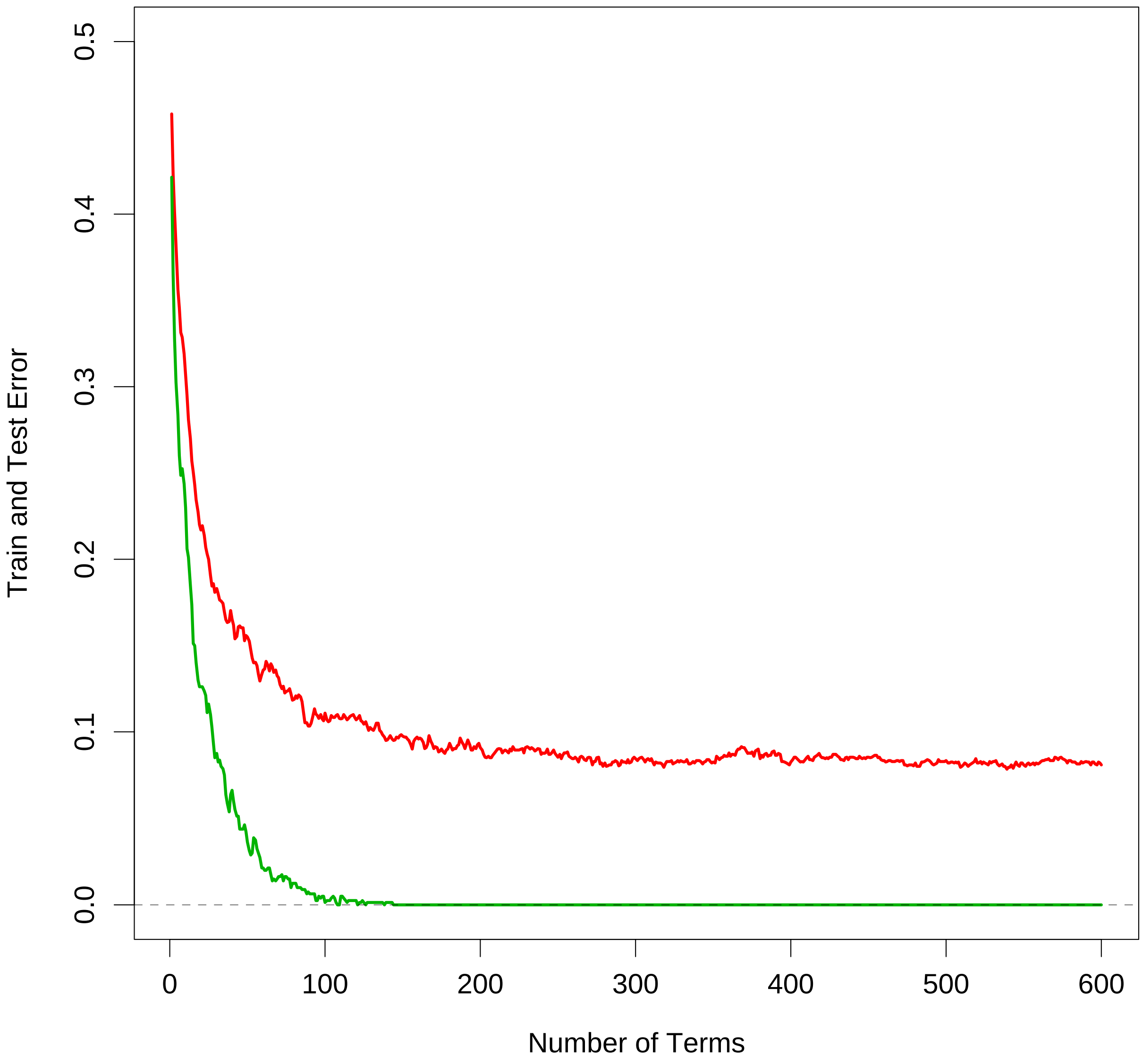
AdaBoost (Freund & Schapire, 1996)

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \dots, N$.
2. For $m = 1$ to M repeat steps (a)–(d):
   (a) Fit a classifier $C_m(x)$ to the training data using weights w_i.
   (b) Compute weighted error of newest tree
   $$\mathrm{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq C_m(x_i))}{\sum_{i=1}^N w_i}.$$
   (c) Compute $\alpha_m = \log[(1 - \mathrm{err}_m)/\mathrm{err}_m]$.
   (d) Update weights for $i = 1, \dots, N$:
   $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq C_m(x_i))]$
   and renormalize to w_i to sum to 1.
3. Output $C(x) = \mathrm{sign}\left[\sum_{m=1}^M \alpha_m C_m(x)\right]$.



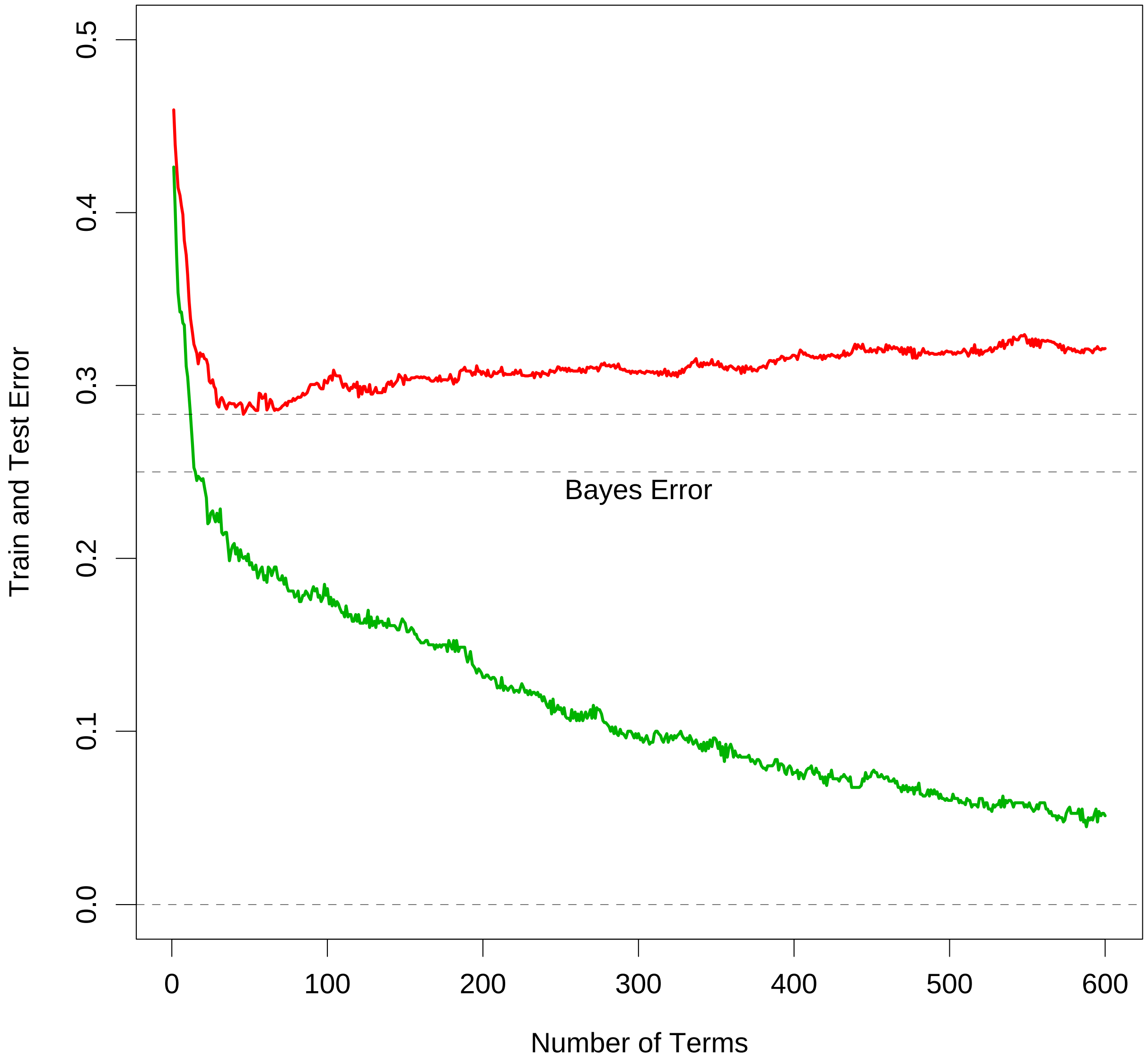
Boosting Stumps

A stump is a two-node tree, after a single split. Boosting stumps works remarkably well on the nested-spheres problem.



Training Error

- Nested spheres in 10-Dimensions.
- Bayes error is 0%.
- Boosting drives the training error to zero.
- Further iterations continue to improve test error in many examples.



Noisy Problems

- Nested Gaussians in 10-Dimensions.
- Bayes error is 25%.
- Boosting with stumps
- Here the test error does increase, but quite slowly.

Stagewise Additive Modeling

Boosting builds an additive model

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m).$$

Here $b(x, \gamma_m)$ is a tree, and γ_m parametrizes the splits.

We do things like that in statistics all the time!

- GAMs: $f(x) = \sum_j f_j(x_j)$
- Basis expansions: $f(x) = \sum_{m=1}^{M} \theta_m h_m(x)$

Traditionally the parameters f_m, θ_m are fit jointly (i.e. least squares, maximum likelihood).

With boosting, the parameters (β_m, γ_m) are fit in a stagewise fashion. This slows the process down, and overfits less quickly.

Additive Trees

- Simple example: stagewise least-squares?
- Fix the past $M-1$ functions, and update the Mth using a tree:

$$\min_{f_M \in Tree(x)} E(Y - \sum_{m=1}^{M-1} f_m(x) - f_M(x))^2$$

- If we define the current residuals to be

$$R = Y - \sum_{m=1}^{M-1} f_m(x)$$

then at each stage we fit a tree to the residuals

$$\min_{f_M \in Tree(x)} E(R - f_M(x))^2$$

Stagewise Least Squares

Suppose we have available a basis family $b(x;\gamma)$ parametrized by γ.

- After $m-1$ steps, suppose we have the model $f_{m-1}(x) = \sum_{j=1}^{m-1} \beta_j b(x;\gamma_j)$.
- At the mth step we solve

$$\min_{\beta,\gamma} \sum_{i=1}^{N} (y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma))^2$$

- Denoting the residuals at the mth stage by $r_{im} = y_i - f_{m-1}(x_i)$, the previous step amounts to

$$\min_{\beta,\gamma} (r_{im} - \beta b(x_i;\gamma))^2,$$

- Thus the term $\beta_m b(x;\gamma_m)$ that best fits the current residuals is added to the expansion at each step.

Adaboost: Stagewise Modeling

- AdaBoost builds an additive logistic regression model

$$f(x) = \log \frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)} = \sum_{m=1}^{M} \alpha_m G_m(x)$$

by stagewise fitting using the loss function

$$L(y, f(x)) = \exp(-y\, f(x)).$$

- Given the current $f_{M-1}(x)$, our solution for (β_m, G_m) is

$$\arg\min_{\beta, G} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta\, G(x))]$$

where $G_m(x) \in \{-1, 1\}$ is a tree classifier and β_m is a coefficient.

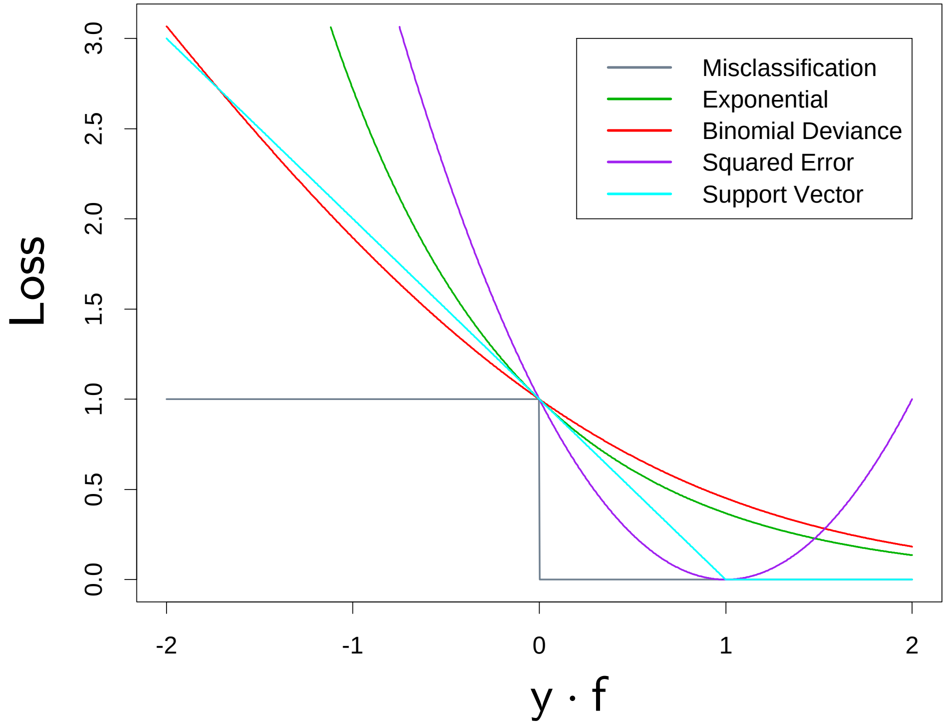
- With $w_i^{(m)} = \exp(-y_i\, f_{m-1}(x_i))$, this can be re-expressed as

$$\arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta\, y_i\, G(x_i))$$

- We can show that this leads to the Adaboost algorithm; See pp 305.

Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of Statistical Learning
Data Mining, Inference, and Prediction
Springer

Why Exponential Loss?



- $e^{-yF(x)}$ is a monotone, smooth upper bound on misclassification loss at x.
- Leads to simple reweighting scheme.
- Has logit transform as population minimizer

$$f^*(x) = \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}$$

- Other more robust loss functions, like binomial deviance.

General Stagewise Algorithm

We can do the same for more general loss functions, not only least squares.

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute $(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$.
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

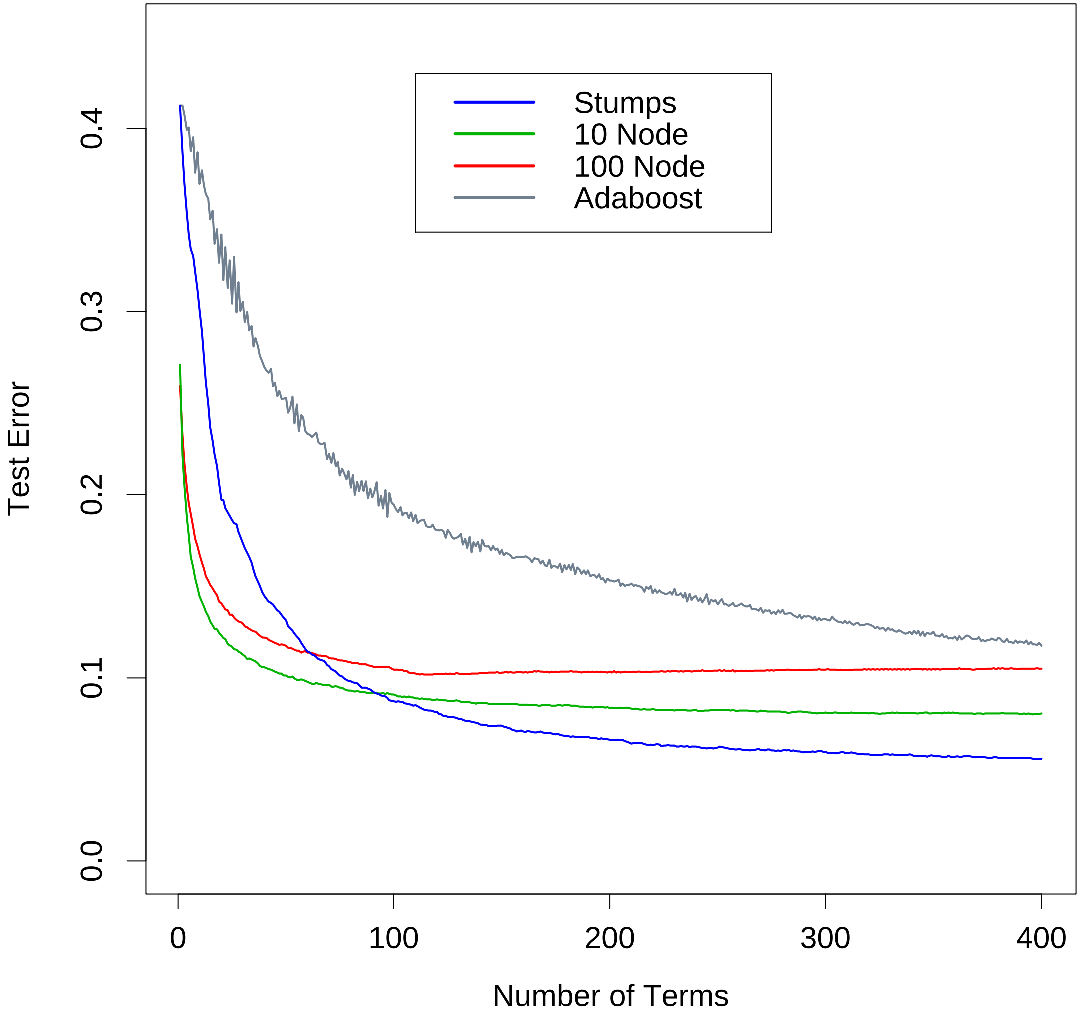
Sometimes we replace step (b) in item 2 by

(b*) Set $f_m(x) = f_{m-1}(x) + \nu\beta_m b(x; \gamma_m)$

Here ν is a shrinkage factor, and often $\nu < 0.1$. Shrinkage slows the stagewise model-building even more, and typically leads to better performance.

Gradient Boosting

- General boosting algorithm that works with a variety of different loss functions. Models include regression, resistant regression, K-class classification and risk modeling.
- Gradient Boosting builds additive tree models, for example, for representing the logits in logistic regression.
- Tree size is a parameter that determines the order of interaction (next slide).
- Gradient Boosting inherits all the good features of trees (variable selection, missing data, mixed predictors), and improves on the weak features, such as prediction performance.
- Gradient Boosting is described in detail in , section 10.10.



Tree Size

The tree size J determines the interaction order of the model:

$$\begin{aligned}\eta(X) &= \sum_j \eta_j(X_j) \\ &\quad + \sum_{jk} \eta_{jk}(X_j, X_k) \\ &\quad + \sum_{jkl} \eta_{jkl}(X_j, X_k, X_l) \\ &\quad + \cdots\end{aligned}$$

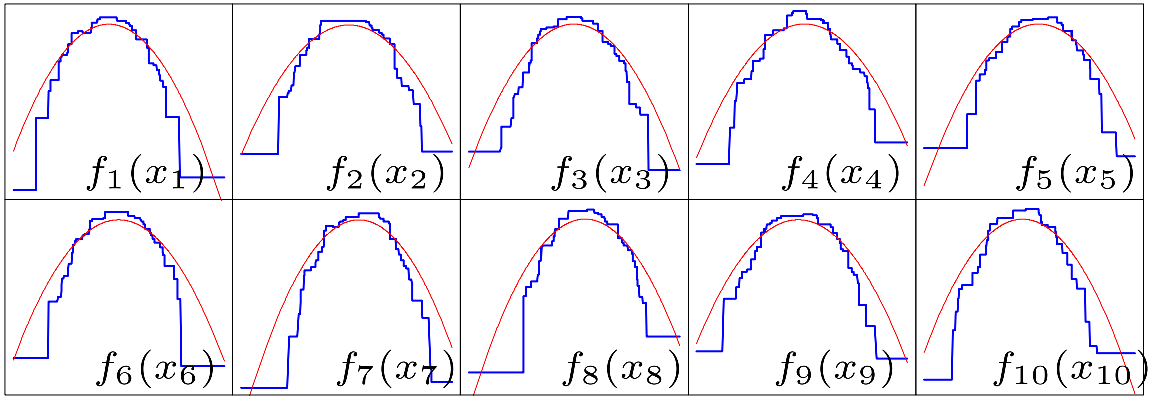
Stumps win!

Since the true decision boundary is the surface of a sphere, the function that describes it has the form

$$f(X) = X_1^2 + X_2^2 + \ldots + X_p^2 - c = 0.$$

Boosted stumps via Gradient Boosting returns reasonable approximations to these quadratic functions.

Coordinate Functions for Additive Logistic Trees



Spam Example Results

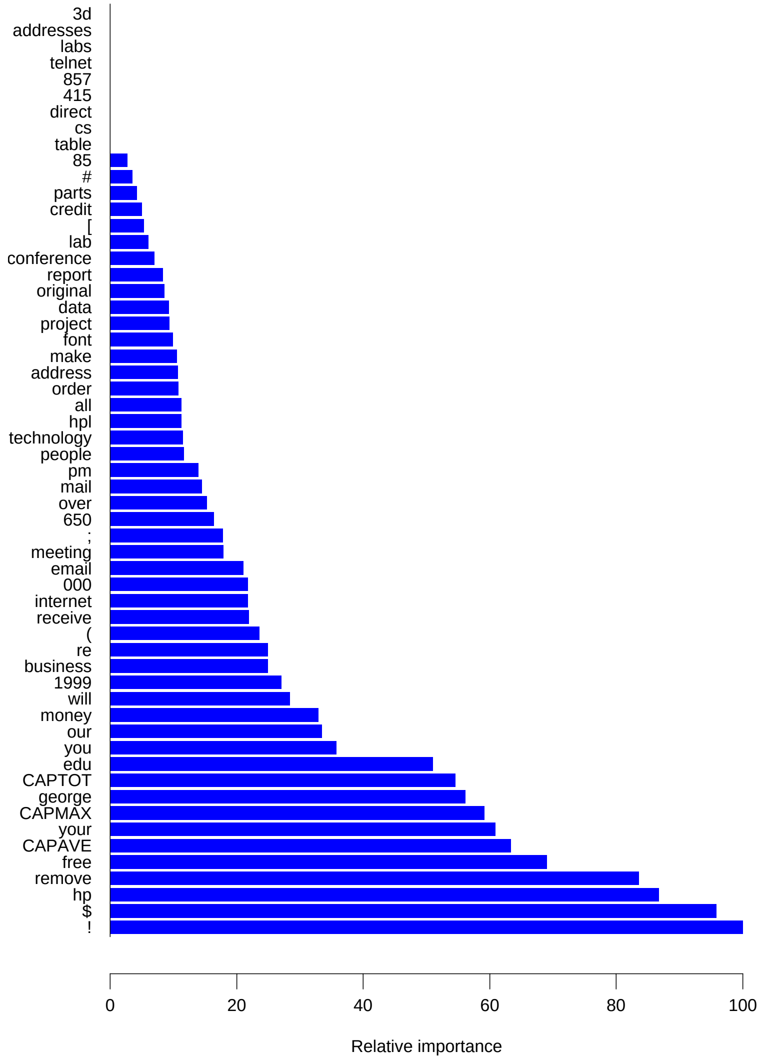
With 3000 training and 1500 test observations, Gradient Boosting fits an additive logistic model

$$f(x) = \log \frac{\Pr(\texttt{spam}|x)}{\Pr(\texttt{email}|x)}$$

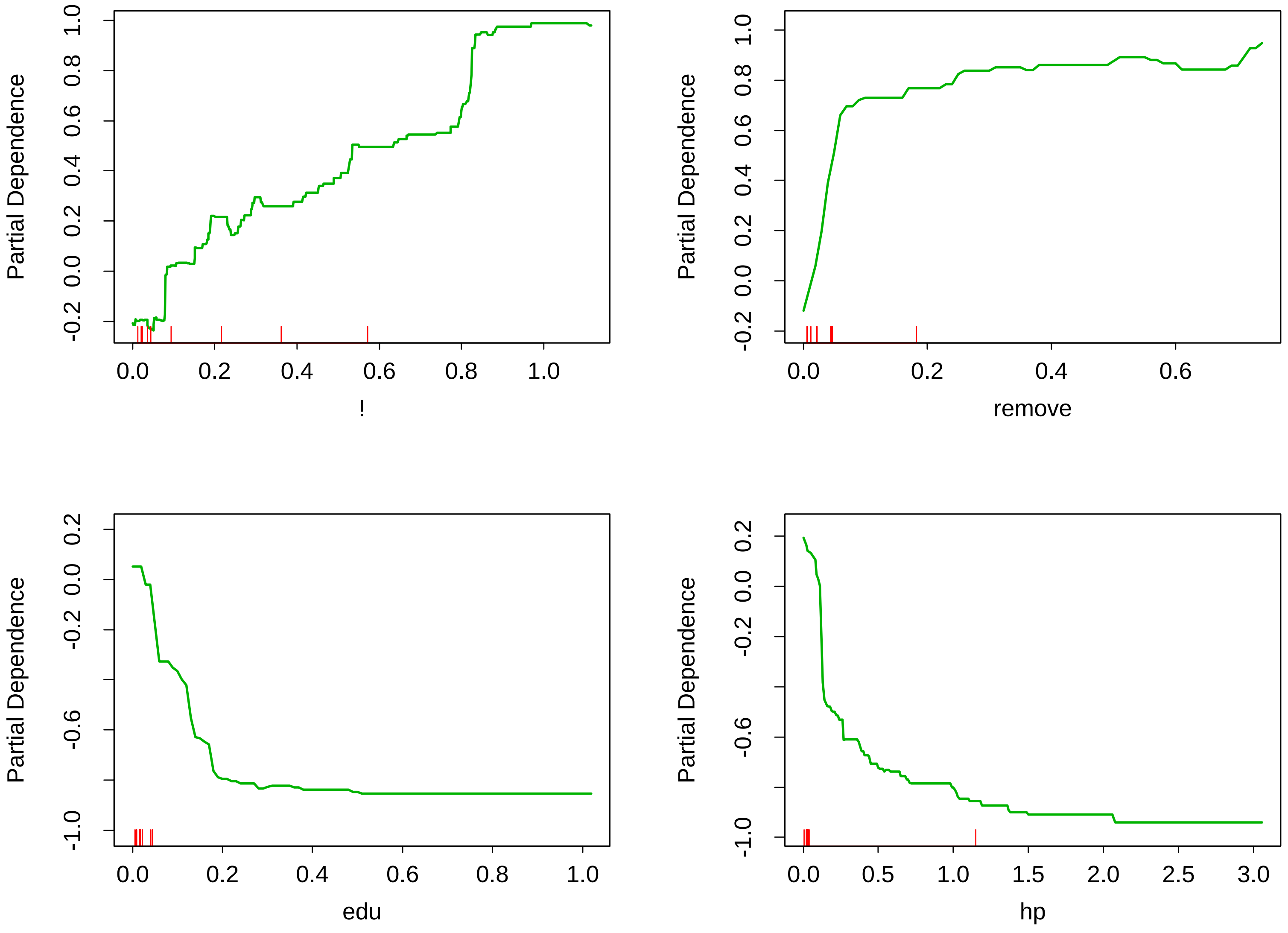
using trees with $J = 6$ terminal-node trees.

Gradient Boosting achieves a test error of 4%, compared to 5.3% for an additive GAM, 5.0% for Random Forests, and 8.7% for CART.

Spam: Variable Importance



Spam: Partial Dependence



Comparison of Learning Methods

Some characteristics of different learning methods.
Key: 🟢= good, 🟡=fair, and 🔴=poor.

Characteristic	Neural Nets	SVM	CART	GAM	KNN, Kernel	Gradient Boost
Natural handling of data of "mixed" type	🔴	🔴	🟢	🟢	🔴	🟢
Handling of missing values	🔴	🔴	🟢	🔴	🟢	🟢
Robustness to outliers in input space	🔴	🔴	🟢	🟡	🟢	🟢
Insensitive to monotone transformations of inputs	🔴	🔴	🟢	🟢	🔴	🟢
Computational scalability (large N)	🔴	🔴	🟢	🟢	🔴	🟢
Ability to deal with irrelevant inputs	🔴	🔴	🟢	🟡	🔴	🟢
Ability to extract linear combinations of features	🟢	🟢	🔴	🔴	🟡	🟡
Interpretability	🔴	🔴	🟡	🟢	🔴	🟢
Predictive power	🟢	🟢	🔴	🔴	🟢	🟢

Software

- R: free GPL statistical computing environment available from CRAN, implements the S language. Includes:
  - randomForest: implementation of Leo Breimans algorithms.
  - rpart: Terry Therneau's implementation of classification and regression trees.
  - gbm: Greg Ridgeway's implementation of Friedman's gradient boosting algorithm.
- Salford Systems: Commercial implementation of trees, random forests and gradient boosting.
- Splus (Insightful): Commerical version of S.
- Weka: GPL software from University of Waikato, New Zealand. Includes Trees, Random Forests and many other procedures.

Ensembles

Léon Bottou

COS 424 – 4/8/2010

Readings

- T. G. Dietterich (2000)
  “Ensemble Methods in Machine Learning”.

- R. E. Schapire (2003):
  “The Boosting Approach to Machine Learning”.
  Sections 1,2,3,4,6.

Summary

1. Why ensembles?
2. Combining outputs.
3. Constructing ensembles.
4. Boosting.

I. Ensembles

Ensemble of classifiers

Ensemble of classifiers

- Consider a set of classifiers $h_1, h_2, \ldots, h_L$.
- Construct a classifier by combining their individual decisions.
- For example by voting their outputs.

Accuracy

- The ensemble works if the classifiers have low error rates.

Diversity

- No gain if all classifiers make the same mistakes.
- What if classifiers make different mistakes?

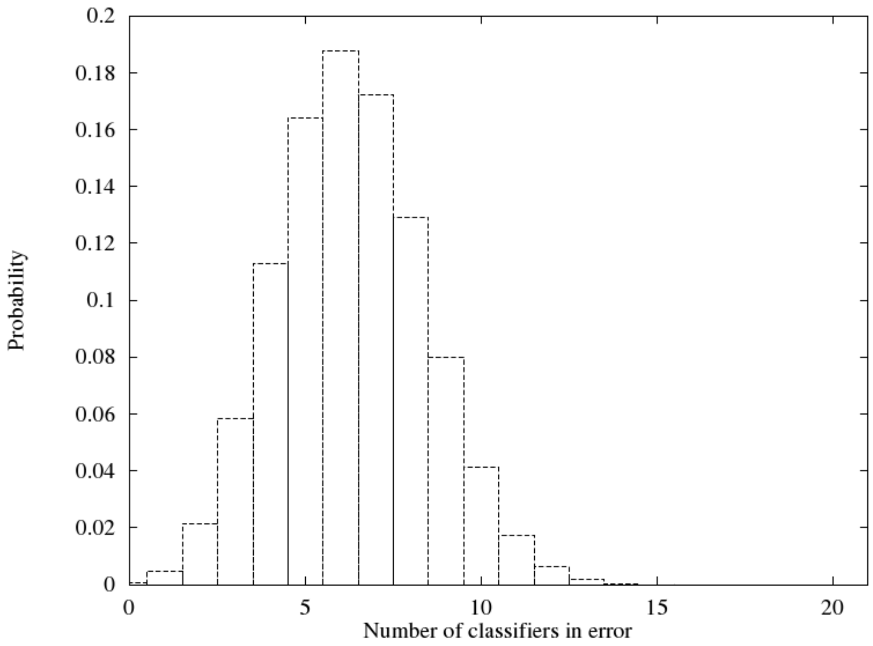
Uncorrelated classifiers

Assume $\forall r \neq s \quad \text{Cov}\left[\, \mathbb{1}\{h_r(x) = y\}\,,\ \mathbb{1}\{h_s(x) = y\}\,\right] = 0$

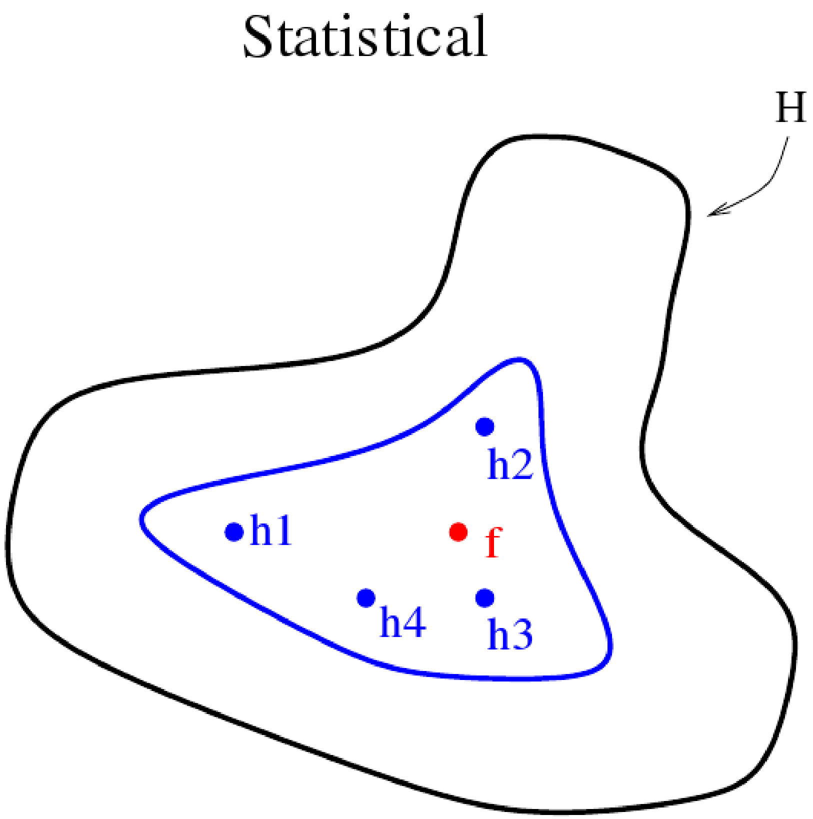
The tally of classifier votes follows a binomial distribution.

Example

Twenty-one uncorrelated classifiers with 30% error rate.



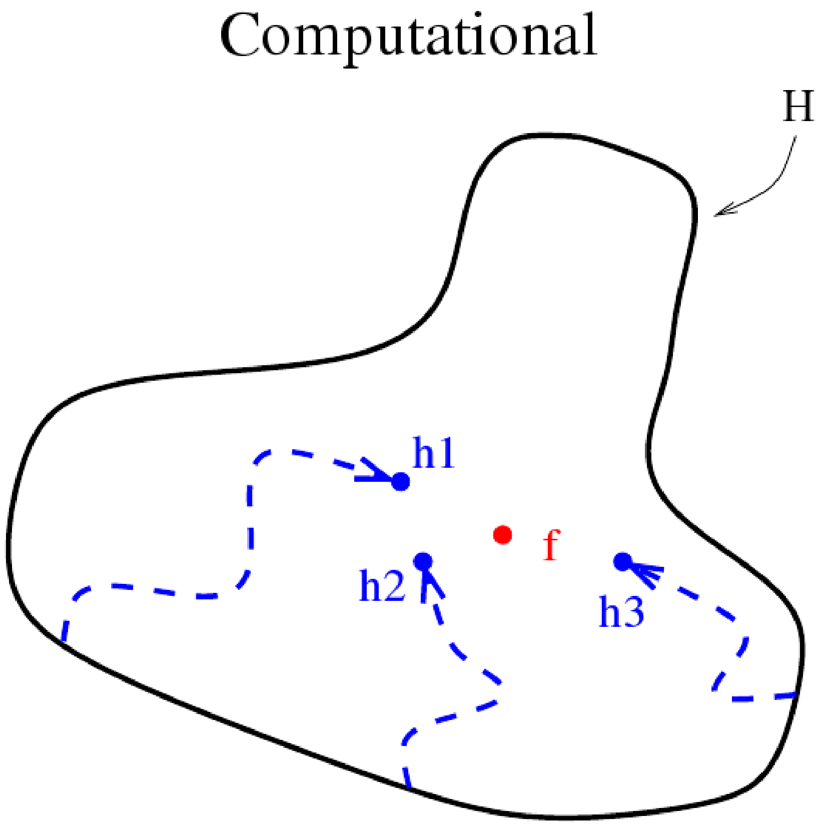
Statistical motivation



blue : classifiers that work well on the training set(s)

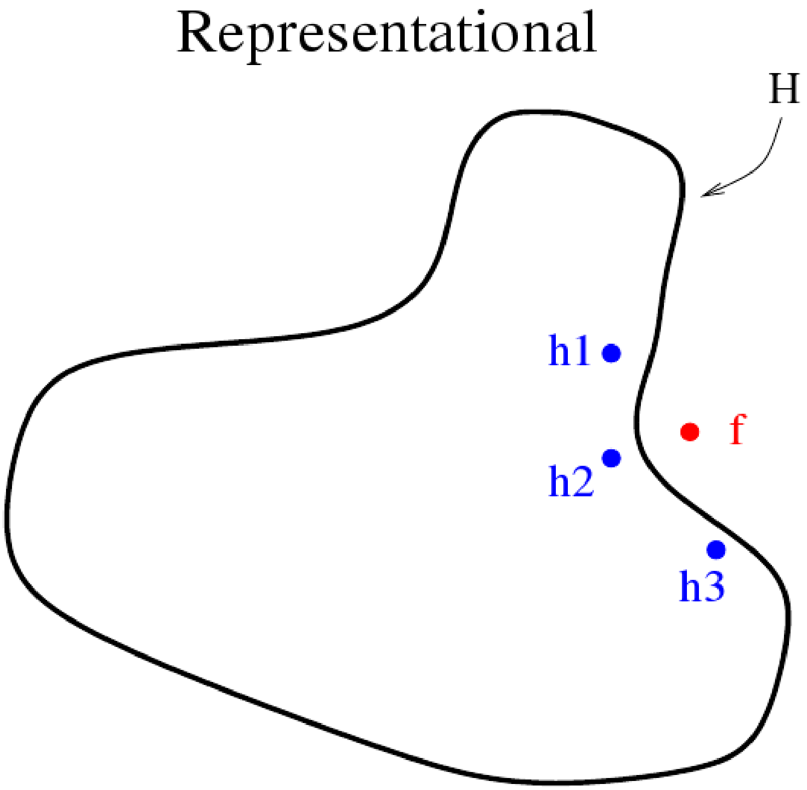
f : best classifier.

Computational motivation



blue : classifier search may reach local optima
f : best classifier.

Representational motivation



blue : classifier space may not contain best classifier

f : best classifier.

Practical success

Recommendation system

– Netflix “movies you may like”.

– Customers sometimes rate movies they rent.

– Input: (movie, customer)

– Output: rating

Netflix competition

– 1M$ for the first team to do 10% better than their system.

Winner: BellKor team and friends

– Ensemble of more than 800 rating systems.

Runner-up: everybody else

– Ensemble of all the rating systems built by the other teams.

Bayesian ensembles

Let D represent the training data.

Enumerating all the classifiers

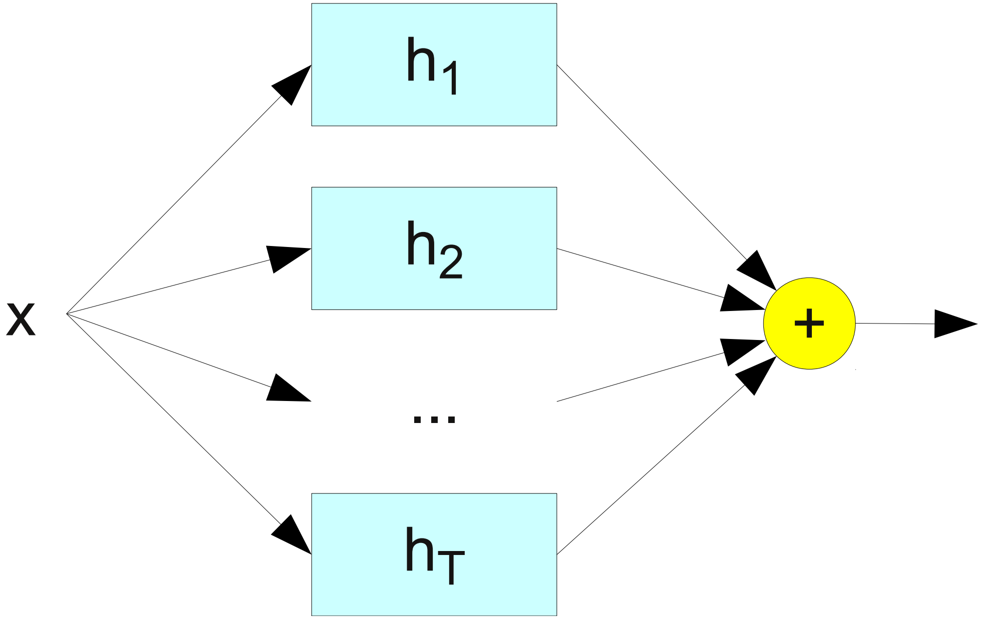
$$\begin{aligned} P(y|x,D) &= \sum_h P(y,h|x,D) \\ &= \sum_h P(h|x,D)\,P(y|h,x,D) \\ &= \sum_h P(h|D)\,P(y|x,h) \end{aligned}$$

$P(h|D)$: how well does h match the training data.
$P(y|x,h)$: what h predicts for pattern x.

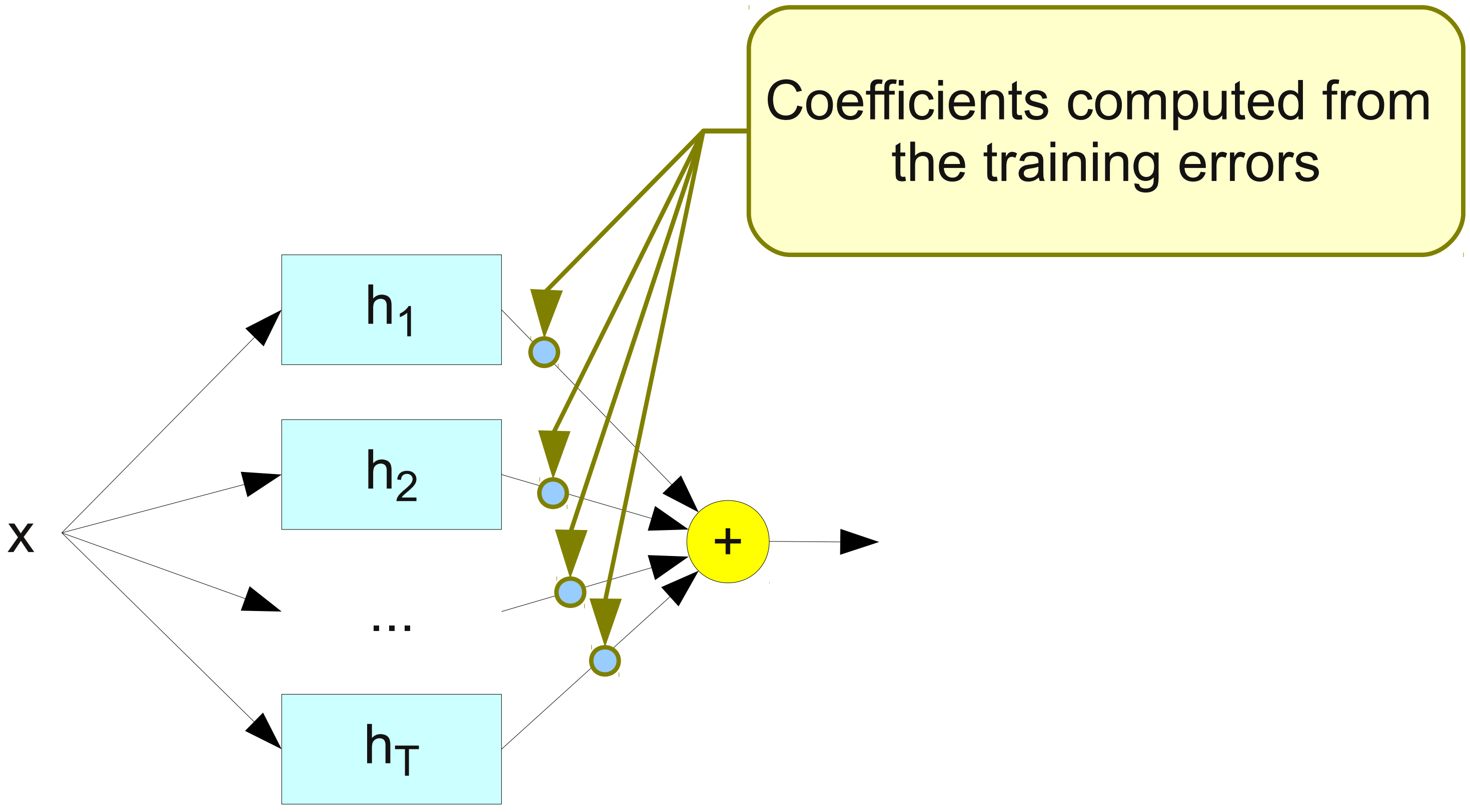
Note that this is a weighted average.

II. Combining Outputs

Simple averaging

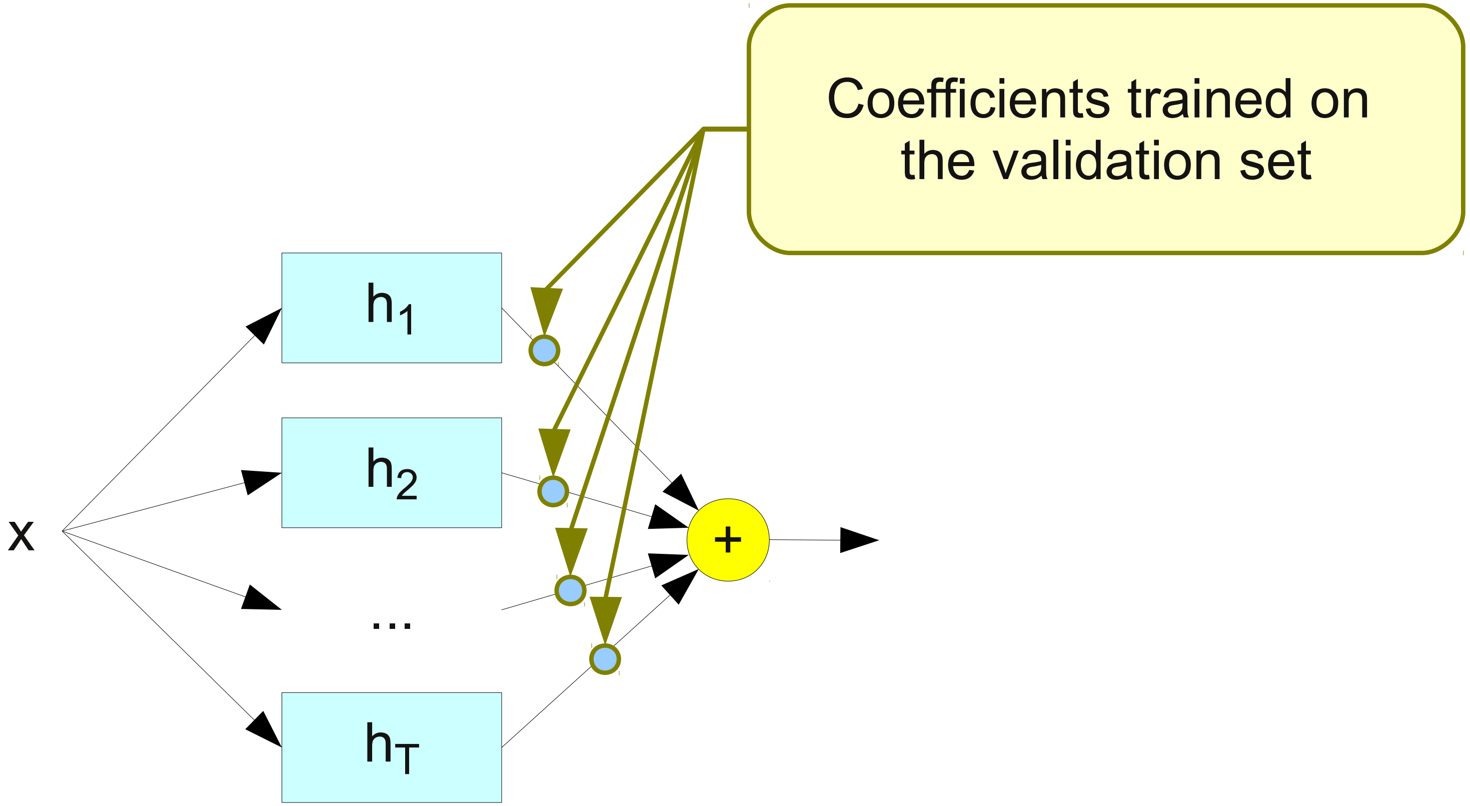


Weighted averaging a priori



Weights derived from the training errors, e.g. $\exp(-\beta\, TrainingError(h_t))$.
Approximate Bayesian ensemble.

Weighted averaging with trained weights

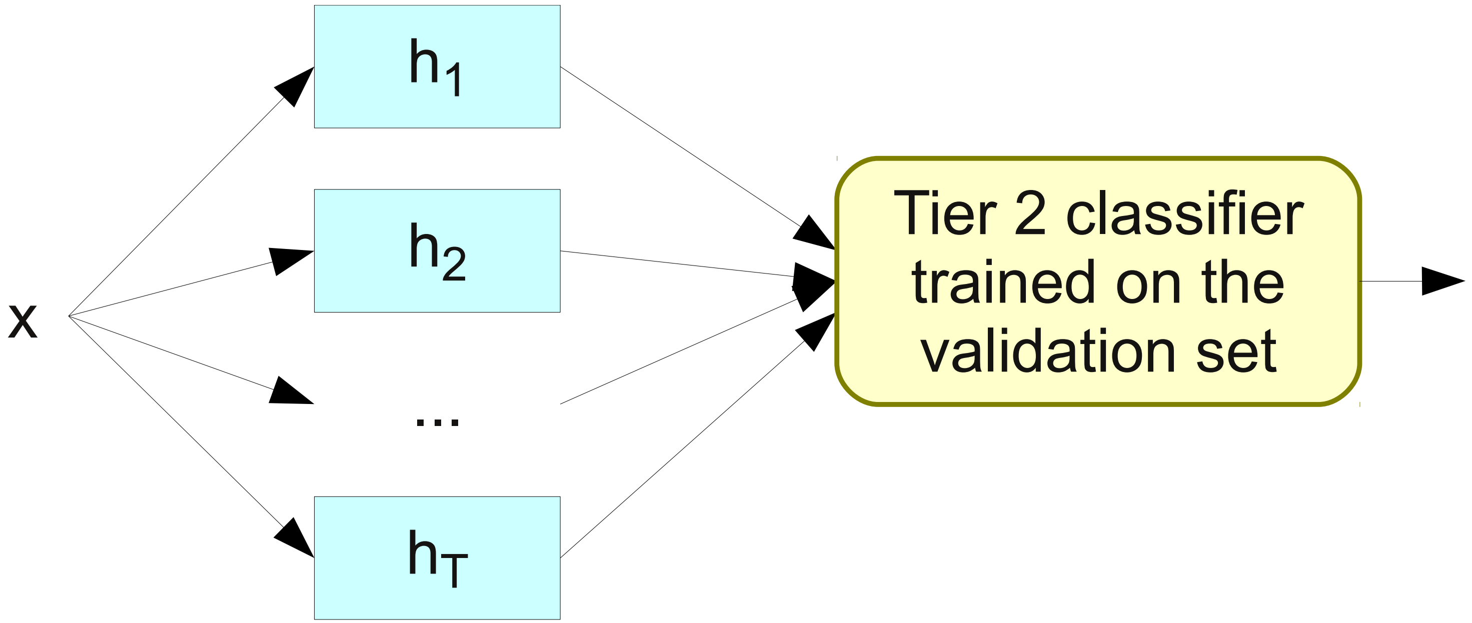


Train weights on the validation set.

Training weights on the training set overfits easily.

You need another validation set to estimate the performance!

Stacked classifiers



Second tier classifier trained on the validation set.

You need another validation set to estimate the performance!

III. Constructing Ensembles

Diversification

Cause of the mistake	Diversification strategy
Pattern was difficult.	hopeless
Overfitting ($\star$)	vary the training sets
Some features were noisy	vary the set of input features
Multiclass decisions were inconsistent	vary the class encoding

Manipulating the training examples

Bootstrap replication simulates training set selection

- Given a training set of size n, construct a new training set by sampling n examples with replacement.
- About 30% of the examples are excluded.

Bagging

- Create bootstrap replicates of the training set.
- Build a decision tree for each replicate.
- Estimate tree performance using out-of-bootstrap data.
- Average the outputs of all decision trees.

Boosting

- See part IV.

Manipulating the features

Random forests

- Construct decision trees on bootstrap replicas.
  Restrict the node decisions to a small subset of features
  picked randomly for each node.
- Do not prune the trees.
  Estimate tree performance using out-of-bootstrap data.
  Average the outputs of all decision trees.

Multiband speech recognition

- Filter speech to eliminate a random subset of the frequencies.
- Train speech recognizer on filtered data.
- Repeat and combine with a second tier classifier.
- Resulting recognizer is more robust to noise.

Manipulating the output codes

Reducing multiclass problems to binary classification

– We have seen one versus all.

– We have seen all versus all.

Error correcting codes for multiclass problems

– Code the class numbers with an error correcting code.

– Construct a binary classifier for each bit of the code.

– Run the error correction algorithm on the binary classifier outputs.

IV. Boosting

Motivation

- Easy to come up with rough rules of thumb for classifying data
  - email contains more than 50% capital letters.
  - email contains expression “buy now”.
- Each alone isnt great, but better than random.
- Boosting converts rough rules of thumb into an accurate classier.

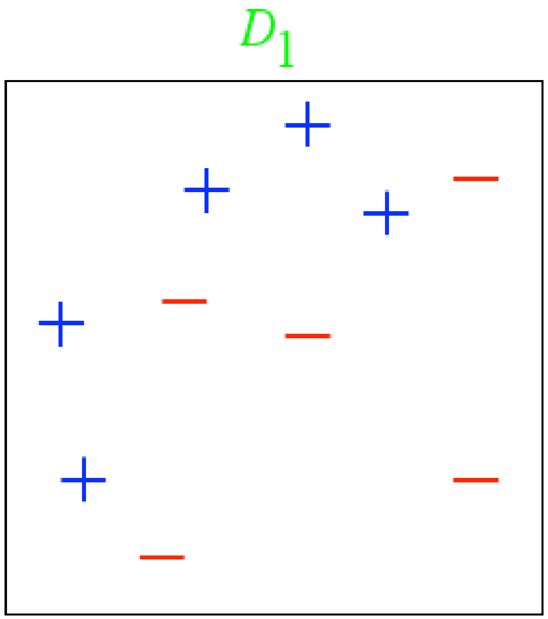
  Boosting was invented by Prof. Schapire.

Adaboost

Given examples $(x_1, y_1) \dots (x_n, y_n)$ with $y_i = \pm 1$.

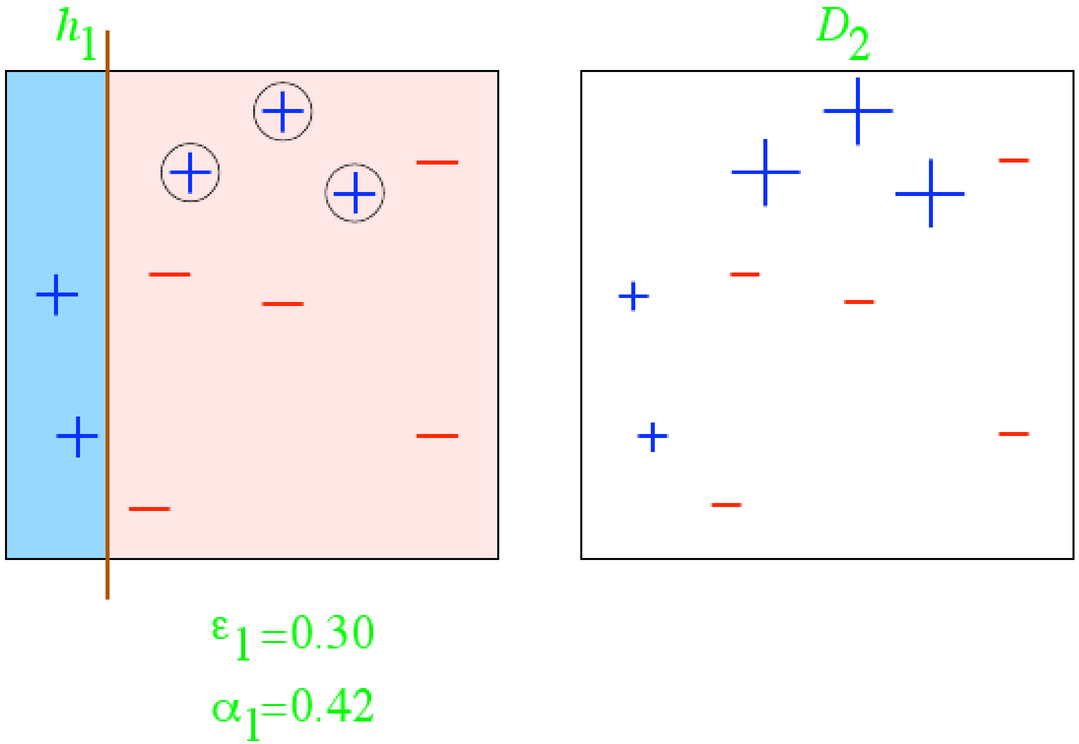
```
Let $D_1(i) = 1/n$ for $i = 1 \dots n$.
For $t = 1 \dots T$ do
    • Run weak learner using examples with weights $D_t$.
    • Get weak classifier $h_t$
    • Compute error: $\varepsilon_t = \sum_i D_t(i)\, \mathbb{1}(h_t(x_i) \neq y_i)$
    • Compute magic coefficient $\alpha_t = \frac{1}{2} \log \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$
    • Update weights $D_{t+1}(i) = \frac{D_t(i)\, e^{-\alpha_t\, y_i\, h_t(x_i)}}{Z_t}$
Output the final classifier $f_T(x) = \text{sign}\left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$
```

Toy example

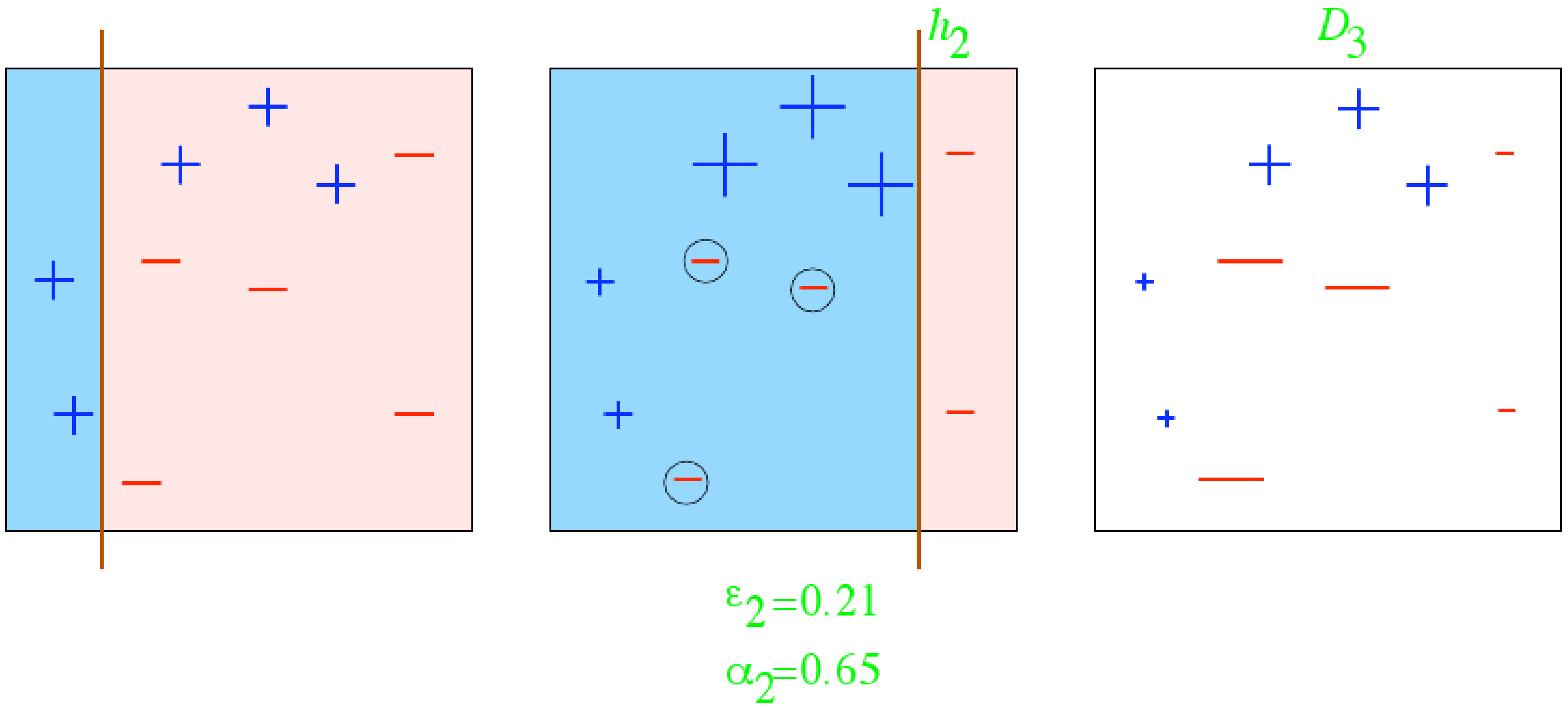


Weak classifiers: vertical or horizontal half-planes.

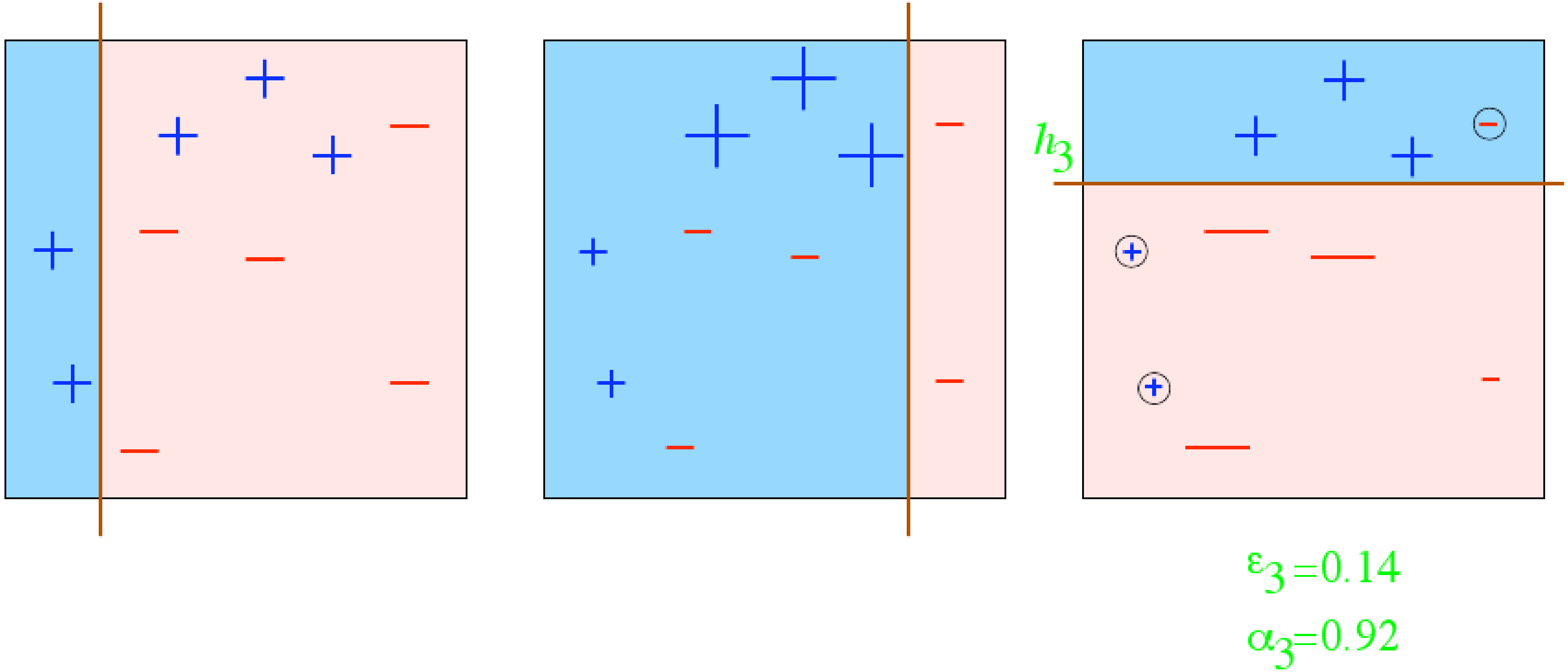
Adaboost round 1



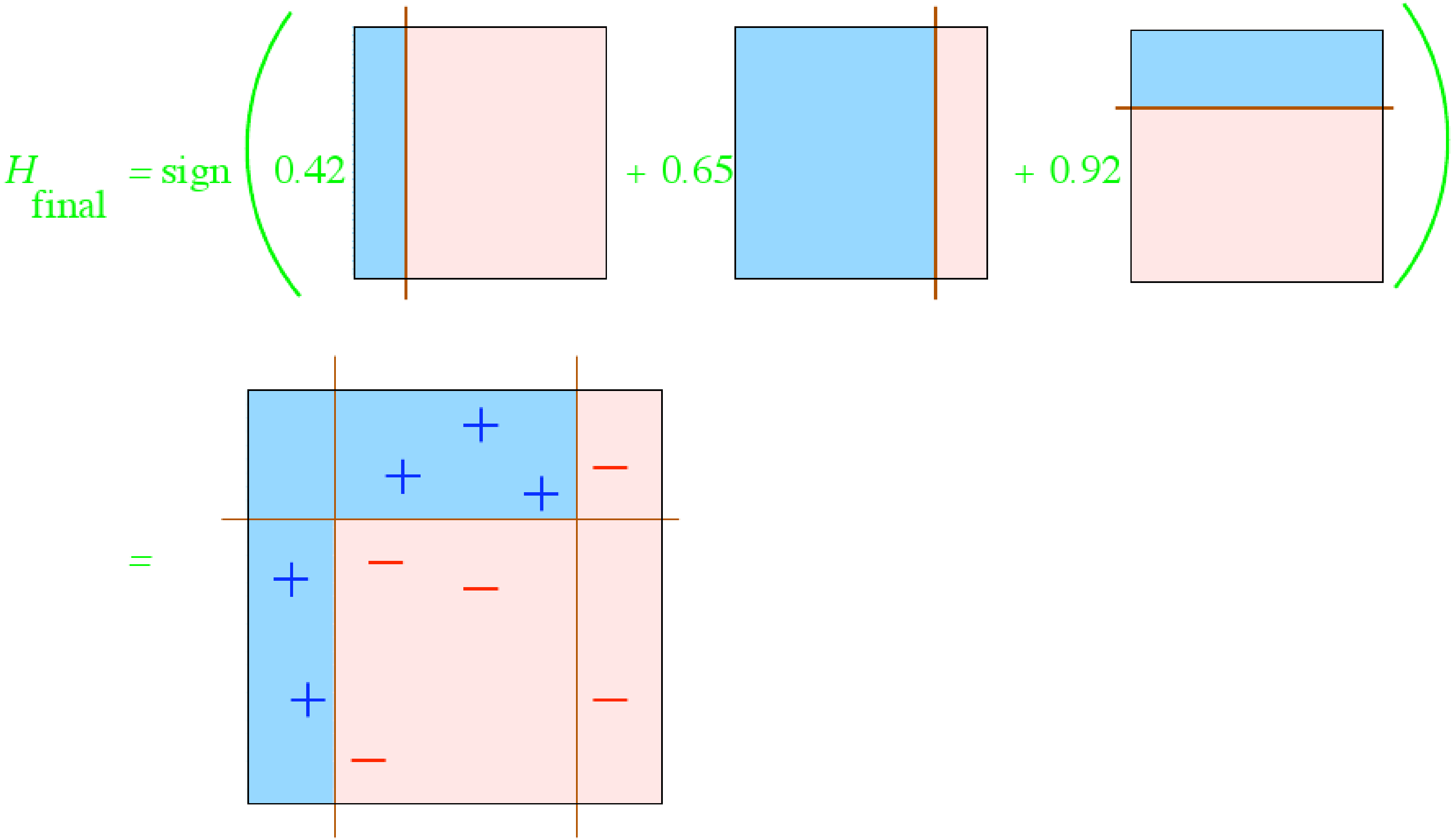
Adaboost round 2



Adaboost round 3



Adaboost final classifier



From weak learner to strong classifier (1)

Preliminary

$$D_{T+1}(i) \;=\; D_1(i)\frac{e^{-\alpha_1\, y_i\, h_1(x_i)}}{Z_1}\cdots\frac{e^{-\alpha_T\, y_i\, h_T(x_i)}}{Z_T} \;=\; \frac{1}{n}\,\frac{e^{-y_i\, f_T(x_i)}}{\prod_t Z_t}$$

Bounding the training error

$$\frac{1}{n}\sum_i \mathbb{I}\{f_T(x_i)\neq y_i\} \;\leq\; \frac{1}{n}\sum_i e{-}y_i\, f_T(x_i) \;=\; \frac{1}{n}\sum_i D_{T+1}(i)\prod_t Z_t \;=\; \prod_t Z_t$$

Idea: make Z_t as small as possible.

$$Z_t \;=\; \sum_{i=1}^{n} D_t(i) e^{-\alpha_t\, y_i\, h_t(x_i)} \;=\; n\,(1-\varepsilon_t)\,e^{-\alpha_t} + n\,\varepsilon_t\,e^{\alpha_t}$$

1. Pick h_t to minimize ε_t.
2. Pick α_t to minimize Z_t.

From weak learner to strong classifier (2)

Pick α_t to minimize Z_t (the magic coefficient)

$$\frac{\partial Z_t}{\partial \alpha_t} = -(1-\varepsilon_t)\, e^{-\alpha_t} + \varepsilon_t\, e^{\alpha_t} = 0 \quad \Longrightarrow \quad \alpha_t = \frac{1}{2} \log \frac{1-\varepsilon_t}{\varepsilon_t}$$

Weak learner assumption: $\gamma_t = \frac{1}{2} - \varepsilon_t$ is positive and small.

$$Z_t \;=\; (1-\varepsilon)\sqrt{\frac{\varepsilon}{1-\varepsilon}} + \varepsilon\sqrt{\frac{1-\varepsilon}{\varepsilon}} \;=\; \sqrt{4\varepsilon(1-\varepsilon)} = \sqrt{1-4\gamma_t^2} \;\leq\; \exp\left(-2\gamma_t^2\right)$$

$$\text{TrainingError}(f_T) \;\leq\; \prod_{t=1}^{T} Z_t \;\leq\; \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right)$$

The training error decreases exponentially if $\inf \gamma_t > 0$.

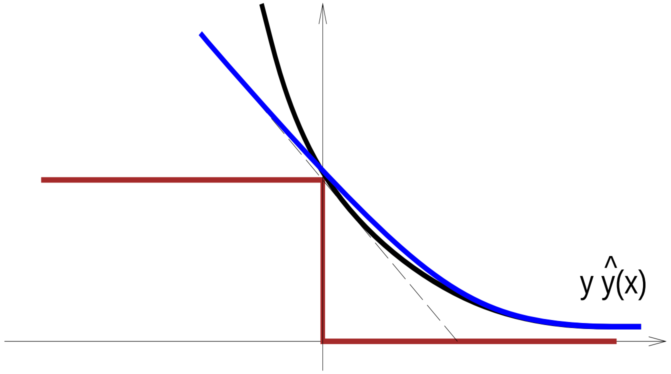
But that does not happen beyond a certain point. . .

Boosting and exponential loss

Proofs are instructive

We obtain the bound

$$\text{TrainingError}(f_T) \;\leq\; \frac{1}{n}\sum_i e^{-y_i H(x_i)} \;=\; \prod_{t=1}^{T} Z_t$$



– without saying how D_t relates to h_t

– without using the value of α_t

Conclusion

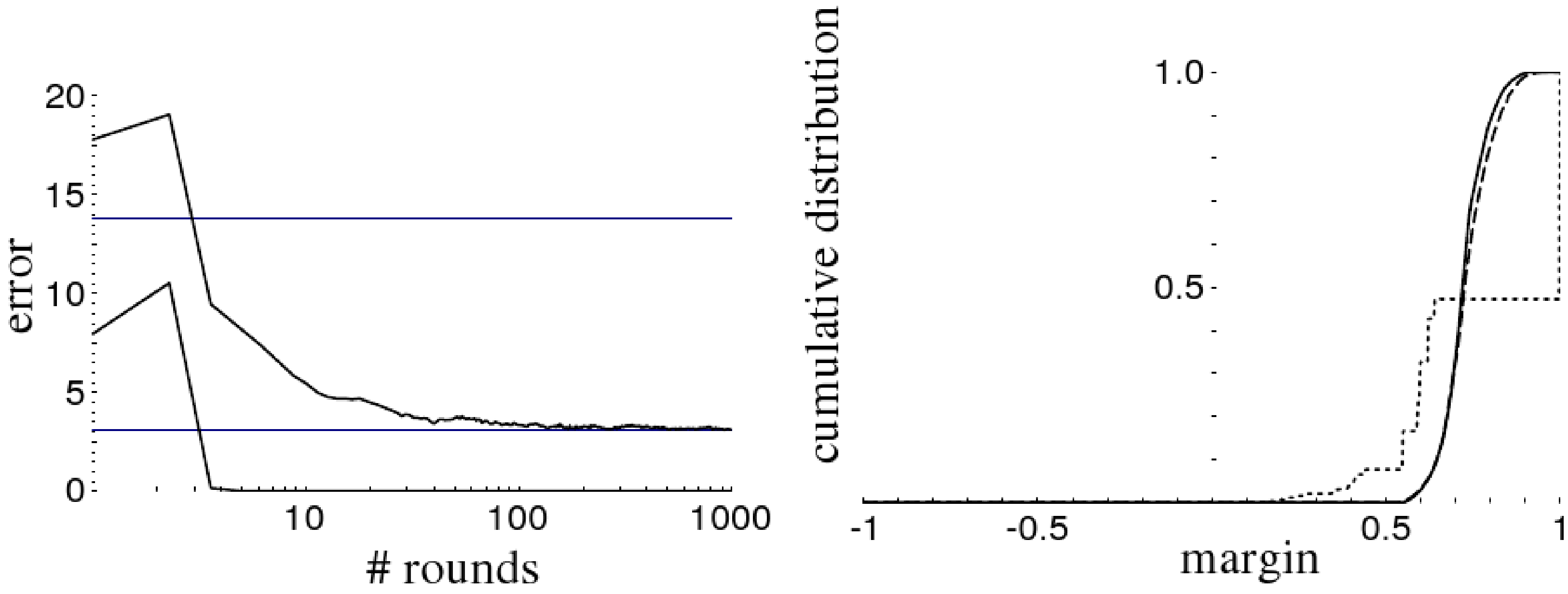
– Round T chooses the h_T and α_T that maximize the exponential loss reduction from f_{T-1} to f_T.

Exercise

– Tweak Adaboost to minimize the log loss instead of the exp loss.

Boosting and margins

$$margin_H(x,y) \;=\; \frac{y\,H(x)}{\sum_t |\alpha_t|} \;=\; \frac{\sum_t \alpha_t\, y\, h_t(x)}{\sum_t |\alpha_t|}$$



Remember support vector machines?

Ensemble learning
Lecture 12

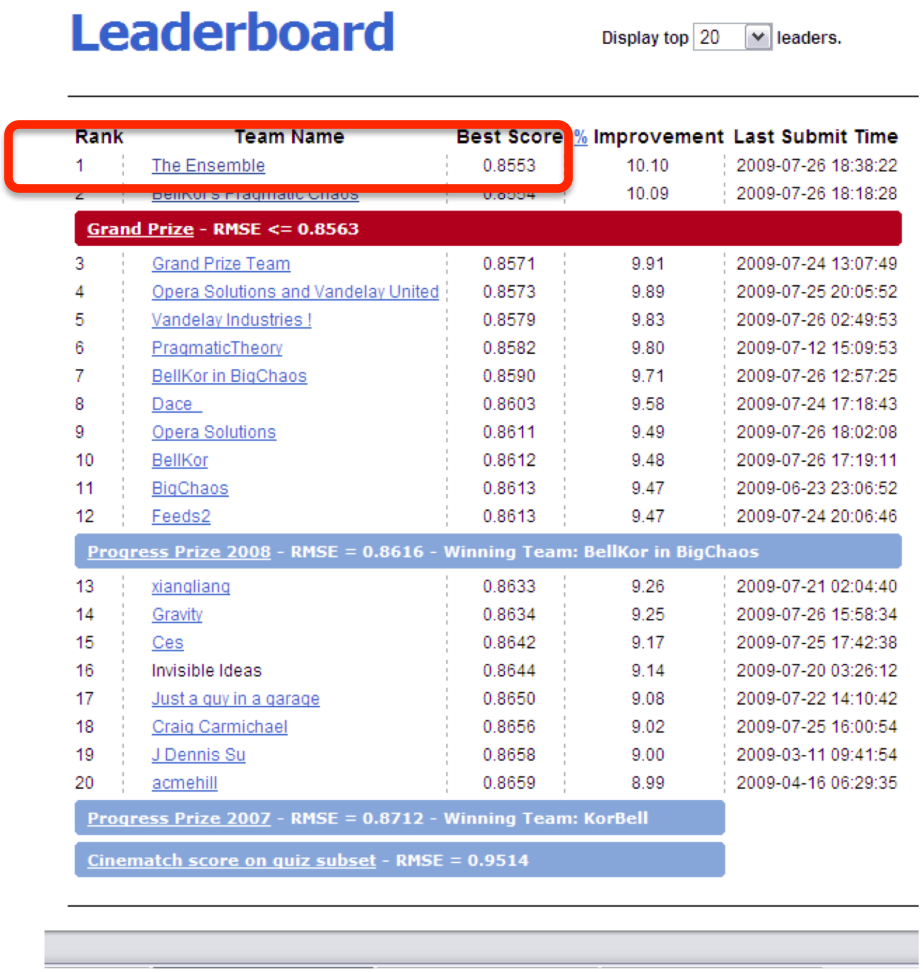
David Sontag

New York University

Slides adapted from Luke Zettlemoyer, Vibhav Gogate, Rob Schapire, and Tommi Jaakkola

Ensemble methods

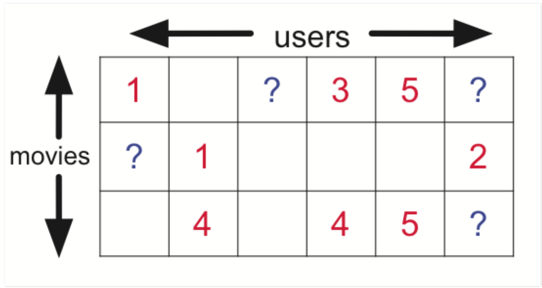
Machine learning competition with a $1 million prize



Leaderboard

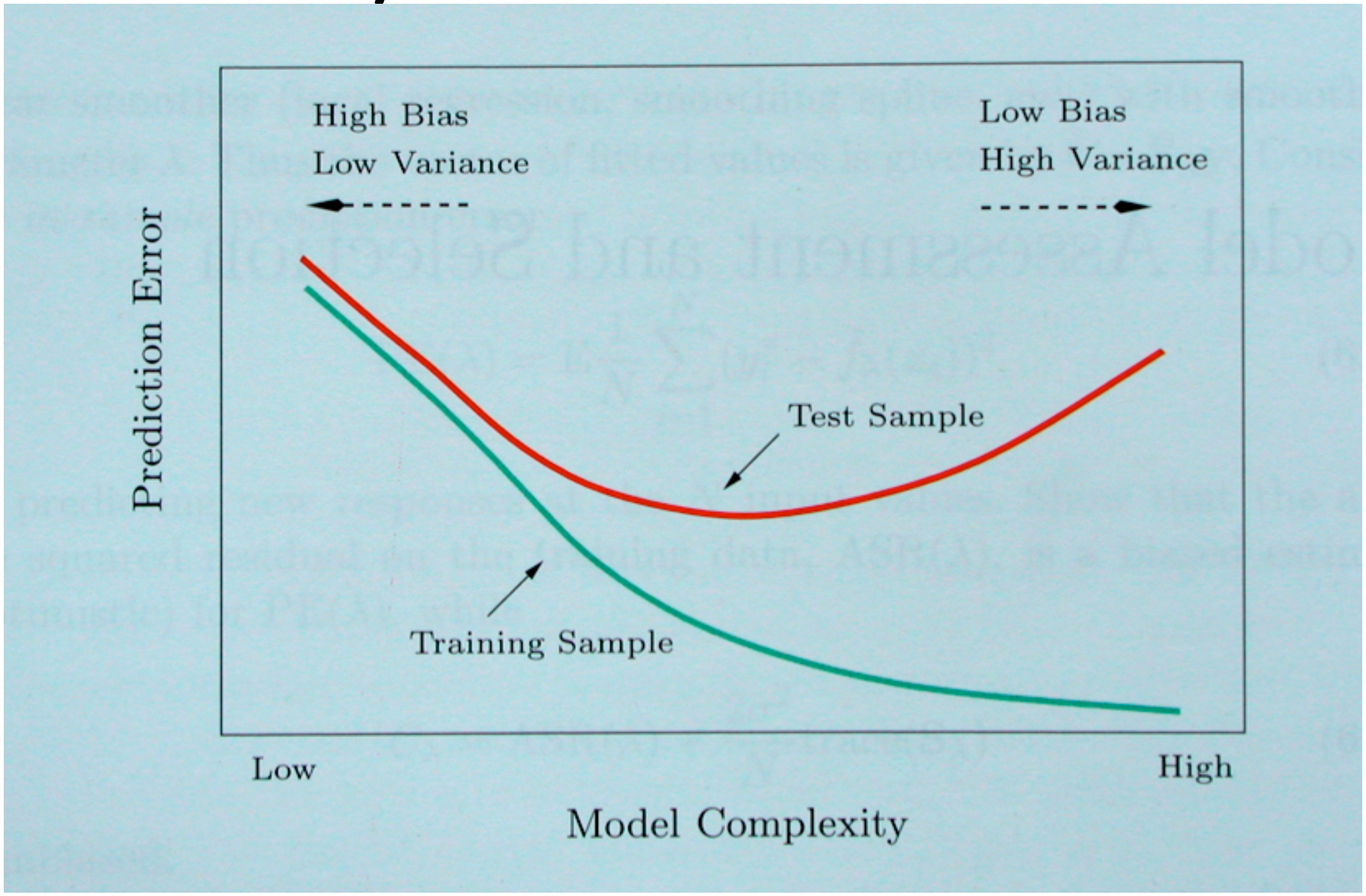
Display top 20 leaders.

Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	The Ensemble	0.8553	10.10	2009-07-26 18:38:22
2	BellKor's Pragmatic Chaos	0.8554	10.09	2009-07-26 18:18:28
Grand Prize - RMSE <= 0.8563				
3	Grand Prize Team	0.8571	9.91	2009-07-24 13:07:49
4	Opera Solutions and Vandelay United	0.8573	9.89	2009-07-25 20:05:52
5	Vandelay Industries !	0.8579	9.83	2009-07-26 02:49:53
6	PragmaticTheory	0.8582	9.80	2009-07-12 15:09:53
7	BellKor in BigChaos	0.8590	9.71	2009-07-26 12:57:25
8	Dace_	0.8603	9.58	2009-07-24 17:18:43
9	Opera Solutions	0.8611	9.49	2009-07-26 18:02:08
10	BellKor	0.8612	9.48	2009-07-26 17:19:11
11	BigChaos	0.8613	9.47	2009-06-23 23:06:52
12	Feeds2	0.8613	9.47	2009-07-24 20:06:46
Progress Prize 2008 - RMSE = 0.8616 - Winning Team: BellKor in BigChaos				
13	xiangliang	0.8633	9.26	2009-07-21 02:04:40
14	Gravity	0.8634	9.25	2009-07-26 15:58:34
15	Ces	0.8642	9.17	2009-07-25 17:42:38
16	Invisible Ideas	0.8644	9.14	2009-07-20 03:26:12
17	Just a guy in a garage	0.8650	9.08	2009-07-22 14:10:42
18	Craig Carmichael	0.8656	9.02	2009-07-25 16:00:54
19	J Dennis Su	0.8658	9.00	2009-03-11 09:41:54
20	acmehill	0.8659	8.99	2009-04-16 06:29:35
Progress Prize 2007 - RMSE = 0.8712 - Winning Team: KorBell				
Cinematch score on quiz subset - RMSE = 0.9514				



	users					
movies	1		?	3	5	?
	?	1				2
		4		4	5	?

Bias/Variance Tradeoff



Hastie, Tibshirani, Friedman "Elements of Statistical Learning" 2001

Reduce Variance Without Increasing Bias

- Averaging reduces variance:

$$Var(\overline{X}) \qquad \frac{Var(X)}{N}$$

(when predictions are **independent**)

Average models to reduce model variance

One problem:

only one training set

where do multiple models come from?

Bagging: Bootstrap Aggregation

- Leo Breiman (1994)
- Take repeated bootstrap samples from training set D.
- *Bootstrap sampling*: Given set D containing N training examples, create D' by drawing N examples at random with replacement from D.

- Bagging:
  - Create k bootstrap samples D_1 … D_k.
  - Train distinct classifier on each D_i.
  - Classify new instance by majority vote / average.

Bagging

- Best case:

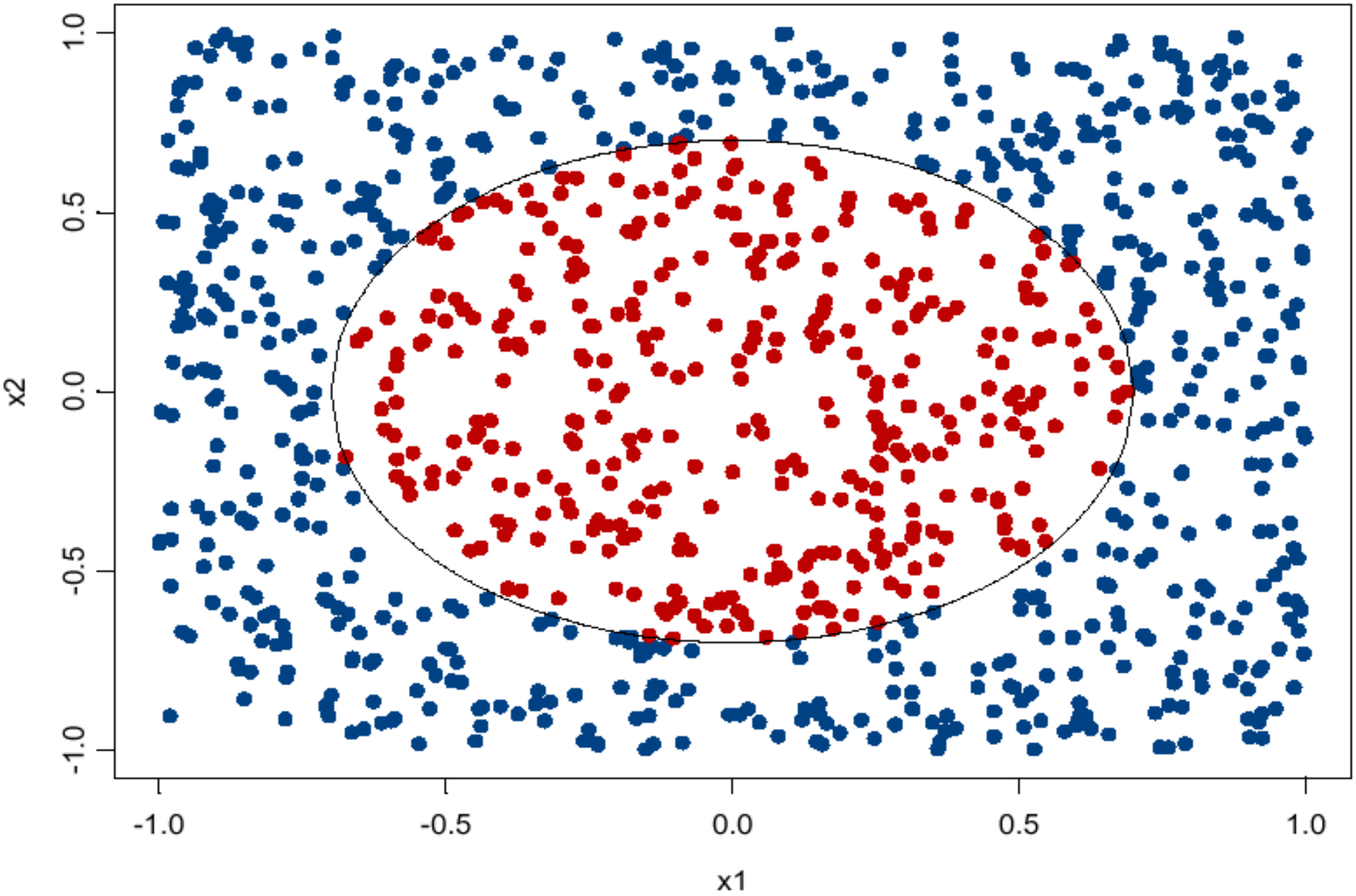
$$Var(Bagging(L(x, D))) \quad \frac{Variance(L(x, D))}{N}$$

In practice:

models are correlated, so reduction is smaller than 1/N

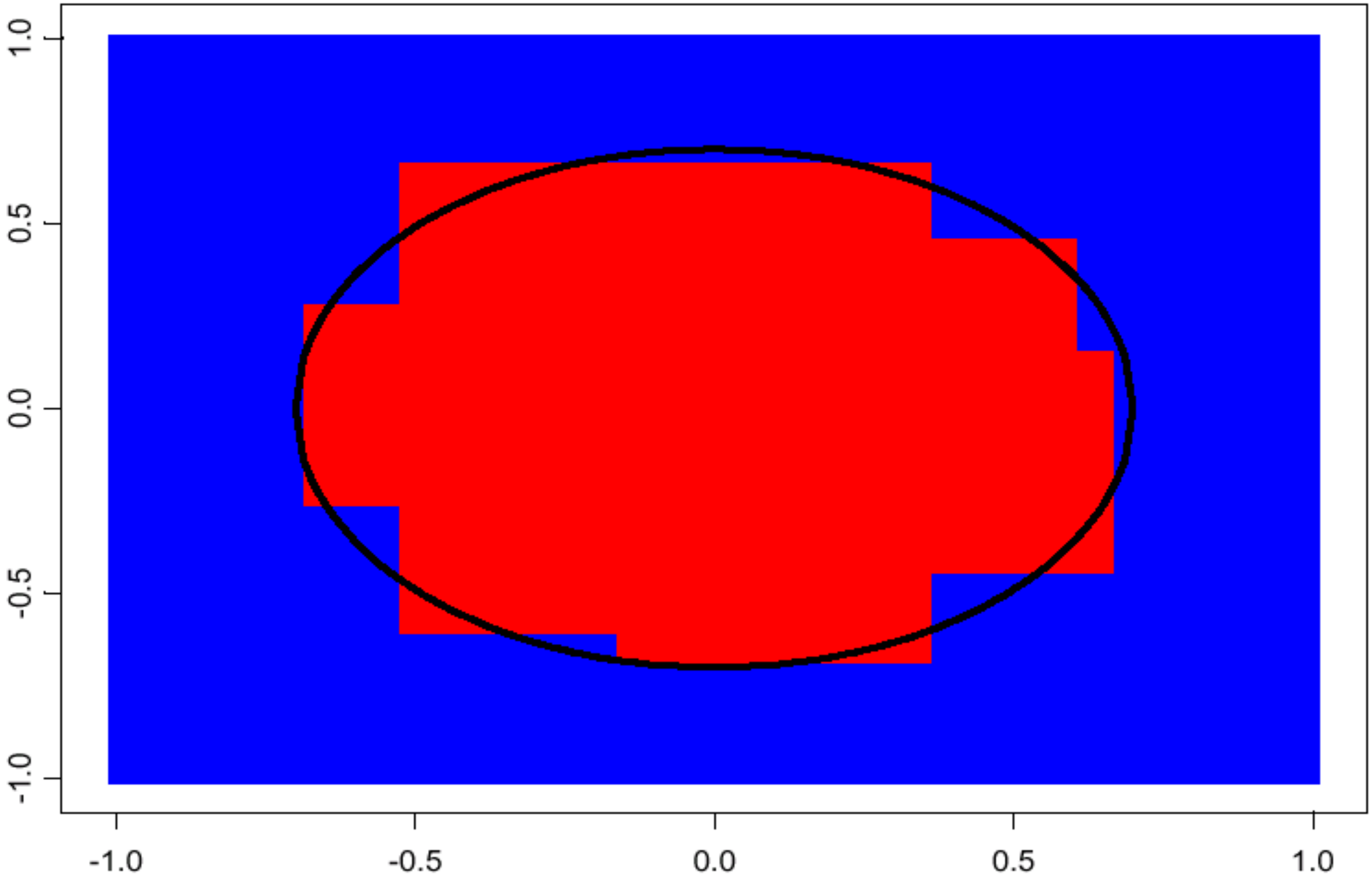
variance of models trained on fewer training cases usually somewhat larger

Bagging Example

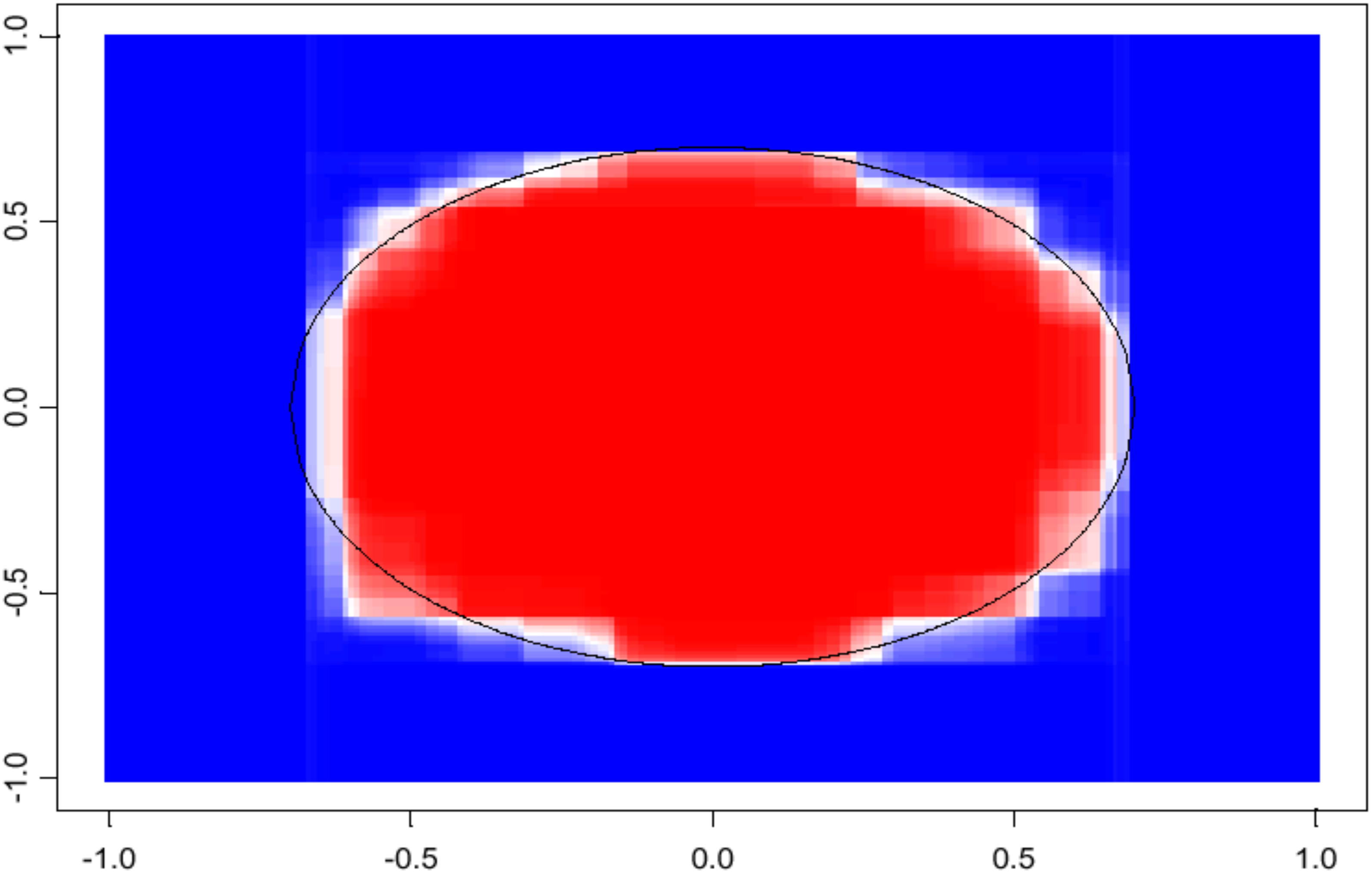


decision tree learning algorithm; very similar to ID3

CART decision boundary



100 bagged trees



shades of blue/red indicate strength of vote for particular classification

Reduce Bias^2 and Decrease Variance?

- Bagging reduces variance by averaging
- Bagging has little effect on bias
- Can we average *and* reduce bias?
- Yes:

  - Boosting

Theory and Applications of Boosting

Rob Schapire

Example: "How May I Help You?"

[Gorin et al.]

- goal: automatically categorize type of call requested by phone customer (Collect, CallingCard, PersonToPerson, etc.)
  - `yes I'd like to place a collect call long distance please` (Collect)
  - `operator I need to make a call but I need to bill it to my office` (ThirdNumber)
  - `yes I'd like to place a call on my master card please` (CallingCard)
  - `I just called a number in sioux city and I musta rang the wrong number because I got the wrong party and I would like to have that taken off of my bill` (BillingCredit)
- observation:
  - easy to find "rules of thumb" that are "often" correct
    - e.g.: "IF '`card`' occurs in utterance
      THEN predict 'CallingCard' "
  - hard to find single highly accurate prediction rule

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

Key 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% (in two-class setting) ["weak learning assumption"]
  - given sufficient data, a boosting algorithm can provably construct single classifier with very high accuracy, say, 99%

Preamble: Early History

Strong and Weak Learnability

- boosting's roots are in "PAC" learning model [Valiant '84]
- get random examples from unknown, arbitrary distribution
- strong PAC learning algorithm:
  - for any distribution
    with high probability
    given polynomially many examples (and polynomial time)
    can find classifier with arbitrarily small generalization error
- weak PAC learning algorithm
  - same, but generalization error only needs to be slightly better than random guessing ($\frac{1}{2} - \gamma$)
- [Kearns & Valiant '88]:
  - does weak learnability imply strong learnability?

If Boosting Possible, Then...

- can use (fairly) wild guesses to produce highly accurate predictions
- if can learn "part way" then can learn "all the way"
- should be able to improve any learning algorithm
- for any learning problem:
  - either can always learn with nearly perfect accuracy
  - or there exist cases where cannot learn even slightly better than random guessing

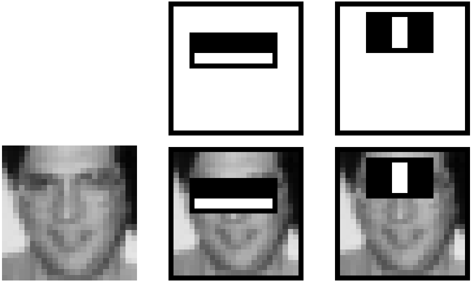
First Boosting Algorithms

- [Schapire '89]:
  - first provable boosting algorithm
- [Freund '90]:
  - "optimal" algorithm that "boosts by majority"
- [Drucker, Schapire & Simard '92]:
  - first experiments using boosting
  - limited by practical drawbacks
- [Freund & Schapire '95]:
  - introduced "AdaBoost" algorithm
  - strong practical advantages over previous boosting algorithms

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

Basic Algorithm and Core Theory

- introduction to AdaBoost
- analysis of training error
- analysis of test error and the margins theory
- experiments and applications

Basic Algorithm and Core Theory

- introduction to AdaBoost
- analysis of training error
- analysis of test error and the margins theory
- experiments and applications

A Formal Description of Boosting

- given training set $(x_1, y_1), \ldots, (x_m, y_m)$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- for $t = 1, \ldots, T$:
    - construct distribution D_t on $\{1, \ldots, m\}$
    - find weak classifier ("rule of thumb")

      $$h_t : X \to \{-1, +1\}$$

      with error ϵ_t on D_t:

      $$\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$$
- output final/combined classifier H_{final}

AdaBoost

[with Freund]

- constructing D_t:
    - $D_1(i) = 1/m$
    - given D_t and h_t:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

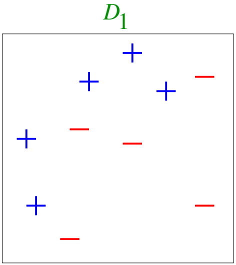
$$= \frac{D_t(i)}{Z_t} \exp(-\alpha_t \, y_i \, h_t(x_i))$$

where $Z_t = \text{normalization factor}$

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) > 0$$

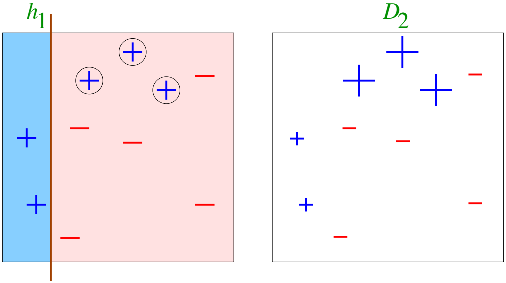
- final classifier:
    - $H_{\text{final}}(x) = \operatorname{sign}\left(\sum_t \alpha_t h_t(x)\right)$

Toy Example



weak classifiers = vertical or horizontal half-planes

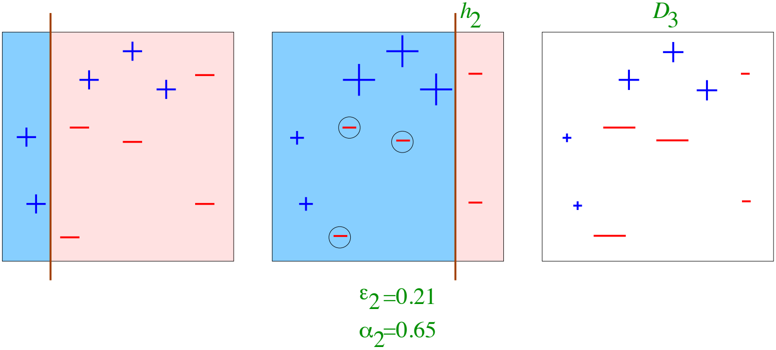
Round 1



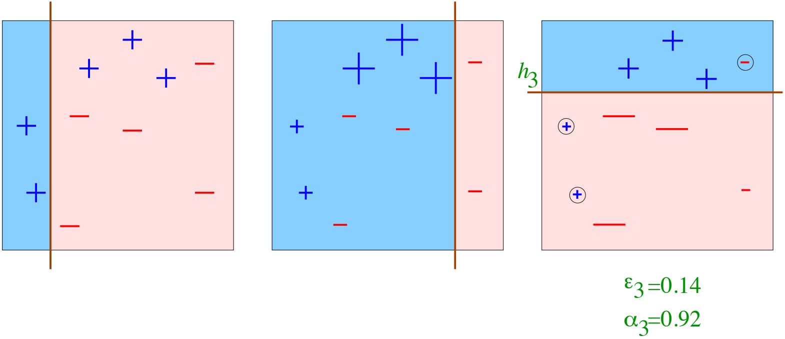
$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$

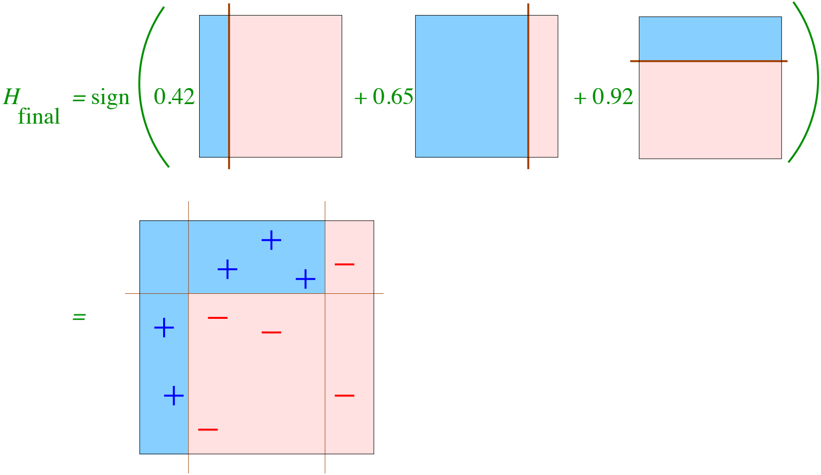
Round 2



Round 3



Final Classifier



$H_{\text{final}} = \text{sign}\left(0.42 \quad + 0.65 \quad + 0.92 \quad\right)$
=
+
+
+
+
+
−
−
−
−
−

Voted combination of classifiers

- The general problem here is to try to combine many simple "weak" classifiers into a single "strong" classifier
- We consider voted combinations of simple binary ± 1 component classifiers

$$h_m(\mathbf{x}) = \alpha_1 \, h(\mathbf{x};\theta_1) + \ldots + \alpha_m \, h(\mathbf{x};\theta_m)$$

where the (non-negative) votes α_i can be used to emphasize component classifiers that are more reliable than others

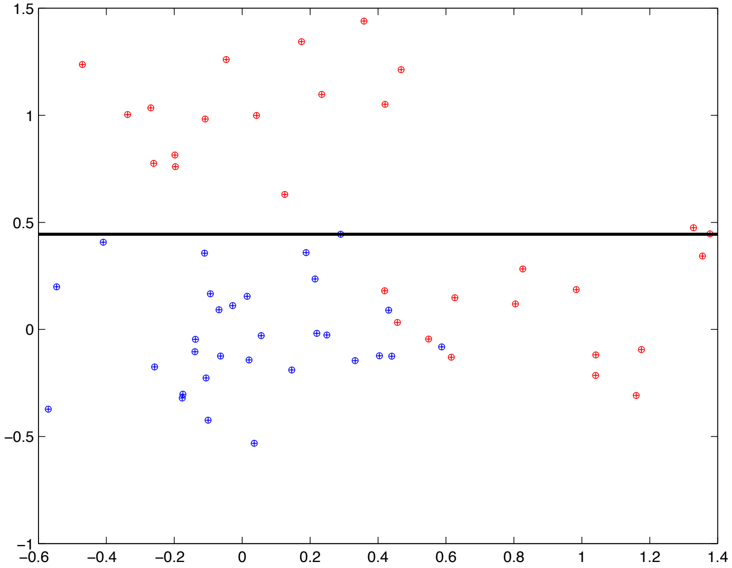
Components: decision stumps

- Consider the following simple family of component classifiers generating ± 1 labels:

$$h(\mathbf{x};\theta) = \text{sign}(\, w_1\, x_k - w_0\,)$$

where $\theta = \{k, w_1, w_0\}$. These are called *decision stumps*.

- Each decision stump pays attention to only a single component of the input vector



Voted combination cont'd

- We need to define a loss function for the combination so we can determine which new component $h(\mathbf{x};\theta)$ to add and how many votes it should receive

$$h_m(\mathbf{x}) = \alpha_1 h(\mathbf{x};\theta_1) + \ldots + \alpha_m h(\mathbf{x};\theta_m)$$

- While there are many options for the loss function we consider here only a simple exponential loss

$$\exp\{\, -y\, h_m(\mathbf{x})\,\}$$

Modularity, errors, and loss

- Consider adding the m^{th} component:

$$\sum_{i=1}^{n} \exp\{\, -y_i[h_{m-1}(\mathbf{x}_i) + \alpha_m h(\mathbf{x}_i;\theta_m)]\,\}$$

$$= \sum_{i=1}^{n} \exp\{\, -y_i h_{m-1}(\mathbf{x}_i) - y_i \alpha_m h(\mathbf{x}_i;\theta_m)\,\}$$

Modularity, errors, and loss

- Consider adding the m^{th} component:

$$\sum_{i=1}^{n} \exp\{\, -y_i[h_{m-1}(\mathbf{x}_i) + \alpha_m h(\mathbf{x}_i;\theta_m)]\,\}$$

$$= \sum_{i=1}^{n} \exp\{\, -y_i h_{m-1}(\mathbf{x}_i) - y_i\alpha_m h(\mathbf{x}_i;\theta_m)\,\}$$

$$= \sum_{i=1}^{n} \underbrace{\exp\{\, -y_i h_{m-1}(\mathbf{x}_i)\}}_{\text{fixed at stage } m} \exp\{-y_i\alpha_m h(\mathbf{x}_i;\theta_m)\,\}$$

Modularity, errors, and loss

- Consider adding the m^{th} component:

$$\begin{aligned}
&\sum_{i=1}^{n} \exp\{\, -y_i[h_{m-1}(\mathbf{x}_i) + \alpha_m h(\mathbf{x}_i;\theta_m)]\,\} \\
&= \sum_{i=1}^{n} \exp\{\, -y_i h_{m-1}(\mathbf{x}_i) - y_i\alpha_m h(\mathbf{x}_i;\theta_m)\,\} \\
&= \sum_{i=1}^{n} \underbrace{\exp\{\, -y_i h_{m-1}(\mathbf{x}_i)\}}_{\text{fixed at stage } m} \exp\{-y_i\alpha_m h(\mathbf{x}_i;\theta_m)\,\} \\
&= \sum_{i=1}^{n} W_i^{(m-1)} \exp\{-y_i\alpha_m h(\mathbf{x}_i;\theta_m)\,\}
\end{aligned}$$

So at the m^{th} iteration the new component (and the votes) should optimize a weighted loss (weighted towards mistakes).

Empirical exponential loss cont'd

- To increase modularity we'd like to further decouple the optimization of $h(\mathbf{x};\theta_m)$ from the associated votes α_m
- To this end we select $h(\mathbf{x};\theta_m)$ that optimizes the rate at which the loss would decrease as a function of α_m

$$\frac{\partial}{\partial\alpha_m}\Big|_{\alpha_m=0}\sum_{i=1}^{n} W_i^{(m-1)}\exp\{-y_i\alpha_m h(\mathbf{x}_i;\theta_m)\,\} =$$

$$\left[\sum_{i=1}^{n} W_i^{(m-1)}\exp\{-y_i\alpha_m h(\mathbf{x}_i;\theta_m)\,\}\cdot\big(-y_i h(\mathbf{x}_i;\theta_m)\big)\right]_{\alpha_m=0}$$

$$=\left[\sum_{i=1}^{n} W_i^{(m-1)}\big(-y_i h(\mathbf{x}_i;\theta_m)\big)\right]$$

Empirical exponential loss cont'd

- We find $h(\mathbf{x}; \hat{\theta}_m)$ that minimizes

$$- \sum_{i=1}^{n} W_i^{(m-1)} y_i h(\mathbf{x}_i; \theta_m)$$

We can also normalize the weights:

$$\begin{aligned} - \sum_{i=1}^{n} \frac{W_i^{(m-1)}}{\sum_{j=1}^{n} W_j^{(m-1)}} y_i h(\mathbf{x}_i; \theta_m) \\ = \; - \sum_{i=1}^{n} \tilde{W}_i^{(m-1)} y_i h(\mathbf{x}_i; \theta_m) \end{aligned}$$

so that $\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} = 1$.

Selecting a new component: summary

- We find $h(\mathbf{x};\hat{\theta}_m)$ that minimizes

$$-\sum_{i=1}^{n} \tilde{W}_i^{(m-1)}\, y_i h(\mathbf{x}_i;\theta_m)$$

where $\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} = 1$.

- α_m is subsequently chosen to minimize

$$\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} \exp\{-y_i\alpha_m h(\mathbf{x}_i;\hat{\theta}_m)\,\}$$

The AdaBoost algorithm

0) Set $\tilde{W}_i^{(0)} = 1/n$ for $i = 1, \ldots, n$

1) At the m^{th} iteration we find (any) classifier $h(\mathbf{x}; \hat{\theta}_m)$ for which the *weighted classification error* ϵ_m

$$\epsilon_m = 0.5 - \frac{1}{2} \left(\sum_{i=1}^{n} \tilde{W}_i^{(m-1)} y_i h(\mathbf{x}_i; \hat{\theta}_m) \right)$$

is better than chance.

2) The new component is assigned votes based on its error:

$$\hat{\alpha}_m = 0.5 \log(\, (1 - \epsilon_m)/\epsilon_m \,)$$

3) The weights are updated according to (Z_m is chosen so that the new weights $\tilde{W}_i^{(m)}$ sum to one):

$$\tilde{W}_i^{(m)} = \frac{1}{Z_m} \cdot \tilde{W}_i^{(m-1)} \cdot \exp\{ -y_i \hat{\alpha}_m h(\mathbf{x}_i; \hat{\theta}_m) \}$$