Ensemble Methods for Machine Learning

COMBINING CLASSIFIERS: ENSEMBLE APPROACHES

Common Ensemble classifiers

- Bagging/Random Forests
- “Bucket of models”
- Stacking
- Boosting

Ensemble classifiers we've studied so far

- Bagging
  - Build many bootstrap replicates of the data
  - Train many classifiers
  - Vote the results

Bagged trees to random forests

- Bagged decision trees:
  - Build many bootstrap replicates of the data
  - Train many tree classifiers
  - Vote the results
- Random forest:
  - Build many bootstrap replicates of the data
  - Train many tree classifiers (no pruning)
    - For each split, restrict to a *random subset* of the original feature set. (Usually a random subset of size *sqrt(d)* where there are *d* original features.)
  - Vote the results
  - Like bagged decision trees, fast and easy to use

A “bucket of models”

- Scenario:
  - I have three learners
    - Logistic Regression, Naïve Bayes, Backprop
  - I’m going to embed a learning machine in some system (eg, “genius”-like music recommender)
    - Each system has a different user ➔ different dataset D
    - In pilot studies there’s no single best system
    - Which learner do embed to get the best result?
    - Can I combine all of them somehow and do better?

Simple learner combination: A "bucket of models"

- Input:
  - your top *T* favorite learners (or tunings)
    - $L_1,\ldots,L_T$
  - A dataset *D*
- Learning algorithm:
  - Use 10-CV to estimate the error of $L_1,\ldots,L_T$
  - Pick the best (lowest 10-CV error) learner L*
  - Train L* on *D* and return its hypothesis h*

Pros and cons of a "bucket of models"

- Pros:
  - Simple
  - Will give results not much worse than the best of the "**base learners**"
- Cons:
  - What if there's not a single best learner?
- Other approaches:
  - Vote the hypotheses (how would you weight them?)
  - Combine them some other way?
  - How about *learning* to combine the hypotheses?

Common Ensemble classifiers

- Bagging/Random Forests
- “Bucket of models”
- Stacking
- Boosting

Stacked learners: first attempt

- Input:
  - your top *T* favorite learn
    - $L_1,\ldots,L_T$
  - A dataset *D* containing (
- Learning algorithm:
  - Train $L_1,\ldots,L_T$ on *D* to g
  - Create a new dataset D containing (**x**',y'),….
    - **x**' is a vector of the T predictions $h_1(\mathbf{x}),\ldots,h_T(\mathbf{x})$
    - y is the label y for **x**
  - Train YFCL on D' to get
- To predict on a new **x:**
  - Construct **x**' as before and predict h'(**x**')

Problem: if L_i overfits the data D, then $h_i(\mathbf{x})$ could be *almost always* the same as y in D'.

But that won't be the case on an out-of-sample-the test example **x.**

The fix: make an **x**' in D' look more like the *out-of-sample test cases.*

Stacked learners: the right way

- Input:
  - your top *T* favorite learn
  - A dataset *D* containing
- Learning algorithm:
  - Train $L_1,\ldots,L_T$ on *D* to get $h_1,\ldots,h_T$
  - Also run 10-CV and collect the *CV test predictions* $(\mathbf{x},L_i(D_{-\mathbf{x}},\mathbf{x}))$ for each example **x** and each learner L_i
  - Create a new dataset D' containing (**x'**,y'),….
    - **x'** is the *CV test predictions* $(L_1(D_{-\mathbf{x}},\mathbf{x}), L_2(D_{-\mathbf{x}},\mathbf{x}),$ …
    - y is the label y for **x**
  - Train YFCL on D' to get h' --- which combines the predictions!
- To predict on a new **x:**
  - Construct **x'** using $h_1,\ldots,h_T$ (as before) and predict h'(**x'**)

$D_{-\mathbf{x}}$ is the CV training set for **x**
$L_i(D^*,\mathbf{x})$ is the label predicted by the hypothesis learned from D* by L_i

we don't do any CV at prediction time

Pros and cons of stacking

- Pros:
  - Fairly simple
  - Slow, but easy to parallelize
- Cons:
  - What if there's not a single best *combination scheme*?
  - E.g.: for movie recommendation sometimes L1 is best for users with many ratings and L2 is best for users with few ratings.

Multi-level stacking/blending

$D_{-\mathbf{x}}$ is the CV training set for **x** and $L_i(D^*,\mathbf{x})$ is a predicted label

- Learning algorithm:
  - Train $L_1,\ldots,L_T$ on *D* to get $h_1,\ldots,h_T$
  - Also run 10-CV and collect the *CV test predictions* $(\mathbf{x},L_i(D_{-\mathbf{x}},\mathbf{x}))$ for each example **x** and each learner L_i
  - Create a new dataset D' containing (**x'**,y'),….
    - **x'** is the *CV test predictions* $(L_1(D_{-\mathbf{x}},\mathbf{x}), L_2(D_{-\mathbf{x}},\mathbf{x}),$ … *combined with additional features from* ***x*** (e.g., numRatings, userAge, …)
    - y is the label y for **x**
  - Train YFCL on D' to get h' --- which combines the predictions!
- To predict on a new **x:**
  - Construct **x'** using $h_1,\ldots,h_T$ (as before) and predict h'(**x'**)

"meta-features"

might create features like "$L_1(D_{-\mathbf{x}},\mathbf{x})$=y* and numRatings>20"

where the choice of classifier to rely on depends on meta-features of **x**

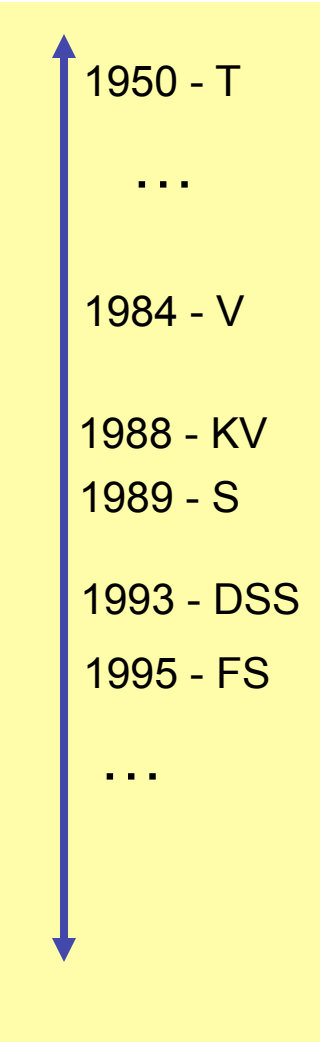
Comments

- Ensembles based on blending/stacking were key approaches used in the netflix competition
  - Winning entries blended many types of classifiers
- Ensembles based on stacking are the main architecture used in Watson
  - Not all of the base classifiers/rankers are learned, however; some are hand-programmed.

Common Ensemble classifiers

- Bagging/Random Forests
- “Bucket of models”
- Stacking
- Boosting

Thanks to A Short Introduction to Boosting. Yoav Freund, Robert E. Schapire, *Journal of Japanese Society for Artificial Intelligence,14(5):771-780, September, 1999*



Boosting

1936 - T

1950 - T

…

1984 - V

1988 - KV

1989 - S

1993 - DSS

1995 - FS

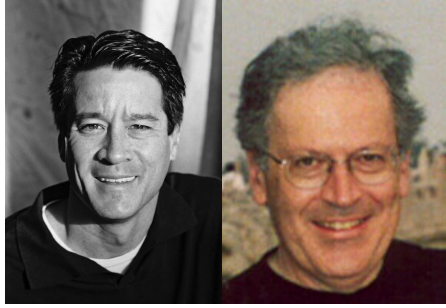
…

- Valiant CACM 1984 and PAC-learning: partly inspired by Turing

	Formal	Informal
AI	*Valiant (1984)*	Turing test (1950)
Complexity	Turing machine (1936)	

Question: what sort of AI questions can we *formalize* and study with *formal methods?*

“Weak” pac-learning (Kearns & Valiant 88)



1950 - T

…

1984 - V

1988 - KV

1989 - S

1993 - DSS

1995 - FS

…

(PAC learning)

Strong Learning	**Weak Learning**
$\exists$ algorithm A	$\exists$ algorithm A
$\forall c \in \mathcal{C}$	$\exists \gamma > 0$
$\forall D$	$\forall c \in \mathcal{C}$
$\forall \epsilon > 0$	$\forall D$
$\forall \delta > 0$	$\forall \epsilon \geq \frac{1}{2} - \gamma$ say, ε=0.49
	$\forall \delta > 0$
A produces $h \in \mathcal{H}$:	A produces $h \in \mathcal{H}$:
$Pr[err(h) > \epsilon] \leq \delta$	$Pr[err(h) > \epsilon] \leq \delta$

“Weak” PAC-learning is equivalent to “strong” PAC-learning (!) (Schapire 89)

1950 - T

…

1984 - V

1988 - KV

1989 - S

1993 - DSS

1995 - FS

…

(PAC learning)

Strong Learning	=	**Weak Learning**
$\exists$ algorithm A		$\exists$ algorithm A
$\forall c \in \mathcal{C}$		$\exists \gamma > 0$
$\forall D$		$\forall c \in \mathcal{C}$
$\forall \epsilon > 0$		$\forall D$
$\forall \delta > 0$		$\forall \epsilon \geq \frac{1}{2} - \gamma$ say, ε=0.49
		$\forall \delta > 0$
A produces $h \in \mathcal{H}$:		A produces $h \in \mathcal{H}$:
$Pr[err(h) > \epsilon] \leq \delta$		$Pr[err(h) > \epsilon] \leq \delta$

"Weak" PAC-learning is equivalent to "strong" PAC-learning (!) (Schapire 89)

1950 - T

…

1984 - V

1988 - KV

1989 - S

1993 - DSS

1995 - FS

…

- The basic idea exploits the fact that you can learn a little on *every* distribution:
  - Learn *h1* from *D0* with error < 49%
  - Modify *D0* so that *h1* has error 50% (call this *D1*)
    - If heads wait for an example x where *h1(x)=f(x),* otherwise wait for an example where *h1(x)!=f(x)*.
  - Learn *h2* from *D1* with error < 49%
  - Modify *D1* so that *h1* and *h2* always disagree (call this *D2)*
  - Learn *h3* from *D2* with error <49%.
  - Now vote *h1,h2,* and *h3.* ***This has error <u>better</u> than any of the "weak" hypotheses*** *h1,h2 or h3.*
  - Repeat this as needed to lower the error rate more….

Boosting can actually help experimentally…but… (Drucker, Schapire, Simard)

1950 - T

…

1984 - V

1988 - KV

1989 - S

1993 - DSS

1995 - FS

…

- The basic idea exploits the fact that you can learn a little on *every* distribution:
  - Learn *h1* from *D0* v
  - Modify *D0* so that *h1* has error 50% (call this *D1*)
    - If heads **wait** for an example x where *h1(x)=f(x),* otherwise **wait** for an example where *h1(x)!=f(x)*.
  - Learn *h2* from *D1* with error < 49%
  - Modify *D1* so that *h1* and *h2* always disagree (call this *D2)*
  - L
  - N ***of the "weak" hypotheses*** *h1,h2 or h3.*
  - Repeat this as needed to lower the error rate more….

Very wasteful of examples

wait for examples where they disagree !?

AdaBoost (Freund and Schapire)

1950 - T

...

1984 - V

1988 - KV

1989 - S

1993 - DSS

1995 - FS

...

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final classifier:

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

Figure 1: The boosting algorithm AdaBoost.

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final classifier:

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

Figure 1: The boosting algorithm AdaBoost.

Sample **with replacement**

Increase weight of x_i if h_t is wrong, decrease weight if h_t is right.

Linear combination of base hypotheses - best weight α_t depends on error of h_t.

AdaBoost: Adaptive Boosting (Freund & Schapire, 1995)

1950 - T

...

1984 - V

1988 - KV

1989 - S

1993 - DSS

1995 - FS

...

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$.
- Update:

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

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

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

Figure 1: The boosting algorithm AdaBoost.

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

Theoretically, one can upper bound the training error of boosting.

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \{-1, +1\}$ with error

$$\epsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$.
- Update:

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

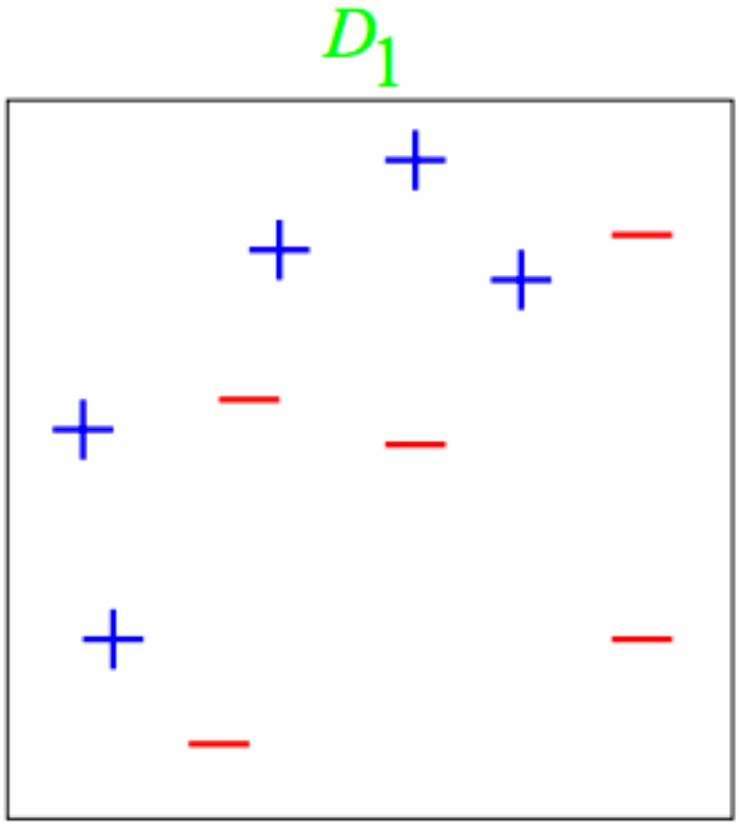
where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

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

Boosting: A toy example

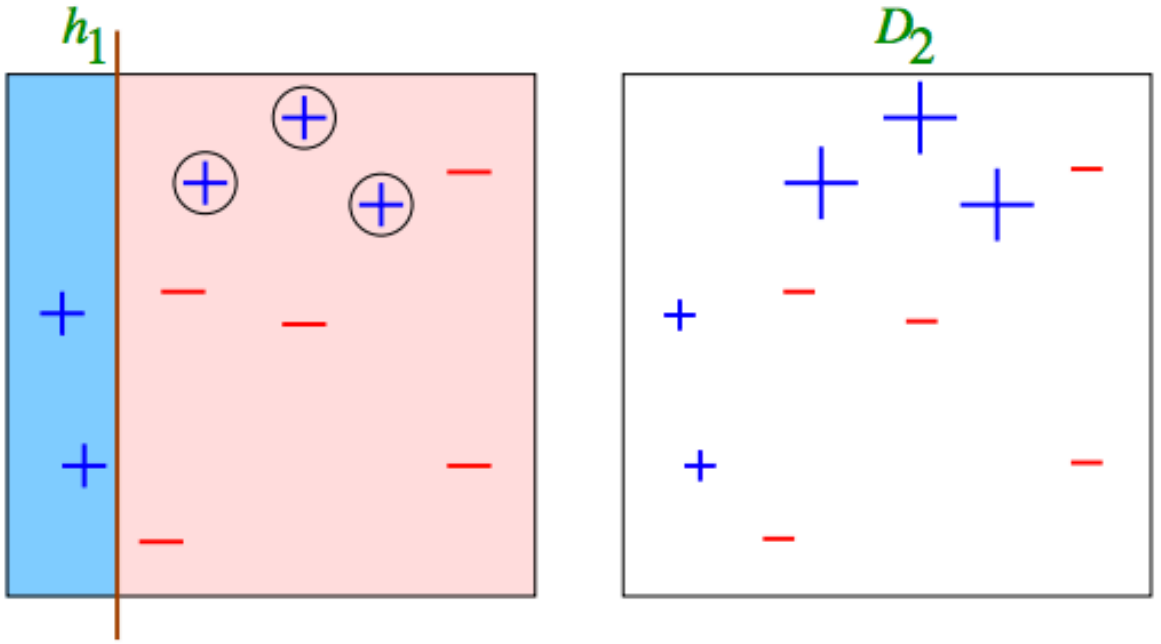
Thanks, Rob Schapire



Thanks, Rob Schapire

Boosting: A toy example

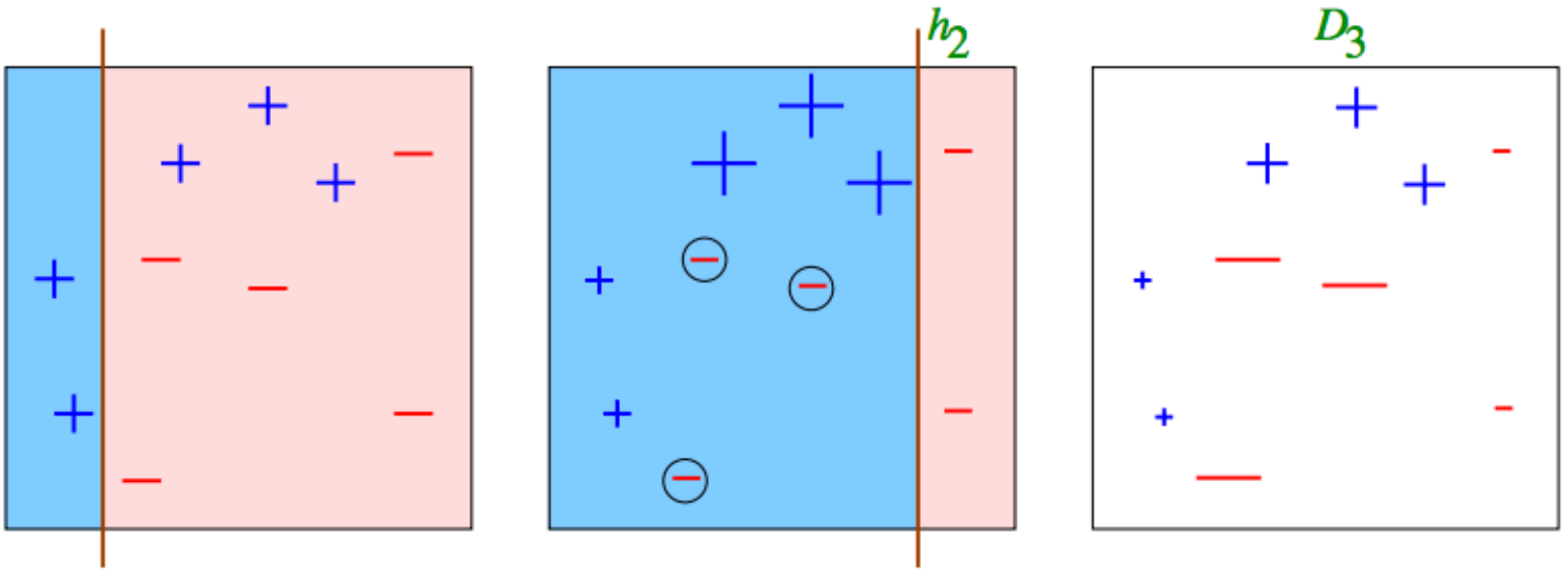
Round 1



Thanks, Rob Schapire

Boosting: A toy example

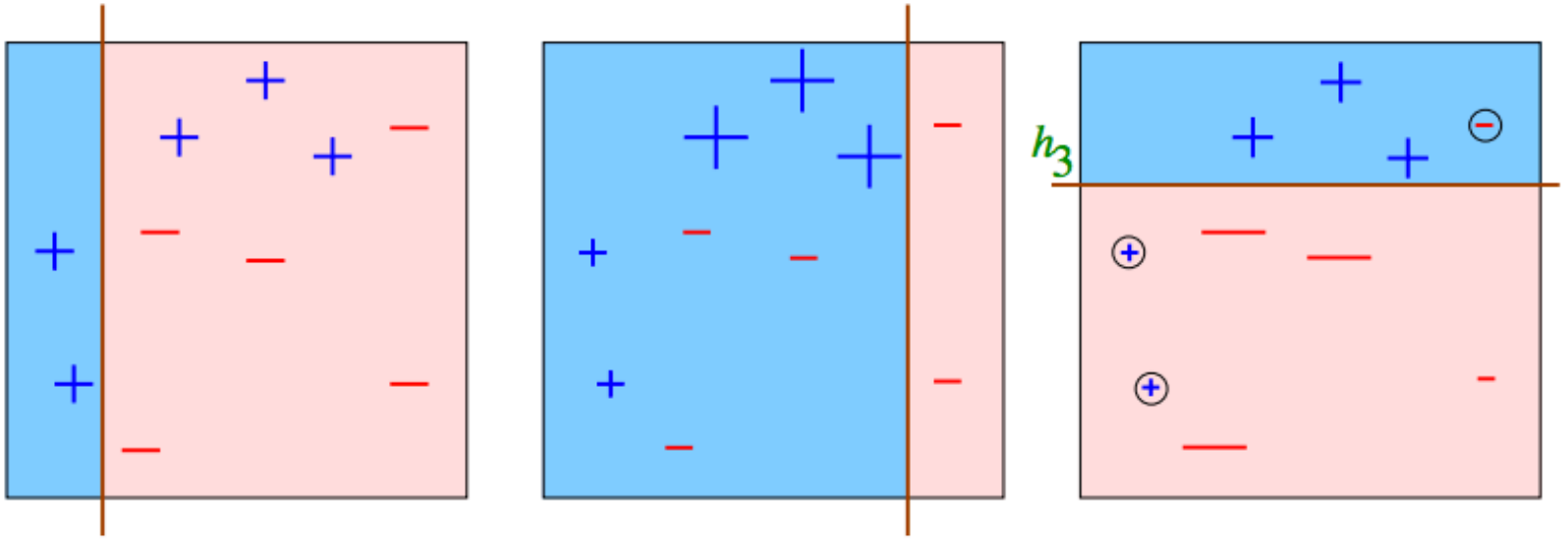
Round 2



Thanks, Rob Schapire

Boosting: A toy example

Round 3



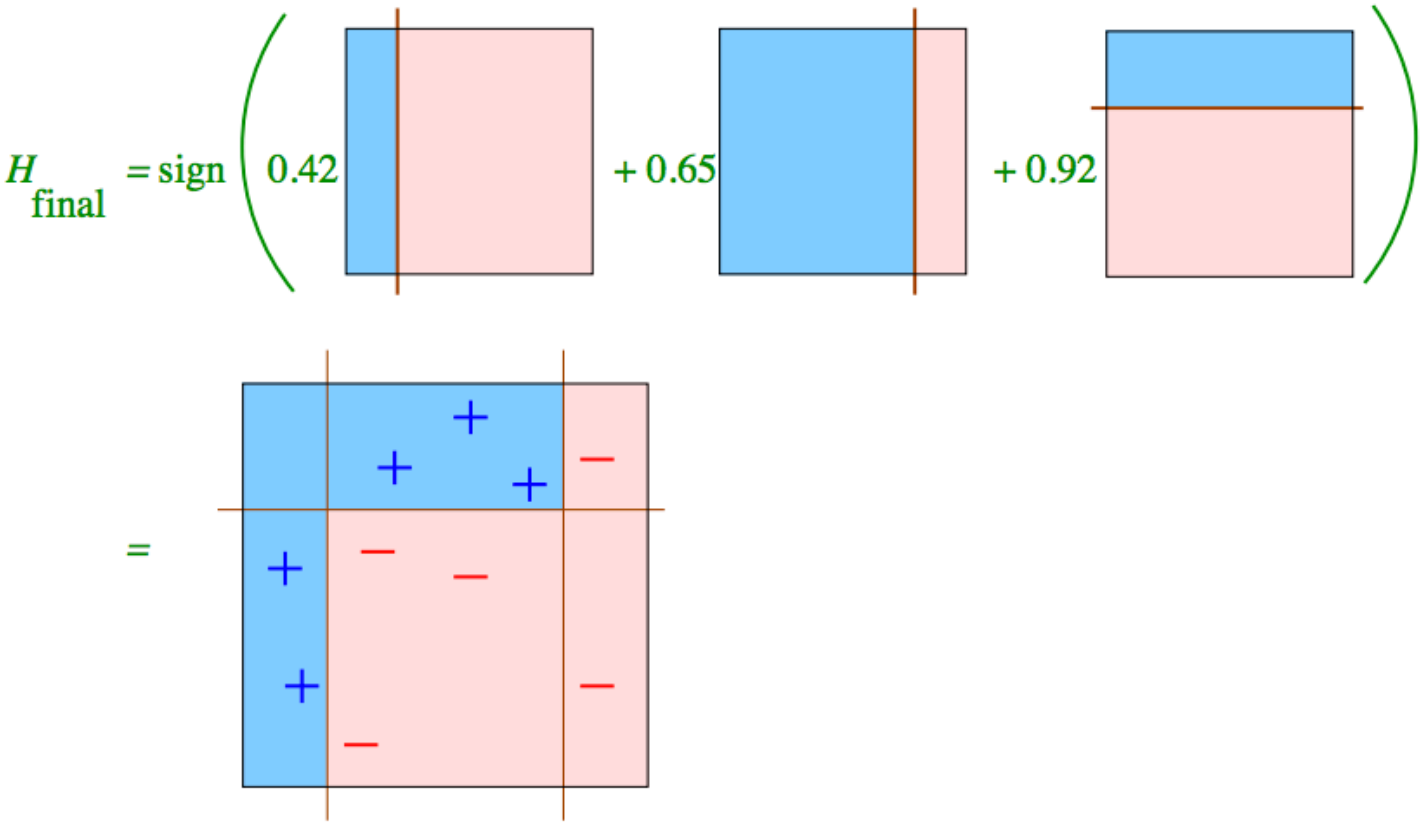
$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

Thanks, Rob Schapire

Boosting: A toy example

Final Classifier



Boosting improved decision trees…

1950 - T

…

1984 - V

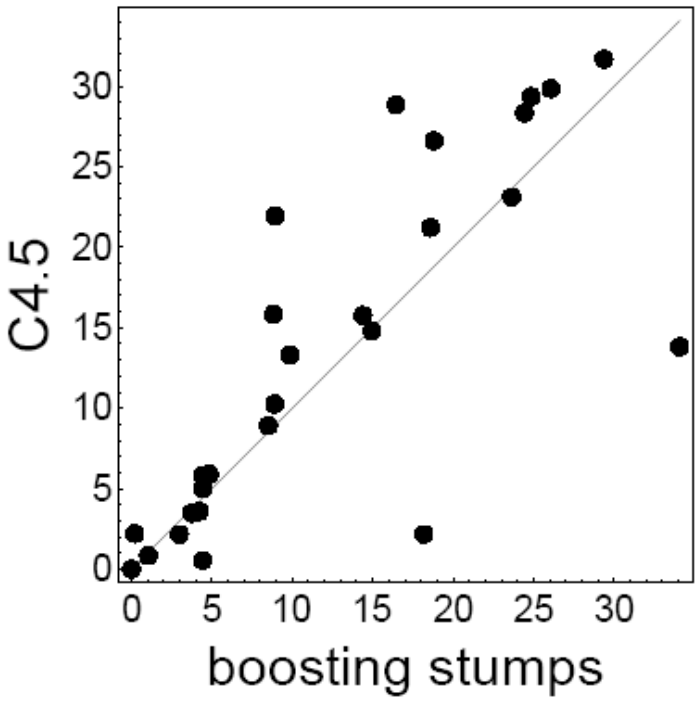
1988 - KV

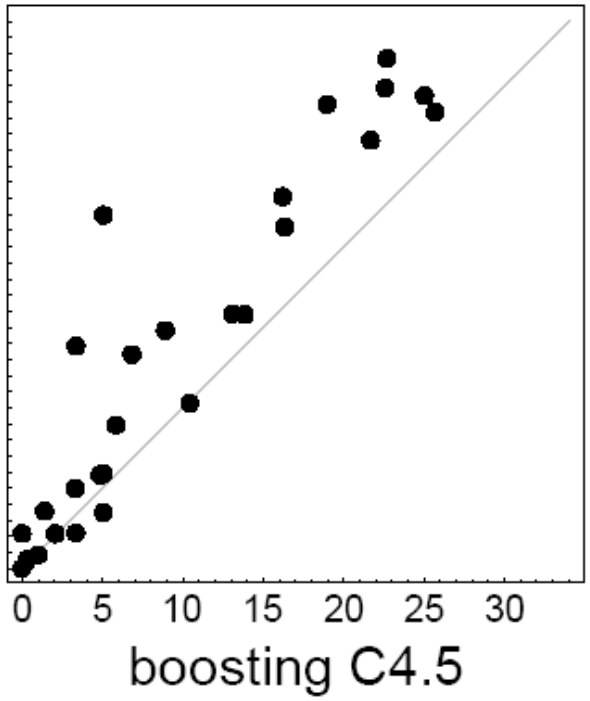
1989 - S

1993 - DSS

1995 - FS

…





Boosting: Analysis

Theorem: if the error at round *t* of the base classifier is $\varepsilon^t = \frac{1}{2}-\gamma_t$, then the *training error* of the boosted classifier is bounded by

$$\begin{aligned}\prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right] &= \prod_t \sqrt{1-4\gamma_t^2} \\ &\leq \exp\left(-2\sum_t \gamma_t^2\right)\end{aligned}$$

The algorithm doesn't need to know any bound on γ_t in advance though -- it *adapts* to the actual sequence of errors.

BOOSTING AS OPTIMIZATION

Even boosting *single features* worked well…

1950 - T

…

1984 - V

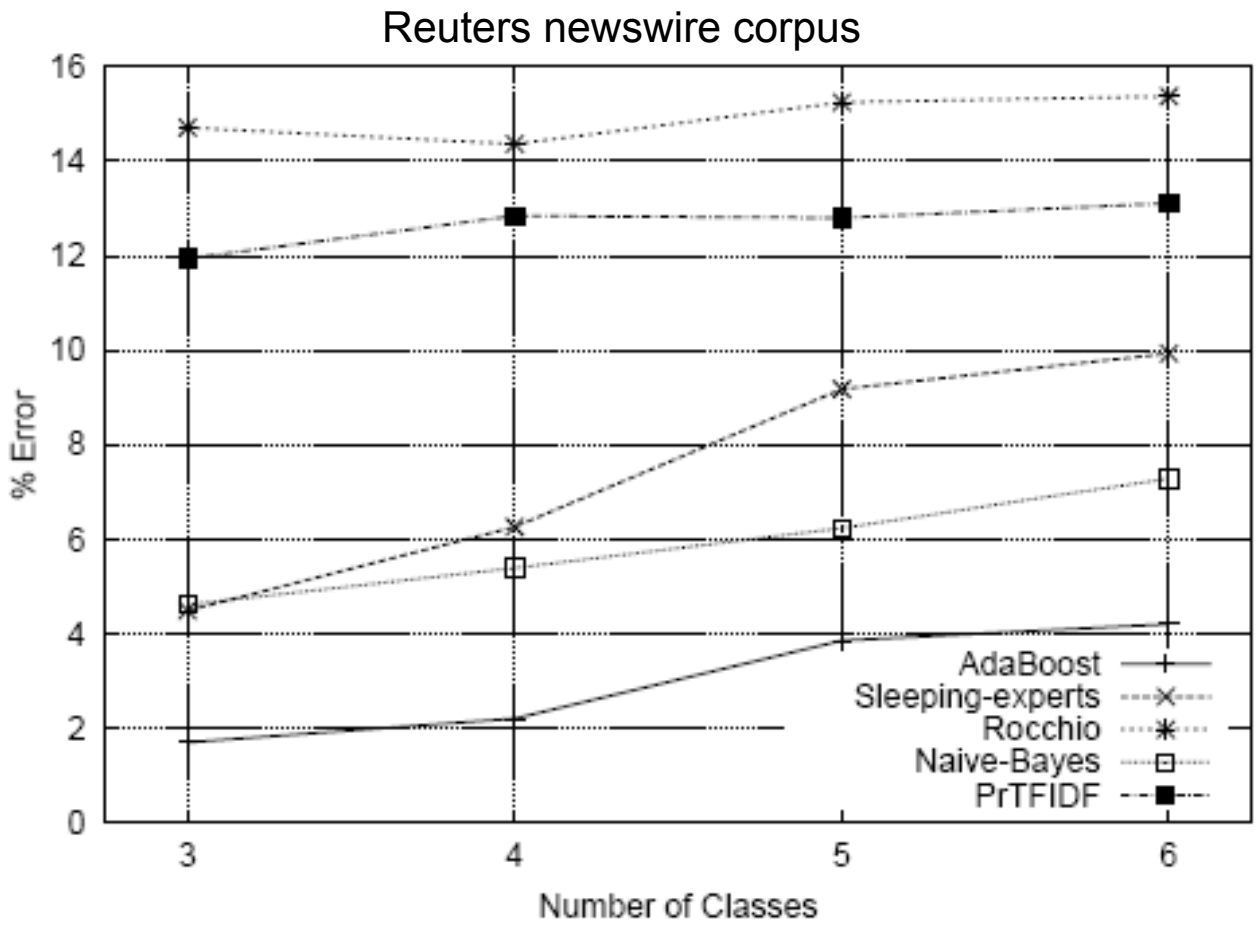
1988 - KV

1989 - S

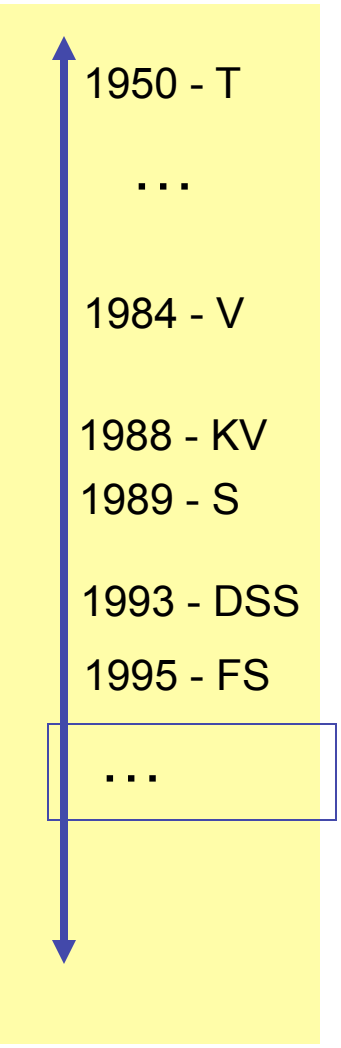
1993 - DSS

1995 - FS

…



Some background facts



Coordinate descent optimization to minimize f**(w)**

- For t=1,…,T or till convergence:
  - For i=1,…,N where **w=<**w_1,…,w_N>
    - Pick w* to minimize f(<w_1,…,w_{i-1},w*,w_{i+1},…,w_N>
    - Set w_i = w*

Boosting as optimization using coordinate descent

With a small number of possible *h's,* you can think of boosting as finding a linear combination of these:

$$H(x) = sign\left(\sum_i w_i h_i(x)\right)$$

So boosting is sort of like stacking:

$$\mathbf{h}(x) \equiv \langle h_1(x),...,h_N(x)\rangle \quad \text{(stacked) instance vector}$$

$$\mathbf{w} \equiv \langle \alpha_1,...,\alpha_N \rangle \quad \text{weight vector}$$

Boosting uses coordinate descent to minimize an *upper bound* on error rate:

$$\sum_{t=i} \exp\left(y_i \sum_i w_i h_i(x)\right)$$

Boosting and optimization

1950 - T

…

1984 - V

1988 - KV

1989 - S

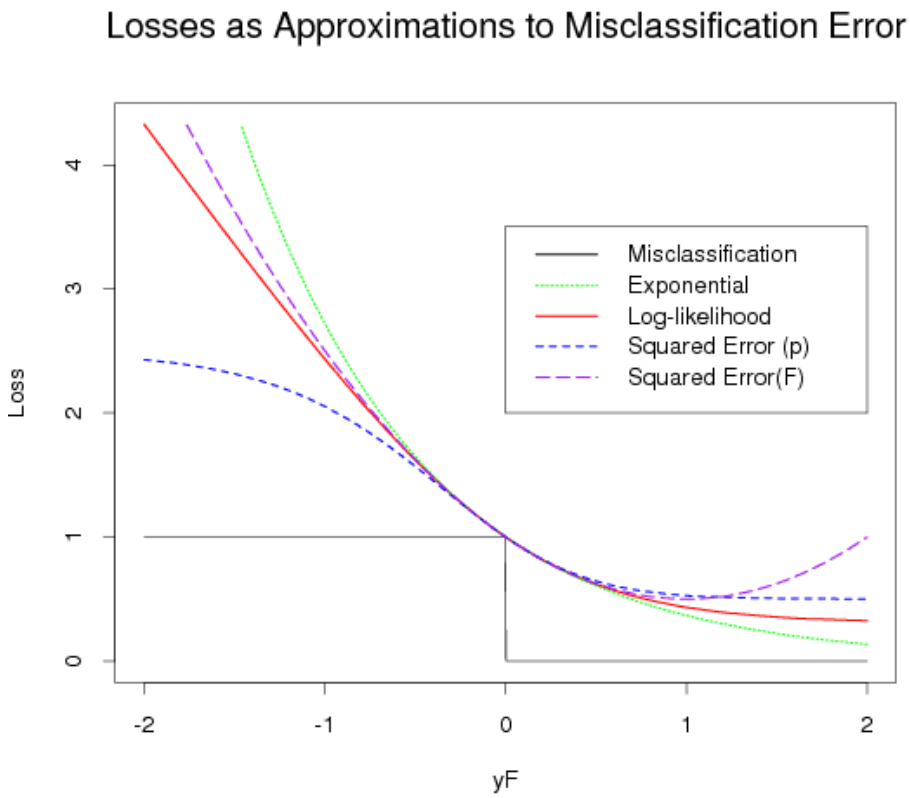
1993 - DSS

1995 - FS

…

1999 - FHT

Jerome Friedman, Trevor Hastie and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 2000.



Compared using AdaBoost to set feature weights *vs* direct optimization of feature weights to minimize log-likelihood, squared error, …

BOOSTING AS MARGIN LEARNING

Boosting didn't seem to overfit…(!)

1950 - T

…

1984 - V

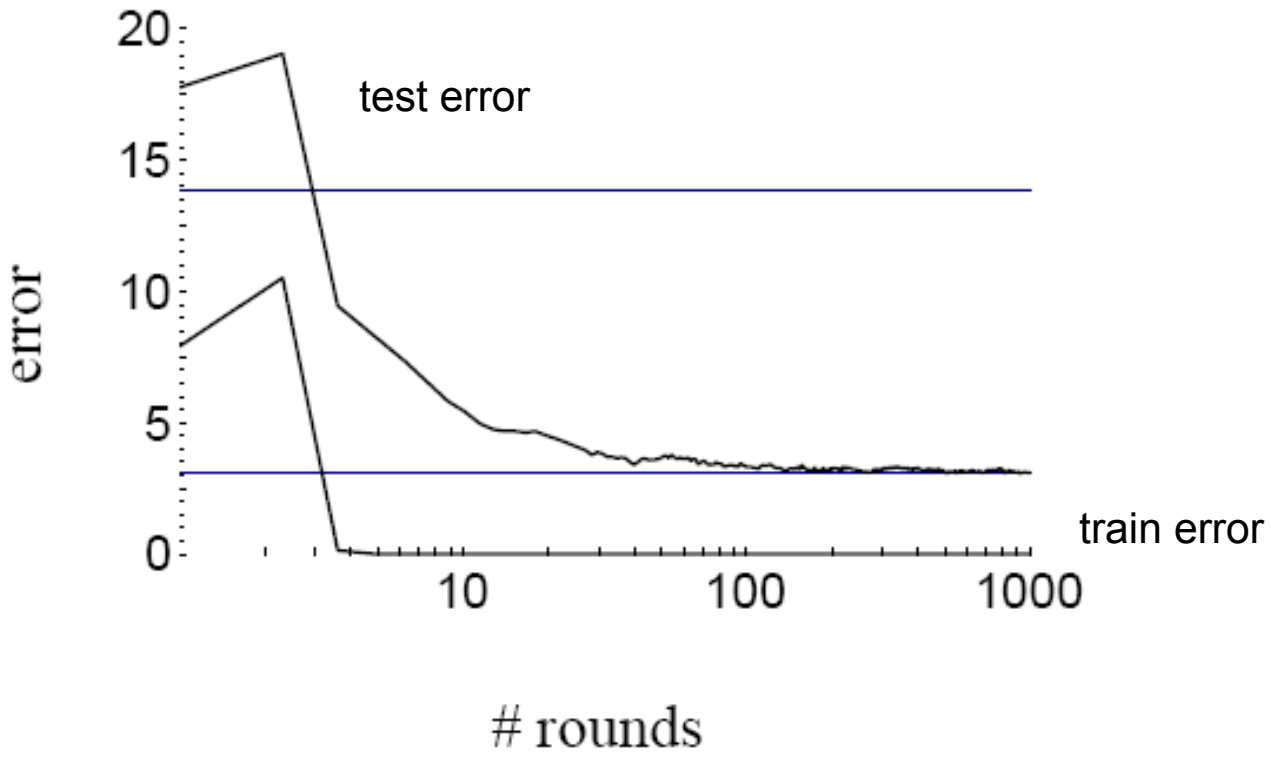
1988 - KV

1989 - S

1993 - DSS

1995 - FS

…



…because it turned out to be increasing the *margin* of the classifier

1950 - T

…

1984 - V

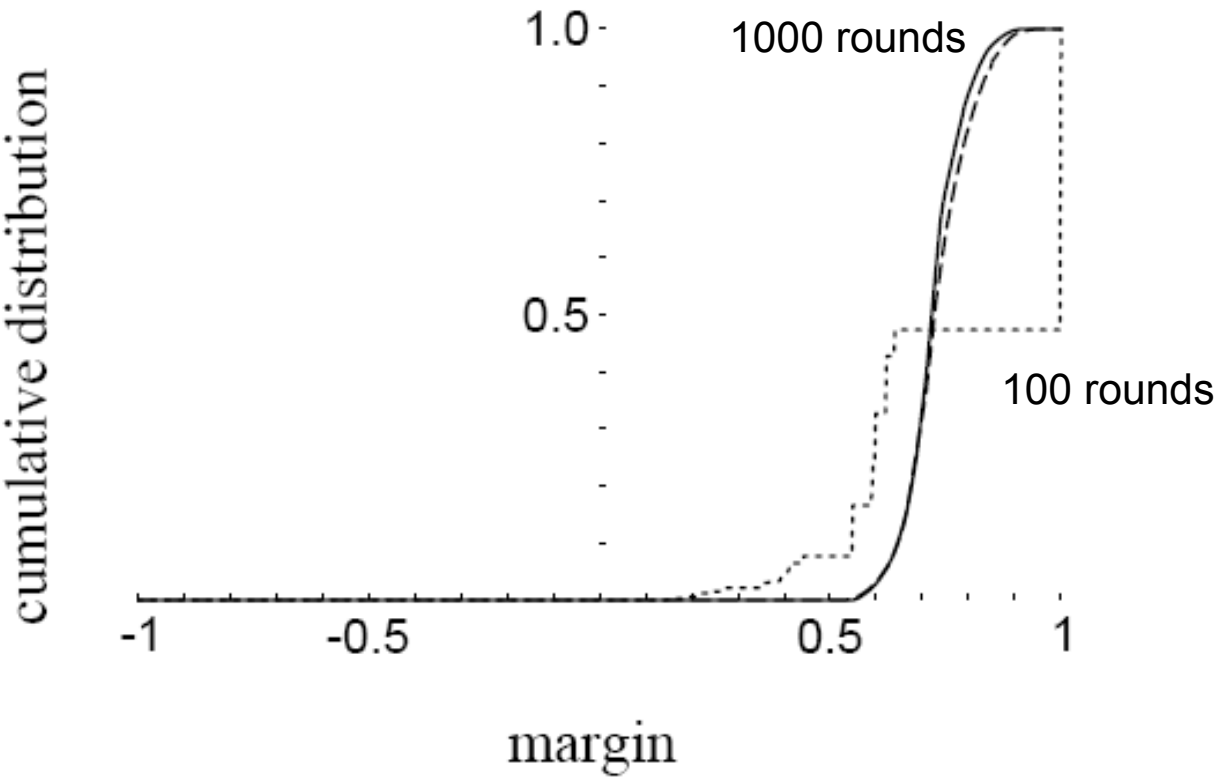
1988 - KV

1989 - S

1993 - DSS

1995 - FS

…



Boosting movie

Some background facts

1950 - T

…

1984 - V

1988 - KV

1989 - S

1993 - DSS

1995 - FS

…

Coordinate descent optimization to minimize f(**w**)

- For t=1,…,T or till convergence:
  - For i=1,…,N where **w=<**w_1,…,w_N>
    - Pick w* to minimize f(<w_1,…,w_{i-1},w*,w_{i+1},…,w_N>
    - Set w_i = w*

$$\|\mathbf{w}\|_k \equiv \sqrt[k]{w_1^k + \ldots + w_T^k}$$

$$\|\mathbf{w}\|_2 \equiv \sqrt{w_1^2 + \ldots + w_T^2}$$

$$\|\mathbf{w}\|_1 \equiv w_1 + \ldots + w_T$$

$$\|\mathbf{w}\|_\infty \equiv \max(w_1, \ldots, w_T)$$

Boosting is closely related to margin classifiers like SVM, voted perceptron, … (!)

1950 - T

…

1984 - V

1988 - KV

1989 - S

1993 - DSS

1995 - FS

…

$$\mathbf{h}(x) \equiv \langle h_1(x),...,h_T(x) \rangle \quad \text{(stacked) instance vector}$$

$$\mathbf{w} \equiv \langle \alpha_1,...,\alpha_N \rangle \quad \text{weight vector}$$

Boosting:

$$\max_{\mathbf{w}} \min_x \frac{\mathbf{w} \cdot \mathbf{h}(x) \cdot y}{\|\mathbf{w}\|_1 * \|\mathbf{h}(x)\|_\infty} \quad \text{optimized by coordinate descent}$$

$$\text{here } \|\mathbf{w}\|_1 = \sum_t \alpha_t \text{ and } \|\mathbf{h}(x)\|_\infty = \max_t h_t(x)$$

The "coordinates" are being extended by one in each round of boosting --- usually, unless you happen to generate the same tree twice

Boosting is closely related to margin classifiers like SVM, voted perceptron, … (!)

1950 - T

…

1984 - V

1988 - KV

1989 - S

1993 - DSS

1995 - FS

…

$$\mathbf{h}(x) \equiv \langle h_1(x),...,h_T(x) \rangle \quad \text{(stacked) instance vector}$$

$$\mathbf{w} \equiv \langle \alpha_1,...,\alpha_N \rangle \quad \text{weight vector}$$

Boosting:

$$\max_{\mathbf{w}} \min_x \frac{\mathbf{w} \cdot \mathbf{h}(x) \cdot y}{\|\mathbf{w}\|_1 * \|\mathbf{h}(x)\|_\infty}$$ optimized by coordinate descent

here $\|\mathbf{w}\|_1 = \sum_t \alpha_t$ and $\|\mathbf{h}(x)\|_\infty = \max_t h_t(x)$

Linear SVMs:

$$\max_{\mathbf{w}} \min_x \frac{\mathbf{w} \cdot \mathbf{h}(x) \cdot y}{\|\mathbf{w}\|_2 * \|\mathbf{h}(x)\|_2}$$ optimized by QP,...

where $\|\mathbf{w}\|_2 = \sqrt{\sum_t \alpha_t^{\,2}}$ and $\|\mathbf{h}(x)\|_2 = \sqrt{h_t(x)^2}$

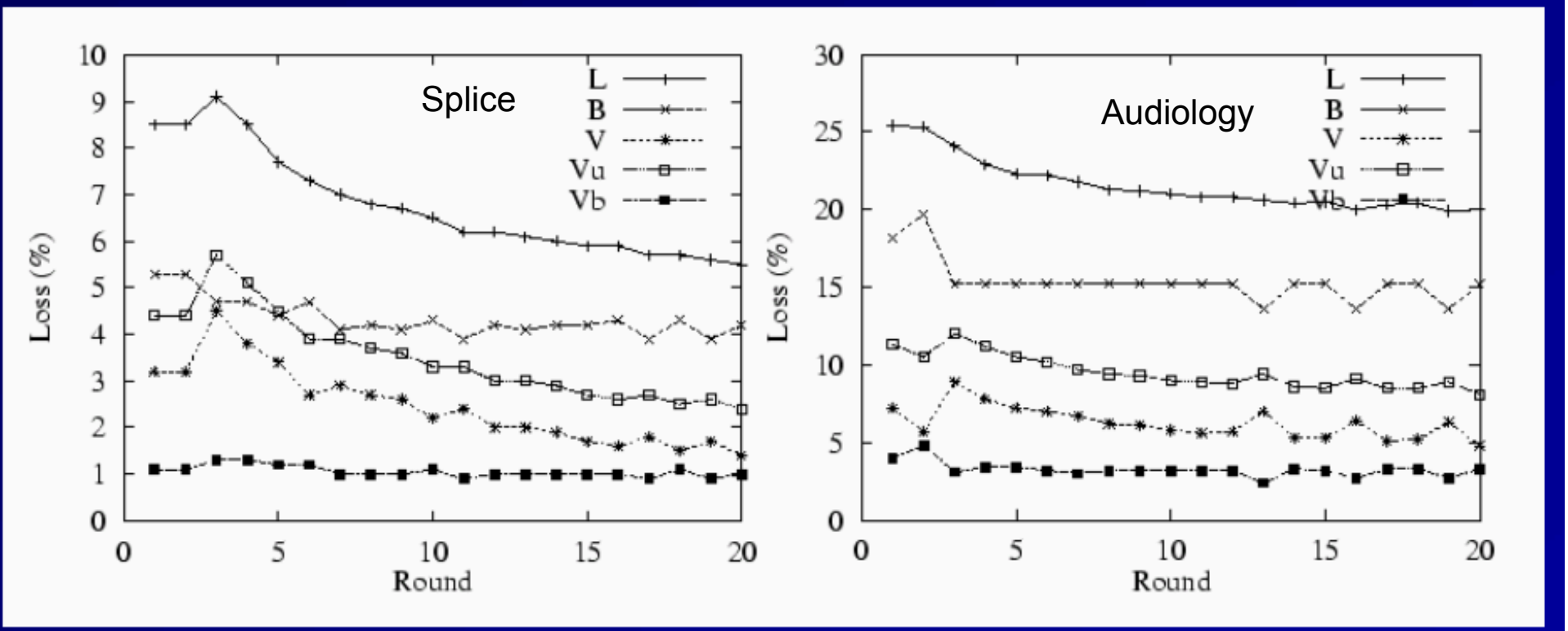
BOOSTING VS BAGGING

Boosting vs Bagging

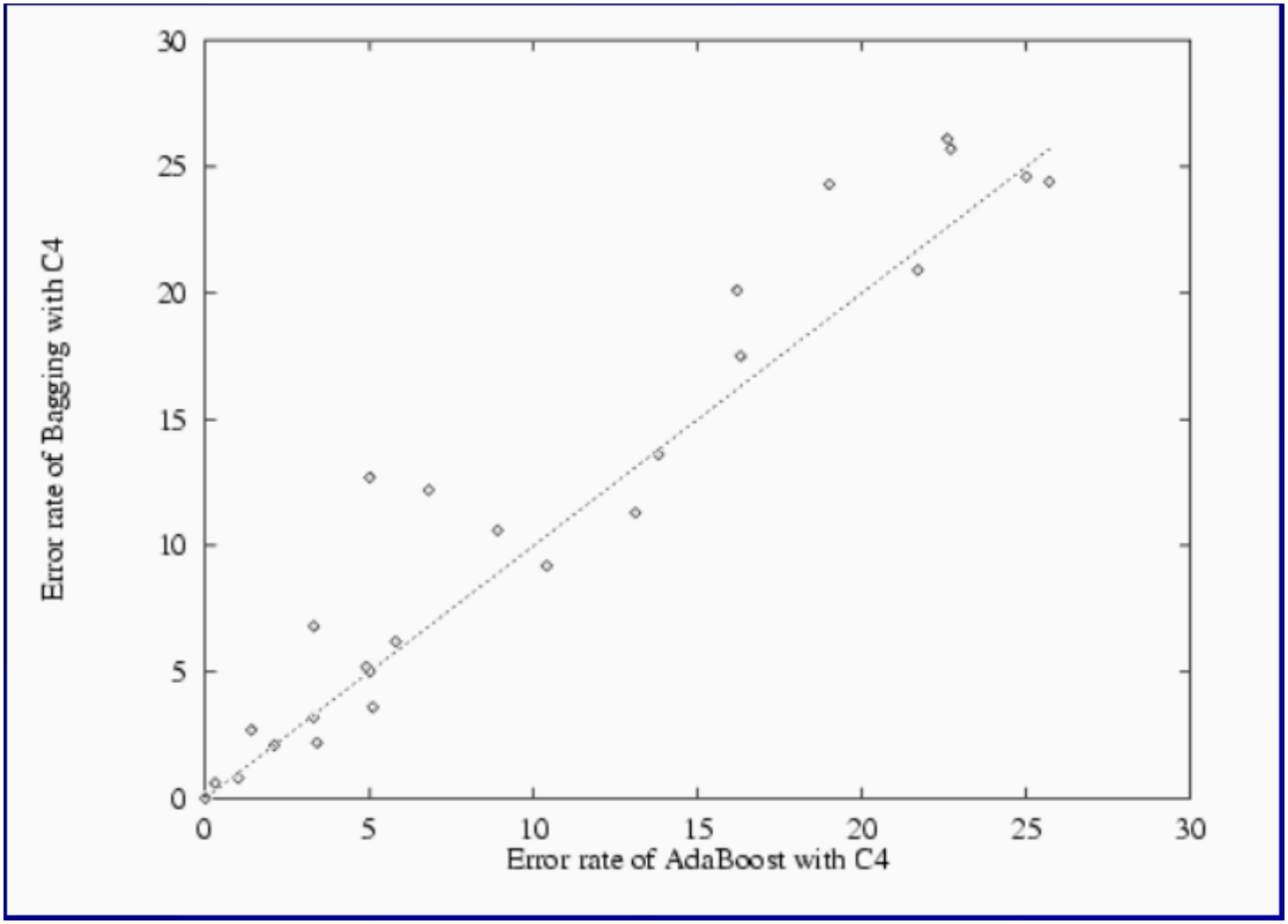
- Both build weighted combinations of classifiers, each classifier being learned on a sample of the original data
- Boosting reweights the distribution in each iteration
- Bagging doesn’t

Boosting vs Bagging

- Boosting finds a linear combination of weak hypotheses
  - Theory says it reduces *bias*
  - In practice is also reduces variance, especially in later iterations

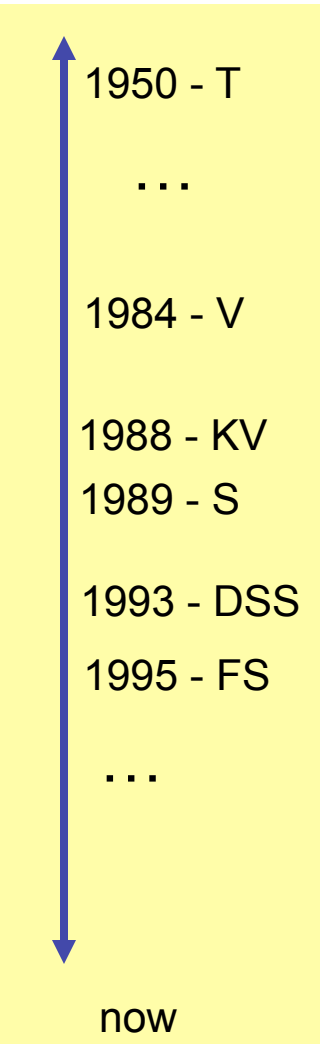


Boosting vs Bagging



WRAPUP ON BOOSTING

Boosting in the real world



- William's wrap up:
  - Boosting is not discussed much in the ML research community any more
    - It's much too well understood
  - It's really useful in practice as a meta-learning method
    - Eg, boosted Naïve Bayes usually beats Naïve Bayes
  - Boosted decision trees are
    - almost always competitive with respect to accuracy
    - very robust against rescaling numeric features, extra features, non-linearities, …
    - somewhat slower to learn and use than many linear classifiers
    - But getting probabilities out of them is a little less reliable.