Machine Learning EN.601.475/675

DR. PHILIP GRAFF

Boosting

EVEN BETTER ENSEMBLES!

Ensemble Learning

Random forests creates a set of decision trees

Can be generalized to any weak learner

Is there anything smarter we can do?

Can weak learners help each other out or learn from each other?

Force weak learners to correct each other's mistakes?

Boosting

Originally developed in computational learning theory for binary classification

One can boost the performance (on training data) of any weak learner arbitrarily high

- Provided weak learner is better than random

Very resistant to over-fitting

Can be interpreted as a form of *gradient descent in function space*

Boosting History

Question first proposed in 1988:

- "Does weak learnability imply strong learnability?"
- Asked about PAC learning, Kearns and Valiant, 1988

Answer came in 1989 by Schapire

- Yes! "boost" a weak learner to get a strong learner!

Schapire 1989

- First boosting algorithm
- Show slight improvements in theory

Freund 1990

- An optimal algorithm that boosts by majority

Drucker, Schapire and Simard 1992

- First experiments using boosting
- Limited by practical considerations

Freund and Schapire 1997

- AdaBoost - the first practical boosting algorithm
- Cited 18,000+ times on Google Scholar

So What is Boosting?

1. Make a simple rule to classify the data (weak learner)
2. Use information from previous iterations to guide next weak learner
3. Repeat to make many rules

Formally

Given a training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ and a weak learner $f(x)$

Binary labels $y_i \in \{-1, +1\}$

For each boosting iteration

- Construct a distribution D_t on the N examples
- Learn a weak hypothesis h_t using f with error: $\varepsilon_t = \Pr\left[h_t(x_i) \neq y_i\right]$

Output final hypothesis

Questions

1. How do we choose subsets of the data?
2. How do we combine all of the rules into a single predictor?
3. The answer: AdaBoost

AdaBoost

AdaBoost = Adaptive Boosting

Given a training set $\{(x_i, y_i)\}_{i=1}^{N}$ where $y_i \in \{-1, +1\}$

Initialize $D_1(i) = {}^{1}/_{N}$

AdaBoost

For each iteration $t=1$ to T

1. Train weak learner using samples weighted by D_t
2. Get weak hypothesis h_t with error

$$\varepsilon_t = \Pr\left[h_t(\mathbf{x}_i) \neq y_i\right] = \frac{\sum_{i=1}^{N} D_t(i) I(h_t(\mathbf{x}_i) \neq y_i)}{\sum_{i=1}^{N} D_t(i)}$$

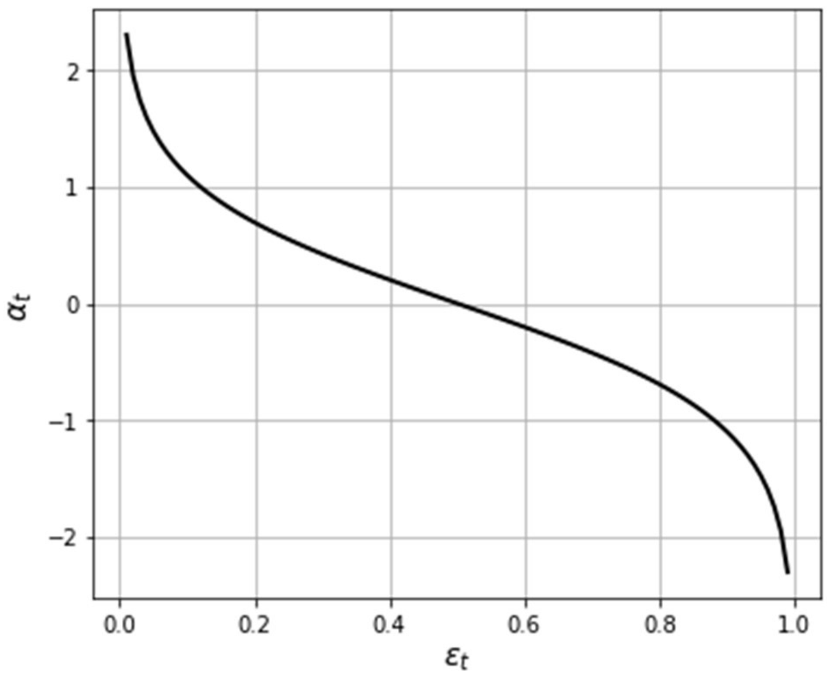
3. Set

$$\alpha_t = \frac{1}{2} \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

- α_t is larger for small error
- α_t is negative for error above 0.5

AdaBoost

$$\alpha_t = \frac{1}{2} \log \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$



- Low error ➔ Large weight
- Error > 0.5 ➔ Weight < 0
  ➔ Do the opposite!

AdaBoost Continued

4. Update weights (where Z_t is a normalization constant)

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

- Strong classifiers make large adjustment
- Correct/incorrect predictions get lower/higher weights for next weak learner
- Next learner can focus on getting right what previous ones got wrong

AdaBoost Continued

Output the final hypothesis:

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

Low-error weak learners are given higher weighting in final prediction

Note

Distribution tends to concentrate on hard examples

- Sound familiar?

SVMs!

- Weight on examples close to the margin
- We’ll come back to this point

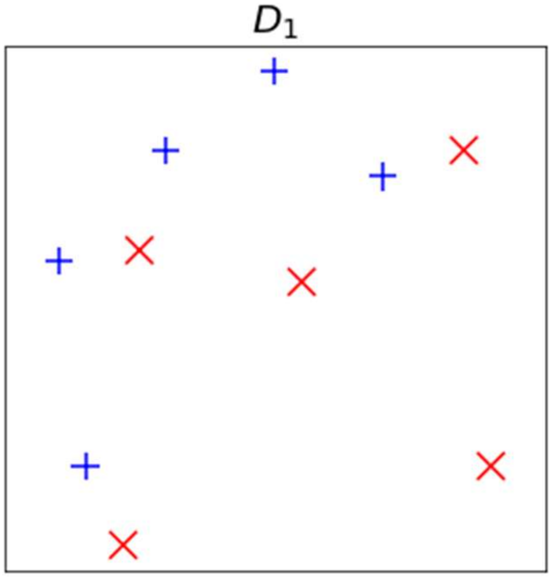
Example

A set of labeled points with a uniform distribution

Fit with *decision stump* model

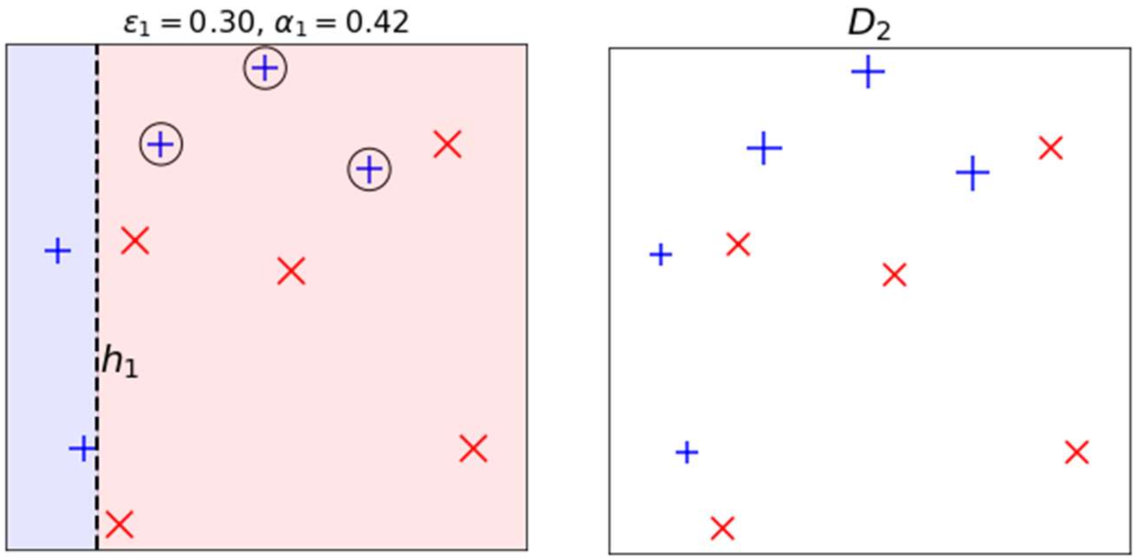
- Decision tree with max depth of 1
- i.e. make only 1 split

w = [0.1 0.1 0.1 0.1 0.1
0.1 0.1 0.1 0.1 0.1]



Round 1

Learn hypothesis, measure error, set α
Re-compute distribution placing more weight on incorrect examples

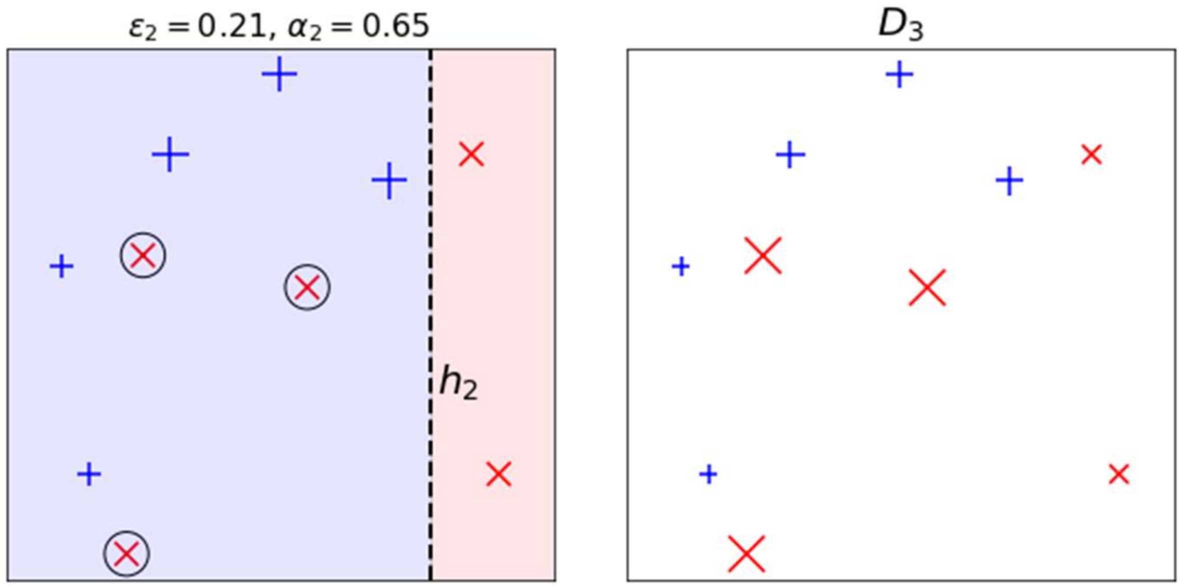


w = [0.0714 0.0714 0.0714 0.0714 0.1667 0.1667 0.1667 0.0714 0.0714 0.0714]

Round 2

Learn hypothesis, measure error, set α

Re-compute distribution placing more weight on incorrect examples

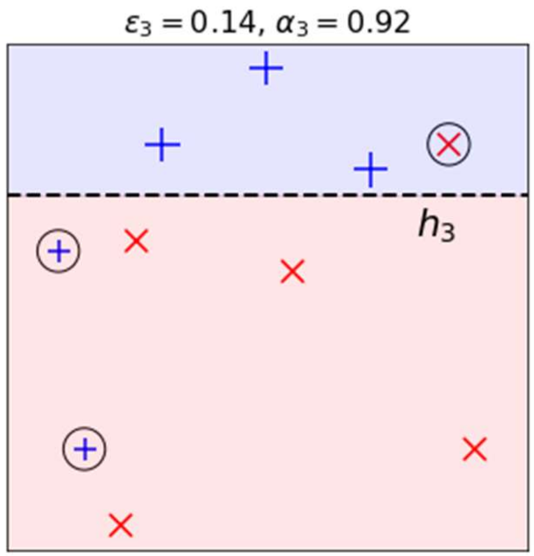


w = [0.0455 0.0455 0.1667 0.1667 0.1061 0.1061 0.1061 0.1667 0.0455 0.0455]

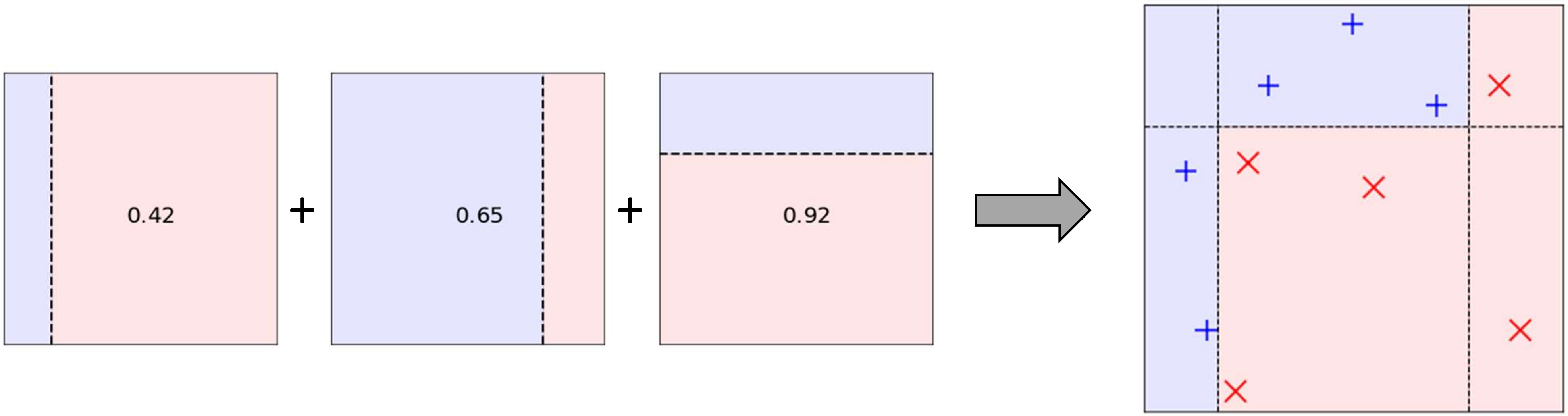
Round 3

Learn hypothesis, measure error, set α

Re-compute distribution placing more weight on incorrect examples



Final Model



Why is Boosting Good?

Boosting achieves good empirical results

Why?

Many answers:

- Statistical View of Boosting
- Boosting and Max Margin
- PAC Learning (learning theory)
- Game theory

Boosting

Fitting a function to data

Fitting: Optimization, what parameters can we change?

Function: Model with a loss function

Data: Weigh data based on previous prediction errors

Statistical View of Boosting

What is boosting doing?

We normally think of classifiers in terms of loss or likelihood

What is the objective function of boosting?

Exponential Loss

AdaBoost can be viewed as an algorithm for minimizing the exponential loss in function space

Choices of α_t and h_t minimize the loss

A form of coordinate descent

- Derivative-free optimization
- At each iteration, from the current point do a line search along one coordinate direction
- Coordinate direction is new function, h_t, to be added

Exponential Loss

Exponential loss given by:

$$E = \sum_{i=1}^{N} \exp\left\{-y_i h_t(\mathbf{x}_i)\right\}$$

h_t is a classifier defined by a linear combination of base classifiers:

Assume as given:

$$h_1(\mathbf{x}), \ldots, h_{t-1}(\mathbf{x})$$

$$\alpha_1, \ldots, \alpha_{t-1}$$

$$h_t(\mathbf{x}) = \frac{1}{2} \sum_{s=1}^{t} \alpha_s f_s(\mathbf{x})$$

Exponential Loss

Re-write *E* separating out fixed values at iteration *t*:

$$E = \sum_{i=1}^{N} \exp\left\{-y_i h_{t-1}(\mathbf{x}_i) - \frac{1}{2} y_i \alpha_t f_t(\mathbf{x}_i)\right\}$$

$$= \sum_{i=1}^{N} w_i^{(t)} \exp\left\{-\frac{1}{2} y_i \alpha_t f_t(\mathbf{x}_i)\right\}$$

$$w_i^{(t)} = \exp\left\{-y_i h_{t-1}(\mathbf{x}_i)\right\}$$

Minimizing this w.r.t. α_t and h_t yields our AdaBoost solution

- See Bishop 14.3.1 for details

Synthetic Data Experiment

Data is $x \in \{-1, +1\}^{10,000}$, 100 training examples and 10,000 test examples

	% test error	[# rounds]				
exp. loss	exhaustive AdaBoost		gradient descent		random AdaBoost	
10^{-10}	0.0	[94]	40.7	[5]	44.0	[24,464]
10^{-20}	0.0	[190]	40.8	[9]	41.6	[47,534]
10^{-40}	0.0	[382]	40.8	[21]	40.9	[94,479]
10^{-100}	0.0	[956]	40.8	[70]	40.3	[234,654]

Table 1 Results of the experiment described in Section 4. The numbers in brackets show the number of rounds required for each algorithm to reach specified values of the exponential loss. The unbracketed numbers show the percent test error achieved by each algorithm at the point in its run at which the exponential loss first dropped below the specified values. All results are averaged over ten random repetitions of the experiment. (Reprinted from [30] with permission of MIT Press.)

Regularization

No explicit regularization in AdaBoost

Observation: Stopping AdaBoost after any number of rounds provides an approximate solution to L1-regularized minimization of exponential loss

- This is a variant of AdaBoost where α_t is set to a small, fixed constant at each round
- Requires stopping while t is still quite small, larger t is less regularization

Illustrative, but doesn't really apply to AdaBoost in practice

Overfitting

Given all of this, AdaBoost seems like it would over-fit

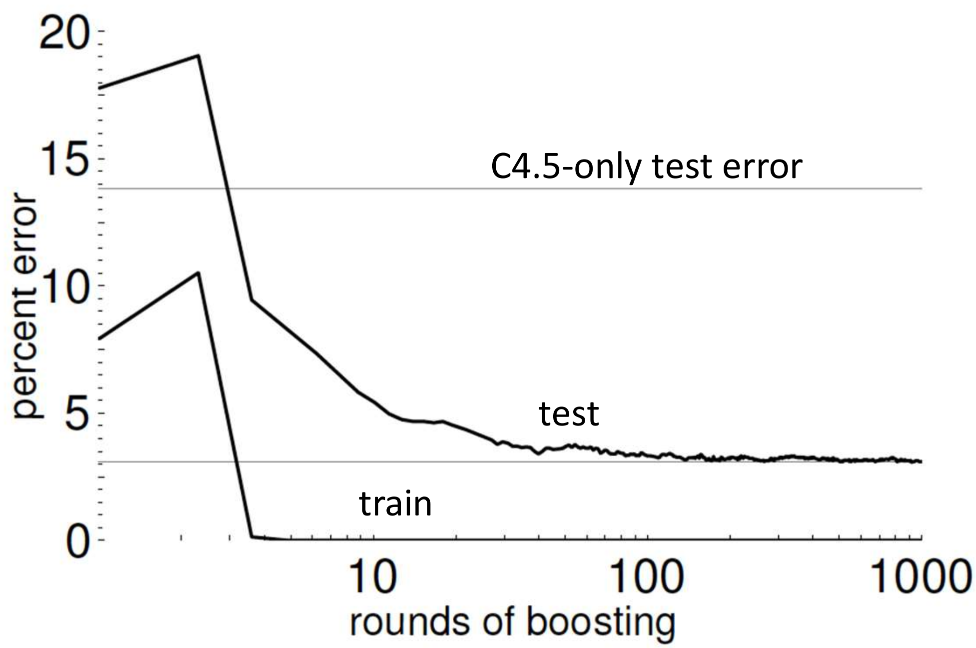
- No training error after $t \sim \mathcal{O}(\log N)$ iterations

Each round focuses more and more on incorrect examples

But it doesn't! *Why?*

Overfitting

Train and test error on an OCR dataset with C4.5 (decision tree) as the weak learner



Margin Explanation

Training error goes to zero very fast

Do we get benefits from additional training?

- As we can see, yes!

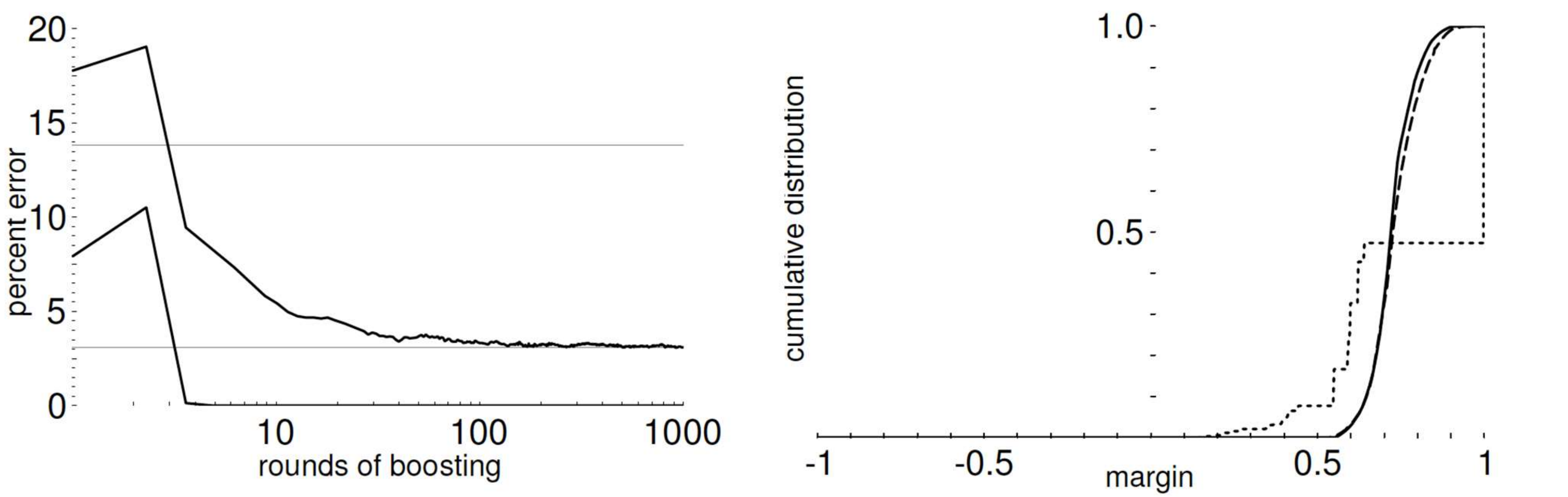
The margins improve

- Better margins → better test error
- Same as in SVMs

What is the margin?

- Confidence in prediction, i.e. magnitude of $\sum_{t=1}^{T} \alpha_t h_t(x)$

Margin Explanation



RHS shows CDF of margins after 5 (dotted), 100 (dashed), and 1000 (solid) rounds of boosting.

Margin Explanation

Step 1: We can prove a generalization bound on AdaBoost that depends on margins of training examples, NOT number of rounds

- Over-fitting no longer a function of rounds
- Depends on margins
- Won't overfit as long as large margins can be achieved

Step 2: AdaBoost generally increases margins of training examples

Idea: design boosting that directly maximizes the margins!

- Hasn't worked. Produces overly complex weak hypotheses, more over-fitting

SVM Connection

Based on Freund and Schapire's generalization bound and margin observations

Write boosting as maximizing the minimum margin

- Assume h functions already known, choosing α values

$$\max_{\boldsymbol{\alpha}} \min_{i} \frac{(\boldsymbol{\alpha} \cdot \mathbf{h}(\mathbf{x}_i))y_i}{\|\boldsymbol{\alpha}\| \|\mathbf{h}(\mathbf{x}_i)\|}$$

Difference in Norms Used

Norms used for boosting are: $\max_{\boldsymbol{\alpha}} \min_i \frac{(\boldsymbol{\alpha} \cdot \mathbf{h}(\mathbf{x}_i))y_i}{\|\boldsymbol{\alpha}\| \|\mathbf{h}(\mathbf{x}_i)\|}$

$$\|\boldsymbol{\alpha}\|_1 \doteq \sum_t |\alpha_t| \qquad \|\mathbf{h}(\mathbf{x})\|_\infty \doteq \max_t |h_t(\mathbf{x})|$$

Norms used for SVMs are Euclidean:

$$\|\boldsymbol{\alpha}\|_2 \doteq \sqrt{\sum_t \alpha_t^2} \qquad \|\mathbf{h}(\mathbf{x})\|_2 \doteq \sqrt{\sum_t h_t(\mathbf{x})^2}$$

Differences

Different norms can result in different margins

- For high-dimensional inputs, the effective margins can be quite different

SVM requires quadratic programming

- Boosting only requires linear programming

Finding high dimensional separators

- SVM uses kernels while boosting uses weak learners
- There is usually a big difference between the learning spaces of the kernels and the weak learners

What Should we Boost?

If boosting improves a base classifier…

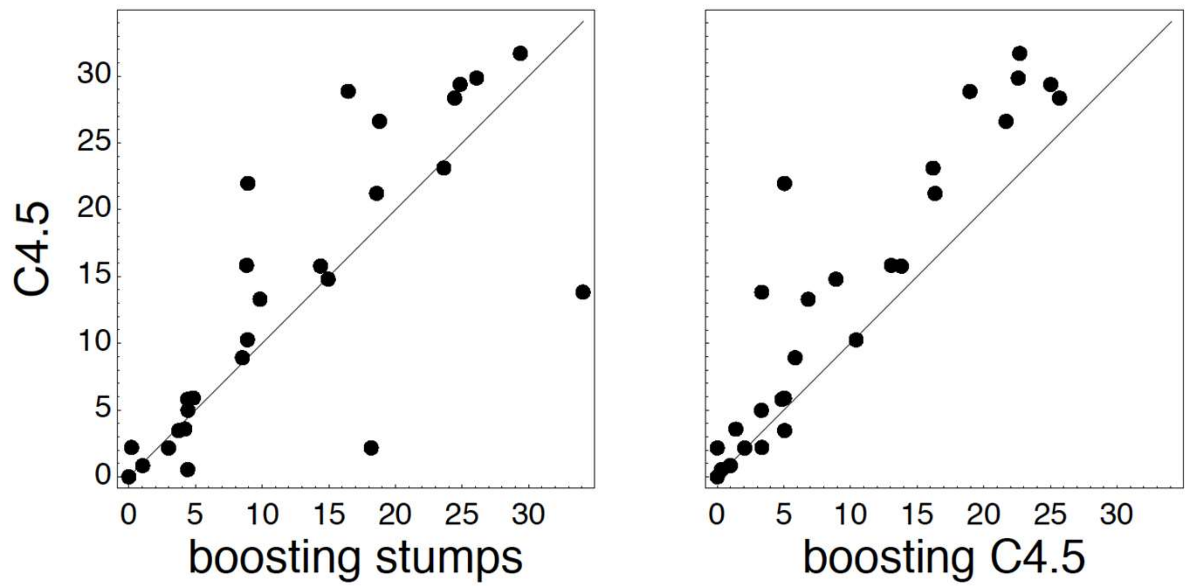
- Boost the best classifier we have!
- Boosted kNN, SVM, perceptron, logistic regression, etc.
- Problem: You have to train many of these, which may not be efficient

But boosting can strengthen a *weak* learner

- Pick a simple model that is easy and fast to train

Boosting Results

Test % error on 27 benchmark datasets (averaged over multiple runs)



Practical Advantages

- Easy to implement
- Very fast
- No parameters to tune
- Not specific to any weak learner
- Well-motivated by learning theory
- Can identify outliers
- Extensions to multi-class, ranking, regression

Boosting

Fitting a function to data

Fitting: Specialized procedure for fitting to data

Function: Exponential loss, linear combination of underlying classifiers

- Underlying classifiers determine hypothesis class

Data: Weigh data based on previous prediction errors

Combining Classifiers

Boosting uses weak learners

What if I have multiple classifiers that I want to combine?

- Is there a general way?
- Yes!
- *Mixtures of experts*

Mixtures of Experts

Assume we have many “experts” giving us advice

- Each expert examines an example and returns a label

How can we combine this mixture of experts to create a single output?

Intuition: some experts are better than others

Solution: Learn which experts are the best and trust them the most

Weighted Majority

Initialize the weight $w_k = 1$ for expert k

- Assume binary classification $y_i \in \{-1, 1\}$

For each example x_i

- Predict
- Update
  - For each expert k
  - If $y_i \neq f_k(x_i)$
    - $w_k = w_k / 2$

$$\hat{y}_i = \text{sign}\left(\sum_k w_k f_k(x_i)\right)$$

Weighted Majority

An online algorithm for learning mixtures of experts

- Have experts, learn the mixture parameters

Bounded by regret to the best expert

- Theorem: The number of mistakes made by the weighted majority algorithm is never more than $2.41(m + \log k)$ where m is the number of mistakes made by the best expert so far
- We will never do much worse than the best expert
- Since we don't know which is best in advance, this is good news
  - Only a log penalty for adding more experts

Weighted Majority

First introduced by Littlestone and Warmuth (1994)

No assumptions about data or expert quality

Widely used in lots of settings

- Stock portfolio balancing
- Experts are individual stocks

Why Combine?

We've talked about several ways to combine

But why are combinations good?

An example: you want to get advice about which stocks to invest in

- What should you do?
- Call the same stockbroker 100 times and average?
- Call 100 different stockbrokers and average?

Diversity in Experts

We can improve by combining multiple classifiers since they have a diversity of opinions

- They won't all make the same mistakes
- If they are all very good, then we can vote them to get even better
  - Reality: they do make some of the same mistakes
- This is the idea behind boosting
  - If you can do a bit better than random (weak), then you can boost that to good performance (strong)

Creating Diversity

We can create diversity by using K different classifiers

We can also create diversity by creating K different datasets

Bagging: create many different datasets by hiding some of the data

- Instance bagging
- Feature bagging

Instance Bagging

Given N examples for training

- Create K datasets
  - Select N examples *with replacement* from the training set
  - Probability of an example being selected: 63.2%
  - Train a classifier on the dataset

Final output: voting of the K classifiers

- Or: weighted majority of the K classifiers

Feature Bagging

Given N examples for training

- Create K datasets
  - Select (K-1)/K of the features to use
  - Ignore the rest
  - Train a classifier on the dataset

Final output: voting of the K classifiers

- Or: weighted majority of the K classifiers

Co-Training

Feature bagging is closely connected to co-training

- Blum and Mitchell 1998

Co-training: A semi-supervised learning algorithm in which you have two representations of each example

- Given: labeled (few) and unlabeled (many) data
- Train a separate classifier on each representation
- Label the unlabeled data
- If one classifier is confident in its prediction, use it for training for both
- Uses diversity of representations to improve performance

Mixed Ensembles

Random forests aren’t the only kind of ensemble

Ensembles don’t even have to be of the same kind of model!

Can build ensembles of any mixture of models you want
- Still average results for a final output
- Can make this a weighted average

Powerful method for building a model where different parts compensate for weaknesses in others

Summary

Boosting

- Turns weak learners into a strong one
- Can do a lot by combining a little

Mixtures of experts

- Weighted majority uses the predictions of the best experts

Diversity

- Create artificial diversity: instance and feature bagging
- Co-training uses diversity in representations for semi-supervised learning

Next time: Neural Networks!