Boosting

Different Approach

- So far we've covered
  - Supervised Learning (Classification)
    - Linear and non-linear models
- Today's idea: combine the outputs of classifiers to get something better
  - Approach: build complex models from simpler models

Combinations

- Can I learn many different classifier using a supervised learning algorithm and combine them to get a better classifier?
  - Yes

- Ensemble learning
  - Combine an ensemble of classifiers

Learning

- weak learner
  - learns a predictor slightly better than random guessing
- strong learner
  - learns a predictor with arbitrary accuracy

Boosting

- Let's say I have a weak learner, can I take this weak learner and make it better?
  - Use sets of weak learners
- 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!

Some Boosting History

- 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 1996
  - AdaBoost- the first practical boosting algorithm
    - Cited 7000+ times on Google Scholar
  - The rest is history

General Boosting Approach

- Look at subset of data
  - Make simple rule to classify data (weak)
  - Repeat (make many rules)
- Boosting
  - Boost weak rules into strong predictors

A Formal View of Boosting

- Given training set $\{x_i, y_i\}_{i=1}^{N}$ and weak learner f
- Binary labels $y_i \in \{+1, -1\}$
- For each boosting iteration
  - Construct distribution D_t on N examples
  - Learn weak hypothesis h_t using f with error:

$$\epsilon_t = P_{D_t}[h_t(x_i) \neq y_i]$$

- Output final hypothesis

Questions

- How do we choose subsets of the data?
- How do we combine all the rules into predictor?

- The answer: AdaBoost

AdaBoost

- Given: $\{x_i, y_i\}_{i=1}^{N}$ where: $y_i \in \{+1, -1\}$
  - Initialize $D_1(i) = 1/N$

AdaBoost

- For each iteration t=1 to T:
  - Train weak learner using distribution D_t
  - Get weak hypothesis h_t with error

$$\varepsilon_t = P_{D_t}[h_t(x_i) \neq y_i]$$

  - Choose

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

  - Update

$$D_{t+1}(i) = D_t(i) \times \frac{e^{-\alpha_i h_t(x_i) y_i}}{Z_t}$$

$$Z_t = \sum_{i=1}^{N} e^{-\alpha_i h_t(x_i) y_i}$$

AdaBoost

- Output the final hypothesis:

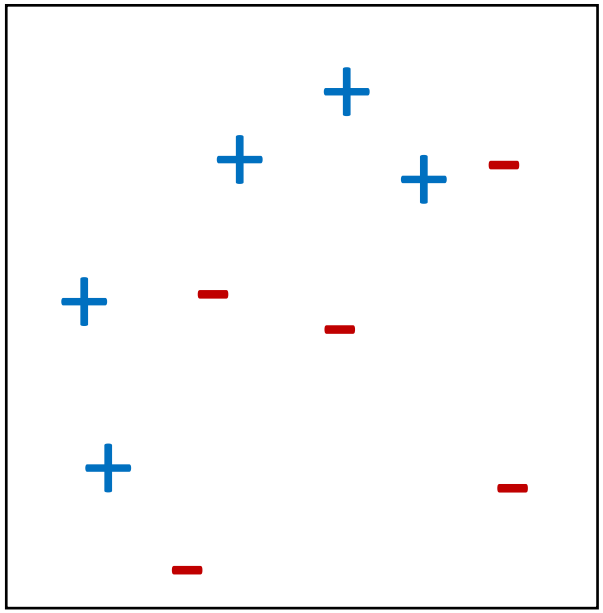
$$f(x) = sign(\sum_{t=1}^{t} \alpha_t \, h_t(x))$$

Notes

- α_t measures importance assigned to h_t
- As α gets larger, error gets smaller
- Distribution tends to concentrate on hard examples
- Sound familiar?
  - SVMs!
  - Weight on examples close to the margin
  - We'll come back to this point

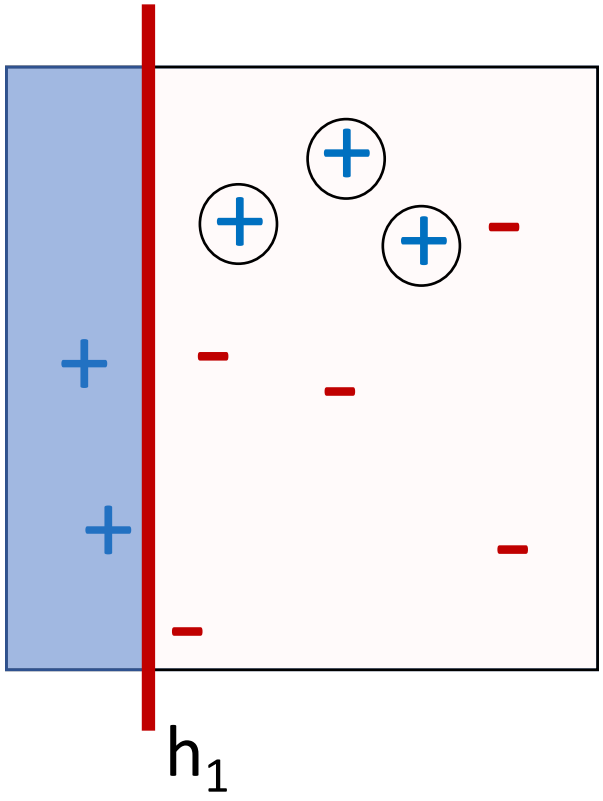
Example

D_1



- A set of labeled points with a uniform distribution

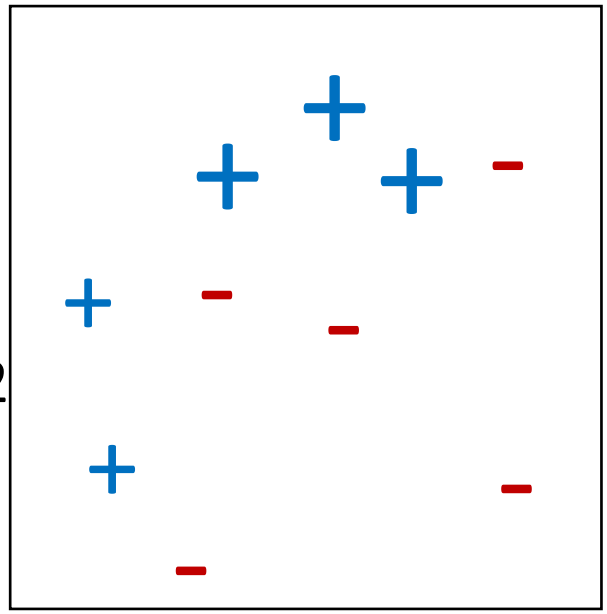
Round 1



h_1

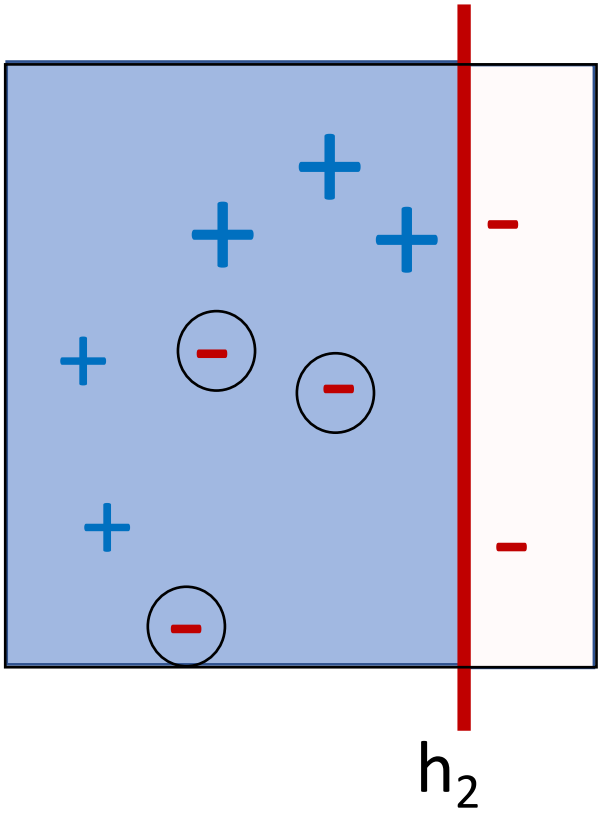
$\varepsilon_1 = 0.30$
$\alpha_1 = 0.42$

D_2



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

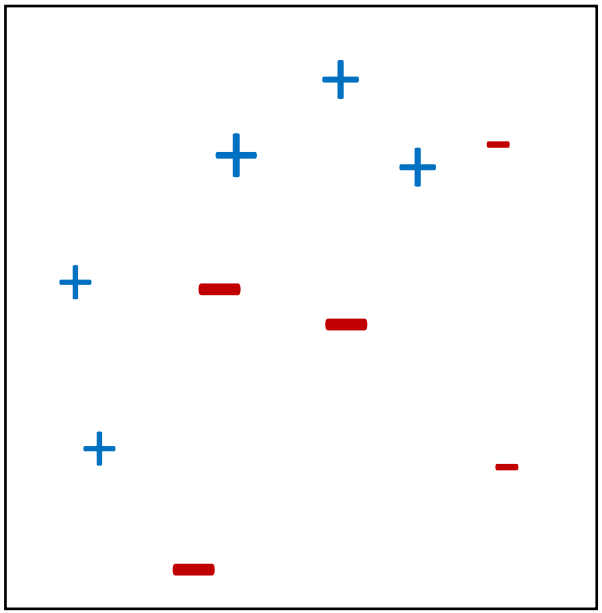
Round 2



h_2

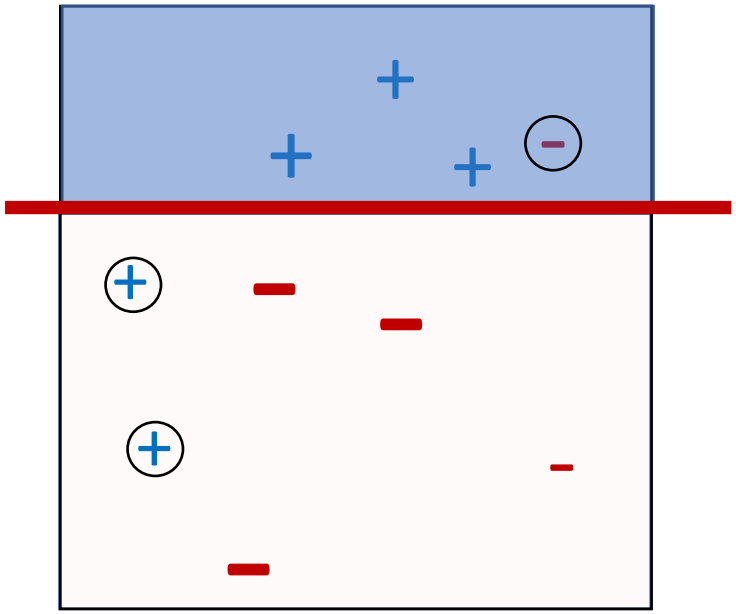
$\varepsilon_2 = 0.21$
$\alpha_2 = 0.65$

D_3



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

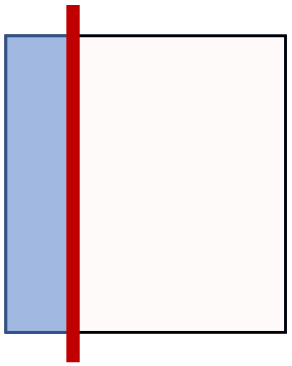
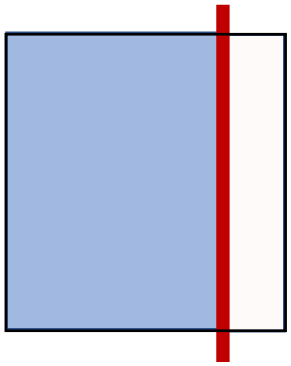
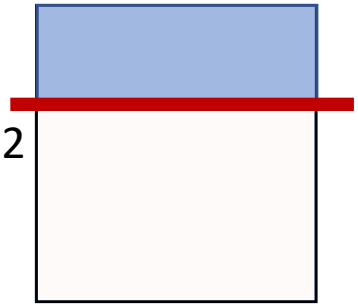
Round 3

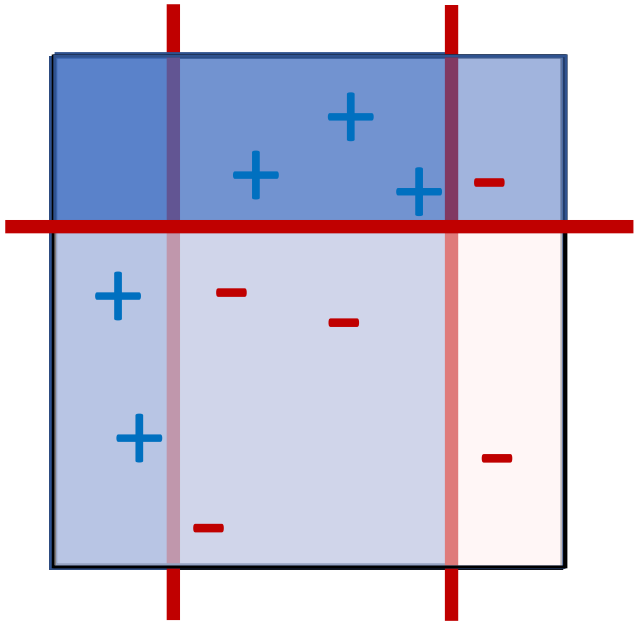


$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

Final Model

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



Why is Boosting Good?

- Boosting achieves good empirical results
- Boosting achieves good empirical results
- Why?
- Many answers
  - Statistical View of Boosting
  - Boosting and Max Margin
  - PAC Learning (learning theory)
  - Game theory

Boosting

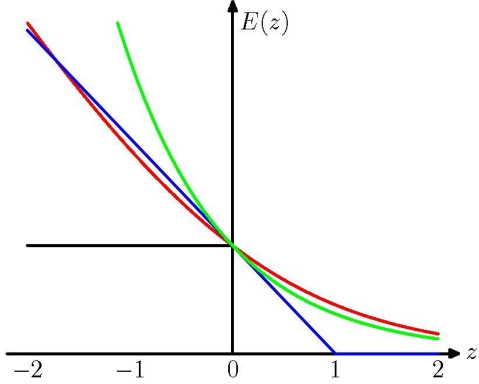
- **Fitting a function to data**

- Fitting: Optimization, what parameters can we change?
- **Function: Model, 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 for boosting?

Exponential Loss



- AdaBoost can be viewed as an algorithm for minimizing exponential loss

$$\sum_i e^{-y_i f(x_i)}$$

- Choices of α_t and h_t minimize this loss
- A form of coordinate descent

Synthetic Data Experiment

	% 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?

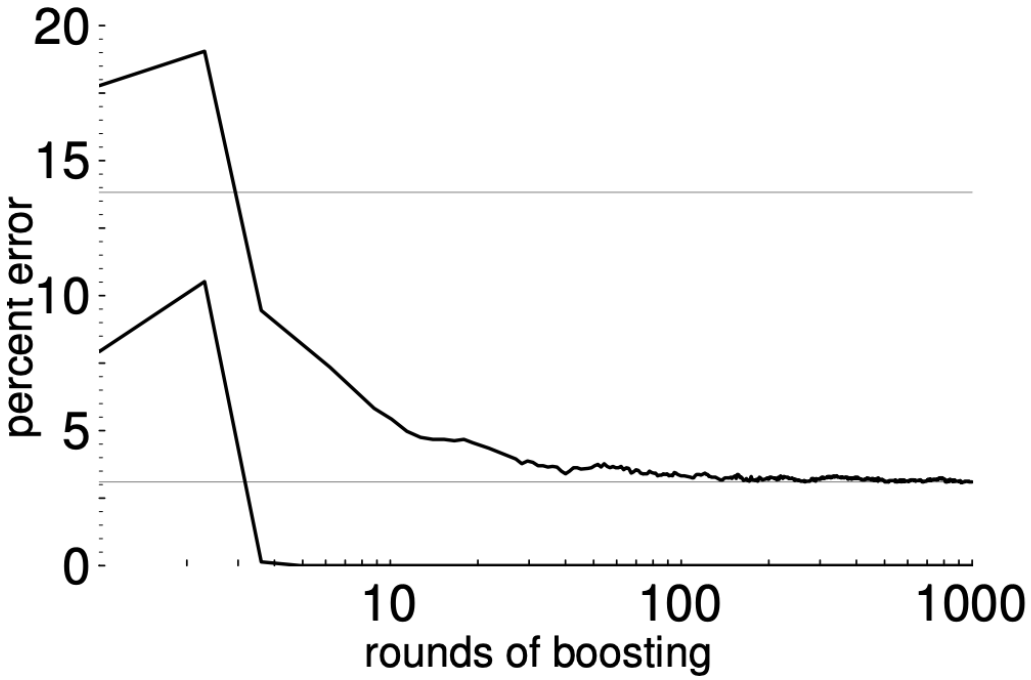
- No explicit form of regularization
- Observation: stopping (variant of) AdaBoost after (any) number of rounds provides an approximate solution to L1 regularized minimization of exponential loss
  - Set α on each round to be small positive constant
  - Resulting classifiers approximate regularized minimization
- In practice, illustrative but doesn't apply to AdaBoost

Overfitting

- AdaBoost seems like it would overfit
  - Each round focuses more on incorrect examples
  - VC theory tells us it will overfit (details not important)
- It doesn't overfit!

Overfitting

- Top: test error of H(x) classifier
- Bottom: train error of H(x) classifier



Margin Explanation

- Train error goes to 0 right away
- Do we get benefits in continued training?
- Yes! The margins improve
  - Better margins should give us better test error
  - Similar idea from SVMs

Margin Explanation

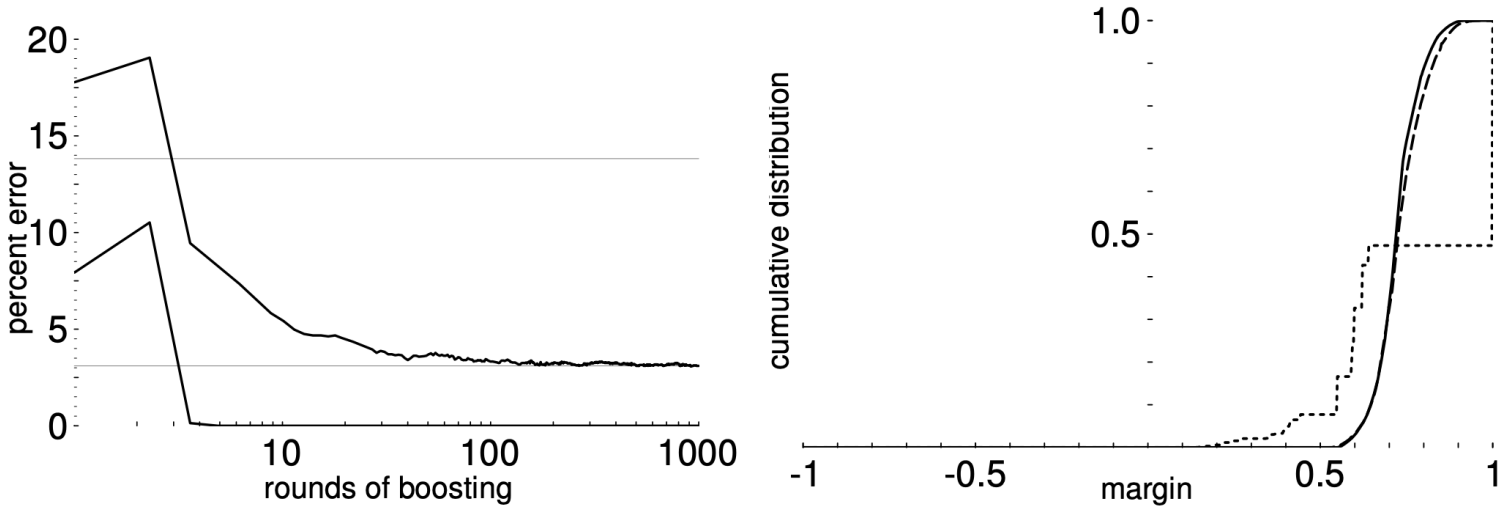


Fig. 3 *Left:* The training and test percent error rates obtained using boosting on an OCR dataset with C4.5 as the base learner. The top and bottom curves are test and training error, respectively. The top horizontal line shows the test error rate using just C4.5. The bottom line shows the final test error rate of AdaBoost after 1000 rounds. *Right:* The margin distribution graph for this same case showing the cumulative distribution of margins of the training instances after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed (mostly hidden) and solid curves, respectively. (Both figures are reprinted from [31] with permission of the Institute of Mathematical Statistics.)

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 the generalization bound, and margin observations
- We can write Boosting as maximizing the minimum margin

$$\max_{\alpha} \min_{i} \frac{(\alpha \cdot h(x_i))y_i}{\|\alpha\|\|h(x_i)\|}$$

Difference in Norms

$$\max_{\alpha} \min_{i} \frac{(\alpha \cdot h(x_i))y_i}{\|\alpha\|\|h(x_i)\|}$$

- The norms for boosting are defined as

$$\|\alpha\|_1 = \sum_t |\alpha_t| \qquad \|h(x)\|_\infty = \max_t |h_t(x)|$$

- For SVMs, the norms are Euclidean

$$\|\alpha\|_2 = \sqrt[2]{\sum_t {\alpha_t}^2} \qquad \|h(x)\|_2 = \sqrt[2]{\sum_t h_t(x)^2}$$

- Boosting and SVMs are very similar

Differences

- The norms are different
  - In high dimensional space, the effective enforced margins may be very different
- SVM requires quadratic programming
  - Boosting requires linear programming
- Finding high dimensional separators
  - SVM uses kernels (inner products of examples)
  - Boosting uses weak learners to handle this
  - There is usually a big difference between the learning spaces of the weak learners and the kernels
- This isn't the whole story
  - Only part of the bound corresponds to maximizing the margin

Story of Boosting

- These explanations illustrate the story of boosting
- Boosting works
  - But why?
- Different analyses yield different conclusions
  - Some not supported by empirical evidence
- Build new boosting algorithms based on these observations
  - Some work, others don't

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
  - The 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={-1,+1}
- For each example x_i
  - Predict

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

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

Weighted Majority

- An online algorithm for learning mixtures of experts
- Bounded by regret to the best expert
  - Theorem states: 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 the best expert, this is great
    - 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 stock broker 100 times and average?
  - Call 100 different stock brokers 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 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

Summary

- We can do a lot by combining a little
  - Boosting: turns weak learners into strong learners
- Mixtures of experts
  - Weighted majority uses the predictions of the best experts
- Diversity helps
  - Create artificial diversity
    - Instance bagging
    - Feature bagging
  - Diversity in representations for semi-supervised learning
    - Co-training