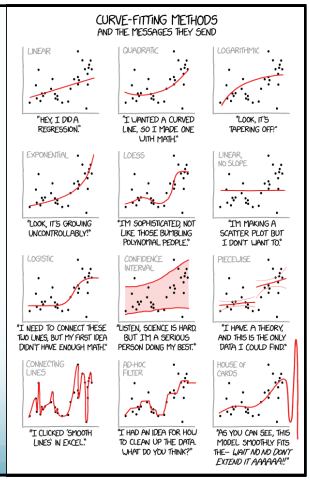
Boosting

Professor: Mark Dredze

Machine Learning CS 601.475

Slides are based on tutorial by Freund and Schapire

Guest lecture by Tim Vieira



Different Approach

- So far we've covered
  - Linear and non-linear models for supervised learning (classification)
- Today's idea:
  - Combine several classifiers to improve accuracy

Ensemble Learning

- Combine multiple classifiers together to get a better classifier
- Basic idea:
  - Train a bunch of classifiers (possibly a mix of SVMs, perceptrons, logistic regressions, & decision trees)
  - Take a majority vote among these to get final classification
- This already works surprisingly well.
  - Can we do better? Ideas?

Consider the extreme case

- Let H be my set of classifiers
- Terminology: H is a …
  - weak learner – if there is always an $h \in H$ that strictly better than random guessing (but possibly not much better)
  - strong learner – if there is always an $h \in H$ with arbitrarily high accuracy (up to label noise)

Theoretical Question

- Does weak learnability imply strong learnability?
  - Posed by Kearns and Valiant (1988)
- Answer
  - Yes! Boosting (Schapire, 1989)
  - We can "boost" the accuracy (on training data) of any weak learner arbitrarily high (up to the noise inherent in the data), i.e., make it a strong learner.
  - Using a carefully designed voting scheme
  - Important! Weak learner must be *strictly* better than chance

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 15,000+ times on Google Scholar

Boosting History

- Very resistant to over-fitting
- Can be interpreted as a form of gradient descent in function space
- Boosting is widely used industry

What is Boosting?

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

The AdaBoost Algorithm

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

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

- Output final hypothesis

Questions

- How do we choose subsets of the data? (D_t)
- How do we combine all the rules into predictor? (Voting scheme)
- The answer: AdaBoost

AdaBoost

- AdaBoost: Adaptive Boosting
- 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

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

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

larger for small error
negative for error above 0.5

AdaBoost

- Update the example distribution used in the next round

$$D_{t+1}(i) = \frac{1}{Z_t} D_t(i) \cdot \exp\left(\begin{cases} \alpha_t, & \text{if } h_t(x_i) \neq y_i \\ -\alpha_t, & \text{otherwise} \end{cases} \right)$$

- where Z_t is a normalization constant (ensures D_{t+1} sums to 1)
- Stronger classifiers make larger adjustments (larger α)
- Correct predictions get lower weights in the next round
- Incorrect predictions get higher weights in the next round
- Next learner can focus on getting right what previous ones got wrong

AdaBoost

- Output the final hypothesis:

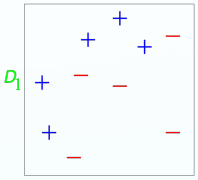
$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

- Low-error h_t are given higher weighting in final prediction (the "adaptive" part of the name)

Notes

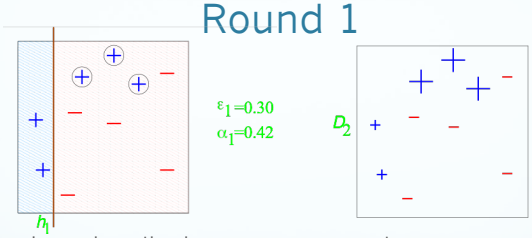
- Distribution tends to concentrate on hard examples
  - Sound familiar?
    - SVMs!
    - Weight on examples close to the margin
    - We'll come back to this point

Example



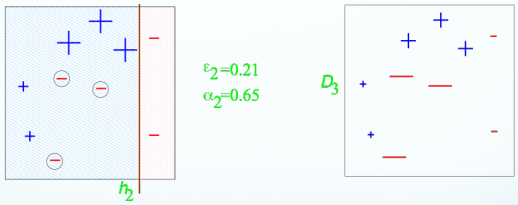
- A set of labeled points with a uniform distribution
- Fit with *decision stump* model
  - Decision tree with max depth 1

Round 1



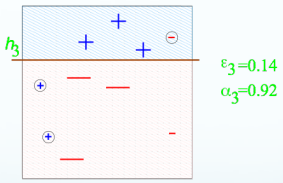
- Learn hypothesis, measure error, set α
- Recompute distribution placing more weight on incorrect examples

Round 2

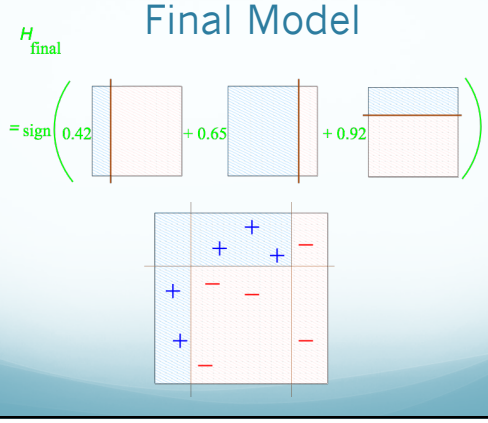


- Learn hypothesis, measure error, set α
- Recompute distribution placing more weight on incorrect examples

Round 3



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, loss function**
- Data: Weigh data based on previous prediction errors?

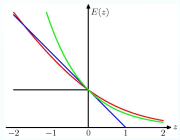
Statistical View of Boosting

- What is boosting doing?
  - We normally think of classifiers in terms of loss or likelihood
  - What is the objective function for boosting?

Exponential Loss

- Exponential loss given by:

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



- f_t is a classifier defined by a linear combination of base classifiers:

$$f_t(\mathbf{x}) = \sum_{k=1}^{t} \alpha_k h_k(\mathbf{x})$$

- Assume as given:

$$h_1, \ldots, h_{t-1} \;\; and \;\; \alpha_1, \ldots, \alpha_{t-1}$$

Exponential Loss

- Rewrite E separating out fixed values at iteration t:

$$E = \sum_{i=1}^{N} \exp\{-y_i f_t(\mathbf{x}_i)\}$$
$$= \sum_{i=1}^{N} \exp\{-y_i f_{t-1}(\mathbf{x}_i) - y_i \alpha_t h_t(\mathbf{x}_i)\}$$
$$= \sum_{i=1}^{N} D_t(i) \exp\{-y_i \alpha_t h_t(\mathbf{x}_i)\}$$

- Minimizing E wrt α_t and h_t yields the AdaBoost solution

Exponential Loss

- AdaBoost can be viewed as an algorithm for minimizing exponential loss
- Choices of α_t and h_t minimize this loss
- A form of coordinate descent
  - Derivative-free optimization
  - At each iteration, from the current point (ensemble) do a "line search" for $\alpha_t h_t$
  - Coordinate direction is new function (h_t) to be added

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.)

Schapire. Explaining AdaBoost

Regularization?

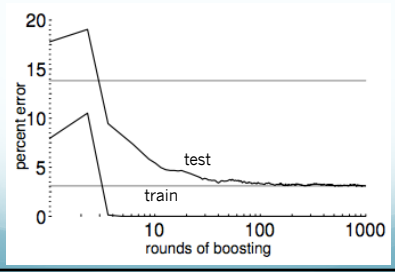
- No explicit form of regularization
- Observation: stopping AdaBoost after any number of rounds provides an approximate solution to L_1 regularized minimization of exponential loss
  - This is a variant of AdaBoost where α is set to a small, fixed constant on each round
  - Requires stopping while t is quite small, larger t is less regularization
- Illustrative but doesn't apply to AdaBoost in practice

Overfitting

- Given all of this, AdaBoost seems like it would overfit
  - No training error after $t \sim \mathcal{O}(\log N)$ iterations
  - Each round focuses more on incorrect examples
- But it doesn't overfit! Why?

Overfitting

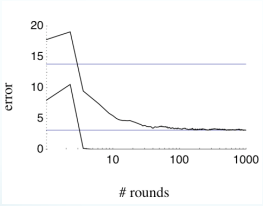
- Train and test error on an OCR dataset with C4.5 (decision tree) as the weak learner



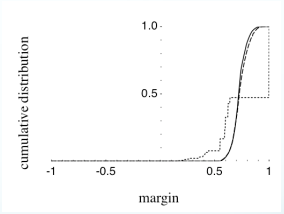
Margin Explanation

- Train error goes to 0 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

Margin Explanation



Learning curve for training (lower) and test (upper)



Cumulative distribution of margins of the training examples
5,100,1k iterations (dashed to solid)
Boosting increases the margin even after training error goes to 0

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
- Writing boosting as maximizing the minimum margin
  - Assume *h* functions already knows, 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

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

- Norms used for boosting

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

- Norms used for SVMs are Euclidean

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

- Boosting and SVMs are very similar

Differences

- Different norms can result in different margins
  - For 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 while Boosting uses weak learners
  - There is usually a big difference between the learning spaces of the kernels and the weak learners
- This isn't the whole story
  - Only part of the bound corresponds to maximizing the margin

Story of Boosting

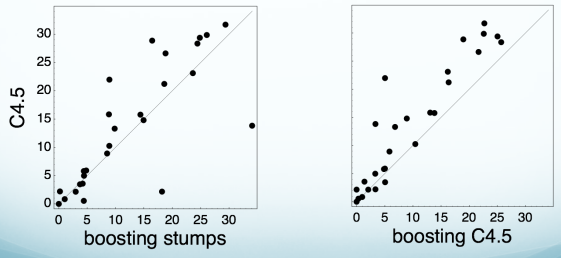
- These explanations illustrate the story of boosting
- Boosting works
  - But why?
- Different analyses yield different conclusions
  - Some not supported by empirical evidence
- Build new boosting algorithms based on these observations
  - Some work, others don't

What Should We Boost?

- If boosting improves a base classifier…
  - Boost the best classifier we can!
  - Boosted SVMs, KNN, Perceptrons, Logistic Regression, etc.
  - Problem: you have to train many of these, may not be efficient
- But boosting can boost 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)



Points above the line are better for boosting

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 $\hat{y}_i = \text{sign}\left(\sum_k w_k f_k(x_i)\right)$
  - Update:
    - For each expert k
      - If $y_i \neq f_k(x_i)$
        - $w_k = w_k / 2$

Weighted Majority

- An online algorithm for learning mixtures of experts
  - Have experts, learn the mixture parameters
- Bounded by regret of 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 the best expert, this is great
    - Only a logarithmic 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
    - 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!

Summary

- Boosting
  - Turns weak learners into a strong one
- Mixtures of experts
  - Weighted majority uses the predictions of the best experts
- Diversity helps
  - Create artificial diversity: instance and feature bagging

References

- Freund & Schapire 1999. "A Short Introduction to Boosting" https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf

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

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

- 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.