CSC 411 Lecture 19-20: Ensembles

Ethan Fetaya, James Lucas and Emad Andrews

University of Toronto

Today

- Ensemble Methods
- Bagging
- Random forest
- Boosting

Ensemble methods

- Back to supervised learning.
- Ensemble of predictors is a set of predictors whose individual decisions are combined in some way to classify new examples
- Simplest approach:
  1. Generate multiple classifiers
  2. Each votes on test instance
  3. Take majority/average as prediction
- Classifiers are different due to different sampling of training data, or randomized parameters within the classification algorithm
- Aim: take simple mediocre algorithm and transform it into a super classifier without requiring any fancy new algorithm

Ensemble methods: Overview

- Differ in training strategy, and combination method
  - Parallel training with different training sets
    1. Bagging (bootstrap aggregation) – train separate models on overlapping training sets, average their predictions
  - Sequential training, iteratively re-weighting training examples so current classifier focuses on hard examples: boosting
  - Parallel training with objective encouraging division of labor: mixture of experts (not covered)
- Notes:
  - Also known as meta-learning

Bias-Variance decomposition

- Best way to understand Bagging is via the bias-variance decomposition
- Consider regression with L_2 loss and define $h^*(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}]$ to be the Bayes-optimal classifier.
- Define the ML algorithm prediction (trained on data $\mathcal{D}$) as $y(\mathbf{x};\mathcal{D})$
- We are interested in breaking $\mathbb{E}_{\mathcal{D},\mathbf{x},t}[(t-y(\mathbf{x};\mathcal{D}))^2]$ into its core components.
    - The expected test error when we sample a random training set.
- First step:

$$\mathbb{E}_t[(t-y(\mathbf{x};\mathcal{D}))^2|\mathcal{D},\mathbf{x}] = \mathbb{E}_t[(t-h^*(\mathbf{x})+h^*(\mathbf{x})-y(\mathbf{x};\mathcal{D}))^2|\mathcal{D},\mathbf{x}] =$$
$$\mathbb{E}_t[(t-h^*(\mathbf{x}))^2|\mathcal{D},\mathbf{x}] + \mathbb{E}_t[(h^*(\mathbf{x})-y(\mathbf{x};\mathcal{D}))^2|\mathcal{D},\mathbf{x}]$$

- The third term disappears because $\mathbb{E}[t|\mathbf{x}] = h^*(\mathbf{x})$
- $\mathbb{E}_{\mathcal{D},\mathbf{x},t}[(t-y(\mathbf{x};\mathcal{D}))^2] = \mathbb{E}_{\mathcal{D},\mathbf{x},t}[(t-h^*(\mathbf{x}))^2] + \mathbb{E}_{\mathcal{D},\mathbf{x},t}[(h^*(\mathbf{x})-y(\mathbf{x};\mathcal{D}))^2]$
- The first term is called the noise and we have no control over it.

Bias-Variance decomposition

$$\mathbb{E}_{\mathcal{D},\mathbf{x},t}[(t - y(\mathbf{x};\mathcal{D}))^2] = \mathbb{E}_{\mathcal{D},\mathbf{x},t}[(t - h^*(\mathbf{x}))^2] + \mathbb{E}_{\mathcal{D},\mathbf{x},t}[(h^*(\mathbf{x}) - y(\mathbf{x};\mathcal{D}))^2]$$

- We can use the same trick to break down the second term

$$\mathbb{E}_{\mathcal{D}}[(h^*(\mathbf{x}) - y(\mathbf{x};\mathcal{D}))^2|\mathbf{x}] = \mathbb{E}_{\mathcal{D}}[(h^*(\mathbf{x}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})] - y(\mathbf{x};\mathcal{D}))^2|\mathbf{x}] =$$
$$\mathbb{E}_{\mathcal{D}}[(h^*(\mathbf{x}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})])^2|\mathbf{x}] + \mathbb{E}_{\mathcal{D}}[(y(\mathbf{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x};\mathcal{D})])^2|\mathbf{x}]$$

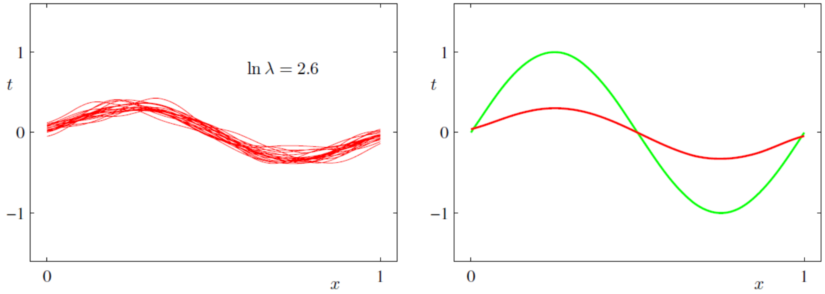
- We get: $Error = (bias)^2 + Variance + noise$

$$\begin{aligned}(bias)^2 &= \mathbb{E}_{\mathbf{x}}[(h^*(\mathbf{x}) - \mathbb{E}[y(\mathbf{x};\mathcal{D})])^2]\\ Variance &= \mathbb{E}_{\mathcal{D},\mathbf{x}}[(y(\mathbf{x};\mathcal{D}) - \mathbb{E}[y(\mathbf{x};\mathcal{D})])^2]\\ noise &= \mathbb{E}_{\mathcal{D},\mathbf{x},t}[(t - h^*(\mathbf{x}))^2]\end{aligned}$$

- If we overfit we have high variance

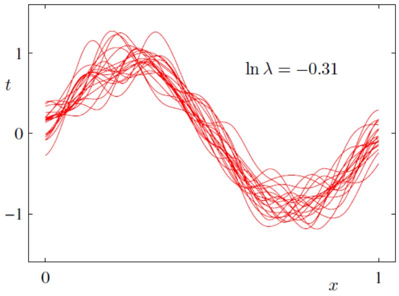
Example:

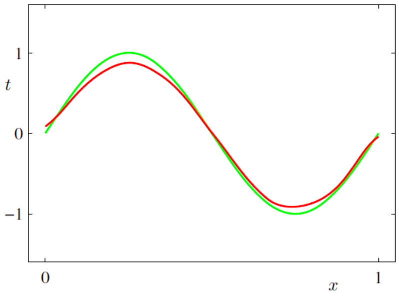
- Fitting a polynomial to 1d input (linear regression) for various regularization values.
- We can sample different datasets and see the variance in predictions and bias (loss of averaged prediction)



- Left: Predictions trained on various sampled datasets. Low variance
- Right: The mean prediction. High bias

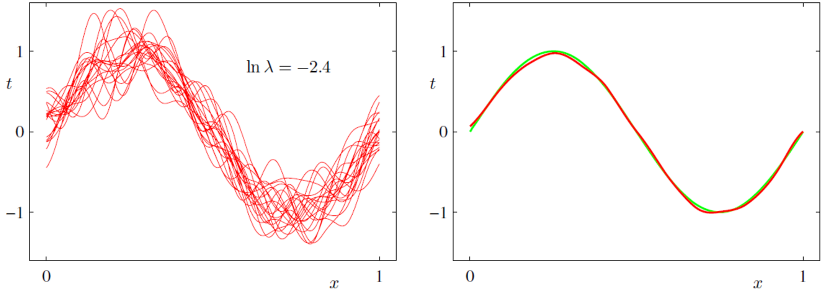
Example:





- Left: Predictions trained on various sampled datasets. high variance
- Right: The mean prediction. Low bias

Example:



- Left: Predictions trained on various sampled datasets. Higher variance
- Right: The mean prediction. Lower bias

Bagging

- We can decompose the error to bias and variance.
- Over-fitting models have high variance (hopefully low bias).
- How can we reduce variance?
- Simple way - you average i.i.d samples - $Var(\frac{1}{m}\sum_i x_i) = \frac{1}{m}Var(x_1)$
- Problem: The samples are other training sets, we only have one.
- Solution: Bootstrapping, generating many training sets from our one set.
- How is this done? By sampling with replacement.
- We can train a separate predictor for each training example and average predictions.
- If samples have correlation ρ: $Var(\frac{1}{m}\sum_i x_i) = \frac{1}{m}(1-\rho)\sigma^2 + \rho\sigma^2$
- Works well if they have low correlation.

Bootstrapping

- Bootstrapping is a classical statistics technique.
- Example: We have an unbiased estimation of some parameter and want to estimate the variance (e.g. for confidence intervals).
- How can you estimate variance? You sample more from the data distribution and estimate the variance using these extra samples.
- What do you do if you cannot sample more? Bootstrap! Replace the data distribution with the empirical distribution.
- Sampling from the empirical distribution is the same as sampling with replacement from your dataset.
- Some theoretical justification (e.g. Glivenko-Cantelli theorem)
- Bagging = Bootstrap Aggregation

Bagging algorithm

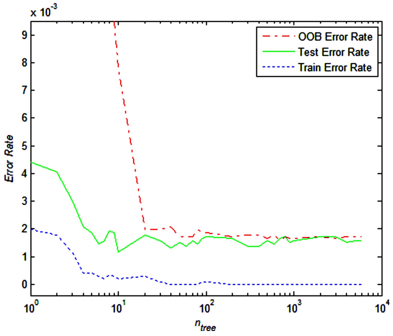
- Input: dataset $\mathcal{D}$, ML algorithm A, number of bags N.
- For i = 1,...,N:
  - Generate dataset $\mathcal{D}_i$ by sampling with replacement from $\mathcal{D}$ (same number of elements).
  - $f_i = A(\mathcal{D}_i)$
- return: $f_1, ..., f_N$.
- Doing predictions: given new example $\mathbf{x}$ return $\frac{1}{N} \sum_i f_i(\mathbf{x})$ (or majority for classification)
- Thats it!

Random Forest

- Bagging reduces overfitting by averaging predictions.
- Works well with decision trees. Why?
  - They overfit easily (high variance to reduce).
  - Fast to perform inference.
  - Powerful method on relational data
- Random forest is decision trees+ bagging + one more trick.
- To reduce correlation even more - each split only considers a random subset of features.
- How do decision boundaries look like?

Out-of-bag estimation

- In bagging there is a nice way to cheaply estimate test loss
- Each training example only appears in some of the "bagged" trees.
  - Probability of not being picked is $\sim 1/e$
- OOB estimation: We predict each training example using all the trees that did not contain this in their training data.



Bagging Overview

- Bagging reduces overfitting by averaging predictions.
- Used in most competition winners
  - Even if a single model is great, a small ensemble usually helps.
- Easy to parallelize.
- Limitations:
  - Does not reduce bias.
  - There is still correlation between classifiers.
- Random forest solution: Add more randomness.
- OOB estimation reduces the need of validation/cross-validation.

Boosting overview

- Boosting is another ensemble method.
- It reduces bias by making each classifier focus on previous mistakes
- Training is sequentially.
- We will talk about AdaBoost (binary classification)
- Started by a theoretical question: Can you take a "weak" classifier that does ϵ better then chance and "boost" it to get low training error?
- Answer is yes! Our classifier is $H(x) = sign\left(\sum_i \alpha_i h(x_i)\right)$ with h_i weak classifiers.
- Note that we sum predictions (after sign) so sum of linear classifiers isn't a linear classifier!

AdaBoost

- The first practical boosting algorithm is adaBoost (adaptive boosting).
- The idea: At each iteration you reweigh the training sample, giving larger weight to points that where classified wrongly and train a new weak classifier.
- The weak learner needs to minimize weighted accuracy.
- Assume the weak learner can get ϵ better then chance $(1/2)$.

AdaBoost Algorithm

- Input: $\{\mathbf{x}^{(n)}, t^{(n)}\}_{n=1}^{N}$, and WeakLearn: learning procedure, produces classifier $H(\mathbf{x})$
- Initialize example weights: $D_n^m(\mathbf{x}) = 1/N$
- For m=1:M
  - $h_m(\mathbf{x}) = WeakLearn(\{\mathbf{x}\}, \mathbf{t}, \mathbf{w})$, fit classifier by minimizing

$$J_m = \sum_{n=1}^{N} D_m^{(n)} [h_m(\mathbf{x}^n) \neq t^{(n)}]$$

  - Compute weighted error rate

$$\epsilon_m = \frac{J_m}{\sum D_m^{(n)}}$$

  - Compute classifier coefficient $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
  - Update data weights

$$D_{m+1}^{(n)} = D_m^{(n)} \exp\left(-t^{(n)} \alpha_m h_m(\mathbf{x}^{(n)})\right)$$

- Final model

$$H(\mathbf{x}) = sign(F(\mathbf{x})) = sign(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x}))$$

AdaBoost Example

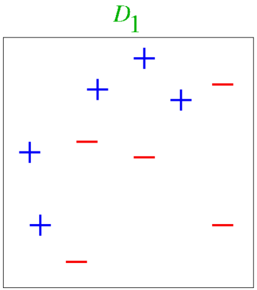
- ϵ_t is the weighted error, assuming less then $1/2$.
- $\alpha_t = \frac{1}{2}\log(\frac{1-\epsilon_t}{\epsilon_t})$ measures the classifier quality.
- Weight the binary prediction of each classifier by the quality of that classifier:

$$H(\mathbf{x}) = \text{sign}(F(\mathbf{x})) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right)$$

- This is how to do inference, i.e., how to compute the prediction for each new example.

AdaBoost Example

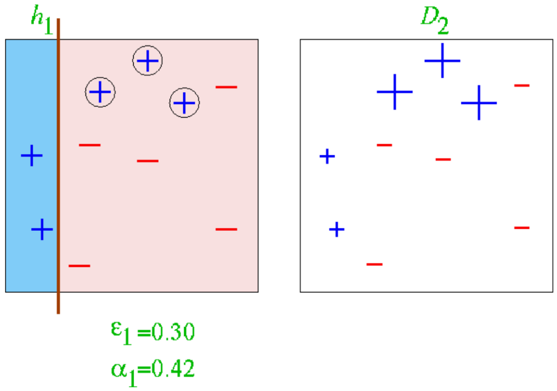
- Training data



[Slide credit: Verma & Thrun]

AdaBoost Example

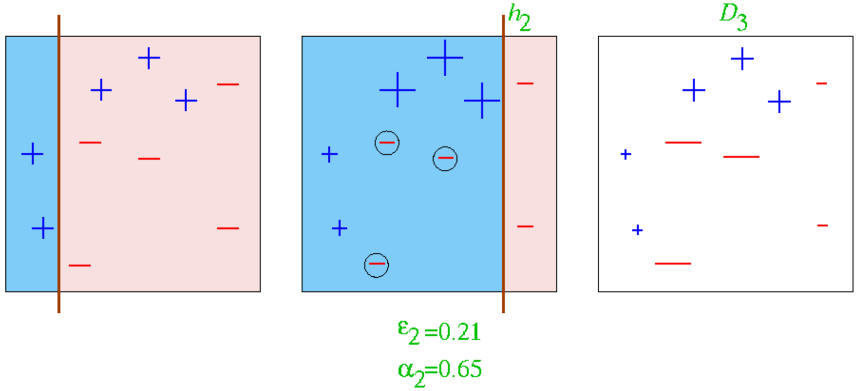
- Round 1



[Slide credit: Verma & Thrun]

AdaBoost Example

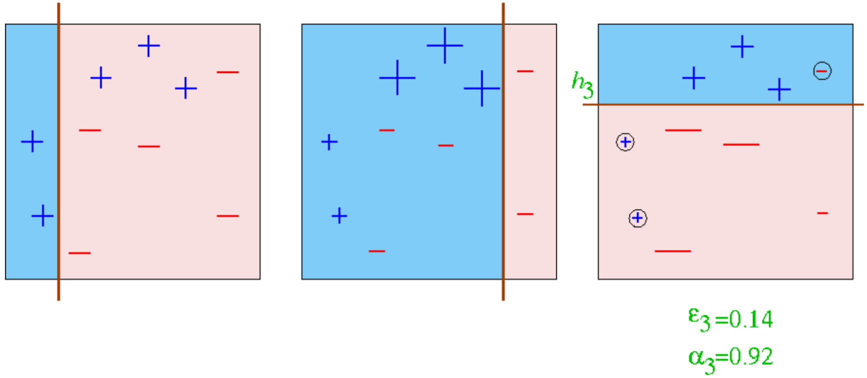
- Round 2



[Slide credit: Verma & Thrun]

AdaBoost Example

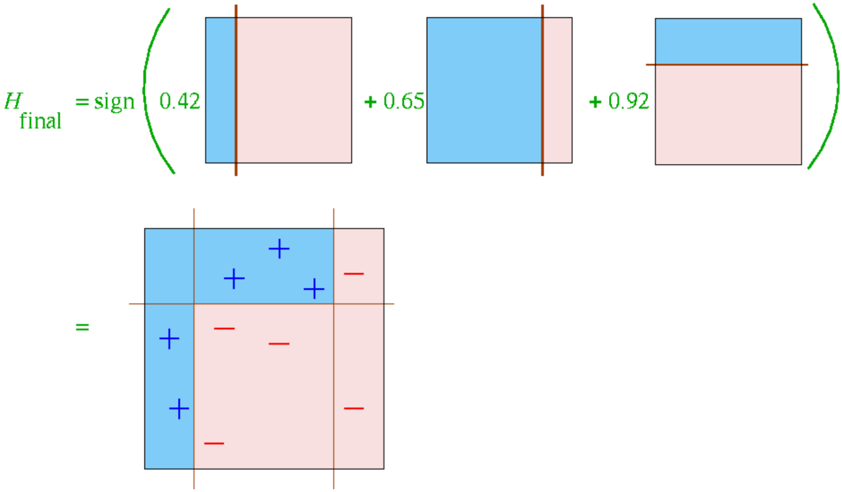
- Round 3



[Slide credit: Verma & Thrun]

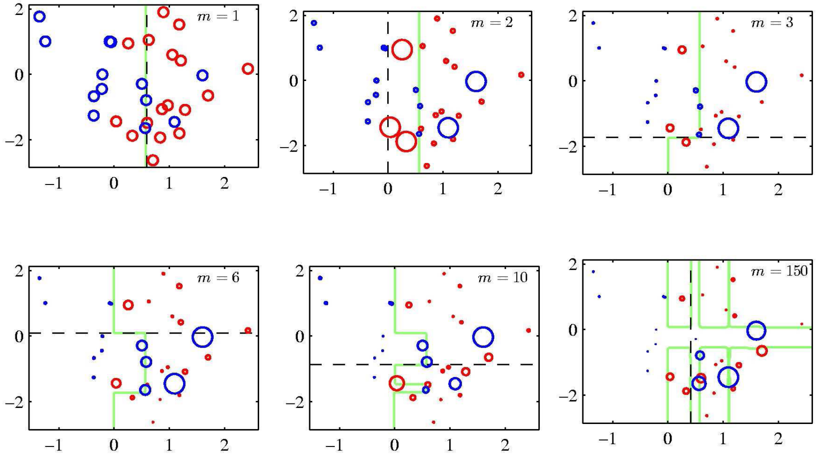
AdaBoost Example

- Final classifier



[Slide credit: Verma & Thrun]

AdaBoost example



- Each figure shows the number *m* of base learners trained so far, the decision of the most recent learner (dashed black), and the boundary of the ensemble (green)

Algorithm analysis

- We will now show that the overall training error decreases exponentially (and it will explain α_t)

Theorem:
Let ϵ_m be the WL error at iteration m and define $\gamma_m = 1/2 - \epsilon_m$. The training loss of the boosted classifier $H(\mathbf{x}) = sign\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$

$$L_S(H) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[H(\mathbf{x}^{(i)}) \neq t^{(i)})] \leq \exp\left(-2\sum_{m=1}^{M}\gamma_m^2\right)$$

- If we assume $\gamma_m \geq \gamma$ then we can simplify the bound to $\exp\left(-2\gamma^2 M\right)$

Proof Intuition

- The idea before we dive into the math:
- The boosted classifier does a (weighted) majority voting.
- For it to make a mistake on $\mathbf{x}^{(i)}$ most (weighted) rounds must be erroneous.
- Weight of $\mathbf{x}^{(i)}$ increases exponentially with mistakes so it has a large weight.
- The weak classifier is better than chance so the total weight decreases.
- This means there can only be few items with large weight so few mistakes.

Proof I *

Proof: We have $F(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})$ and define $H(\mathbf{x}) = sign(F(\mathbf{x}))$.
We can write $\mathbf{D}_M^{(i)}$ using the algorithm recursive formula:

$$\mathbf{D}_M^{(i)} = \mathbf{D}_{M-1}^{(i)} \exp(-\alpha_M t^{(i)} h_M(\mathbf{x})) =$$
$$\mathbf{D}_{M-2}^{(i)} \exp(-\alpha_{M-1} t^{(i)} h_{M-1}(\mathbf{x})) \exp(-\alpha_M t^{(i)} h_M(\mathbf{x})) = D_1^{(i)} \exp(-t^{(i)} F(\mathbf{x}^{(i)}))$$

Next we note that the 0-1 loss is bounded by the exponential loss
$\mathbb{1}[H(\mathbf{x}) \neq t] \leq \exp(-tF(\mathbf{x}))$
We now have

$$L_S(H) = \sum_{i=1}^{N} \mathbf{D}_1^{(i)} \mathbb{1}[H(\mathbf{x}^{(i)}) \neq t^{(i)}] \leq \sum_{i=1}^{N} \mathbf{D}_1^{(i)} \exp(-t^{(i)} F(\mathbf{x}^{(i)})) = \sum_{i=1}^{N} \mathbf{D}_M^{(i)}$$

The number of mistakes is bounded by the total weight!

Proof II *

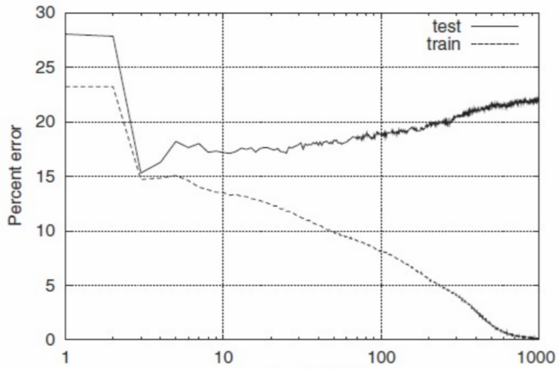
We need to bound the total weight $Z_M = \sum_{i=1}^{N} \mathbf{D}_M^{(i)}$.

$$
\begin{aligned}
Z_{m+1} &= \sum_{i=1}^{N} \mathbf{D}_{m+1}^{(i)} = \sum_{i=1}^{N} \mathbf{D}_m^{(i)} \exp(-\alpha_{m+1} t^{(i)} h_{m+1}(\mathbf{x}^{(i)})) \\
&= \sum_{t^{(i)} = h_{m+1}(\mathbf{x}^{(i)})} D_m^{(i)} e^{-\alpha_{m+1}} + \sum_{t^{(i)} \neq h_{m+1}(\mathbf{x}^{(i)})} D_m^{(i)} e^{\alpha_{m+1}} \\
&= Z_m \left(e^{-\alpha_{m+1}} (1 - \epsilon_{m+1}) + \epsilon_{m+1} e^{\alpha_{m+1}} \right)
\end{aligned}
$$

Can show α_{m+1} picked by the algorithm minimizes this term and it is equal to $Z_m \sqrt{1 - 4\gamma_{m+1}^2}$.
This gives a bound of $\prod_{m=1}^{M} \sqrt{1 - 4\gamma_m^2} \leq \exp(-2 \sum \gamma_m^2)$ finishing the proof (with a few last steps skipped)

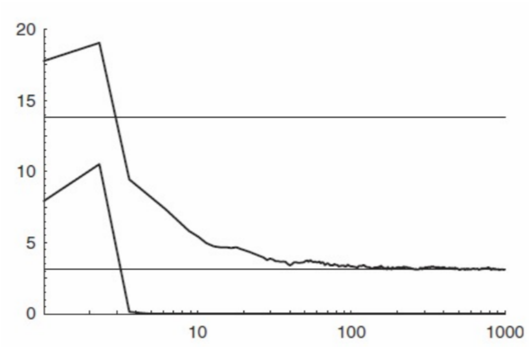
AdaBoost generalization

- We have seen how AdaBoost training loss converges to zero, what about test loss?
- Can show the complexity (defined in some manner) grows linearly with iterations.
- If you run AdaBoost long enough it can overfit.



AdaBoost generalization

- However, many times it does not.
- Sometimes the test error decreases even after the training error is zero!



- How does that happen?

AdaBoost alternative viewpoint

- Another way to see AdaBoost sheds some light on this.
- We defined $F(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})$ and define $H(\mathbf{x}) = sign(F(\mathbf{x}))$. Can think of AdaBoost as a greedy optimization of the exponential loss $\exp(-t^{(i)} F(\mathbf{x}^{(i)}))$
- Can show this leads to a large margin.
- Can show the margin leads to good generalization.
- The paper for whoever is interested https://www.cc.gatech.edu/~isbell/tutorials/boostingmargins.pdf

AdaBoost alternative viewpoint

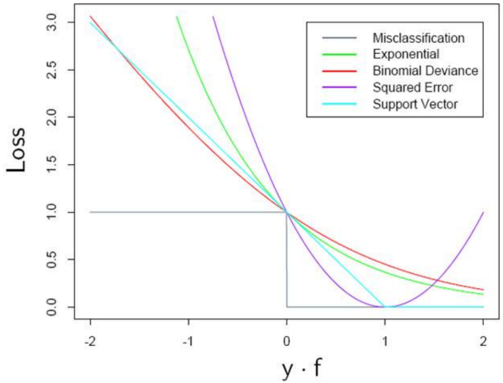
- How do we see this other viewpoint?
- Define $F_t(\mathbf{x}) = \sum_{m=1}^{t} \alpha_m h_m(\mathbf{x})$. If we fix F_t and try to find F_{t+1} that maximizes

$$\frac{1}{N}\sum_{i=1}^{N} \exp(-t^{(i)} F_{t+1}(\mathbf{x}^{(i)})) = \frac{1}{N}\sum_{i=1}^{N} \exp(-t^{(i)} F_t(\mathbf{x}^{(i)})) \exp(-t^{(i)} \alpha_{t+1} h_{t+1} \mathbf{x}^{(i)})$$

$$\sum_{i=1}^{N} \mathbf{D}_t^{(i)} \exp(-t^{(i)} \alpha_{t+1} h_{t+1} \mathbf{x}^{(i)})$$

- If h_{t+1} has weighted accuracy ϵ then the optimal α is the one used by AdaBoost and the total loss is $2\sqrt{\epsilon(1-\epsilon)}$
- This is minimized when ϵ is minimized - so h_{t+1} should minimized the weighted accuracy which is what AdaBoost does.

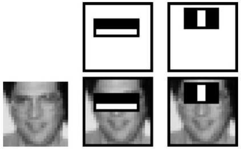
Loss Functions



- Misclassification: 0/1 loss
- Exponential loss: $\exp(-t \cdot f(x))$ (AdaBoost)
- Squared error: $(t - f(x))^2$
- Soft-margin support vector (hinge loss): $\max(0, 1 - t \cdot y)$

An impressive example of boosting

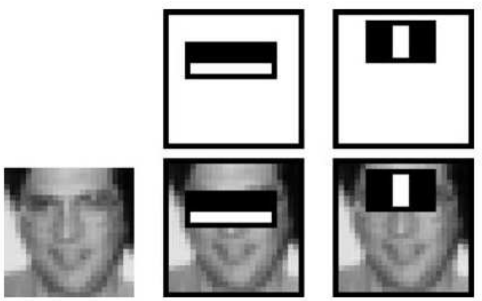
- Viola and Jones created a very fast face detector that can be scanned across a large image to find the faces.



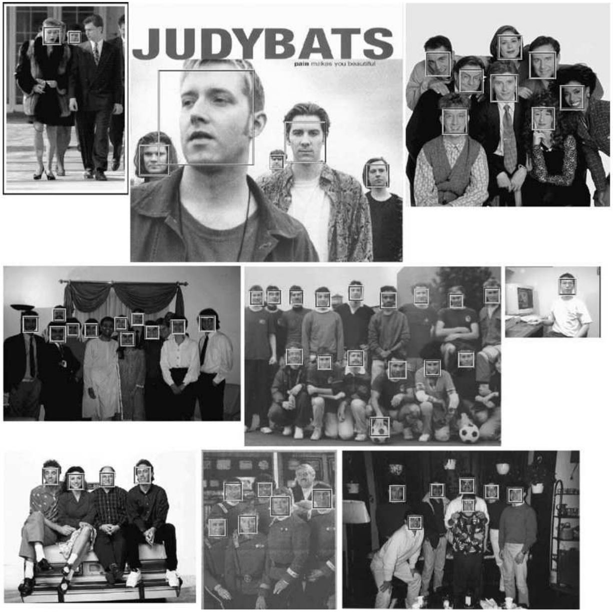
- The base classifier/weak learner just compares the total intensity in two rectangular pieces of the image.
    - There is a neat trick for computing the total intensity in a rectangle in a few operations.
        - So its easy to evaluate a huge number of base classifiers and they are very fast at runtime.
    - The algorithm adds classifiers greedily based on their quality on the weighted training cases.

AdaBoost in Face Detection

- Famous application of boosting: detecting faces in images
- Few twists on standard algorithm
  - Pre-define weak classifiers, so optimization=selection
  - Change loss function for weak learners: false positives less costly than misses
  - Smart way to do inference in real-time (in 2001 hardware)



AdaBoost Face Detection Results



Boosting recap

- Boosting is an ensemble method that reduces bias
- We have shown AdaBoost a boosting algorithm for binary classification.
- Viewing AdaBoost as a greedy optimization of the exponential loss lead to many extensions.
  - Boosting for ranking (RankBoost)
  - Boosting for multiclass classification
  - Boosting for regression
  - Gradient boosting (in tutorial)
- Exponential loss also shows this isn't robust to outliers (some extensions try to fix this)
- Quiet resistant to overfitting but can still overfit
- Usually used with decision stumps, axis aligned or linear classifiers.

Ensembles recap

- Ensembles combine classifiers to improve performance.
- Boosting
  - reduce bias.
  - Increases variance (large ensemble can cause overfitting).
  - Sequential.
  - High dependency between ensemble elements.
- Bagging can reduce variance
  - reduce variance (large ensemble can't cause overfitting).
  - Bias isn't changed
  - Parallel.
  - Minimizes correlation between ensemble elements.

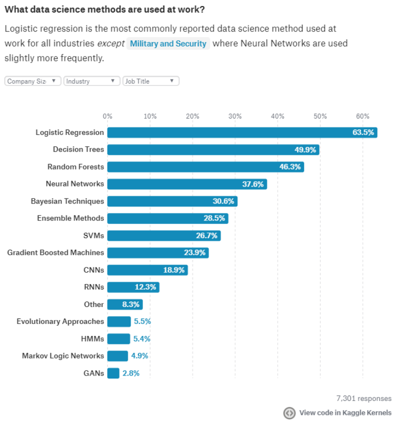
Supervised learning recap

This was the last lecture about supervised learning, what have we seen so far?

- We have seen various ML algorithms - each has its pros and cons.
  - No silver-bullet (not even deep learning), need to fit your solution to the problem.
- Need to understand the inductive bias of each algorithm
  - Can be explicit, e.g. linear classifier.
  - Can be implicit, e.g. nearest neighbor.
- Many times (classification) you cannot optimize the loss you care about.
  - Be sure you understand what it is your optimizing.
  - Does it makes sense as a surrogate loss?
- How do you optimize?
  - Analytic solution (rare cases)
  - Gradient descent/SGD
  - EM algorithm
  - Other alternatives exist, but SGD is the most common.

Kaggle survey

- Recent survey on Kaggle on what ML methods people use



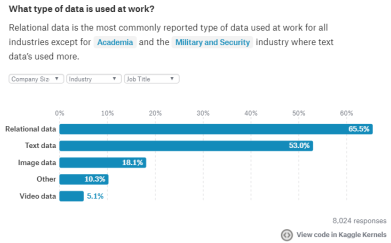
What data science methods are used at work?

Logistic regression is the most commonly reported data science method used at work for all industries *except* Military and Security where Neural Networks are used slightly more frequently.

- Classic methods like logistic regression still dominate!
- Most where covered in this course.

Kaggle survey - data

- Why isn't everyone just using deep learning?
- Deep learning is great for vision/text but not the best at relational data.



What type of data is used at work?

Relational data is the most commonly reported type of data used at work for all industries except for Academia and the Military and Security industry where text data's used more.

- Vision benchmarks dominate the academic ML community, but in industry there are a lot different tasks.
- Most kaggle competitions are won with random forest/gradient boosting.