Sparse vs. Ensemble Approaches to Supervised Learning

Greg Grudic

Modified by Longin Jan Latecki

Some slides are by Piyush Rai

Goal of Supervised Learning?

- Minimize the probability of model prediction errors on ***future*** data

- Two Competing Methodologies
  - Build **one** really good model
    - Traditional approach
  - Build **many** models and average the results
    - Ensemble learning (more recent)

The Single Model Philosophy

- Motivation: Occam's Razor
  - "one should not increase, beyond what is necessary, the number of entities required to explain anything"
- Infinitely many models can explain any given dataset
  - Might as well pick the smallest one…

Which Model is Smaller?

$$\hat{y} = f_1(x) = \sin(x)$$

or

$$\hat{y} = f_2(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

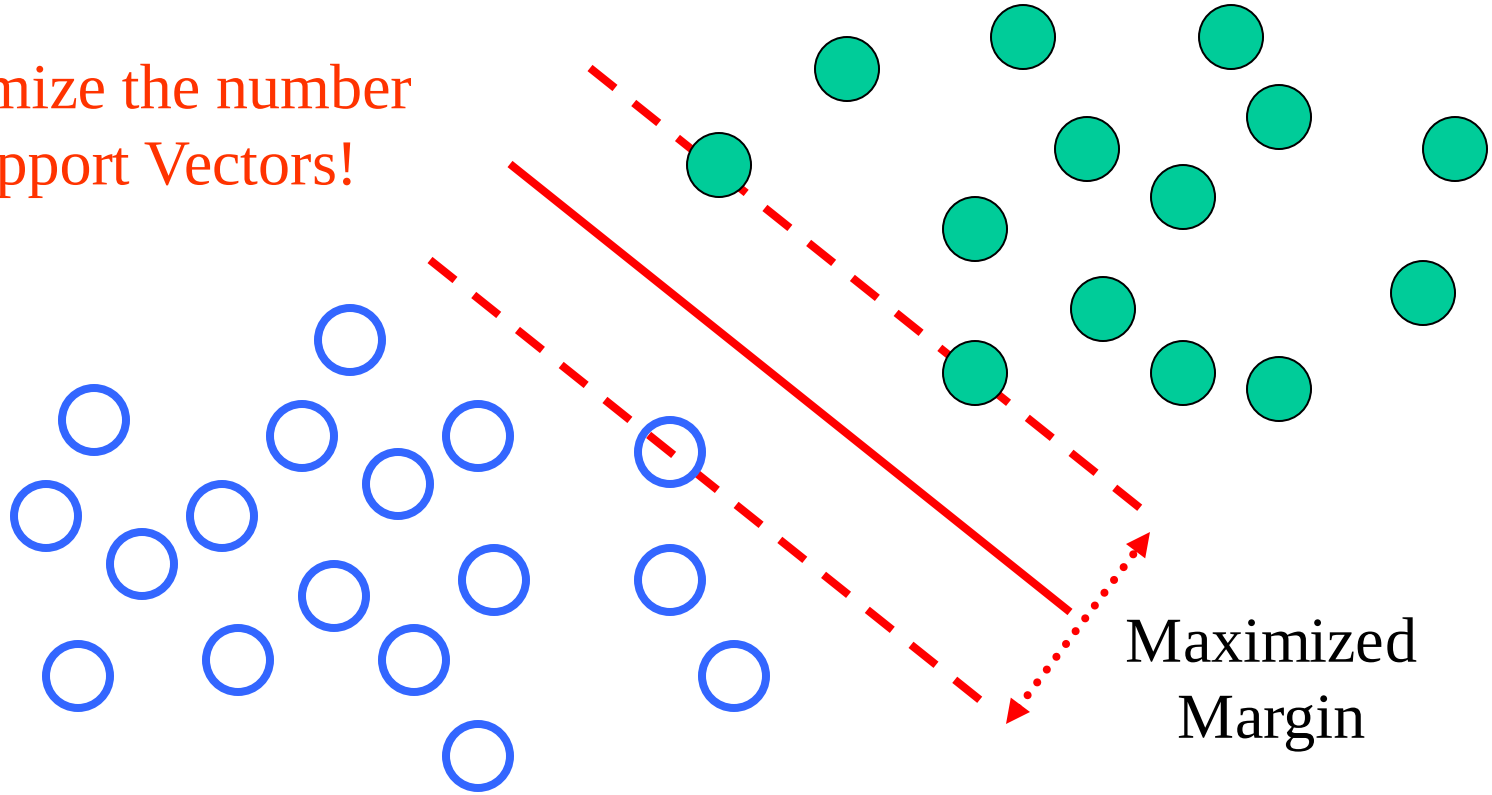
- In this case $f_1(x) \equiv f_2(x)$
- It's not always easy to define small!

Exact Occam's Razor Models

- Exact approaches find optimal solutions
- Examples:
  - Support Vector Machines
    - Find a model structure that uses the smallest percentage of training data (to explain the rest of it).
  - Bayesian approaches
    - Minimum description length

How Do Support Vector Machines Define Small?

Minimize the number of Support Vectors!



Approximate Occam's Razor Models

- Approximate solutions use a greedy search approach which is not optimal
- Examples
  - Kernel Projection Pursuit algorithms
    - Find a minimal set of kernel projections
  - Relevance Vector Machines
    - Approximate Bayesian approach
  - Sparse Minimax Probability Machine Classification
    - Find a minimum set of kernels and features

Other Single Models: Not Necessarily Motivated by Occam's Razor

- Minimax Probability Machine (MPM)
- Trees
  - Greedy approach to sparseness
- Neural Networks
- Nearest Neighbor
- Basis Function Models
  - e.g. Kernel Ridge Regression

Ensemble Philosophy

- Build many models and combine them
- Only through averaging do we get at the truth!
- It's too hard (*impossible*?) to build a single model that works best
- Two types of approaches:
  - Models that don't use randomness
  - Models that incorporate randomness

Ensemble Approaches

- Bagging
  - **B**ootstrap aggregating
- Boosting
- Random Forests
  - Bagging reborn

Bagging

- Main Assumption:
  - Combining many unstable predictors to produce a ensemble (stable) predictor.
  - Unstable Predictor: small changes in training data produce large changes in the model.
    - e.g. Neural Nets, trees
    - Stable: SVM (sometimes), Nearest Neighbor.
- Hypothesis Space
  - Variable size (nonparametric):
    - Can model any function if you use an appropriate predictor (e.g. trees)

The Bagging Algorithm

Given data: $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

For $m = 1 : M$

- Obtain bootstrap sample D_m from the training data D
- Build a model $G_m(\mathbf{x})$ from bootstrap data D_m

The Bagging Model

- Regression

$$\hat{y} = \frac{1}{M}\sum_{m=1}^{M} G_m(\mathbf{x})$$

- Classification:
  - Vote over classifier outputs $G_1(\mathbf{x}),\ldots,G_M(\mathbf{x})$

Bagging Details

- Bootstrap sample of N instances is obtained by drawing N examples at random, with replacement.
- On average each bootstrap sample has 63% of instances
  - Encourages predictors to have uncorrelated errors
    - This is why it works

Bagging Details 2

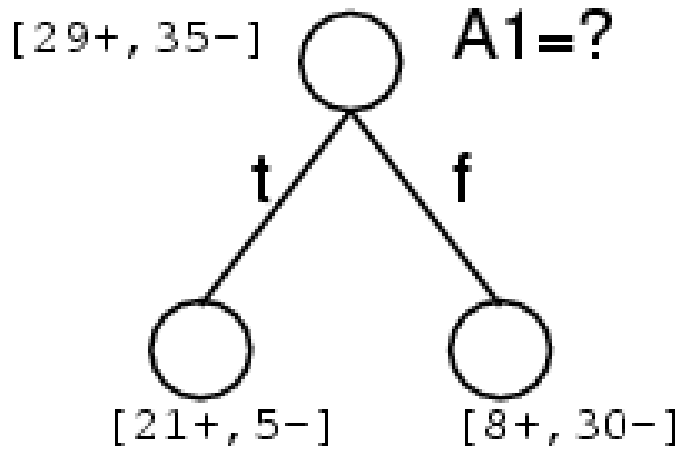
- Usually set $M = \sim 30$
  - Or use validation data to pick M
- The models $G_m(\mathbf{x})$ need to be unstable
  - Usually full length (or slightly pruned) decision trees.

Boosting

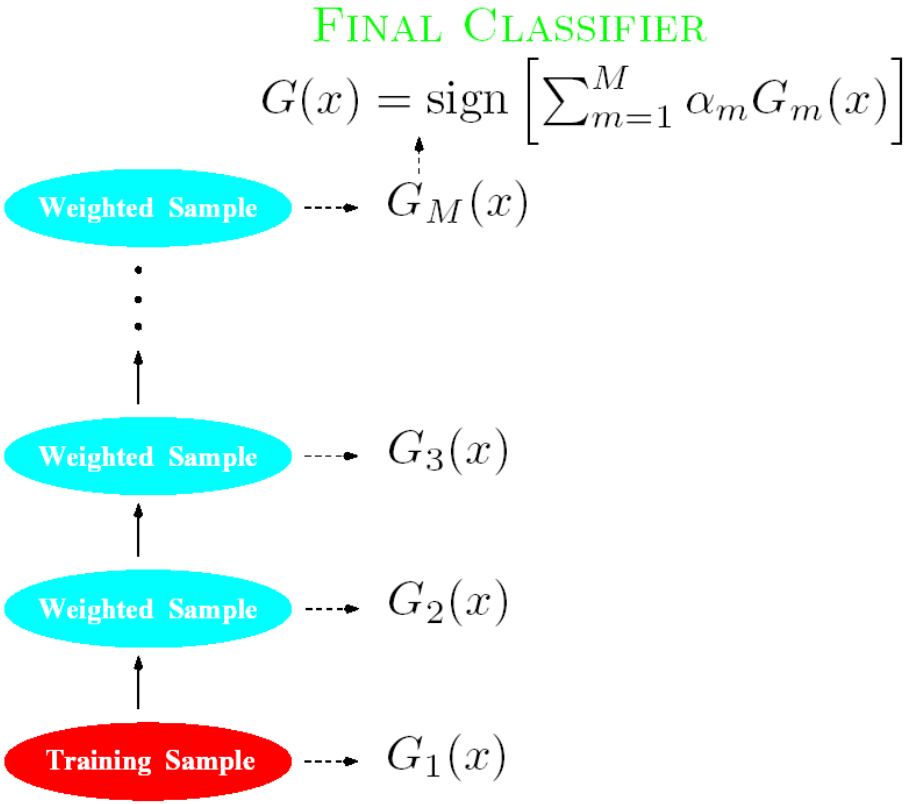
- Main Assumption:
  - Combining many weak predictors (e.g. tree stumps or 1-R predictors) to produce an ensemble predictor
  - The weak predictors or classifiers need to be **stable**
- Hypothesis Space
  - Variable size (nonparametric):
    - Can model any function if you use an appropriate predictor (e.g. trees)

Commonly Used Weak Predictor (or classifier)

A Decision Tree Stump (1-R)



Boosting



Each classifier $G_m(\mathbf{x})$ is trained from a weighted Sample of the training Data

Boosting (Continued)

- Each predictor is created by using a biased sample of the training data
  - Instances (training examples) with high error are weighted higher than those with lower error
- Difficult instances get more attention
  - This is the motivation behind boosting

Background Notation

- The $I(s)$ function is defined as:

$$I(s) = \begin{cases} 1 \text{ if } s \text{ is true} \\ 0 \quad \text{otherwise} \end{cases}$$

- The $\log(x)$ function is the natural logarithm

The AdaBoost Algorithm

(Freund and Schapire, 1996)

Given data: $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

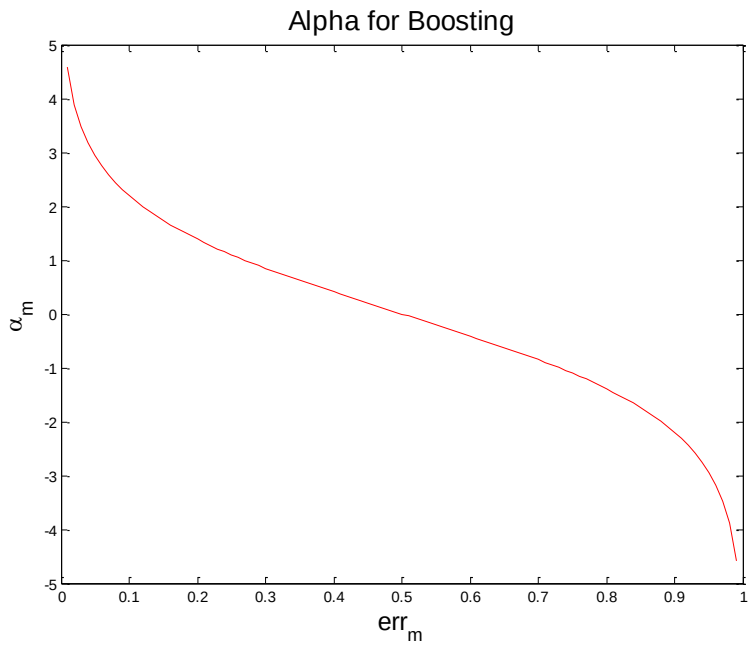
1. Initialize weights $w_i = 1/N, i = 1, \ldots, N$
2. For $m = 1 : M$
   a) Fit classifier $G_m(\mathbf{x}) \in \{-1, 1\}$ to data using weights w_i
   b) Compute $$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(\mathbf{x}_i))}{\sum_{i=1}^{N} w_i}$$
   c) Compute $\alpha_m = \log((1 - err_m)/err_m)$
   d) Set $w_i \leftarrow w_i \exp[\alpha_m I(y_i \neq G_m(\mathbf{x}_i))], \quad i = 1, \ldots, N$

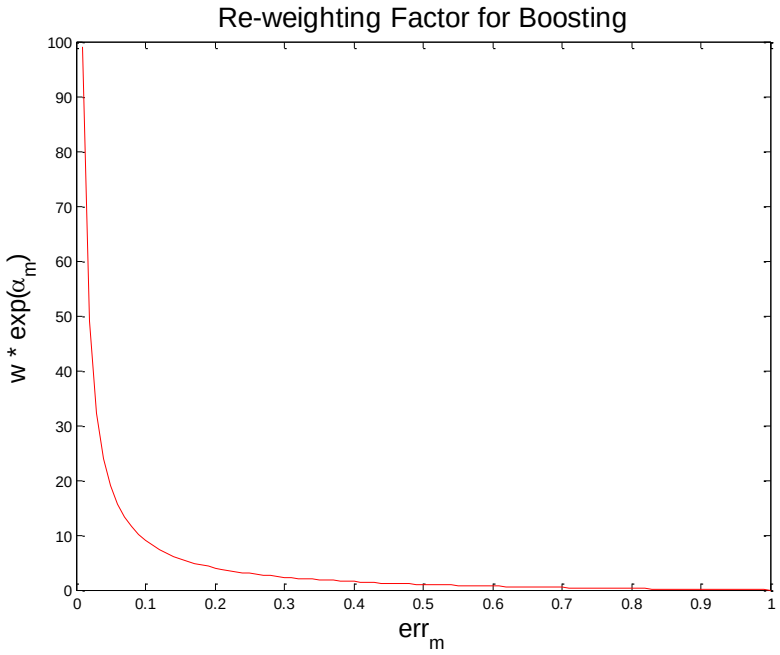
The AdaBoost Model

$$\hat{y} = \text{sgn}\left[\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right]$$

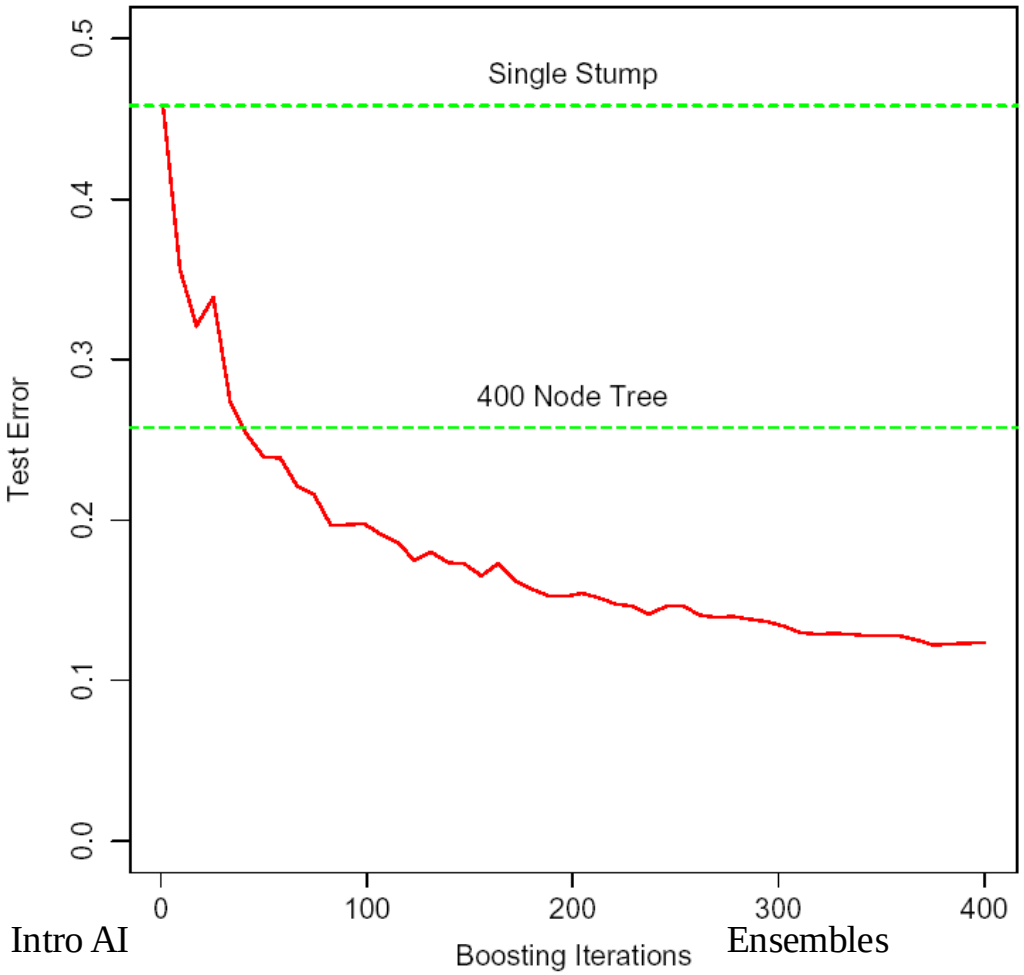
AdaBoost is NOT used for Regression!

The Updates in Boosting



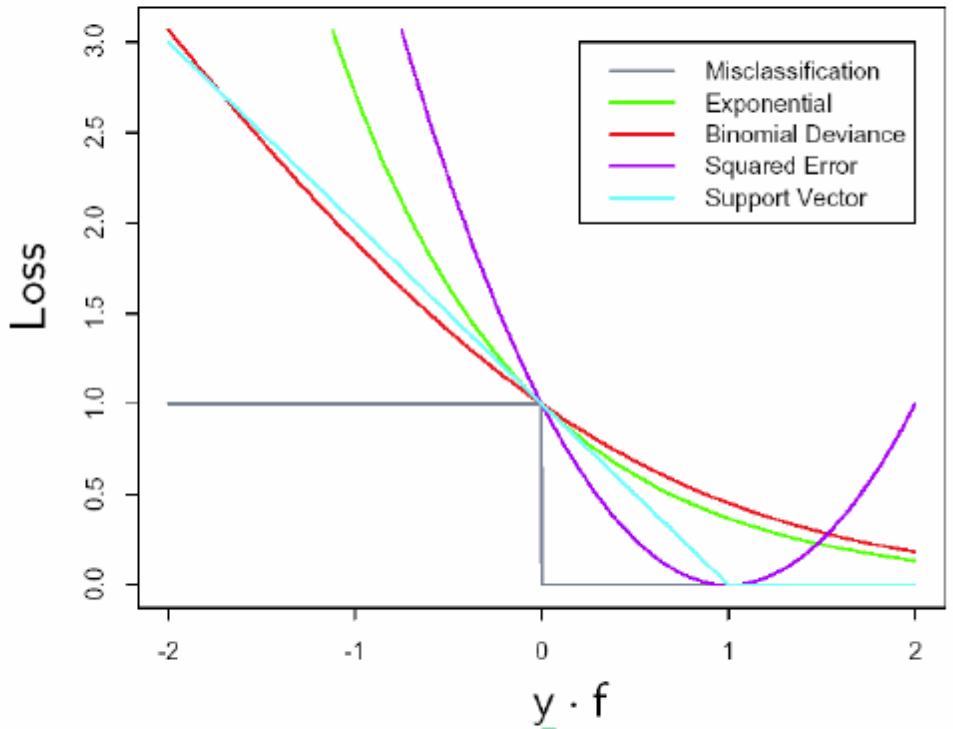


Boosting Characteristics



Simulated data: test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 400 *node tree.*

Loss Functions for $y \in \{-1, +1\}, f \in \Re$



Incorrect Classification | **Correct Classification**

- Misclassification

$$I(\operatorname{sgn}(f) \neq y)$$

- Exponential (Boosting)

$$\exp(-yf)$$

- Binomial Deviance
(Cross Entropy)

$$\log(1+\exp(-2yf))$$

- Squared Error

$$(y-f)^2$$

- Support Vectors

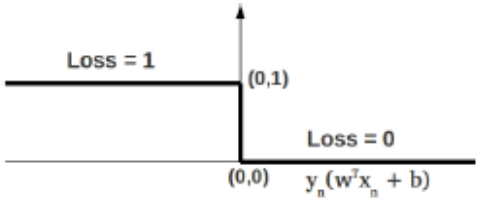
$$(1-yf) \cdot I(yf > 1)$$

Loss Functions for Linear Classification

- We have seen two linear binary classification algorithms (Perceptron, SVM)
- Linear binary classification written as a general optimization problem:

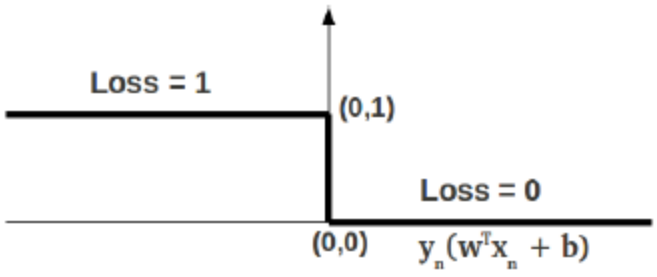
$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T \mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- $\mathbb{I}(.)$ is the indicator function (1 if $(.)$ is true, 0 otherwise)
- The objective is sum of two parts: the loss function and the regularizer
  - Want to fit training data well and also want to have simple solutions
- The above loss function called the 0-1 loss



- The 0-1 loss is NP-hard to optimize (exactly/approximately) in general
- **Different loss function approximations and regularizers lead to specific algorithms** (e.g., Perceptron, SVM, Logistic Regression, etc.).

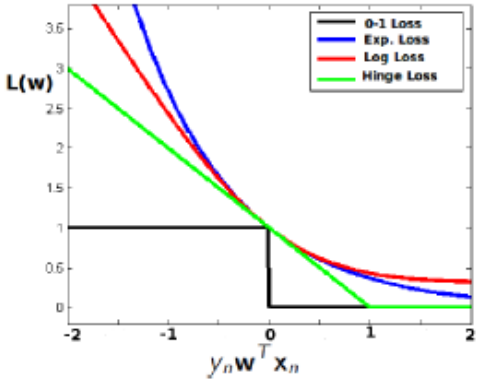
Why is the 0-1 loss hard to optimize?



- It's a combinatorial optimization problem
- Can be shown to be NP-hard
  - .. using a reduction of a variant of the satisfiability problem
- No polynomial time algorithm
- Loss function is non-smooth, non-convex
- Small changes in $\mathbf{w}, b$ can change the loss by a lot

Approximations to the 0-1 loss

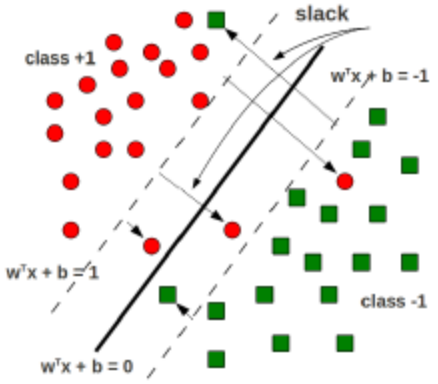
- We use loss functions that are convex approximations to the 0-1 loss
  - These are called surrogate loss functions
- Examples of surrogate loss functions (assuming $b = 0$):
  - Hinge loss: $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$
  - Log loss: $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$
  - Exponential loss: $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$
  - All are convex upper bounds on the 0-1 loss
  - Minimizing a convex upper bound also pushes down the original function
  - Unlike 0-1 loss, these loss functions depend on how far the examples are from the hyperplane



- Apart from convexity, smoothness is the other desirable for loss functions
  - Smoothness allows using gradient (or stochastic gradient) descent
  - Note: hinge loss is not smooth at (1,0) but subgradient descent can be used

Loss functions for specific algorithms

- Recall **SVM** non-separable case: we minimized the sum of slacks $\sum_{n=1}^{N} \xi_n$



  - No penalty ($\xi_n = 0$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$
  - Linear penalty ($\xi_n = 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)$) if $y_n(\mathbf{w}^T\mathbf{x}_n + b) < 1$
  - It's precisely the hinge loss $\max\{0, 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$
  - Note: Some SVMs minimize the sum of **squared** slacks $\sum_{n=1}^{N} \xi_n^2$
- **Perceptron** uses a variant of the hinge loss: $\max\{0, -y_n(\mathbf{w}^T\mathbf{x}_n + b)\}$
- **Logistic Regression** uses the log loss
  - Misnomer: Logistic Regression does classification, not regression!
- **Boosting** uses the exponential loss

Regularizers

- Recall: The optimization problem for regularized linear binary classification:

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- We have already seen the approximation choices for the 0-1 loss function
- What about the regularizer term $R(\mathbf{w}, b)$ to ensure simple solutions?
- The regularizer $R(\mathbf{w}, b)$ determines what each entry w_d of $\mathbf{w}$ looks like
- Ideally, we want most entries w_d of $\mathbf{w}$ be zero, so prediction depends only on a small number of features (for which $w_d \neq 0$). Desired minimization:

$$R^{cnt}(\mathbf{w}, b) = \sum_{d=1}^{D} \mathbb{I}(w_d \neq 0)$$

- $R^{cnt}(\mathbf{w}, b)$ is NP-hard to minimize, so its approximations are used
  - A good approximation is to make the individual w_d's small
  - Small $w_d \Rightarrow$ small changes in some feature x_d won't affect prediction by much
  - Small individual weights w_d is a notion of function simplicity

Norm based Regularizers

- Norm based regularizers are used as approximations to $R^{cnt}(\mathbf{w}, b)$
  - ℓ_2 squared norm: $||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$
  - ℓ_1 norm: $||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$
  - ℓ_p norm: $||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$

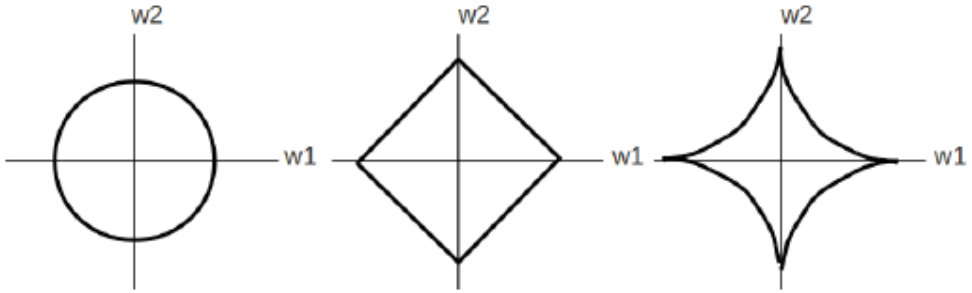


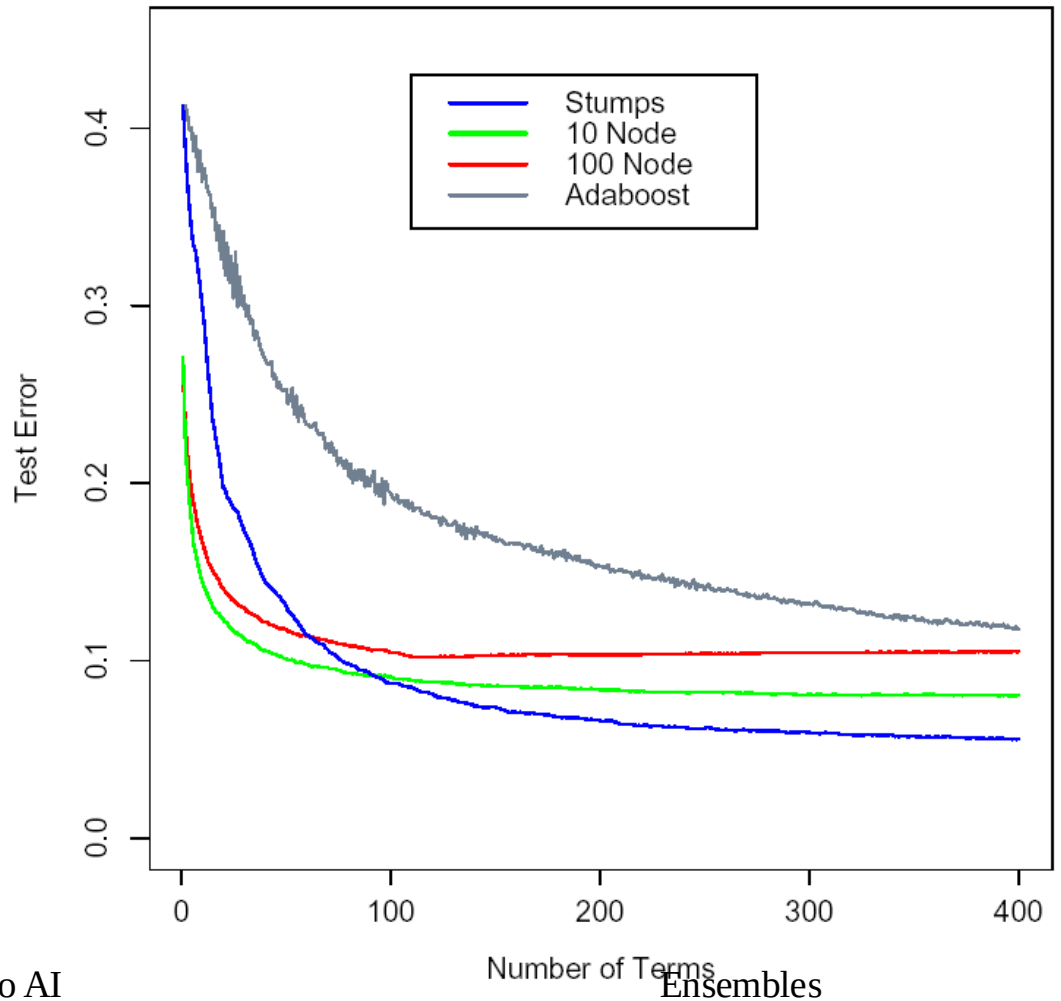
Figure: Contour plots. Left: ℓ_2 norm, Center: ℓ_1 norm, Right: ℓ_p norm (for $p < 1$)

- Smaller p favors sparser vector $\mathbf{w}$ (most entries of $\mathbf{w}$ close/equal to 0)
  - But the norm becomes non-convex for $p < 1$ and is **hard to optimize**
- The ℓ_1 norm is the most preferred regularizer for sparse $\mathbf{w}$ (many w_d's zero)
  - Convex, but it's not smooth at the axis points
  - .. but several methods exists to deal with it, e.g., subgradient descent
- The ℓ_2 squared norm tries to keep the individual w_d's small
  - Convex, smooth, and the easiest to deal with

Other Variations of Boosting

- Gradient Boosting
  - Can use any cost function
- Stochastic (Gradient) Boosting
  - Bootstrap Sample: Uniform random sampling (with replacement)
  - Often outperforms the non-random version

Gradient Boosting



Boosting Summary

- Good points
  - Fast learning
  - Capable of learning any function (given appropriate weak learner)
  - Feature weighting
  - Very little parameter tuning
- Bad points
  - Can overfit data
  - Only for binary classification
- Learning parameters (picked via cross validation)
  - Size of tree
  - When to stop
- Software
  - http://www-stat.stanford.edu/~jhf/R-MART.html