

Applied Machine Learning

Bootstrap, Bagging and Boosting

Siamak Ravanbakhsh

COMP 551 (winter 2020)

Learning objectives

bootstrap for uncertainty estimation

bagging for variance reduction

- random forests

boosting *reduce bias*

- AdaBoost
- gradient boosting
- relationship to L1 regularization

Bootstrap

a simple approach to estimate the uncertainty in prediction

non-parametric bootstrap

given the dataset $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^N$

subsample **with replacement** B datasets of size N

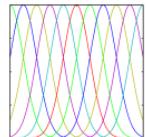
$$\mathcal{D}_b = \{(x^{(n,b)}, y^{(n,b)})\}_{n=1}^N, b = 1, \dots, B$$

Picking a sample and
put it back
(can have duplicates)

train a model on each of these bootstrap datasets (called *bootstrap samples*)

produce a measure of uncertainty from these models

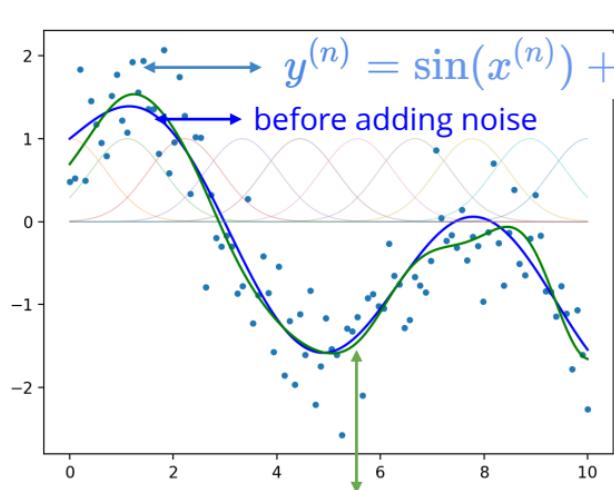
- for model parameters
- for predictions



$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

Bootstrap: example

Recall: linear model with nonlinear Gaussian bases (N=100)

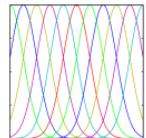


noise

ϵ



```
1 #x: N
2 #y: N
3 plt.plot(x, y, 'b.')
4 phi = lambda x,mu: np.exp(-(x-mu)**2)
5 mu = np.linspace(0,10,10) #10 Gaussians bases
6 Phi = phi(x[:,None], mu[None,:]) #N x 10
7 w = np.linalg.lstsq(Phi, y)[0]
8 yh = np.dot(Phi,w)
9 plt.plot(x, yh, 'g-')
```



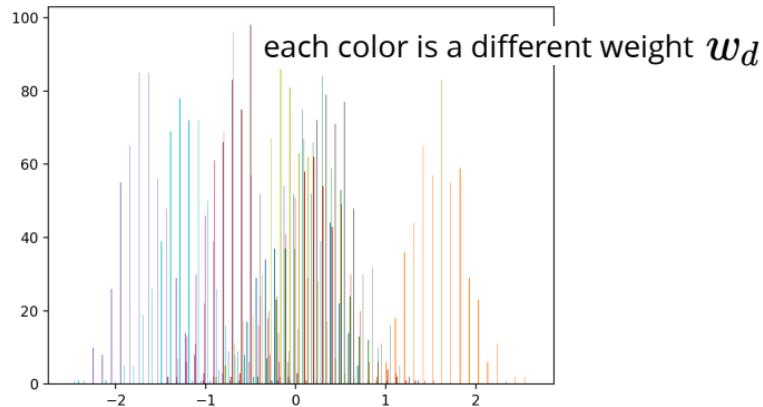
$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

Bootstrap: example

Recall: linear model with nonlinear Gaussian bases ($N=100$)

using $B=500$ bootstrap samples

gives a measure of uncertainty of the parameters



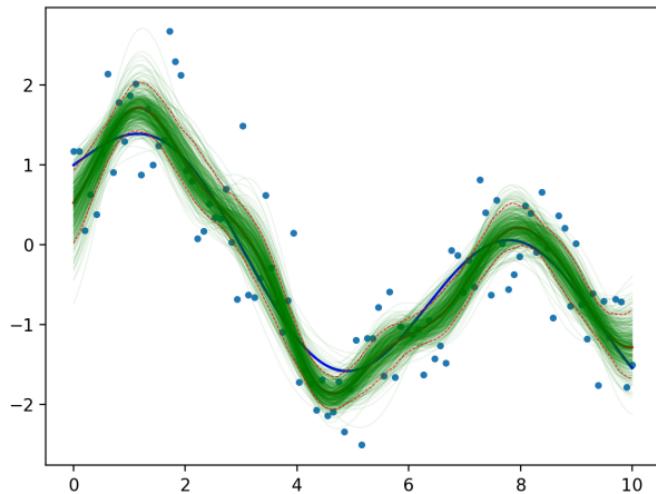
```
1 #Phi: N x D
2 #y: N
3 B = 500
4 ws = np.zeros((B,D))
5 for b in range(B):
6     inds = np.random.randint(N, size=(N))
7     Phi_b = Phi[inds,:] #N x D
8     y_b = y[inds] #N
9     #fit the subsampled data
10    ws[b,:] = np.linalg.lstsq(Phi_b, y_b[:,b])[0]
11
12 plt.hist(ws, bins=50)
```



$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

Bootstrap: example

Recall: linear model with nonlinear Gaussian bases (N=100)
using B=500 bootstrap samples
also gives a measure of **uncertainty of the predictions**



*different from cross validation:
bootstrap samples might overlap
⇒ They're not independent!*

the red lines are 5% and 95% quantiles
(for each point we can get these across bootstrap model predictions)



```
1 #Phi: N x D
2 #Phi_test: Nt x D
3 #y: N
4 #ws: B x D from previous code
5 y_hats = np.zeros((B, Nt))
6 for b in range(B):
7     wb = ws[b,:]
8     y_hats[b,:] = np.dot(Phi_test, wb)
9
10 # get 95% quantiles
11 y_5 = np.quantile(y_hats, .05, axis=0)
12 y_95 = np.quantile(y_hats, .95, axis=0)
```



Bagging

use bootstrap for **more accurate prediction** (not just uncertainty)

variance of sum of random variables

$$\begin{aligned}\text{Var}(z_1 + z_2) &= \mathbb{E}[(z_1 + z_2)^2] - \mathbb{E}[z_1 + z_2]^2 \\ &= \mathbb{E}[z_1^2 + z_2^2 + 2z_1z_2] - (\mathbb{E}[z_1] + \mathbb{E}[z_2])^2 \\ &= \mathbb{E}[z_1^2] + \mathbb{E}[z_2^2] + \mathbb{E}[2z_1z_2] - \mathbb{E}[z_1]^2 - \mathbb{E}[z_2]^2 - 2\mathbb{E}[z_1]\mathbb{E}[z_2] \\ &= \text{Var}(z_1) + \text{Var}(z_2) + 2\text{Cov}(z_1, z_2)\end{aligned}$$

for uncorrelated variables this term is zero

Bagging

use bootstrap for **more accurate prediction** (not just uncertainty)

average of uncorrelated random variables has a lower variance

z_1, \dots, z_B are uncorrelated random variables with mean μ and variance σ^2

the average $\bar{z} = \frac{1}{B} \sum_b z_b$ has mean μ and variance

$$\text{Var}\left(\frac{1}{B} \sum_b z_b\right) = \frac{1}{B^2} \text{Var}\left(\sum_b z_b\right) = \frac{1}{B^2} B \sigma^2 = \frac{1}{B} \sigma^2$$

use this to reduce the variance of our models (bias remains the same)

regression: average the model predictions $\hat{f}(x) = \frac{1}{B} \sum_b \hat{f}_b(x)$

issue: model predictions are not uncorrelated (trained using the same data)

bagging (bootstrap aggregation) use **bootstrap samples** to reduce correlation

*although NOT completely
uncorrelated.*

*it's not gonna work if
the model is linear.*

Bagging for classification

averaging makes sense for regression, how about classification?

wisdom of crowds

$z_1, \dots, z_B \in \{0, 1\}$ are IID Bernoulli random variables with mean $\mu = .5 + \epsilon^{> 0}$

for $\bar{z} = \frac{1}{B} \sum_b z_b$ we have $p(\bar{z} > .5)$ goes to 1 as B grows

mode of iid classifiers that are better than chance is a better classifier

- use voting

crowds are wiser when

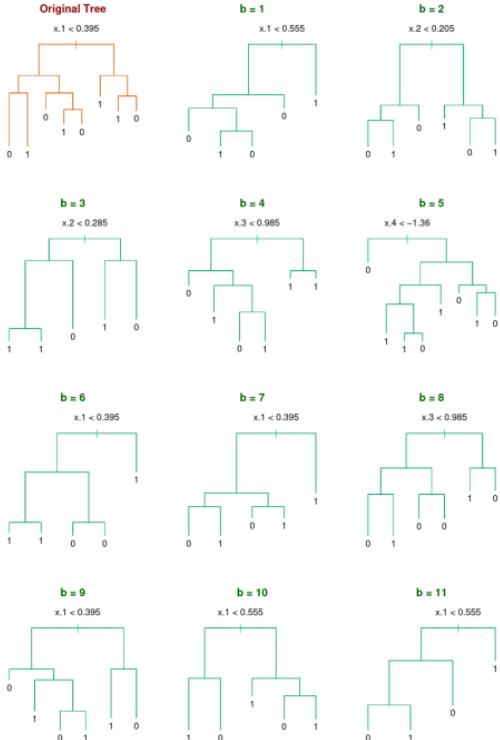
- individuals are better than random
- votes are uncorrelated

bagging (bootstrap aggregation)

use **bootstrap samples** to reduce correlation

example

Bagging decision trees

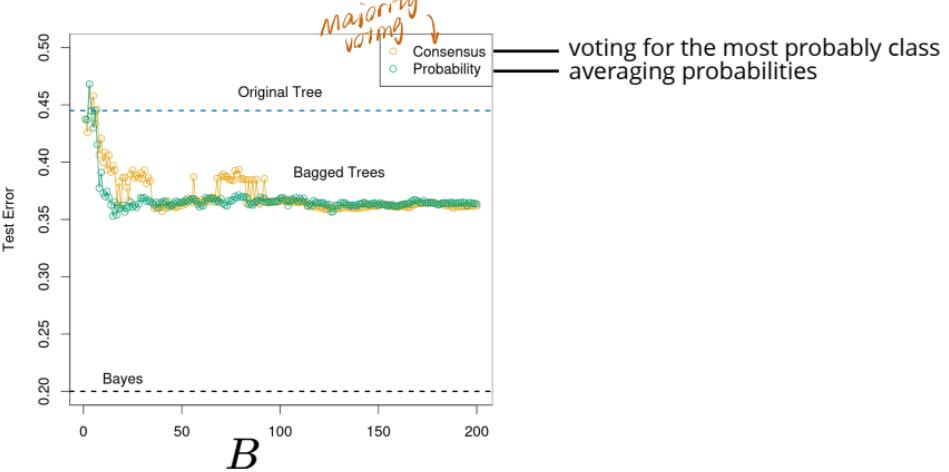


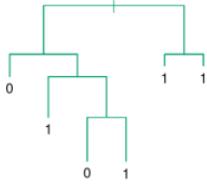
setup

- synthetic dataset
- 5 correlated features
- 1st feature is a noisy predictor of the label

Bootstrap samples create different decision trees (due to high variance)

compared to decision trees, no longer **interpretable!**





Random forests

further reduce the correlation between decision trees

feature sub-sampling

only a random subset of features are available for split at each step

further reduce the dependence between decision trees

magic number?

$$\sqrt{D}$$

this is a hyper-parameter, can be optimized using CV

Out Of Bag (OOB) samples:

- the instances not included in a bootstrap dataset can be used for validation
- simultaneous validation of decision trees in a forest
- no need to set aside data for **cross validation**

Example: spam detection

Dataset

N=4601 emails

binary classification task: *spam - not spam*

D=57 features:

- **48** words: percentage of words in the email that match these words
 - e.g., business,address,internet, free, George (customized per user)
- **6** characters: again percentage of characters that match these
 - ch; , ch(,ch[,ch! ,ch\$, ch#
- average, max, sum of length of uninterrupted sequences of capital letters:
 - CAPAVE, CAPMAX, CAPTOT

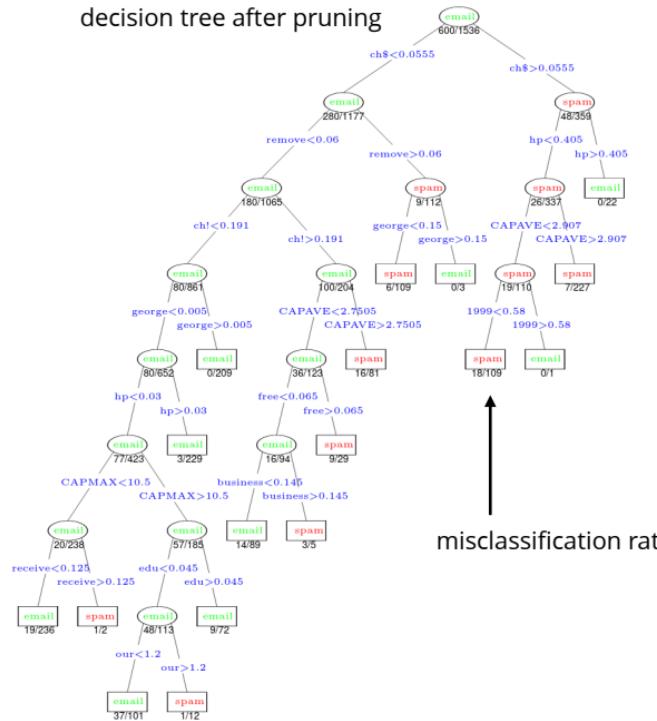
an example of
feature engineering

average value of these features in the spam and non-spam emails

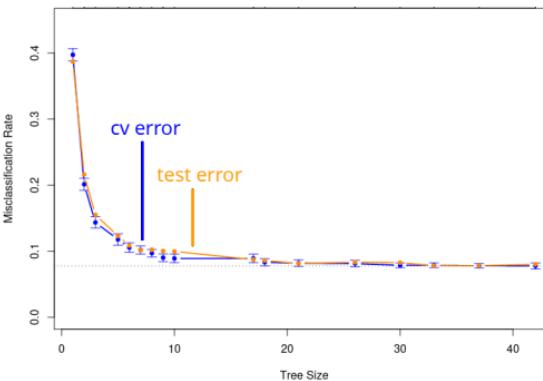
| | george | you | your | hp | free | hpl | ! | our | re | edu | remove |
|-------|--------|------|------|------|------|------|------|------|------|------|--------|
| spam | 0.00 | 2.26 | 1.38 | 0.02 | 0.52 | 0.01 | 0.51 | 0.51 | 0.13 | 0.01 | 0.28 |
| email | 1.27 | 1.27 | 0.44 | 0.90 | 0.07 | 0.43 | 0.11 | 0.18 | 0.42 | 0.29 | 0.01 |

Example: spam detection

decision tree after pruning



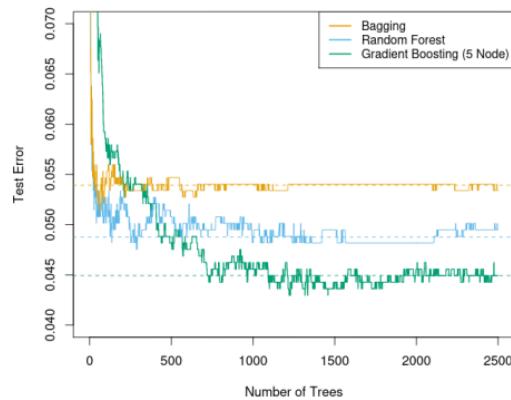
number of leaves (17) in optimal pruning
decided based on cross-validation error



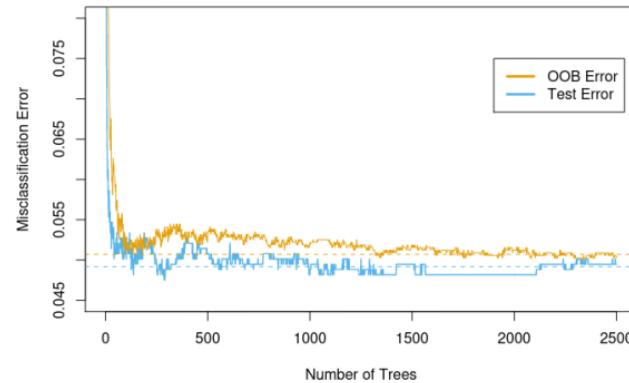
misclassification rate on test data

Example: spam detection

Bagging and Random Forests do much better than a single decision tree!



Out Of Bag (OOB) error can be used for parameter tuning
(e.g., size of the forest)



Summary so far...

- Bootstrap is a powerful technique to get uncertainty estimates
- Bootstrep aggregation (Bagging) can reduce the variance of unstable models
- Random forests:
 - Bagging + further de-corelation of features at each split
 - OOB validation instead of CV
 - destroy interpretability of decision trees
 - perform well in practice
 - can fail if only few relevant features exist (due to feature-sampling)

Adaptive bases

several methods can be classified as *learning these bases adaptively*

- decision trees
- generalized additive models
- **boosting**
- neural networks

$$f(x) = \sum_d w_d \phi_d(x; v_d)$$



in boosting each basis is a classifier or regression function (**weak learner, or base learner**)
create a **strong learner** by sequentially combining **weak learners**

Forward stagewise additive modelling

model $f(x) = \sum_{t=1}^T w^{\{t\}} \phi(x; v^{\{t\}})$ a simple model, such as decision stump (decision tree with one node)

cost $J(\{w^{\{t\}}, v^{\{t\}}\}_t) = \sum_{n=1}^N L(y^{(n)}, f(x^{(n)}))$
so far we have seen L2 loss, log loss and hinge loss

optimizing this cost is difficult given the form of f

optimization idea add one weak-learner in each stage t , to reduce the error of previous stage

1. find the best weak learner

$$v^{\{t\}}, w^{\{t\}} = \arg \min_{v, w} \sum_{n=1}^N L(y^{(n)}, f^{\{t-1\}}(x^{(n)}) + w \phi(x^{(n)}; v))$$

2. add it to the current model

$$f^{\{t\}}(x) = f^{\{t-1\}}(x^{(n)}) + w^{\{t\}} \phi(x^{(n)}; v^{\{t\}})$$

L_2 loss & forward stagewise linear model

model consider **weak learners** that are individual features $\phi^{\{t\}}(x) = w^{\{t\}} x_{d^{\{t\}}}$ *Pick one feature at each step.*

cost using L2 loss for regression *Linear regression.*

$$\text{at stage } t \quad \arg \min_{d, w_d} \frac{1}{2} \sum_{n=1}^N \left(y^{(n)} - (\mathbf{f}^{\{t-1\}}(x^{(n)}) + w_d x_d^{(n)}) \right)^2$$

optimization **recall:** optimal weight for each d is $w_d = \frac{\sum_n x_d^{(n)} r_d^{(n)}}{\sum_n x_d^{(n)2}}$

pick the feature that most significantly reduces the residual

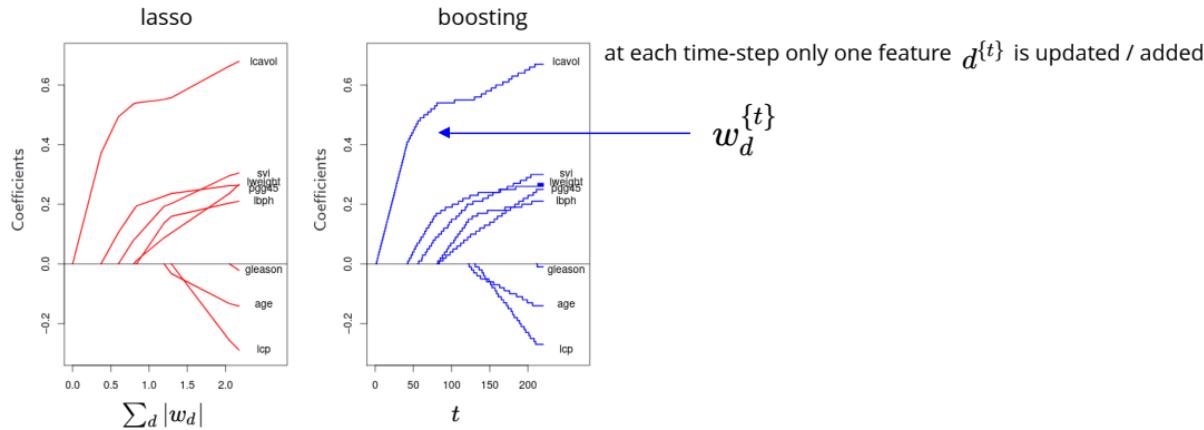
the model at time-step t : $f^{\{t\}}(x) = \sum_t \alpha w_{d^{\{t\}}}^{(t)} x_{d^{\{t\}}}$
using a small α helps with test error

is this related to L1-regularized linear regression?

example

L_2 loss & forward stagewise linear model

using small learning rate $\alpha = .01$ L2 Boosting has a similar regularization path to lasso



we can view boosting as doing feature (base learner) selection in exponentially large spaces (e.g., all trees of size K)
the number of steps t plays a similar role to (the inverse of) regularization hyper-parameter

Exponential loss & AdaBoost

loss functions for **binary classification** $y \in \{-1, +1\}$

predicted label is $\hat{y} = \text{sign}(f(x))$

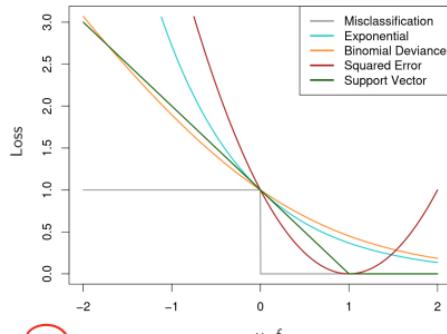
misclassification loss $L(y, f(x)) = \mathbb{I}(yf(x) > 0)$
(0-1 loss)

log-loss $L(y, f(x)) = \log(1 + e^{-yf(x)})$
(aka cross entropy loss or binomial deviance)

Hinge loss $L(y, f(x)) = \max(0, 1 - yf(x))$
support vector loss

yet another loss function is **exponential loss** $L(y, f(x)) = e^{-yf(x)}$

note that the loss grows faster than the other surrogate losses (more sensitive to outliers)



useful property when working with additive models:

$$L(y, f^{\{t-1\}}(x) + w^{\{t\}} \phi(x, v^{\{t\}})) = L(y, f^{\{t-1\}}(x)) \cdot L(y, w^{\{t\}} \phi(x, v^{\{t\}}))$$

treat this as a weight **q** for an instance

instances that are not properly classified before receive a higher weight

Exponential loss & AdaBoost

cost using exponential loss

$$J(\{w^{(t)}, v^{(t)}\}_t) = \sum_{n=1}^N L(y^{(n)}, f^{(t-1)}(x^{(n)}) + w^{(t)} \phi(x^{(n)}, v^{(t)})) = \sum_n q^{(n)} L(y^{(n)}, w^{(t)} \phi(x^{(n)}, v^{(t)}))$$

loss for this instance at previous stage $L(y^{(n)}, f^{(t-1)}(x^{(n)}))$

discrete AdaBoost: assume this is a simple classifier, so its output is +/- 1

optimization objective is to find the weak learner minimizing the cost above

$$\begin{aligned} J(\{w^{(t)}, v^{(t)}\}_t) &= \sum_n q^{(n)} e^{-y^{(n)} w^{(t)} \phi(x^{(n)}, v^{(t)})} \\ &= e^{-w^{(t)}} \sum_n q^{(n)} \mathbb{I}(y^{(n)} = \phi(x^{(n)}, v^{(t)})) + e^{w^{(t)}} \sum_n q^{(n)} \mathbb{I}(y^{(n)} \neq \phi(x^{(n)}, v^{(t)})) \\ &= e^{-w^{(t)}} \sum_n q^{(n)} + (e^{w^{(t)}} - e^{-w^{(t)}}) \sum_n q^{(n)} \mathbb{I}(y^{(n)} \neq \phi(x^{(n)}, v^{(t)})) \end{aligned}$$

does not depend on
the weak learner

assuming $w^{(t)} \geq 0$ the weak learner should minimize this cost
this is classification with weighted instances

Exponential loss & AdaBoost

cost

$$\begin{aligned} J(\{w^{(t)}, v^{(t)}\}_t) &= \sum_n q^{(n)} L(y^{(n)}, w^{(t)} \phi(x^{(n)}, v^{(t)})) \\ &= e^{-w^{(t)}} \sum_n q^{(n)} + (e^{w^{(t)}} - e^{-w^{(t)}}) \sum_n q^{(n)} \mathbb{I}(y^{(n)} \neq \phi(x^{(n)}, v^{(t)})) \end{aligned}$$

does not depend on
the weak learner

assuming $w^{(t)} \geq 0$ the weak learner should minimize this cost
this is classification with weighted instances
this gives $v^{(t)}$

still need to find the optimal $w^{(t)}$

setting $\frac{\partial J}{\partial w^{(t)}} = 0$ gives $w^{(t)} = \frac{1}{2} \log \frac{1-\ell^{(t)}}{\ell^{(t)}}$

weight-normalized misclassification error

$$\ell^{(t)} = \frac{\sum_n q^{(n)} \mathbb{I}(\phi(x^{(n)}; v^{(t)}) \neq y^{(n)})}{\sum_n q^{(n)}}$$

since weak learner is better than chance $\ell^{(t)} < .5$ and so $w^{(t)} \geq 0$

negative if signs disagree.

we can now update instance weights q for next iteration $q^{(n), \{t+1\}} = q^{(n), \{t\}} e^{-w^{(t)} y^{(n)} \phi(x^{(n)}; v^{(t)})}$

(multiply by the new loss)

since $w > 0$, the weight q of misclassified points increase and the rest decrease

Exponential loss & AdaBoost

overall algorithm for discrete AdaBoost

initialize $q^{(n)} := \frac{1}{N} \quad \forall n$

for $t=1:T$

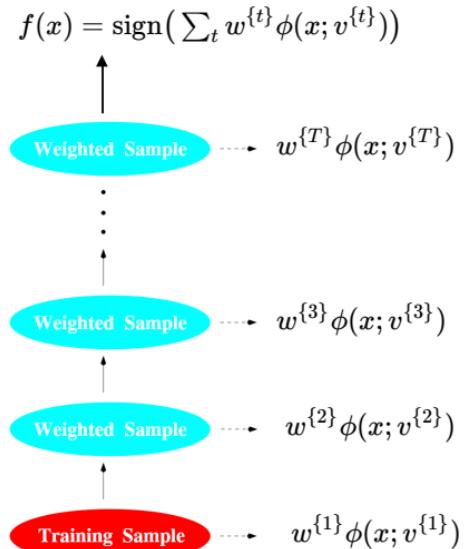
fit the simple classifier $\phi(x, v^{(t)})$ to the weighted dataset

$$\ell^{(t)} := \frac{\sum_n q^{(n)} \mathbb{I}(\phi(x^{(n)}; v^{(t)}) \neq y^{(n)})}{\sum_n q^{(n)}}$$

$$w^{(t)} := \frac{1}{2} \log \frac{1-\ell^{(t)}}{\ell^{(t)}}$$

$$q^{(n)} := q^{(n)} e^{-w^{(t)} y^{(n)} \phi(x^{(n)}; v^{(t)})} \quad \forall n$$

return $f(x) = \text{sign}(\sum_t w^{(t)} \phi(x; v^{(t)}))$

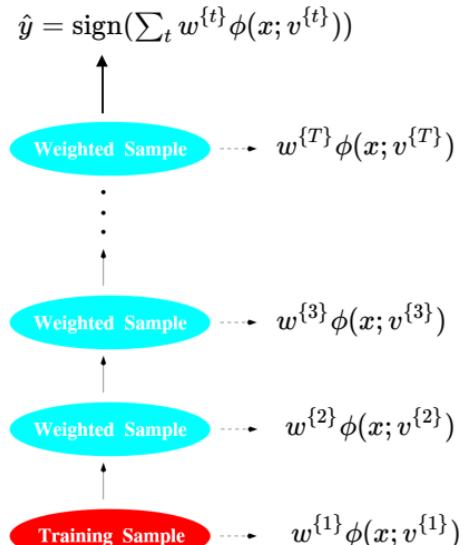
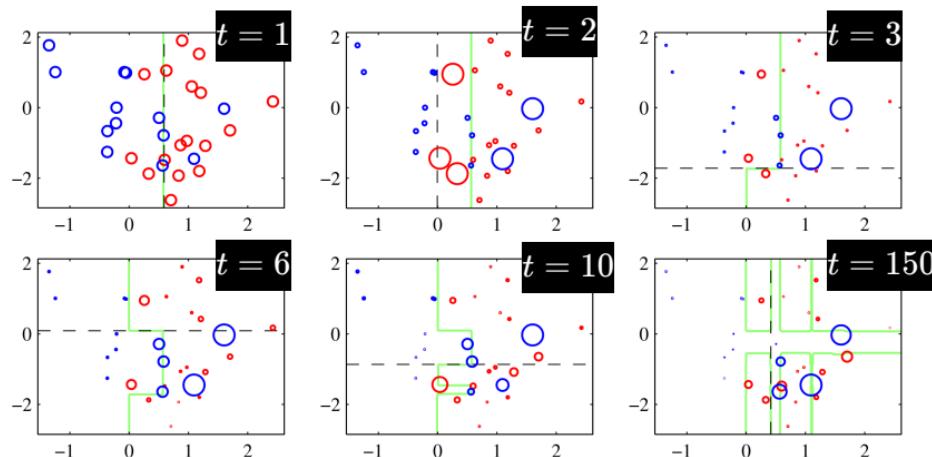


example AdaBoost

each weak learner is a decision stump (dashed line)

circle size is proportional to $q^{n,\{t\}}$

green is the decision boundary of $f^{\{t\}}$



example

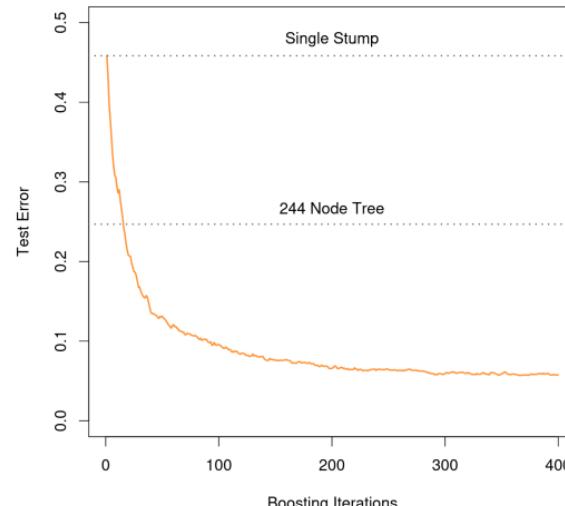
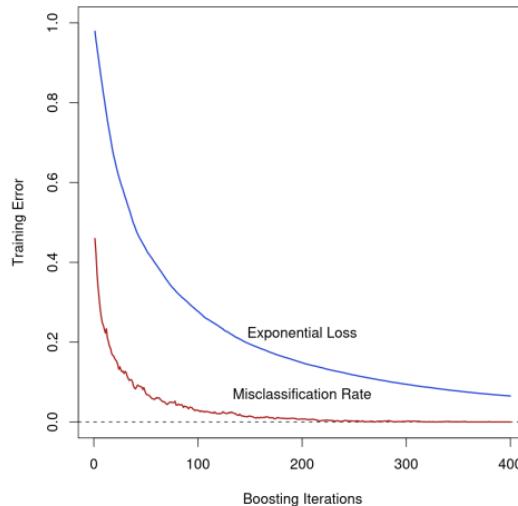
AdaBoost

features $x_1^{(n)}, \dots, x_{10}^{(n)}$ are samples from standard Gaussian

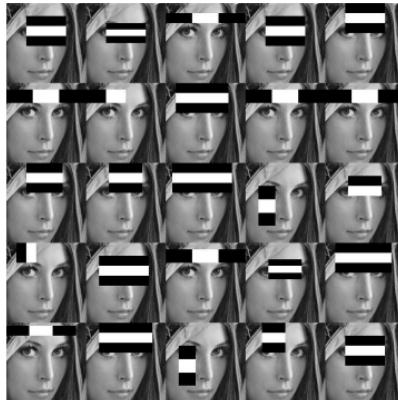
label $y^{(n)} = \mathbb{I}(\sum_d x_d^{(n)} > 9.34)$

N=2000 training examples

notice that test error does not increase
AdaBoost is very slow to overfit



application: Viola-Jones face detection



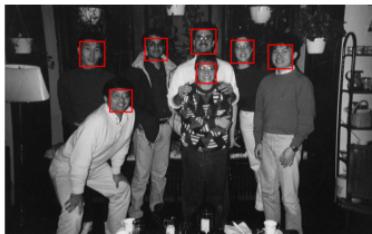
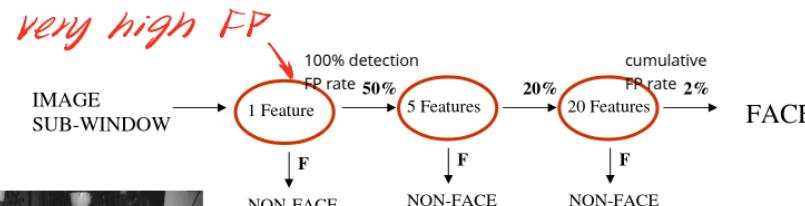
Haar features are computationally efficient

each feature is a weak learner

AdaBoost picks one feature at a time (label: face/no-face)

Still can be inefficient:

- use the fact that faces are rare (.01% of subwindows are faces)
- cascade of classifiers due to small rate



cascade is applied over all image subwindows
fast enough for real-time (object) detection

image source: David Lowe slides

Gradient boosting

idea fit the weak learner to the gradient of the cost

let $\mathbf{f}^{\{t\}} = [f^{\{t\}}(x^{(1)}), \dots, f^{\{t\}}(x^{(N)})]^\top$ and true labels $\mathbf{y} = [y^{(1)}, \dots, y^{(N)}]^\top$

ignoring the structure of \mathbf{f}

if we use gradient descent to minimize the loss $\hat{\mathbf{f}} = \arg \min_{\mathbf{f}} L(\mathbf{f}, \mathbf{y})$

write $\hat{\mathbf{f}}$ as a sum of steps

$$\hat{\mathbf{f}} = \mathbf{f}^{\{T\}} = \mathbf{f}^{\{0\}} - \sum_{t=1}^T w^{\{t\}} \mathbf{g}^{\{t\}}$$

$$w^{\{t\}} = \arg \min_w L(\mathbf{f}^{\{t-1\}} - w \mathbf{g}^{\{t\}}) \frac{\partial}{\partial \mathbf{f}} L(\mathbf{f}^{\{t-1\}}, \mathbf{y})$$

we can look for the optimal step size

gradient vector
its role is similar to residual

so far we treated \mathbf{f} as a parameter vector

fit the weak-learner to negative of the gradient $v^{\{t\}} = \arg \min_v \frac{1}{2} \|\phi_v - (-\mathbf{g})\|_2^2$

we are fitting the gradient using L2 loss regardless of the original loss function

$$\phi_v = [\phi(x^{(1)}; v), \dots, \phi(x^{(N)}; v)]^\top$$

Gradient tree boosting

apply gradient boosting to CART (classification and regression trees)

initialize $\mathbf{f}^{\{0\}}$ to predict a constant

for $t=1:T$ decide T using a validation set (early stopping)

calculate the negative of the gradient $\mathbf{r} = -\frac{\partial}{\partial \mathbf{f}} L(\mathbf{f}^{\{t-1\}}, \mathbf{y})$

fit a regression tree to $\mathbf{X}_{N \times D}, \mathbf{r}_N$ and produce regions $\mathbb{R}_1, \dots, \mathbb{R}_K$ shallow trees of $K = 4-8$ leaf usually work well as weak learners

re-adjust predictions per region $w_k = \arg \min_w \sum_{x^{(n)} \in \mathbb{R}_k} L(y^{(n)}, f^{\{t-1\}}(x^{(n)}) + w_k)$ this is effectively the line-search

update $f^{\{t\}}(x) = f^{\{t-1\}}(x) + \alpha \sum_{k=1}^K w_k \mathbb{I}(x \in \mathbb{R}_k)$

return $f^{\{T\}}(x)$ using a small learning rate here improves test error (**shrinkage**)

stochastic gradient boosting

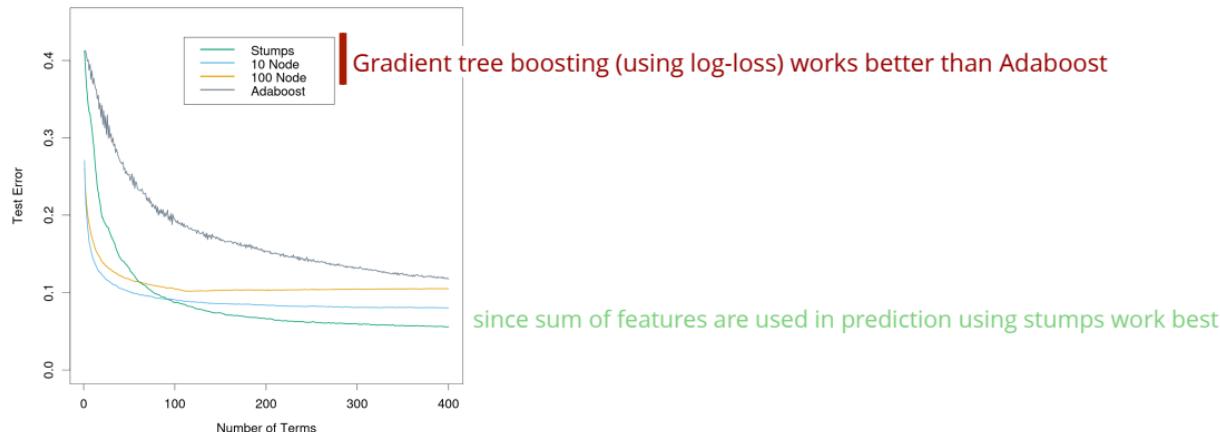
- combines bootstrap and boosting
- use a subsample at each iteration above
- similar to stochastic gradient descent

example

Gradient tree boosting

recall the synthetic example:

features $x_1^{(n)}, \dots, x_{10}^{(n)}$ are samples from standard Gaussian
label $y^{(n)} = \mathbb{I}(\sum_d x_d^{(n)} > 9.34)$
N=2000 training examples



example

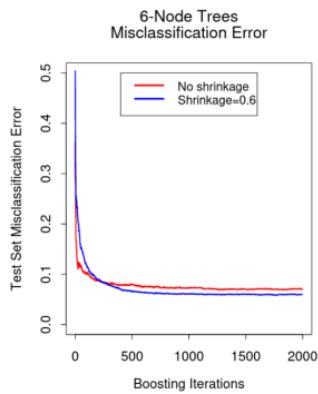
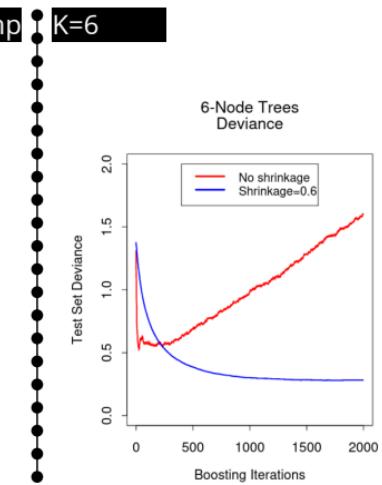
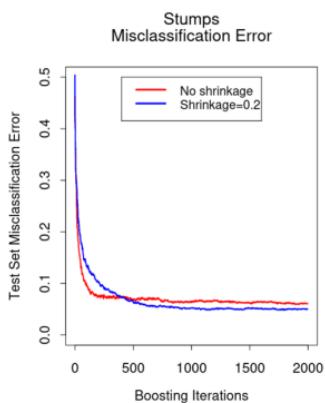
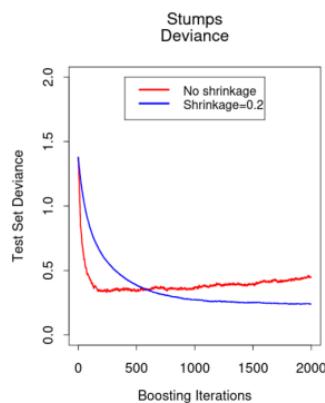
Gradient tree boosting

recall the synthetic example:

features $x_1^{(n)}, \dots, x_{10}^{(n)}$ are samples from standard Gaussian
label $y^{(n)} = \mathbb{I}(\sum_d x_d^{(n)} > 9.34)$
N=2000 training examples

deviance = cross entropy = log-loss

(K=2) stump K=6



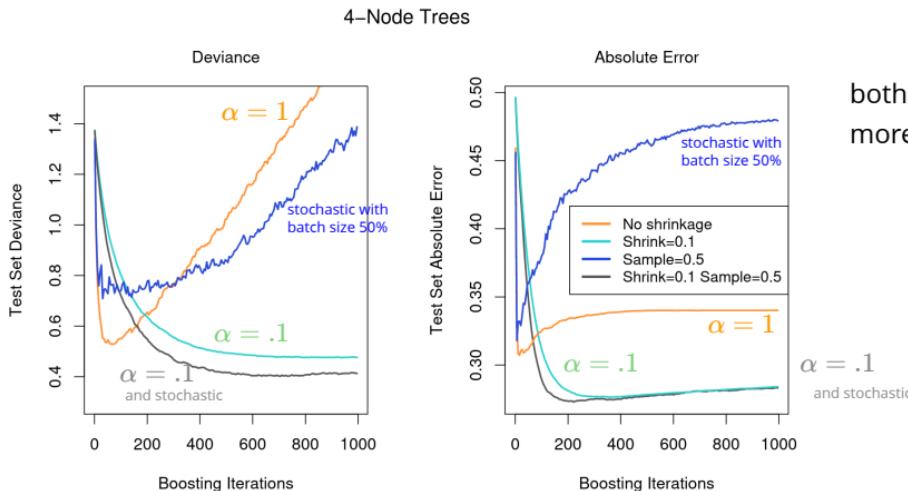
in both cases using shrinkage $\alpha = .2$ helps
while test loss may increase, test misclassification error does not

example

Gradient tree boosting

recall the synthetic example:

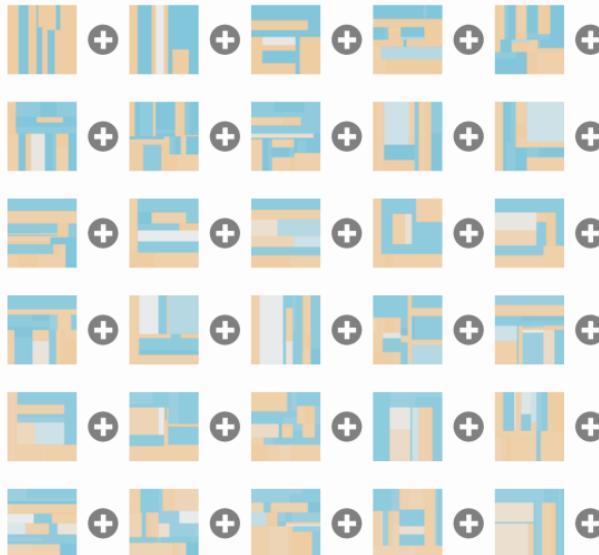
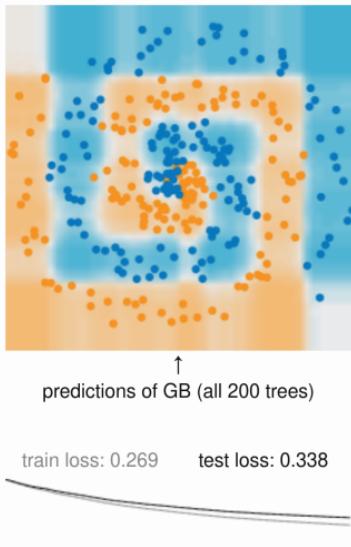
features $x_1^{(n)}, \dots, x_{10}^{(n)}$ are samples from standard Gaussian
label $y^{(n)} = \mathbb{I}(\sum_d x_d^{(n)} > 9.34)$
N=2000 training examples



both shrinkage and **subsampling** can help more hyper-parameters to tune

example

Gradient tree boosting



see the interactive demo: https://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html

Summary

two **ensemble methods**

- bagging & random forests (reduce variance)
 - produce models with minimal correlation
 - use their average prediction
- boosting (reduces the bias of the weak learner)
 - models are added in steps
 - a single cost function is minimized
 - for exponential loss: interpret as re-weighting the instance (AdaBoost)
 - gradient boosting: fit the weak learner to the negative of the gradient
 - interpretation as L1 regularization for "weak learner"-selection
 - also related to max-margin classification (for large number of steps T)
- random forests and (gradient) boosting generally perform very well

Gradient boosting

Gradient for some loss functions $\hat{\mathbf{f}} = \mathbf{f}^{\{T\}} = \mathbf{f}^{\{0\}} - \sum_{t=1}^T w^{\{t\}} \frac{\partial}{\partial \mathbf{f}} L(\mathbf{f}^{\{t-1\}}, \mathbf{y})$

| setting | loss function | $-\frac{\partial}{\partial \mathbf{f}} L(\mathbf{f}^{\{t-1\}}, \mathbf{y})$ |
|------------------------------|---|---|
| regression | $\frac{1}{2} \ \mathbf{y} - \mathbf{f}\ _2^2$ | $\mathbf{y} - \mathbf{f}$ |
| regression | $\ \mathbf{y} - \mathbf{f}\ _1$ | $\text{sign}(\mathbf{y} - \mathbf{f})$ |
| binary classification | $\exp(-\mathbf{y}\mathbf{f})$ exponential loss | $-\mathbf{y} \exp(-\mathbf{y}\mathbf{f})$ |
| multiclass classification | multi-class cross-entropy <small>one-hot coding for C-class classification</small> | $\mathbf{Y} - \mathbf{P}$ <small>$N \times C$ $N \times C$</small> predicted class probabilities $\mathbf{P}_{c,:} = \text{softmax}(f_{[c]})$ |