Applied Machine Learning

Bootstrap, Bagging and Boosting

Reihaneh Rabbany

McGill School of Computer Science

COMP 551 (winter 2020)

Learning objectives

bootstrap for uncertainty estimation

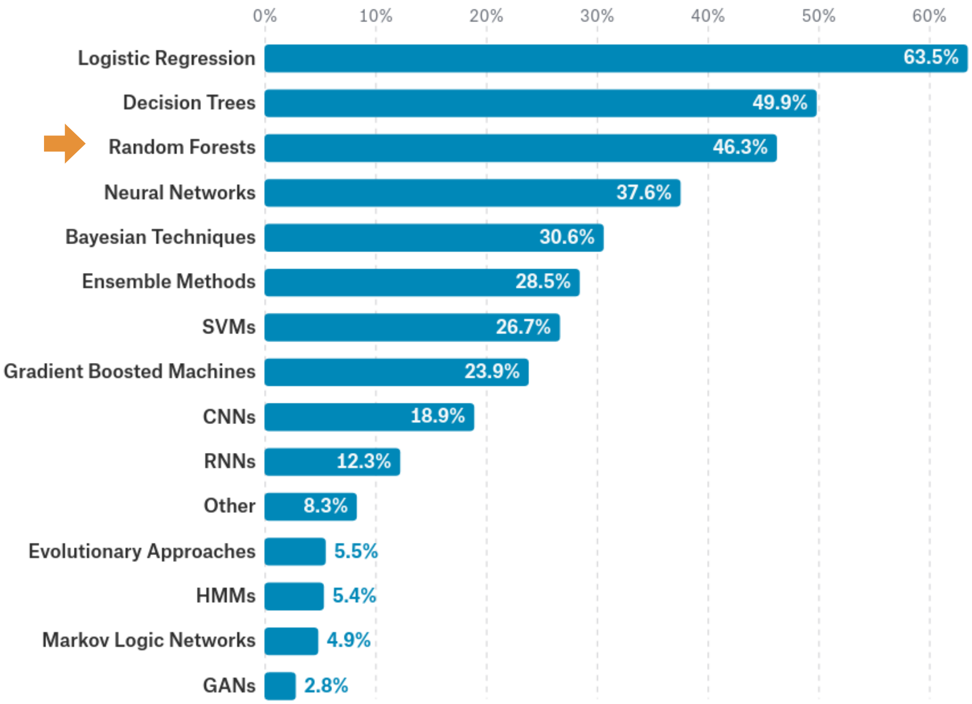
bagging for variance reduction

- random forests

boosting

- AdaBoost
- gradient boosting
- relationship to L1 regularization

Commonly used in practice



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

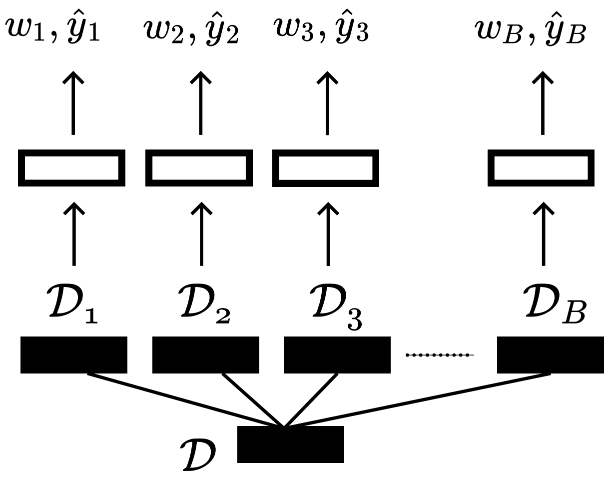
$$\mathcal{D}_b = \{(x^{(n,b)}, y^{(n,b)})\}_{n=1}^{N}, b = 1, \ldots, B$$

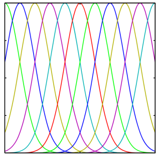
train a model on each of these bootstrap datasets
(called *bootstrap samples*)

produce a measure of uncertainty from these models

- for model parameters
- for predictions

sample the same size as
the original training set





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

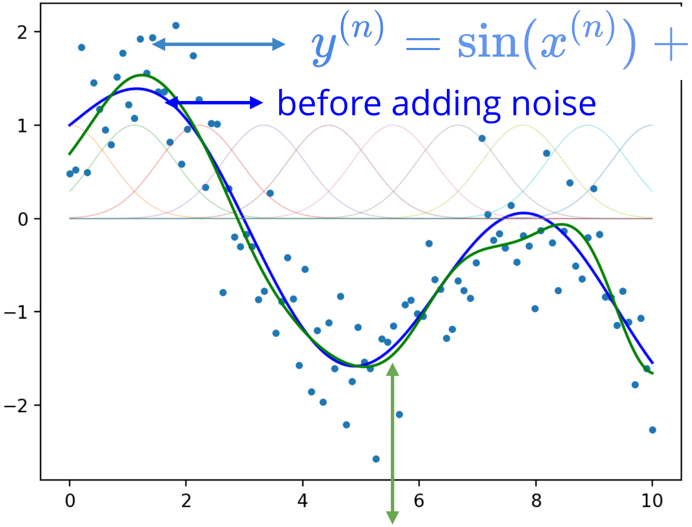
Bootstrap: example

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

noise

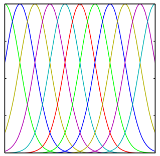
$$y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \epsilon$$

before adding noise



our fit to data using 10 Gaussian bases

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



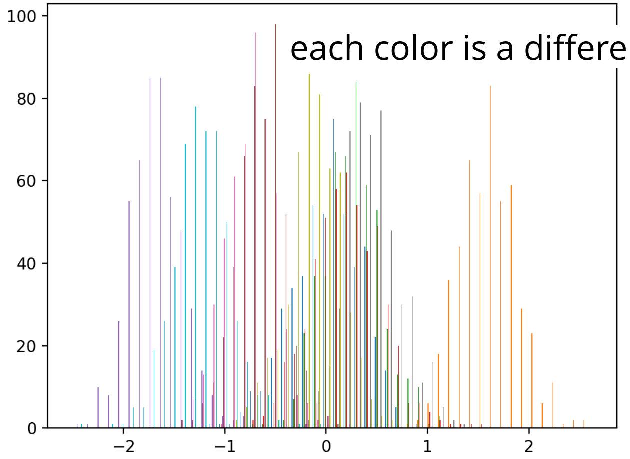
$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

Bootstrap: example

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

using B=500 bootstrap samples

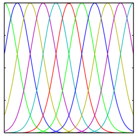
gives a measure of uncertainty of the parameters



each color is a different weight w_d

```
#Phi: N x D
#Phi_test: Nt x D
#y: N
B = 500
ws = np.zeros((B,D))
for b in range(B):
  inds = np.random.randint(N, size=(N))
  Phi_b = Phi[inds,:] #N x D
  y_b = y[inds] #N
  #fit the subsampled data
  ws[b,:] = np.linalg.lstsq(Phi_b, y_b[:,b])[0]

plt.hist(ws, bins=50)
```



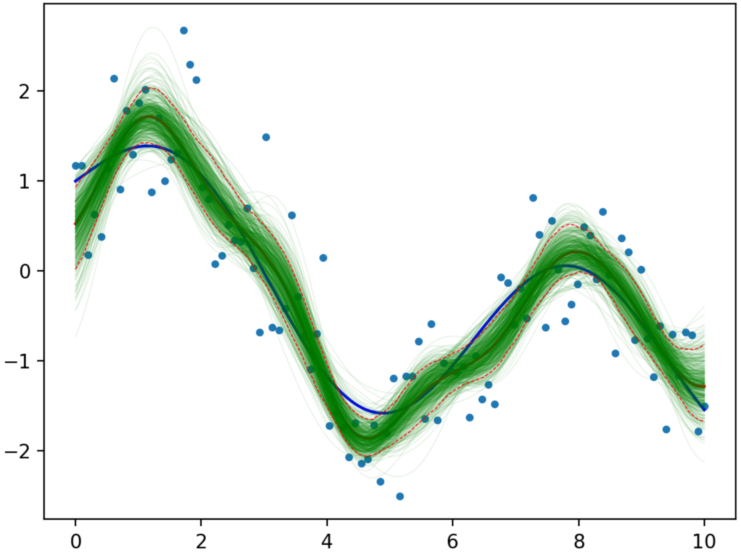
$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

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**



the red lines are 5% and 95% quantiles

(for each point we can get these across bootstrap model predictions)

```
#Phi: N x D
#Phi_test: Nt x D
#y: N
#ws: B x D from previous code
y_hats = np.zeros((B, Nt))
for b in range(B):
  wb = ws[b,:]
  y_hats[b,:] = np.dot(Phi_test, wb)

# get 95% quantiles
y_5 = np.quantile(y_hats, .05, axis=0)
y_95 = np.quantile(y_hats, .95, axis=0)
```

Bagging Bootstrap aggregating

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

variance of sum of random variables

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

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

$z_1, \ldots, z_B$ 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

$$\mathrm{Var}(\tfrac{1}{B} \sum_b z_b) = \tfrac{1}{B^2} \mathrm{Var}(\sum_b z_b) = \tfrac{1}{B^2} B\sigma^2 = \tfrac{1}{B}\sigma^2$$

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

Bagging for classification

averaging makes sense for regression, how about classification?

> *" the collective knowledge of a diverse and independent body of people typically exceeds the knowledge of any single individual, and can be harnessed by voting*

wisdom of crowds

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

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

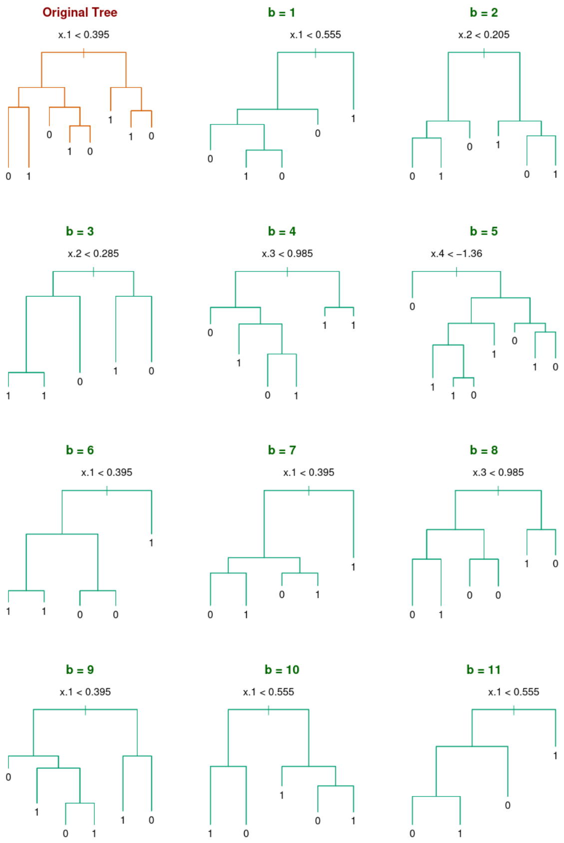
- use voting

crowds are wiser when

- individuals are better than random
- votes are uncorrelated

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

Bagging decision trees

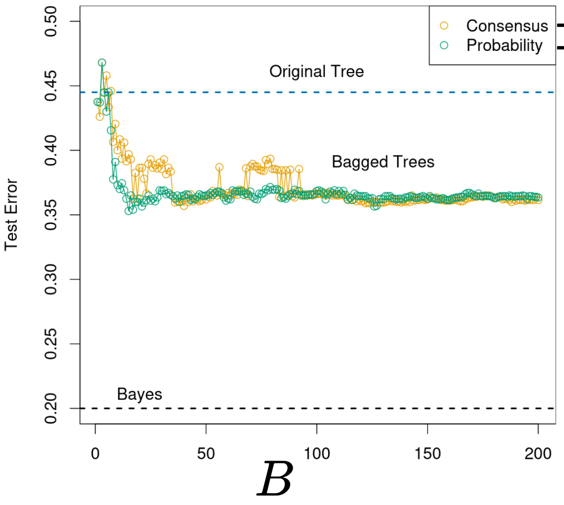


setup

- *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**!



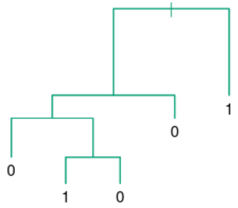
voting for the most probably class

averaging probabilities

Probability: average the the class proportions in the terminal node

Random forests

de-correlated tree



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

function greedy-test ($\mathbb{R}_{\text{node}}$, $\mathcal{D}$)

```
best-cost = -inf
inds = np.random.randint(D, size=(m))
```

for $d \in inds, s_{d,n} \in \mathbb{S}_d$

$\mathbb{R}_{\text{left}} = \mathbb{R}_k \cup \{x_d < s_{d,n}\}$

$\mathbb{R}_{\text{right}} = \mathbb{R}_k \cup \{x_d \geq s_{d,n}\}$

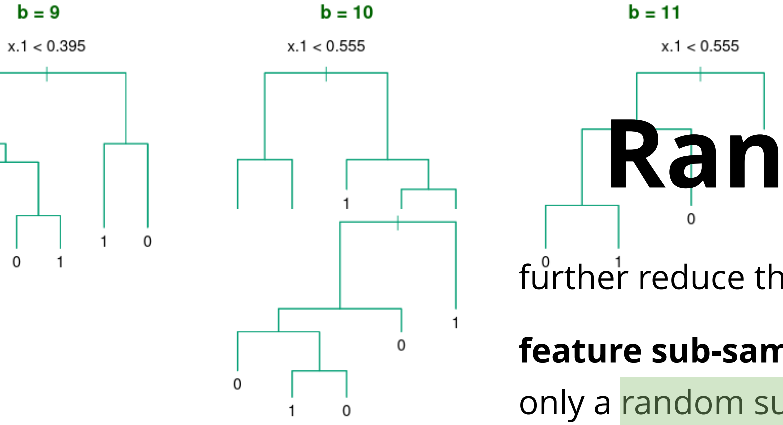
split-cost $= \frac{N_{\text{left}}}{N_{\text{node}}} \text{cost}(\mathbb{R}_{\text{left}}, \mathcal{D}) + \frac{N_{\text{right}}}{N_{\text{node}}} \text{cost}(\mathbb{R}_{\text{right}}, \mathcal{D})$

```
if split-cost < best-cost:
    best-cost = split-cost
```

$\mathbb{R}^*_{\text{left}} = \mathbb{R}_{\text{left}}$

$\mathbb{R}^*_{\text{right}} = \mathbb{R}_{\text{right}}$

return $\mathbb{R}^*_{\text{left}}, \mathbb{R}^*_{\text{right}}$



Random forests

de-correlated tree

further reduce the correlation between decision trees

feature sub-sampling

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

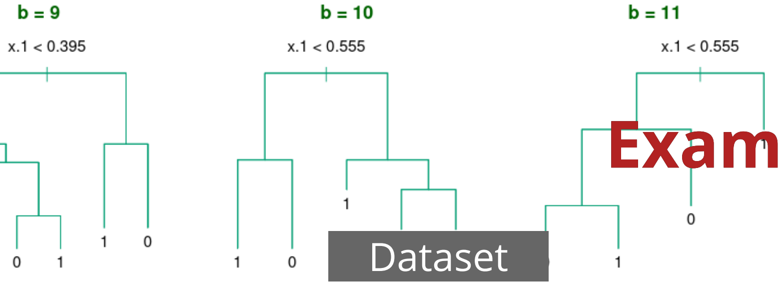
further reduce the dependence between decision trees

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 bootsrap 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

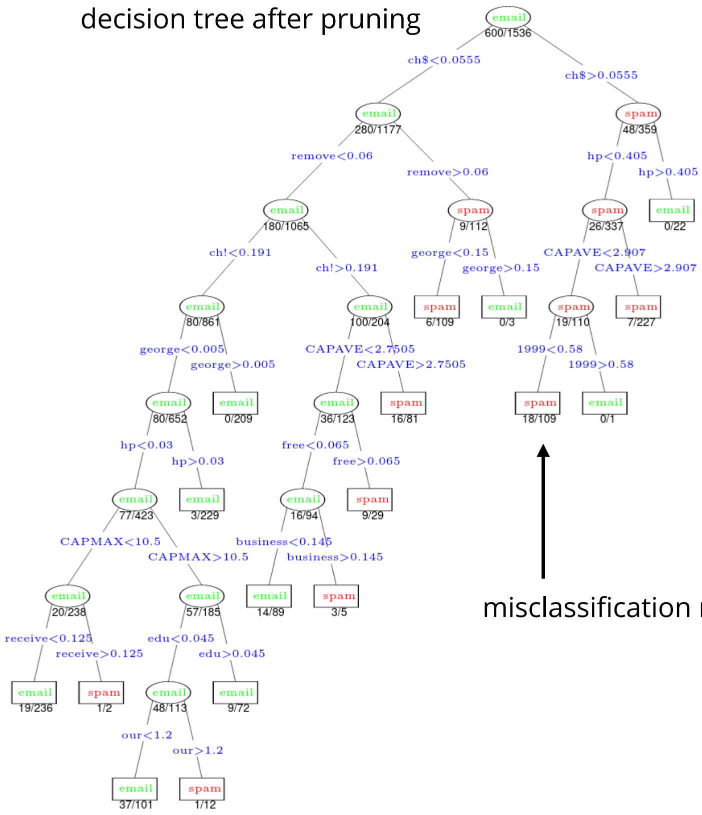
an example of
feature engineering

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

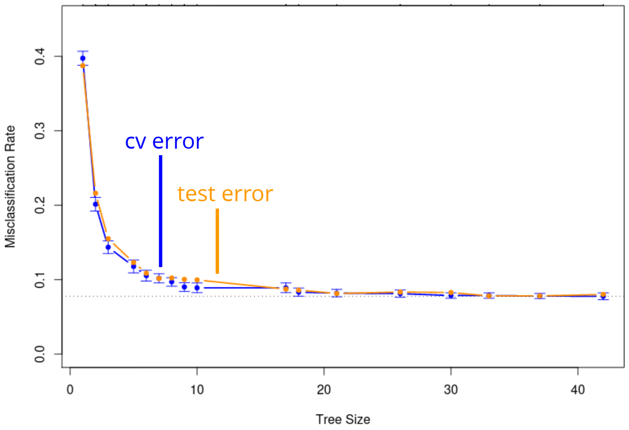
	george	you	your	hp	free	hpl	!	our	re	edu	remove
spam	0.00	2.26	1.38	0.02	0.52	0.01	0.51	0.51	0.13	0.01	0.28
email	1.27	1.27	0.44	0.90	0.07	0.43	0.11	0.18	0.42	0.29	0.01

Example: spam detection

decision tree after pruning



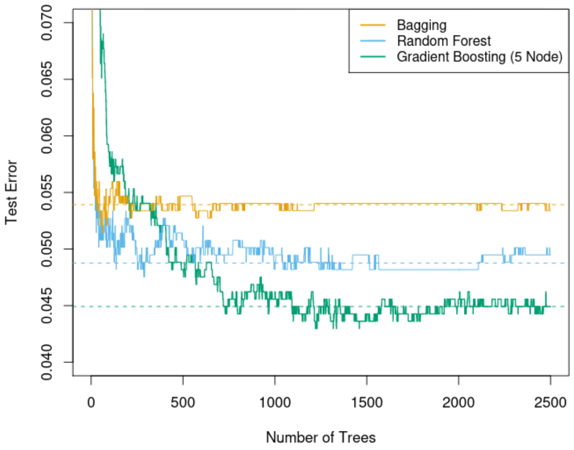
number of leaves (17) in optimal pruning decided based on cross-validation error



misclassification rate on test data

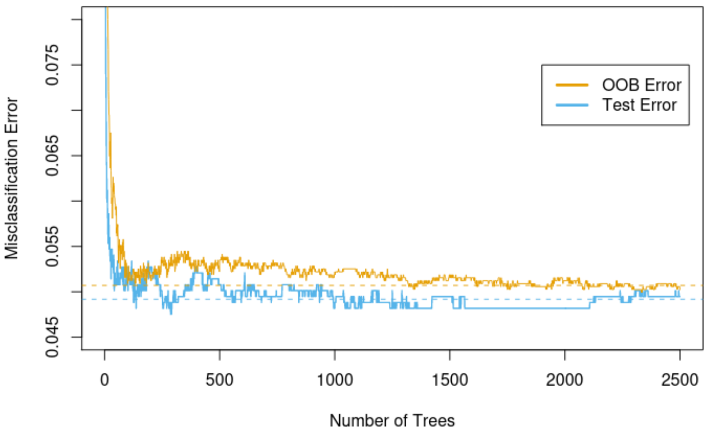
Example: spam detection

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...

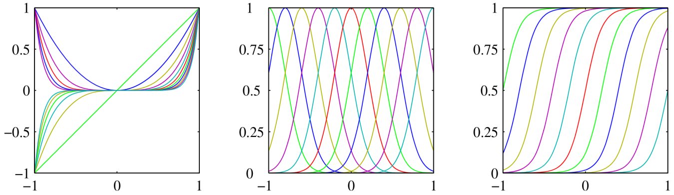
- Bootstrap is a powerful technique to get uncertainty estimates
- Bootstrep aggregation (Bagging) can reduce the variance of unstable models
- Random forests:
  - Bagging + further de-corelation of features at each split
  - OOB validation instead of CV
  - destroy interpretability of decision trees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

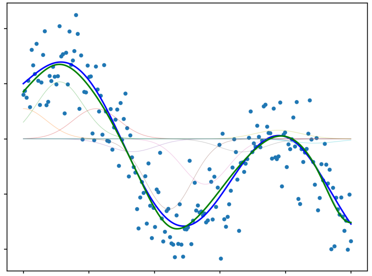
Adaptive bases

fixed set of bases in $f(x) = \sum_d w_d \phi_d(x)$

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

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





Gaussian bases example

Adaptive bases

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

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

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

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

model consider **weak learners** that are individual features $\phi^{\{t\}}(x) = w^{\{t\}} x_{d^{\{t\}}}$

cost using L2 loss for regression $\frac{1}{2}(y - f(x))^2$

at stage t $\arg\min_{d, w_d} \frac{1}{2} \sum_{n=1}^{N} \Big(\underbrace{y^{(n)} - (f^{\{t-1\}}(x^{(n)})}_{\text{residual } r^{(n)}} + w_d x_d^{(n)}) \Big)^2$

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

similar to 1D linear regression

$J(w) = \frac{1}{2} \sum_n (y^{(n)} - w x^{(n)})^2$

$\frac{\mathrm{d}J}{\mathrm{d}w} = \sum_n x^{(n)} (w x^{(n)} - y^{(n)})$

$w^* = \frac{\sum_n x^{(n)} y^{(n)}}{\sum_n x^{(n)} x^{(n)}}$

pick the feature that most significantly reduces the residual

the model at time-step t: $f^{\{t\}}(x) = \sum_t \alpha w_{d^{\{t\}}}^{\{t\}} x_{d^{\{t\}}}$

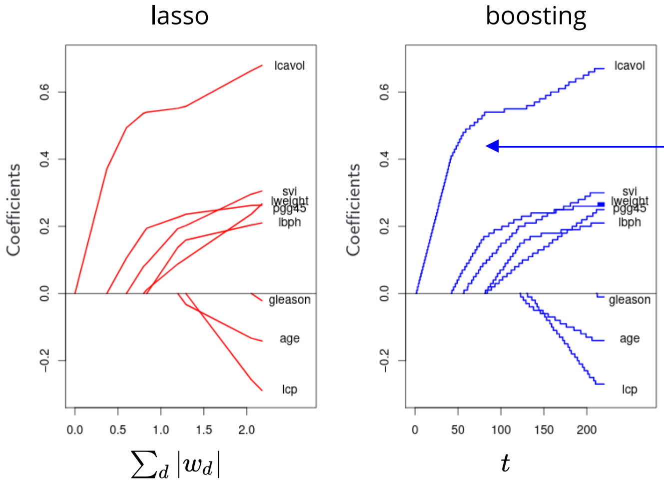
using a small α helps with test error

is this related to L1-regularized linear regression?

example

L_2 loss & linear modelling

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



at each time-step only one feature $d^{\{t\}}$ is updated / added

$w_d^{\{t\}}$

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

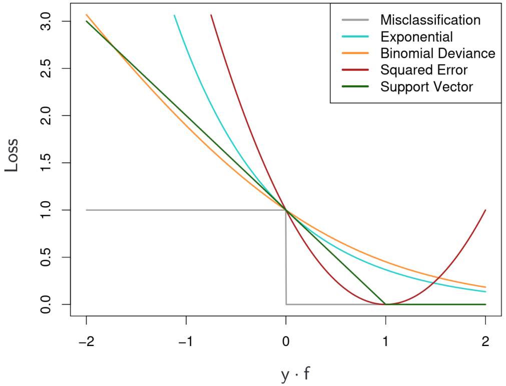
loss functions for **binary classification** $y \in \{-1, +1\}$

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

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

log-loss $L(y, f(x)) = \log\left(1 + e^{-yf(x)}\right)$
(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

$$J(\{w^{\{t\}}, v^{\{t\}}\}_t) = \sum_n q^{(n)} e^{-y^{(n)} w^{\{t\}} \phi(x^{(n)}, v^{\{t\}})}$$

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

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

does not depend on the weak learner

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

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

$$J(\{w^{\{t\}}, v^{\{t\}}\}_t) = \sum_n q^{(n)} e^{-y^{(n)} w^{\{t\}} \phi(x^{(n)}, v^{\{t\}})}$$

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

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

does not depend on the weak learner

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

Exponential loss & AdaBoost

cost $J(\{w^{\{t\}}, v^{\{t\}}\}_t) = \sum_n q^{(n)} L(y^{(n)}, w^{\{t\}} \phi(x^{(n)}, v^{\{t\}}))$

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

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 $w^{\{t\}} \geq 0$

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

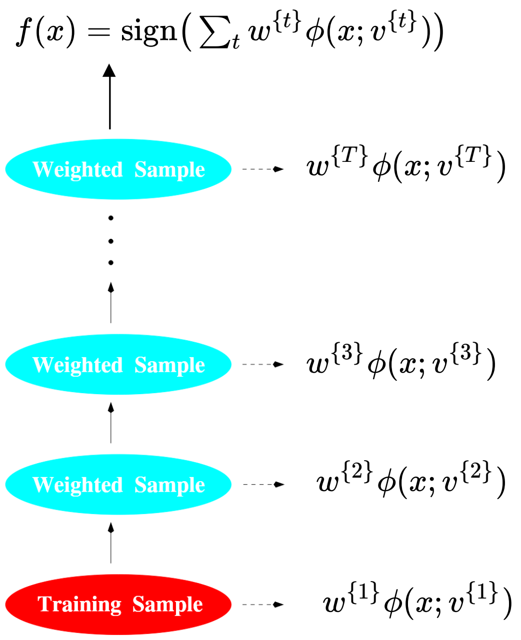
  fit the simple classifier $\phi(x, v^{\{t\}})$ to the weighted dataset

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

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

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

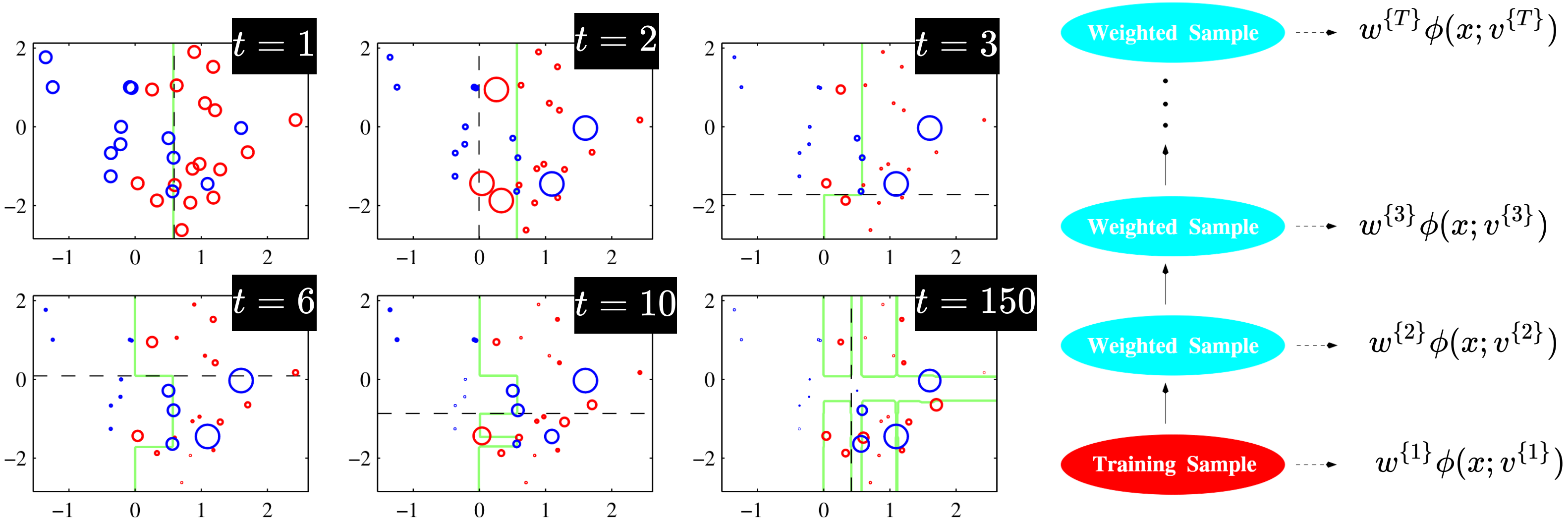
return $f(x) = sign\left(\sum_t w^{\{t\}} \phi(x; v^{\{t\}})\right)$



AdaBoost

each weak learner is a decision stump (dashed line)

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



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)

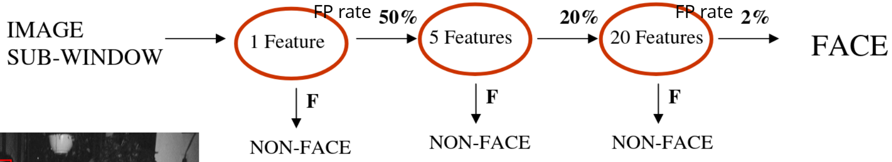
Still can be inefficient:

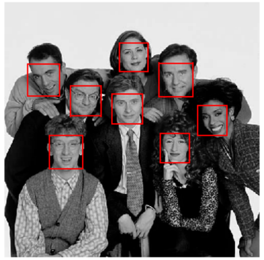
- use the fact that faces are rare (.01% of subwindows are faces)
- cascade of classifiers due to small rate

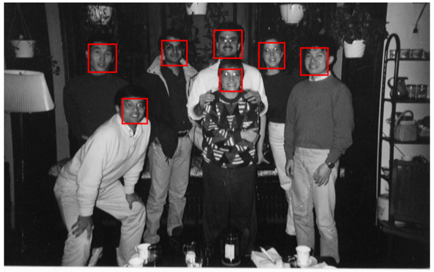
100% detection

cumulative

FP rate







cascade is applied over all image subwindows

fast enough for real-time (object) detection

image credit: David Lowe

Revisit & detailed derivations

AdaBoost Adaptive boosting

model is additive $f^{\{t\}}(x) = f^{\{t-1\}}(x^{(n)}) + w^{\{t\}}\phi(x^{(n)}; v^{\{t\}})$ base/weak learner

trained in sequence

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

$$
\begin{aligned}
L(y, f(x)) &= L(y, f^{\{t-1\}}(x) + w^{\{t\}}\phi(x, v^{\{t\}})) \\
&= e^{-y\left(f^{\{t-1\}}(x)+w^{\{t\}}\phi(x,v^{\{t\}})\right)} \\
&= e^{-yf^{\{t-1\}}(x)-yw^{\{t\}}\phi(x,v^{\{t\}})} \\
&= e^{-yf^{\{t-1\}}(x)}e^{-yw^{\{t\}}\phi(x,v^{\{t\}})} \\
&= L(y, f^{\{t-1\}}(x)) \cdot L(y, w^{\{t\}}\phi(x, v^{\{t\}}))
\end{aligned}
$$

useful property when working with additive models

overall cost function is:

$$
\begin{aligned}
J^{\{t\}} &= \sum_{n=1}^{N} L(y^{(n)}, f^{\{t\}}(x^{(n)})) = \sum_n L(y^{(n)}, f^{\{t-1\}}(x^{(n)})) \cdot L(y^{(n)}, w^{\{t\}}\phi(x^{(n)}, v^{\{t\}})) \\
&= \sum_n q^{(n)} L(y^{(n)}, w^{\{t\}}\phi(x^{(n)}, v^{\{t\}}))
\end{aligned}
$$

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

instances that are not properly classified before receive a higher weight

Discrete AdaBoost

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

assume output is +/- 1 $\quad \phi(x^{(n)}, v^{\{t\}}) = \pm 1$

$L(y, f(x)) = e^{-yf(x)}$

$$L(y^{(n)}, w^{\{t\}}\phi(x^{(n)}, v^{\{t\}})) = e^{-y^{(n)} w^{\{t\}} \phi(x^{(n)}, v^{\{t\}})} = \begin{cases} e^{-w^{\{t\}}} & y^{(n)} = \phi(x^{(n)}; v^{\{t\}}) \quad \text{correctly classified} \\ e^{w^{\{t\}}} & y^{(n)} \neq \phi(x^{(n)}; v^{\{t\}}) \quad \text{misclassified} \end{cases}$$

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

Discrete AdaBoost

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

does not depend on the weak learner

assuming $w^{\{t\}} \geq 0$ the weak learner should minimize this cost

classification with weighted intances to get $v^{\{t\}}$

to find the optimal $w^{\{t\}}$ set $\frac{\partial J}{\partial w^{\{t\}}} = 0$ which gives

$$\frac{\partial J}{\partial w^{\{t\}}} = (e^{w^{\{t\}}} + e^{-w^{\{t\}}}) \sum_n q^{(n)} \mathbb{I}(y^{(n)} \neq \phi(x^{(n)}, v^{\{t\}})) - e^{-w^{\{t\}}} \sum_n q^{(n)} = 0$$

$$= (e^{2w^{\{t\}}} + 1) \sum_n q^{(n)} \mathbb{I}(y^{(n)} \neq \phi(x^{(n)}, v^{\{t\}})) - \sum_n q^{(n)} = 0$$

$$\Rightarrow (e^{2w^{\{t\}}} + 1) = \boxed{\frac{\sum_n q^{(n)}}{\sum_n q^{(n)} \mathbb{I}(y^{(n)} \neq \phi(x^{(n)}, v^{\{t\}}))}} \qquad \ell^{\{t\}} = \frac{\sum_n q^{(n)} \mathbb{I}(\phi(x^{(n)}; v^{\{t\}}) \neq y^{(n)})}{\sum_n q^{(n)}}$$

$$\Rightarrow (e^{2w^{\{t\}}} + 1) = \frac{1}{\ell^{\{t\}}} \quad \Rightarrow 2w^{\{t\}} = \log(\frac{1}{\ell^{\{t\}}} - 1) \Rightarrow w^{\{t\}} = \frac{1}{2} \log(\frac{1 - \ell^{\{t\}}}{\ell^{\{t\}}})$$

Note that this implies the basis for this log is e

Discrete AdaBoost

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

does not depend on the weak learner

assuming $w^{\{t\}} \geq 0$ the weak learner should minimize this cost

classification with weighted intances to get $v^{\{t\}}$

to find the optimal $w^{\{t\}}$ set $\frac{\partial J}{\partial w^{\{t\}}} = 0$ which 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 $w^{\{t\}} \geq 0$

$$q^{(n)} = L(y^{(n)}, f^{\{t-1\}}(x^{(n)}))$$

$$L(y^{(n)}, f^{\{t\}}(x^{(n)})) = q^{(n)} L(y^{(n)}, w^{\{t\}} \phi(x^{(n)}, v^{\{t\}}))$$

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

$$q^{(n),\{t+1\}} = \begin{cases} q^{(n),\{t\}} e^{-w^{\{t\}}} & y^{(n)} = \phi(x^{(n)}; v^{\{t\}}) \\ q^{(n),\{t\}} e^{w^{\{t\}}} & y^{(n)} \neq \phi(x^{(n)}; v^{\{t\}}) \end{cases} = \begin{cases} q^{(n),\{t\}} \sqrt{\frac{\ell^{\{t\}}}{1-\ell^{\{t\}}}} & \text{correctly classified } \searrow \\ q^{(n),\{t\}} \sqrt{\frac{1-\ell^{\{t\}}}{\ell^{\{t\}}}} & \text{misclassified } \nearrow \end{cases}$$

Discrete AdaBoost Algorithm

```
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`

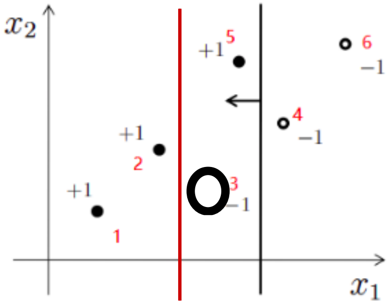
$$\ell^{\{t\}} := \frac{\sum_n q^{(n)} \mathbb{I}(\phi(x^{(n)}; v^{\{t\}}) \neq y^{(n)})}{\sum_n q^{(n)}}$$

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

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

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

Example



$$q := [\tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}]$$

$$\phi^{\{1\}} = [1, 1, 1, -1, 1, -1]$$

$$\ell = \frac{\frac{1}{6}}{\sum q} = \frac{1}{6}$$

$$w = \frac{1}{2} \log(\frac{1 - \frac{1}{6}}{\frac{1}{6}}) = .5 \log(5) \approx 0.8$$

$$q := [\tfrac{1}{6\sqrt{5}}, \tfrac{1}{6\sqrt{5}}, \tfrac{\sqrt{5}}{6}, \tfrac{1}{6\sqrt{5}}, \tfrac{1}{6\sqrt{5}}, \tfrac{1}{6\sqrt{5}}]$$

$$\phi^{\{2\}} = [1, 1, -1, -1, -1, -1]$$

$$\ell = \frac{\frac{1}{6\sqrt{5}}}{\sum q} = \frac{1}{10}$$

$$w = \frac{1}{2} \log(\frac{1 - \frac{1}{10}}{\frac{1}{10}}) = .5 \log(9) \approx 1.1$$

$$f = [sign(.8 + 1.1), sign(.8 + 1.1), sign(.8 - 1.1), sign(-.8 - 1.1), sign(.8 - 1.1), sign(-.8 - 1.1)]$$

Discrete AdaBoost Algorithm

Simplified version, same procedure only update misclassified points

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

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

$$\hat{w}^{\{t\}} := \log \frac{1-\ell^{\{t\}}}{\ell^{\{t\}}}$$

$\hat{w}^{\{t\}} = 2w^{\{t\}}$

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

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

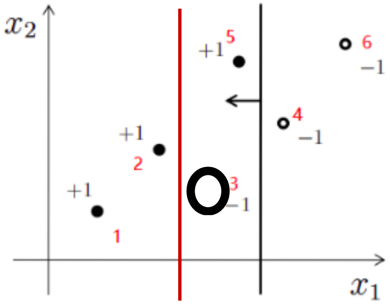
same results as sign is not affected

$$q^{(n)} := q^{(n)} e^{-w^{\{t\}} y^{(n)} \phi(x^{(n)}; v^{\{t\}})} = q^{(n)} e^{w^{\{t\}} (2\mathbb{I}(y^{(n)} \neq \phi(x^{(n)}; v^{\{t\}}))-1)} =$$

$$q^{(n)} e^{2w^{\{t\}} \mathbb{I}(y^{(n)} \neq \phi(x^{(n)}; v^{\{t\}}))} e^{-w^{\{t\}}}$$

cancels out with normalization

Example



$$q := [\tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}]$$

$$\ell = \frac{\frac{1}{6}}{\sum q} = \frac{1}{6}$$

$$\hat{w} = \log\left(\frac{1-\frac{1}{6}}{\frac{1}{6}}\right) = \log(5) \approx 1.6$$

$$q := [\tfrac{1}{6}, \tfrac{1}{6}, \tfrac{5}{6}, \tfrac{1}{6}, \tfrac{1}{6}, \tfrac{1}{6}]$$

$$\ell = \frac{\frac{1}{6}}{\sum q} = \frac{1}{10}$$

$$\hat{w} = \log\left(\frac{1-\frac{1}{10}}{\frac{1}{10}}\right) = \log(9) \approx 2.2$$

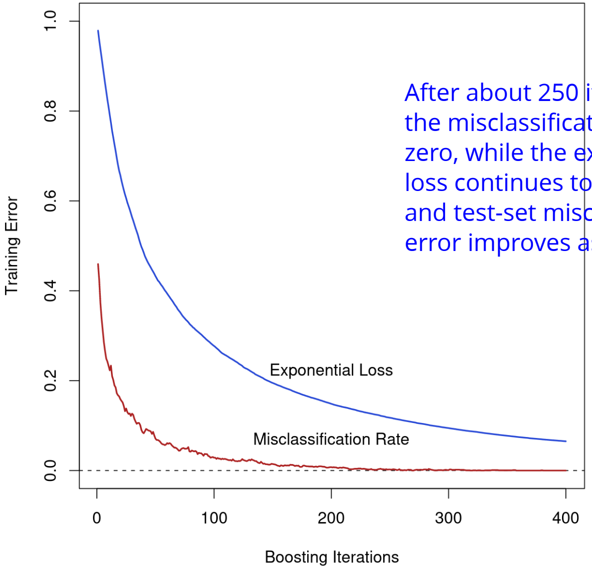
$$f = [sign(2(.8 + 1.1)), sign(2(.8 + 1.1)), sign(2(.8 - 1.1)), sign(2(-.8 - 1.1)), sign(2(.8 - 1.1)), sign(2(-.8 - 1.1))]$$

example AdaBoost

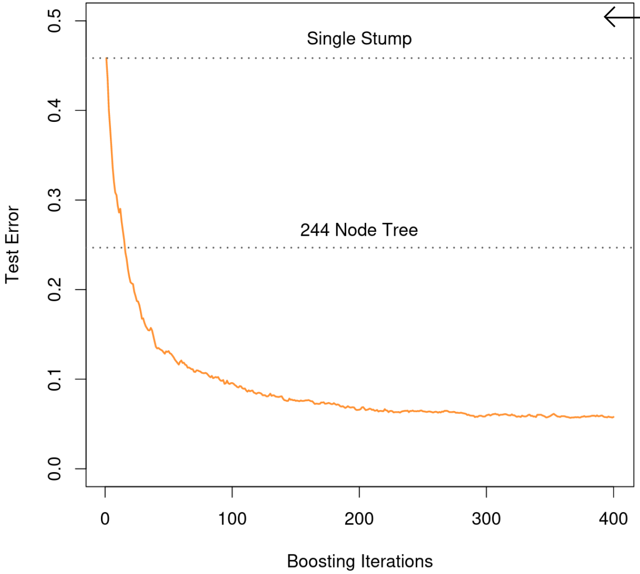
features $x_1^{(n)}, \ldots, x_{10}^{(n)}$ are samples from standard Gaussian

label $y^{(n)} = \mathbb{I}(\sum_d x^{(n)}{}_d^2 > 9.34)$

N=2000 training examples, (~1000+,~1000-)



After about 250 iterations, the misclassification error is zero, while the exponential loss continues to decrease and test-set misclassification error improves as well



random performance (50%)

2 leaf node three

notice that test error does not increase

AdaBoost is very slow to overfit

Boosting

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

optimizing this cost is difficult given the form of f

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

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

L2Boosting $L(y, f(x)) = \frac{1}{2}(y - f(x))^2$

AdaBoost $L(y, f(x)) = e^{-yf(x)}$

General algorithm for any loss ?

Gradient boosting

idea fit the weak learner to the gradient of the cost

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

ignoring the structure of **f**

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

$$\mathbf{f}^{\{t\}} = \mathbf{f}^{\{t-1\}} - w^{\{t\}} \mathbf{g}^{\{t\}}$$

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

we can look for the optimal step size

optional

gradient vector

its role is similar to residual

write $\hat{\mathbf{f}}$ as a sum of steps $\hat{\mathbf{f}} = \mathbf{f}^{\{T\}} = \mathbf{f}^{\{0\}} - \sum_{t=1}^{T} w^{\{t\}} \mathbf{g}^{\{t\}}$

Gradient boosting

idea fit the weak learner to the gradient of the cost

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

ignoring the structure of **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 $\hat{\mathbf{f}} = \mathbf{f}^{\{T\}} = \mathbf{f}^{\{0\}} - \sum_{t=1}^{T} w^{\{t\}} \mathbf{g}^{\{t\}}$

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

we can look for the optimal step size

gradient vector

its role is similar to residual

so far we treated **f** as a parameter vector of input size, **to generalize:**

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

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

$\boldsymbol{\phi}_v = \left[\phi(x^{(1)}; v), \ldots, \phi(x^{(N)}; v)\right]^\top$

$v^{\{t\}} = \arg\min_v \sum_n ((-g) - \phi(x^{(n)}, v))^2$

Gradient boosting

```
initialize
```
$f^{\{0\}}(x)$ `using a base learner` $\arg\min_v \sum_n L(y^{(n)}, \phi(x^{(n)}, v))$

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

```
    calculate the gradient
```
$g^{(n),\{t\}} = \frac{\partial}{\partial f^{\{t-1\}}(x^{(n)})} L(f^{\{t-1\}}(x^{(n)}), y(x^{(n)}))$

```
    fit a weak learner to negative of gradient using
```
$v^{\{t\}} = \arg\min_v \sum_n (g^{(n),\{t\}} + \phi(x^{(n)}, v))^2$

```
    find the optimal step size
```
$w^{\{t\}} = \arg\min_w \sum_n L(f^{\{t-1\}}(x^{(n)}) - w g^{(n),\{t\}})$ optional, can use fixed rate as well

```
    Update the function
```
$f^{\{t\}}(x) = f^{\{t-1\}}(x) + w^{\{t\}} \phi(x, v^{\{t\}})$

```
return
```
$f^{\{T\}}(x)$

We can use different loss functions for example:

$$L(y, f(x)) = \frac{1}{2}(y - f(x))^2 \quad \Rightarrow \quad g = y - f(x)$$

L2Boosting

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 $\underset{N \times D}{\mathbf{X}}, \underset{N}{\mathbf{r}}$ and produce regions $\mathbb{R}_1, \ldots, \mathbb{R}_K$ shallow trees of K = 4-8 leaf usually work well as weak learners

re-adjust predictions per region $w_k = \arg\min_w \sum_{x^{(n)} \in \mathbb{R}_k} L(y^{(n)}, f^{\{t-1\}}(x^{(n)}) + w)$ refinement over the generic algorithm, re-adjust the weak learner

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

using a small learning rate here improves test error (**shrinkage**)

return $f^{\{T\}}(x)$

a.k.a **MART**: multiple additive regression trees

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:

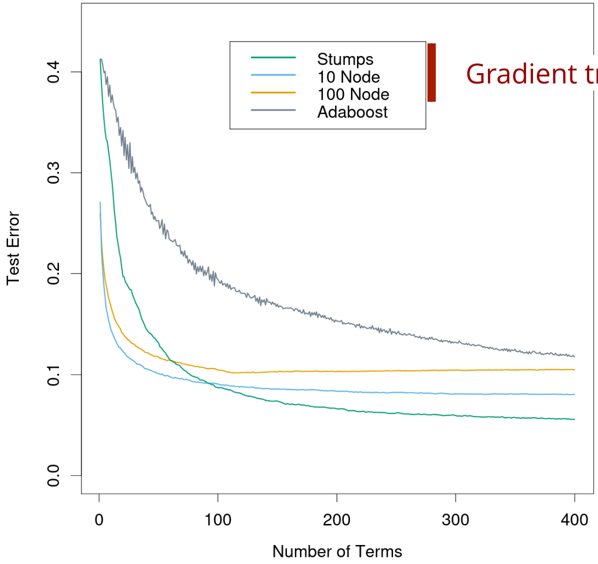
features $x_1^{(n)}, \ldots, x_{10}^{(n)}$ are samples from standard Gaussian

label $y^{(n)} = \mathbb{I}(\sum_d x_d^{(n)\,2} > 9.34)$

N=2000 training examples, (~1000+,~1000-)

Boosting with different sized trees

Entropy (a.k.a deviance) for the trees loss



Gradient tree boosting (using log-loss) works better than Adaboost

since sum of features are used in prediction using stumps work best example **Gradient tree boosting**

recall the synthetic example:

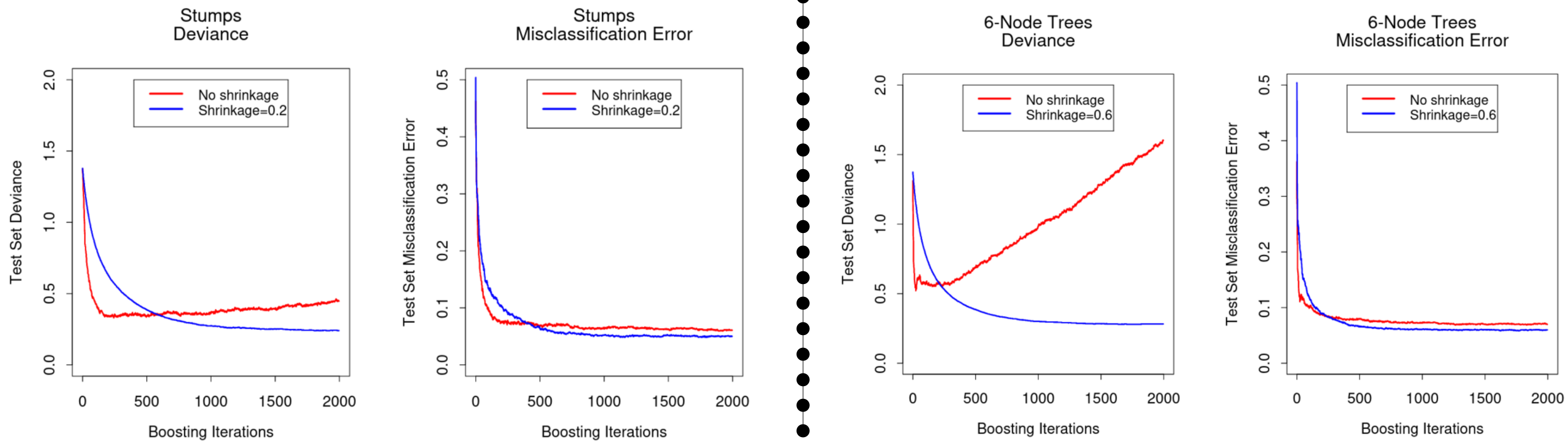
features $x_1^{(n)}, \ldots, x_{10}^{(n)}$ are samples from standard Gaussian

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

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

scale the contribution of each tree by a factor
to control the learning rate of boosting procedure
smaller rate needs more iterations

example

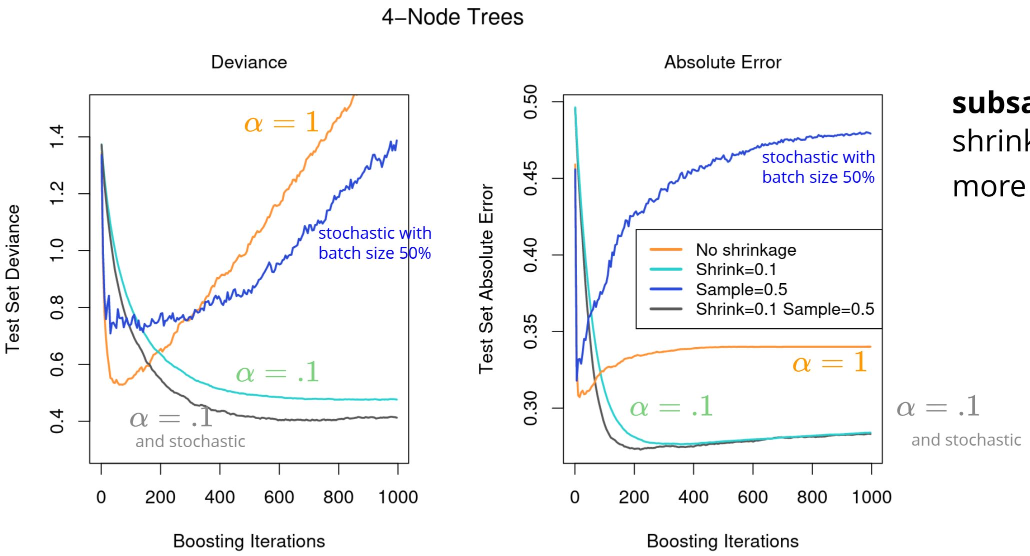
Stochastic Gradient tree boosting

recall the synthetic example:

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

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

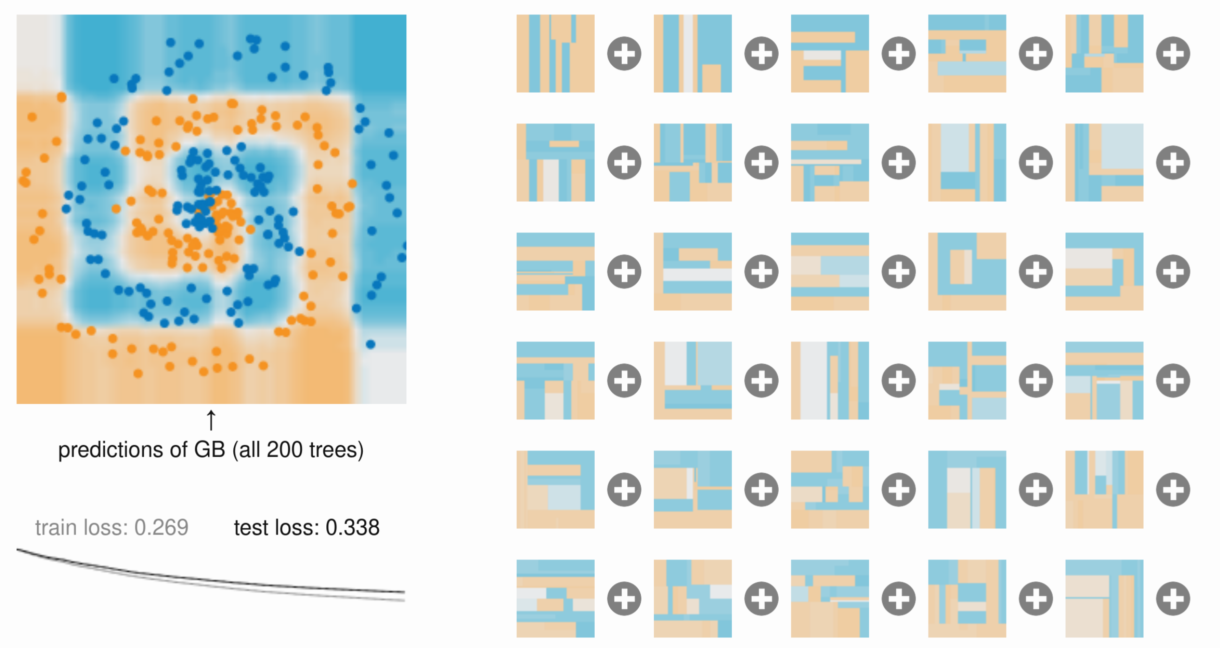
N=2000 training examples



subsampling (a fraction of input) on top of shrinkage 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	$\|\|\mathbf{y} - \mathbf{f}\|\|_1$	$\mathrm{sign}(\mathbf{y} - \mathbf{f})$
binary classification	$\exp(-\mathbf{y}\mathbf{f})$ exponential loss	$-\mathbf{y}\exp(-\mathbf{y}\mathbf{f})$
multiclass classification	multi-class cross-entropy	$\underset{N \times C}{\mathbf{Y}} - \underset{N \times C}{\mathbf{P}}$

one-hot coding for C-class classification

predicted class probabilities

$\mathbf{P}_{c,:} = \mathrm{softmax}(f_{[c]})$