Applied Machine Learning

Bootstrap, Bagging and Boosting

Siamak Ravanbakhsh
Learning objectives

bootstrap for uncertainty estimation
bagging for variance reduction
  • random forests
boosting
  • AdaBoost
  • gradient boosting
  • relationship to L1 regularization
Bootstrap

a simple approach to estimate the uncertainty in prediction
Bootstrap

a simple approach to estimate the uncertainty in prediction

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

a simple approach to estimate the uncertainty in prediction

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

subsample **with replacement** \( B \) datasets of size \( N \)

\[
D_b = \{ (x^{(n,b)}, y^{(n,b)}) \}_{n=1}^{N}, b = 1, \ldots, B
\]
Bootstrap

a simple approach to estimate the uncertainty in prediction

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)
Bootstrap

a simple approach to estimate the uncertainty in prediction

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
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
Bootstrap: example

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

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

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

![Graph showing the model before adding noise and the fit to data using 10 Gaussian bases.]

```python
#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-')
```
Recall: linear model with nonlinear Gaussian bases (N=100) using B=500 bootstrap samples gives a measure of uncertainty of the parameters.

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

```
# Phi: N 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,:]
    y_b = y[inds]
    #fit the subsampled data
    ws[b,:] = np.linalg.lstsq(Phi_b, y_b[:,b])[0]
plt.hist(ws, bins=50)
```
Bootstrap: example

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

ϕ_k(x) = e^{-(x - \mu_k)^2 / \sigma_k^2}

```
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 gives a measure of uncertainty of the parameters.

```python
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)
```
Bootstrap: example

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

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

```
# Phi: N x D
# y: N
B = 500
ws = np.zeros((B,D))
for b in range(B):
    # Phi: N x D
    # y: N
    inds = np.random.randint(N, size=(N))
    Phi_b = Phi[inds, :]
    y_b = y[inds]
    # fit the subsampled data
    ws[b, :] = np.linalg.lstsq(Phi_b, y_b[:, b])[0]
plt.hist(ws, bins=50)
```
$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{\sigma^2}}$

Recall: linear model with nonlinear Gaussian bases (N=100)
using B=500 bootstrap samples
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
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

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

```python
#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)
```
Bootstrap: example

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

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

---

```python
for b in range(B):
    wb = ws[b, :]
y_hats[b, :] = np.dot(Phi_test, wb)
```

---

The **red lines** are 5% and 95% quantiles (for each point we can get these across bootstrap model predictions)
$$\phi_k(x) = e^{-\frac{(x - \mu_k)^2}{s^2}}$$

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)

```python
#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

use bootstrap for more accurate prediction (not just uncertainty)

variance of sum of random variables

\[ \text{Var}(z_1 + z_2) = \mathbb{E}[(z_1 + z_2)^2] - \mathbb{E}[z_1 + z_2]^2 \]
Bagging

use bootstrap for more accurate prediction (not just uncertainty)

variance of sum of random variables

\[ \text{Var}(z_1 + z_2) = \mathbb{E}[(z_1 + z_2)^2] - \mathbb{E}[z_1 + z_2]^2 \]

\[ = \mathbb{E}[z_1^2 + z_2^2 + 2z_1z_2] - (\mathbb{E}[z_1] + \mathbb{E}[z_2])^2 \]
Bagging

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

variance of sum of random variables

\[ \text{Var}(z_1 + z_2) = \mathbb{E}[(z_1 + z_2)^2] - \mathbb{E}[z_1 + z_2]^2 \]

\[ = \mathbb{E}[z_1^2 + z_2^2 + 2z_1z_2] - (\mathbb{E}[z_1] + \mathbb{E}[z_2])^2 \]

\[ = \mathbb{E}[z_1^2] + \mathbb{E}[z_2^2] + \mathbb{E}[2z_1z_2] - \mathbb{E}[z_1]^2 - \mathbb{E}[z_2]^2 - 2\mathbb{E}[z_1]\mathbb{E}[z_2] \]
Bagging

use bootstrap for more accurate prediction (not just uncertainty)

variance of sum of random variables

\[
\text{Var}(z_1 + z_2) = \mathbb{E}[(z_1 + z_2)^2] - \mathbb{E}[z_1 + z_2]^2
\]

\[
= \mathbb{E}[z_1^2 + z_2^2 + 2z_1z_2] - (\mathbb{E}[z_1] + \mathbb{E}[z_2])^2
\]

\[
= \mathbb{E}[z_1^2] + \mathbb{E}[z_2^2] + \mathbb{E}[2z_1z_2] - \mathbb{E}[z_1]^2 - \mathbb{E}[z_2]^2 - 2\mathbb{E}[z_1]\mathbb{E}[z_2]
\]

\[
= \text{Var}(z_1) + \text{Var}(z_2) + 2\text{Cov}(z_1, z_2)
\]
Bagging

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

variance of sum of random variables

$$\text{Var}(z_1 + z_2) = \mathbb{E}[(z_1 + z_2)^2] - \mathbb{E}[z_1 + z_2]^2$$

$$= \mathbb{E}[z_1^2 + z_2^2 + 2z_1z_2] - (\mathbb{E}[z_1] + \mathbb{E}[z_2])^2$$

$$= \mathbb{E}[z_1^2] + \mathbb{E}[z_2^2] + \mathbb{E}[2z_1z_2] - \mathbb{E}[z_1]^2 - \mathbb{E}[z_2]^2 - 2\mathbb{E}[z_1]\mathbb{E}[z_2]$$

$$= \text{Var}(z_1) + \text{Var}(z_2) + 2\text{Cov}(z_1, z_2)$$

*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
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 \( \mu \) and variance \( \sigma^2 \)

the average \( \bar{z} = \frac{1}{B} \sum_b z_b \) has mean \( \mu \) and variance
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 $\mu$ and variance $\sigma^2$

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

$$\text{Var}(\frac{1}{B} \sum_b z_b) = \frac{1}{B^2} \text{Var}(\sum_b z_b) = \frac{1}{B^2} B \sigma^2 = \frac{1}{B} \sigma^2$$
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 \( \mu \) and variance \( \sigma^2 \)

the average \( \bar{z} = \frac{1}{B} \sum_b z_b \) has mean \( \mu \) and variance

\[
\text{Var}(\frac{1}{B} \sum_b z_b) = \frac{1}{B^2} \text{Var}(\sum_b z_b) = \frac{1}{B^2} B \sigma^2 = \frac{1}{B} \sigma^2
\]

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

**regression:** average the model predictions

\[
\hat{f}(x) = \frac{1}{B} \sum_b \hat{f}_b(x)
\]
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 \( \mu \) and variance \( \sigma^2 \)

the average \( \bar{z} = \frac{1}{B} \sum_b z_b \) has mean \( \mu \) and variance

\[
\text{Var}(\frac{1}{B} \sum_b z_b) = \frac{1}{B^2} \text{Var}(\sum_b z_b) = \frac{1}{B^2} B \sigma^2 = \frac{1}{B} \sigma^2
\]

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
Bagging for classification

averaging makes sense for regression, how about classification?
Bagging for classification

Averaging makes sense for regression, how about classification?

**Wisdom of crowds**

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

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

Averaging makes sense for regression, how about classification?

**Wisdom of crowds**

\[ z_1, \ldots, z_B \in \{0, 1\} \text{ are IID Bernoulli random variables with mean } \mu = 0.5 + \epsilon > 0 \]

For \[ \bar{z} = \frac{1}{B} \sum_b z_b \] we have \[ p(\bar{z} > 0.5) \text{ goes to 1 as } B \text{ grows} \]

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

- Use voting
Bagging for classification

Averaging makes sense for regression, how about classification?

**Wisdom of crowds**

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

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 for classification

Averaging makes sense for regression, how about classification?

- Use voting

Crowds are wiser when

- Individuals are better than random
- Votes are uncorrelated

\textbf{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
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!*
Bagging decision trees

**Example**

**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!

![Graph showing voting for the most probably class and averaging probabilities](image)
Random forests further reduce the correlation between decision trees
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
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
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#
- **average, max, sum of length of uninterrupted sequences of capital letters:**
  - CAPAVE, CAPMAX, CAPTOT
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
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#*
- **average, max, sum of length of uninterrupted sequences of capital letters:**
  - CAPAVE, CAPMAX, CAPTOT

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

<table>
<thead>
<tr>
<th></th>
<th>george</th>
<th>you</th>
<th>your</th>
<th>hp</th>
<th>free</th>
<th>hpl</th>
<th>!</th>
<th>our</th>
<th>re</th>
<th>edu</th>
<th>remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>spam</td>
<td>0.00</td>
<td>2.26</td>
<td>1.38</td>
<td>0.02</td>
<td>0.52</td>
<td>0.01</td>
<td>0.51</td>
<td>0.51</td>
<td>0.13</td>
<td>0.01</td>
<td>0.28</td>
</tr>
</tbody>
</table>
| email    | 1.27   | 1.27  | 0.44  | 0.90  | 0.07  | 0.43 | 0.11  | 0.18  | 0.42  | 0.29  | 0.01   

an example of feature engineering
Example: spam detection

decision tree after pruning
Example: spam detection

decision tree after pruning

misclassification rate on test data
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!
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
- Bootstrap aggregation (Bagging) can reduce the variance of unstable models
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*

\[ f(x) = \sum_d w_d \phi_d(x; v_d) \]

- 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 *weak learners*
Forward stagewise additive modelling
Forward stagewise additive modelling

\[ f(x) = \sum_{t=1}^{T} w^{\{t\}} \phi(x; v^{\{t\}}) \]

A simple model, such as decision stump (decision tree with one node)
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**

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

so far we have seen L2 loss, log loss and hinge loss

optimizing this cost is difficult given the form of f
Forward stagewise additive modelling

\[ f(x) = \sum_{t=1}^{T} w^{\{t\}} \phi(x; v^{\{t\}}) \]

A simple model, such as decision stump (decision tree with one node)

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

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

\[ v^{\{t\}}, w^{\{t\}} = \text{arg min}_{v, w} \sum_{n=1}^{N} L(y^{(n)}, f^{\{t-1\}}(x^{(n)}) + w \phi(x^{(n)}; v)) \]

2. Add it to the current model

\[ f^{\{t\}}(x) = f^{\{t-1\}}(x^{(n)}) + w^{\{t\}} \phi(x^{(n)}; v^{\{t\}}) \]
Consider weak learners that are individual features: \( \phi^{\{t\}}(x) = w^{\{t\}} x_{d(t)} \)
consider weak learners that are individual features $\phi^{(t)}(x) = w^{(t)} x_{d^{(t)}}$

using L2 loss for regression

at stage $t$ \[
\arg\min_{d,w_d} \frac{1}{2} \sum_{n=1}^{N} \left( y^{(n)} - (f^{(t-1)}(x^{(n)}) + w_d x_{d}^{(n)}) \right)^2
\]
$L_2$ loss & forward stagewise linear model

consider **weak learners** that are individual features $\phi^t(x) = w^t x_d^t$

using L2 loss for regression

at stage $t$  

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

residual $r^{(n)}$
**$L_2$** loss & forward stagewise **linear** model

**model**

consider **weak learners** that are individual features $\phi_t(x) = w_t x_d(t)$

**cost**

using L2 loss for regression

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

**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
$L_2$ loss & forward stagewise linear model

**model**

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

**cost**

using L2 loss for regression

at stage $t$ \[ \arg\min_{d, w_d} \frac{1}{2} \sum_{n=1}^{N} \left( r^{(n)} \right)^2 \]

**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 $t$: $f^{t}(x) = \sum_{t} \alpha w^{t}_{d^{(t)}} x_{d^{(t)}}$

using a small $\alpha$ helps with test error

is this related to L1-regularized linear regression?
using small learning rate $\alpha = .01$  L2 Boosting has a similar regularization path to lasso
L₂ 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 $\hat{y} = \text{sign}(f(x))$
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)
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)
loss functions for **binary classification** $y \in \{-1, +1\}$

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

misclassification loss $L(y, f(x)) = 1(yf(x) > 0)$ (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
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)
loss functions for **binary classification** $y \in \{-1, +1\}$

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

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

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

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

support vector loss

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

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

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

---

**cost** using exponential loss

\[ L(y^{(n)}, f^{(t-1)}(x^{(n)})) \]
Exponential loss & AdaBoost

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

Cost using exponential loss

\[ \text{Discrete AdaBoost: assume this is a simple classifier, so its output is +/- 1} \]
Exponential loss & AdaBoost

Using exponential loss:

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

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

Objective is to find the weak learner minimizing the cost above.

\[ J(\{w^t, v^t\}_t) = \sum_n q^{(n)} e^{-y^{(n)} w^t \phi(x^{(n)}, v^t)} \]
Exponential loss & AdaBoost

Cost using exponential loss

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

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

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

\[ = e^{-w^t} \sum_n q(n) \mathbb{1}(y^{(n)} = \phi(x^{(n)}, v^t)) + e^{w^t} \sum_n q(n) \mathbb{1}(y^{(n)} \neq \phi(x^{(n)}, v^t)) \]
Exponential loss & AdaBoost

**Cost** using exponential loss

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

loss for this instance at previous stage

\[
\sum_n q^{(n)} L(y^{(n)}, f^{(t-1)}(x^{(n)}))
\]

**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

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

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

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

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

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

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

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 $v^t$
Exponential loss & AdaBoost

\[ J(w_t, v_t) = \sum_n q(n) L(y(n), w_t \phi(x(n), v_t)) \]

\[ = e^{-w(t)} \sum_n q(n) + (e^{w(t)} - e^{-w(t)}) \sum_n q(n) \mathbb{1}(y(n) \neq \phi(x(n), v_t)) \]

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 \( w_t \)
Exponential loss & AdaBoost

\[
J(\{w^t, v^t\}_t) = \sum_n q^{(n)} L(y^{(n)}, w^t \phi(x^{(n)}, v^t))
\]
\[
= e^{-w^t} \sum_n q^{(n)} + (e^{w^t} - e^{-w^t}) \sum_n q^{(n)} \mathbb{1}(y^{(n)} \neq \phi(x^{(n)}, v^t))
\]

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:

\[
\ell^t = \frac{\sum_n q^{(n)} \mathbb{1}(\phi(x^{(n)}, v^t) \neq y^{(n)})}{\sum_n q^{(n)}}
\]
Exponential loss & AdaBoost

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

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

assuming \( w^t \geq 0 \) the weak learner should minimize this cost

this is classification with weighted instances

this gives \( w^t \)

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 \( w^t \geq 0 \)
Exponential loss & AdaBoost

\[ J(\{w^t\}, v^t) = \sum_n q^{(n)} L(y^{(n)}, w^t \phi(x^{(n)}, v^t)) \]

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

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

\[ \ell^t = \sum_n q^{(n)} (\phi(x^{(n)}, v^t) \neq y^{(n)}) \]

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)
Exponential loss & AdaBoost

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

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

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

\[ \ell^t = \frac{\sum_n q^{(n)} I(\phi(x^{(n)}, v^t) \neq y^{(n)})}{\sum_n q^{(n)}} \]

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
overall algorithm for discrete AdaBoost
overall algorithm for discrete AdaBoost
overall algorithm for discrete AdaBoost
overall algorithm for discrete AdaBoost
overall algorithm for discrete AdaBoost
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

$\ell^{(t)} := \frac{\sum_n q^{(n)} 1(\phi(x^{(n)}; v^{(t)}) \neq y^{(n)})}{\sum_n q^{(n)}}$

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

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

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

each weak learner is a decision stump (dashed line)
circle size is proportional to $q_{n,t}$
green is the decision boundary of $f^t$

\[
\hat{y} = \text{sign} \left( \sum_t w^t \phi(x; v^t) \right)
\]
features $x_1^{(n)}, \ldots, x_{10}^{(n)}$ are samples from standard Gaussian
features $x_1^{(n)}, \ldots, x_{10}^{(n)}$ are samples from standard Gaussian label $y^{(n)} = \mathbb{I}(\sum_d x_d^{(n)} > 9.34)$
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
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
example AdaBoost

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

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

---

Winter 2020 | Applied Machine Learning (COMP551)
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
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

image source: David Lowe slides
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
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
FP rate 50%
cumulative
FP rate 2%
cascade is applied over all image subwindows
fast enough for real-time (object) detection
Gradient boosting

fit the weak learner to the gradient of the cost
Gradient boosting

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

fit the weak learner to the gradient of the cost
Gradient boosting

**idea** fit the weak learner to the gradient of the cost

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

ignoring the structure of $f$

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

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

ingoing the structure of $f$

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

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

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

its role is similar to residual
**Gradient boosting**

**idea** fit the weak learner to the gradient of the cost

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

ignoring the structure of \( f \)

if we use gradient descent to minimize the loss \( \hat{f} = \text{arg min}_f L(f, y) \)

write \( \hat{f} \) as a sum of steps

\[
\hat{f} = f^{\{T\}} = f^{\{0\}} - \sum_{t=1}^{T} w^{\{t\}} g^{\{t\}}
\]

\[
w^{\{t\}} = \text{arg min}_w L(f^{\{t-1\}} - w g^{\{t\}}) \frac{\partial}{\partial f} L(f^{\{t-1\}}, y)
\]

we can look for the optimal step size

its role is similar to residual
Gradient boosting

**idea** fit the weak learner to the gradient of the cost

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

ignoring the structure of $f$

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

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

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

$w^{(t)} = \arg \min_{w} L(f^{(t-1)} - wg^{(t)}) \frac{\partial}{\partial f} L(f^{(t-1)}, y)$

we can look for the optimal step size

its role is similar to residual

so far we treated $f$ as a parameter vector
Gradient boosting

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

ignoring the structure of $f$

if we use gradient descent to minimize the loss

$$\hat{f} = \arg \min_f L(f, y)$$

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

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

so far we treated $f$ as a parameter vector

fit the weak-learner to negative of the gradient

$$v^t = \arg \min_v \frac{1}{2} \| \phi_v - (-g) \|^2_2$$

we can look for the optimal step size

its role is similar to residual
Gradient boosting

Let \( f^t = \begin{bmatrix} f^t(x^{(1)}) , \ldots , f^t(x^{(N)}) \end{bmatrix}^\top \) and true labels \( y = \begin{bmatrix} y^{(1)} , \ldots , y^{(N)} \end{bmatrix}^\top \)

ignoring the structure of \( f \)

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

write \( \hat{f} \) as a sum of steps

\[
\hat{f} = f^T = f^0 - \sum_{t=1}^{T} \frac{\partial}{\partial f} L(f^{t-1}, y) \frac{g^t}{\|g^t\|_2}
\]

so far we treated \( f \) as a parameter vector

fit the weak-learner to negative of the gradient

\[
v^t = \arg \min_v \frac{1}{2} \| \phi_v - (-g) \|_2^2
\]

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

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

its role is similar to residual

we can look for the optimal step size

fit the weak learner to the gradient of the cost

fit the weak-learner to negative of the gradient
Gradient tree boosting

apply gradient boosting to CART (classification and regression trees)

initialize $f^0$ to predict a constant

for $t=1:T$

- calculate the negative of the gradient $r = -\frac{\partial}{\partial f} L(f^{t-1}, y)$
- fit a regression tree to $X, r$ and produce regions $\mathbb{R}_1, \ldots, \mathbb{R}_K$
- re-adjust predictions per region $w_k = \arg\min_w \sum_{x^{(n)} \in \mathbb{R}_k} L(y^{(n)}, f^{t-1}(x^{(n)}) + w_k)$
- update $f^t(x) = f^{t-1}(x) + \alpha \sum_{k=1}^{K} w_k \mathbb{I}(x \in \mathbb{R}_k)$

return $f^T(x)$
Gradient tree boosting

apply gradient boosting to CART (classification and regression trees)

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

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

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

fit a regression tree to $X, r$ and produce regions $\mathbb{R}_1, \ldots, \mathbb{R}_K$

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

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

return $f^{T}(x)$
Gradient tree boosting

apply gradient boosting to CART (classification and regression trees)

initialize $f^{(0)}$ to predict a constant

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

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

* fit a regression tree to $X_{N \times D}, \mathbf{r}$ 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 $w_k = \arg\min_w \sum_{x^{(n)} \in \mathbb{R}_k} L(y^{(n)}, f^{(t-1)}(x^{(n)}) + w_k)$

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

return $f^{(T)}(x)$
apply gradient boosting to CART (classification and regression trees)

**Gradient tree boosting**

<table>
<thead>
<tr>
<th>Gradient tree boosting</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize $f^{(0)}$ to predict a constant</td>
</tr>
<tr>
<td>for $t=1:T$ decide T using a validation set (early stopping)</td>
</tr>
<tr>
<td>calculate the negative of the gradient $r = -\frac{\partial}{\partial f} L(f^{(t-1)}, y)$</td>
</tr>
<tr>
<td>fit a regression tree to $X_N, r$ and produce regions $\mathbb{R}_1, \ldots, \mathbb{R}_K$</td>
</tr>
<tr>
<td>re-adjust predictions per region $w_k = \arg\min_w \sum_{x} L(y, f^{(t-1)}(x) + w_k)$</td>
</tr>
<tr>
<td>update $f^{(t)}(x) = f^{(t-1)}(x) + \alpha \sum_{k=1}^K w_k I(x \in \mathbb{R}_k)$</td>
</tr>
<tr>
<td>return $f^{(T)}(x)$</td>
</tr>
</tbody>
</table>

shallow trees of $K = 4-8$ leaf usually work well as weak learners

this is effectively the line-search
Gradient tree boosting

apply gradient boosting to CART (classification and regression trees)

initialize $f^{(0)}$ to predict a constant

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

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

fit a regression tree to $X_{N \times D}, r$ and produce regions $R_1, \ldots, R_K$ shallow trees of $K = 4-8$ leaf usually work well as weak learners

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

update $f^{(t)}(x) = f^{(t-1)}(x) + \alpha \sum_{k=1}^{K} w_k \mathbb{I}(x \in R_k)$ using a small learning rate here improves test error (shrinkage)

return $f^{(T)}(x)$
Gradient tree boosting

apply gradient boosting to CART (classification and regression trees)

initialize $f^{[0]}$ to predict a constant

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

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

fit a regression tree to $X, r$ 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 $w_k = \arg \min_w \sum_{x^{(n)} \in \mathbb{R}_k} L(y^{(n)}, f^{[t-1]}(x^{(n)}) + w_k)$

this is effectively the line-search

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

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
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
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

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

Gradient tree boosting (using log-loss) works better than Adaboost since sum of features are used in prediction using stumps work best
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

$\alpha = .2$
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

\[ \alpha = .2 \]
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

$\alpha = .2$
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

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

\( \alpha = .2 \) helps

while test loss may increase, test misclassification error does not
Gradient tree boosting

recall the synthetic example:

\[\begin{align*}
\text{features} & \quad x_1^{(n)}, \ldots, x_{10}^{(n)} \text{ are samples from standard Gaussian} \\
\text{label} & \quad y^{(n)} = \mathbb{I}(\sum_d x_d^{(n)} > 9.34) \\
N & = 2000 \text{ training examples}
\end{align*}\]

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

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

- bagging & random forests (reduce variance)
  - produce models with minimal correlation
  - use their average prediction
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)*
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

\[ \hat{f} = f^T = f^0 - \sum_{t=1}^{T} w^t \frac{\partial}{\partial f} L(f^t, y) \]

<table>
<thead>
<tr>
<th>setting</th>
<th>loss function</th>
<th>(- \frac{\partial}{\partial f} L(f^t, y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>regression</td>
<td>(\frac{1}{2} | y - f |^2 )</td>
<td>(y - f)</td>
</tr>
<tr>
<td>regression</td>
<td>(| y - f |_1 )</td>
<td>(\text{sign}(y - f))</td>
</tr>
<tr>
<td>binary classification</td>
<td>(\exp(-yf))</td>
<td>(-y \exp(-yf))</td>
</tr>
<tr>
<td>classification</td>
<td>multi-class cross-entropy</td>
<td>(Y - P)</td>
</tr>
<tr>
<td></td>
<td>exponential loss</td>
<td></td>
</tr>
<tr>
<td></td>
<td>one-hot coding for C-class</td>
<td></td>
</tr>
<tr>
<td></td>
<td>classification</td>
<td></td>
</tr>
<tr>
<td></td>
<td>predicted class probabilities</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(P_{c,:} = \text{softmax}(f_c))</td>
<td></td>
</tr>
</tbody>
</table>