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
  - 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

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

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)

\[ \phi_k(x) = e^{-\frac{(x-mu)^2}{2k^2}} \]

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

before adding noise

our 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.

```
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]
```

Each color is a different weight $w_d$. 

$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$
Recall: linear model with nonlinear Gaussian bases (N=100) using B=500 bootstrap samples also gives a measure of uncertainty of the predictions.

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

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

variance of sum of random variables

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

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

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

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

*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

\[ z_1, \ldots, z_B \text{ are uncorrelated random variables with mean } \mu \text{ and variance } \sigma^2 \]

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?

Wisdom of crowds

\[ z_1, \ldots, z_B \in \{0, 1\} \] 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
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#*
- 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>0.13</td>
<td>0.13</td>
<td>0.01</td>
<td>0.28</td>
</tr>
<tr>
<td>email</td>
<td>1.27</td>
<td>1.27</td>
<td>0.44</td>
<td>0.90</td>
<td>0.07</td>
<td>0.43</td>
<td>0.11</td>
<td>0.18</td>
<td>0.42</td>
<td>0.29</td>
<td>0.01</td>
</tr>
</tbody>
</table>
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

cv error
test error
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.
- Bootstrapping 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

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

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

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

optimizing this cost is difficult given the form of \( f \)

\[
\text{optimization idea } \quad \text{add one weak-learner in each stage } t, \text{ to reduce the error of previous stage}
\]

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 $\phi^t(x) = w^t x_{d^t}$

**cost** using L2 loss for regression

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

**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?
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

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

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

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

Assuming \(w^{(t)} \geq 0\) the weak learner should minimize this cost.

This is classification with weighted instances.

Assuming \(w^{(t)} \geq 0\) the weak learner should minimize this cost.

This is classification with weighted instances.
Exponential loss & AdaBoost

Cost:

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

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

This 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)

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

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:

\[ q^{(n),\{t+1\}} = q^{(n),\{t\}} e^{-w^{(t)}_i y^{(n)} \phi(x^{(n)}_i; v^{(t)})} \]

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

for $t=1:T$

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

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

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

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

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

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

Weighted Sample $\rightarrow w^{(T)} \phi(x; v^{(T)})$

Weighted Sample $\rightarrow w^{(3)} \phi(x; v^{(3)})$

Weighted Sample $\rightarrow w^{(2)} \phi(x; v^{(2)})$

Training Sample $\rightarrow w^{(1)} \phi(x; v^{(1)})$
AdaBoost

Each weak learner is a decision stump (dashed line).
Circle size is proportional to $q^n_{t}\{t\}$.
Green is the decision boundary of $f^{t}\{t\}$.

$$
\hat{y} = \text{sign}(\sum_t w^{t}\phi(x; v^{t}))
$$
features \( x_1^{(n)}, \ldots, x_{10}^{(n)} \) are samples from standard Gaussian

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

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

\[
\hat{f} = \text{arg min}_f L(f, y)
\]

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

ignoring the structure of \( f \)

if we use gradient descent to minimize the loss

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

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

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

we can look for the optimal step size

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

fit the weak-learner to negative of the gradient

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

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

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

apply gradient boosting to CART (classification and regression trees)

\[ f^{[0]} \text{ to predict a constant} \]

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

\[ r = - \frac{\partial}{\partial f} L(f^{[t-1]}, y) \]

fit a regression tree to \( X, r \) and produce regions \( R_1, \ldots, R_K \)

re-adjust predictions per region \( w_k = \arg \min_w \sum_{x^{(n)} \in R_k} L(y^{(n)}, f^{[t-1]}(x^{(n)}) + w_k) \)

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

return \( f^{[T]}(x) \)

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 \( x^{(n)}_1, \ldots, x^{(n)}_{10} \) are samples from standard Gaussian
- label \( y^{(n)} = I(\sum_d x^{(n)}_d > 9.34) \)
- 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

deviance = cross entropy = log-loss

in both cases using shrinkage $\alpha = .2$ helps

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

example

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

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
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

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

<table>
<thead>
<tr>
<th>setting</th>
<th>loss function</th>
<th>(- \frac{\partial}{\partial f} L(f^{(t-1)}, y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>regression</td>
<td>(\frac{1}{2}</td>
<td></td>
</tr>
<tr>
<td>regression</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>binary classification</td>
<td>(\exp(-yf))</td>
<td>(-y \exp(-yf))</td>
</tr>
<tr>
<td>multiclass classification</td>
<td>multi-class cross-entropy</td>
<td>(Y - P)</td>
</tr>
</tbody>
</table>

one-hot coding for C-class classification

\[P_{c,:} = \text{softmax}(f_{[c]})\]

predicted class probabilities

\[P_{c,:} = \text{softmax}(f_{[c]})\]