Overview

Topics covered in the lecture:

- Ensemble Methods
- Bias-Variance Decomposition
- Bagging
- Random Forests
- Boosting and Gradient Boosting
- Gradient Boosted Trees
Ensemble Methods

- Inspired in *Wisdom of the crowd*
  - (weighted) averaging or taking majority vote
  - cancelling effect of noise of individual opinions,
  - examples: politics, trial by jury (vs. trial by judge), sports (figure skating, gymnastics), Wikipedia, Quora, Stack Overflow, . . .

- Learning and aggregating multiple predictors

- Ensemble may be built using single or different types of predictors
Ensembling Approaches

◆ Bagging (Bootstrap AGGregatING):
  • sample different training sets from the original training set
  • train high variance low bias predictors based on these sets and average them
  • exploits independence between predictors

◆ Boosting:
  • sequentially train low variance high bias predictors
  • subsequent predictors learn to fix the mistakes of the previous ones
  • exploits dependence between learners
Stacking and Mixture of Experts

- Combine *base-learners* with *meta-learner*

[Diagram of Stacking and Mixture of Experts]

https://www.commonlounge.com/discussion/9331c0d004704e89bd4d1da08fd7c7bc
Prediction Problem: Expected Risk and Error Decomposition

Expected risk for data generated by $p(x, y)$:

$$R(h) = \mathbb{E}_{(x, y) \sim p}[\ell(y, h(x))]$$

- The best attainable (Bayes) risk is $R^* = \inf_{h \in \mathcal{Y} \times \mathcal{X}} R(h)$
- The best predictor in $\mathcal{H}$ is $h_{\mathcal{H}} \in \text{Argmin}_{h \in \mathcal{H}} R(h)$
- The predictor $h_m = A(T^m)$ learned from $T^m$ has risk $R(h_m)$

Excess error measures deviation of the learned predictor from the best one:

$$
\begin{align*}
\begin{pmatrix}
R(h_m) - R^*
\end{pmatrix}
&= 
\begin{pmatrix}
R(h_m) - R(h_{\mathcal{H}})
\end{pmatrix}
+ 
\begin{pmatrix}
R(h_{\mathcal{H}}) - R^*
\end{pmatrix}

\text{excess error} & \text{estimation error} & \text{approximation error}
\end{align*}
$$
Risk Averaged over Datasets

How will our predictor behave when sampling different training sets?

We can define the errors considering average over models constructed using all possible datasets $\mathcal{T}^m$, i.e., $\mathbb{E}_{\mathcal{T}^m}[R(h_m)]$

The errors can be redefined as:

$\left( \mathbb{E}_{\mathcal{T}^m}[R(h_m)] - R^* \right) = \left( \mathbb{E}_{\mathcal{T}^m}[R(h_m)] - R(h_H) \right) + \left( R(h_H) - R^* \right)$

excess error

estimation error

approximation error
Predictors Averaged over Datasets

- Let us also define a model averaged over all possible datasets:

\[ g_m(x) = \mathbb{E}_{\mathcal{T}^m} [h_m(x)] \]

- Unlike individual \( h_m \) models, \( g_m \) has an access to the whole \( p(x, y) \).

- Note: in general \( g_m \neq h_\mathcal{H} \) due to training algorithm \( A \) involved in \( h_m \).

- Also: \( g_m \) can’t be actually evaluated for infinite number of \( \mathcal{T}^m \) datasets.
Consider a regression problem with data generated as follows:

\[ y = h^*(x) + \epsilon \]

where \( \epsilon \) is noise: \( \mathbb{E}[\epsilon] = 0 \) and \( \text{Var}(\epsilon) = \sigma^2 \), e.g., \( \epsilon \sim \mathcal{N}(0, \sigma^2) \)

Use squared loss:

\[ \ell(y, h(x)) = (h(x) - y)^2 \]

The optimal predictor \( h^*(x) \) has a nonzero risk (for \( \sigma^2 > 0 \)):

\[ R^* = \mathbb{E}_{x,y} \left[ (h^*(x) - y)^2 \right] = \mathbb{E}_\epsilon \left[ \epsilon^2 \right] = \text{Var}(\epsilon) = \sigma^2 \]
Bias-Variance Decomposition for Regression 2

- The expected risk for $h_m$ can be decomposed:

$$
\mathbb{E}_{\mathcal{T}^m}[R(h_m)] = \mathbb{E}_{x,y,\mathcal{T}^m}[(h_m(x) - y)^2] = \ldots
$$

$$
= \mathbb{E}_{x,\mathcal{T}^m}[(h_m(x) - g_m(x))^2] + \\
\phantom{=} \underbrace{\mathbb{E}_x[(g_m(x) - h^*(x))^2]}_{\text{bias}^2} + \sigma^2_{\text{noise}}
$$

- The error splits into three terms
  - variance: difference of $h_m$ from the averaged predictor $g_m$,
  - bias$^2$: difference of the averaged predictor $g_m$ from the optimal one,
  - noise: irreducible determined by data
Excess Error vs. Bias and Variance

- The excess error is defined as:

$$\mathbb{E}_{\mathcal{T}^m} \left[ R(h_m) \right] - R^*$$

- As $R^* = \sigma^2$ we get:

$$\mathbb{E}_{\mathcal{T}^m} \left[ R(h_m) \right] - R^* = \mathbb{E}_x \left[ \left( g_m(x) - h^*(x) \right)^2 \right]$$

$$+ \mathbb{E}_{x, \mathcal{T}^m} \left[ \left( h_m(x) - g_m(x) \right)^2 \right]$$

- Compare
  - $\text{bias}^2$ vs. approximation error,
  - variance vs. estimation error
  - averaged model $g_m$ vs. best predictor $h_{\mathcal{H}}$
Derivation of the Bias-Variance Decomposition

\[ \mathbb{E}_{\mathcal{T}^m} \left[ R(h_m) \right] = \mathbb{E}_{x,y,\mathcal{T}^m} \left[ (h_m(x) - y)^2 \right] \]

\[ = \mathbb{E}_{x,y,\mathcal{T}^m} \left[ (h_m(x) - g_m(x) + g_m(x) - y)^2 \right] \]

\[ = \mathbb{E}_{x,y,\mathcal{T}^m} \left[ (h_m(x) - g_m(x))^2 + (g_m(x) - y)^2 \right. \]

\[ + 2 \left( h_m(x) - g_m(x) \right) \left( g_m(x) - y \right) \] \]

\[ = \mathbb{E}_{x,\mathcal{T}^m} \left[ (h_m(x) - g_m(x))^2 \right] + \mathbb{E}_{x,y} \left[ (g_m(x) - y)^2 \right] \]

\[ + \mathbb{E}_{x,y} \left[ 2 \left( \underbrace{\mathbb{E}_{\mathcal{T}^m} [ h_m(x) ] - g_m(x)}_{g_m(x)} \right) \left( g_m(x) - y \right) \right] \]
Derivation of the Bias-Variance Decomposition 2

We get:

\[
\mathbb{E}_{\mathcal{T}^m} \left[ R(h_m) \right] = \mathbb{E}_{x,\mathcal{T}^m} \left[ \left( h_m(x) - g_m(x) \right)^2 \right] + \mathbb{E}_{x,y} \left[ \left( g_m(x) - y \right)^2 \right]
\]

Note that the second term does not depend on \( \mathcal{T}^m \).
Let us continue with the second term:

\[
\mathbb{E}_{x,y} \left[ (g_m(x) - y)^2 \right] = \mathbb{E}_{x,\epsilon} \left[ (g_m(x) - h^*(x) - \epsilon)^2 \right] \\
= \mathbb{E}_{x,\epsilon} \left[ (g_m(x) - h^*(x))^2 + \epsilon^2 - 2\epsilon (g_m(x) - h^*(x)) \right] \\
= \mathbb{E}_x \left[ (g_m(x) - h^*(x))^2 \right] + \mathbb{E}_\epsilon \left[ \epsilon^2 \right] \\
- 2\mathbb{E}_{x,\epsilon} \left[ \epsilon (g_m(x) - h^*(x)) \right] = 0 \\
= \mathbb{E}_x \left[ (g_m(x) - h^*(x))^2 \right] + \sigma^2
\]
Pointwise Bias-Variance

We can express the bias and variance as function of $x$ by not integrating over in expected values

$$
\mathbb{E}_{y|x, \mathcal{T}^m} \left[ \ell(y, h_m(x)) \right] = \mathbb{E}_{y|x, \mathcal{T}^m} \left[ (h_m(x) - y)^2 \right]
$$

$$
= \text{Var}_{\mathcal{T}^m} \left( h_m(x) \right) + \text{variance}(x)
$$

$$
+ \left( g_m(x) - h^*(x) \right)^2 + \sigma(x)^2
$$

bias(x)^2

noise
Bias-Variance: Example

◆ Polynomial regression with a varying degree of polynomial

Degree = 1

Degree = 5

Degree = 15

Figure 4.2: Bias-variance decomposition of the expected generalization error for polynomials of degree 1, 5, and 15.

Also, because of low complexity, none of them really fits the trend of the training points, even approximately, which implies that the average model is far from approximating the Bayes model. This results in high bias. On the other hand, polynomials of degree 15 (right) suffer from overfitting. In terms of bias and variance, the situation is the opposite. Predictions have low bias but high variance, as shown in the lower right plot of Figure 4.2. The variability of the predictions is large because the high degree of the polynomials (i.e., the high model complexity) captures noise in the learning set. Indeed, compare the gray line with the blue dots – they almost all intersect. Put otherwise, small changes in the learning set result in large changes in the obtained model and therefore in its predictions. By contrast, the average model is now quite close from the Bayes model, which results in low bias. Finally, polynomials of degree 5 (middle) are neither too simple nor too complex. In terms of bias and variance, the trade-off is well-balanced between the two extreme situations. Bias and variance are neither too low nor too large.

4.1.2 Classification

In direct analogy with the bias-variance decomposition for the squared error loss, similar decompositions have been proposed in the literature for the expected generalization error based on the zero-one loss, i.e., for $E_{Y \mid X = x} \{1 - L(x) \mid Y = L(x) \} = P_{L, Y \mid X = x}(Y = L(x))$. Most no-

1 Note however the Gibbs-like phenomenon resulting in both high variance and high bias at the boundaries of $X$. 
Decision/Regression Trees

- Nodes at the same level correspond to mutually exclusive subsets of the original training data as well as mutually exclusive subsets of the input space $\mathcal{X}$
- Inner node further splits its subset

Hastie et al.: *The Elements of Statistical Learning*, 2009
Decision/Regression Trees (contd.)

- Training set: $\mathcal{T}^m = \{(x_i, y_i) \mid i = 1, \ldots, m\}$, $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$

- Input space split into regions defined in leaves: $R_r$, $r \in \{1, \ldots, M\}$

- We can model region responses by constants $c_r$, $r \in \{1, \ldots, M\}$ but other possibilities, e.g., linear regression are possible

- Prediction:
  \[
  h(x) = \sum_{r=1}^{M} c_r [x \in R_r]
  \]

- For sum of squares loss function $\sum_{i=1}^{m}(y_i - h(x_i))^2$ we set the responses to be the averages over regions:
  \[
  \hat{c}_r = \frac{1}{|S_r|} \sum_{(x_i, y_i) \in S_r} y_i \quad \text{(see seminar)}
  \]

where $S_r = \{(x_i, y_i) : (x_i, y_i) \in \mathcal{T}^m \land x_i \in R_r\}$
Greedy Learning of Decision/Regression Trees

- How many distinct decision trees with \( p \) Boolean attributes for binary classification?
  - at least as many as boolean functions of \( p \) attributes
  - \( = \) number of distinct truth tables with \( 2^p \) rows: \( 2^{2^p} \)
  - For 6 Boolean attributes at least
    \( 18,446,744,073,709,551,616 \) trees!

- Learning is NP-complete: [Hyafil and Rivest 1976]

- We need heuristics \( \Rightarrow \) greedy approach

- Recursively choose the "most important" attribute to find a small tree consistent with the training data

- Split points:
  - **nominal attribute**: try all possibilities
  - **ordinal/continuous attribute**: try attribute values based on all training data samples or their subset
Regression Trees: Which Attribute to Split?

- The "most important" attribute for regression trees would be the one, for which the split reduces the loss (sum of squared errors) by the greatest amount.

- We have:

\[
h(x) = \sum_{r=1}^{M} c_r [x \in R_r]
\]

- Consider splitting attribute \( j \) and split point \( s \), we split an original region \( R \) into a pair of half-planes for an ordinal (e.g., continuous) attribute:

\[
R_L(j, s) = \{x | x \in R \land x_j \leq s\} \quad \text{and} \quad R_R(j, s) = \{x | x \in R \land x_j > s\}
\]

Similarly for a nominal attribute:

\[
R_L(j, s) = \{x | x \in R \land x_j = s\} \quad \text{and} \quad R_R(j, s) = \{x | x \in R \land x_j \neq s\}
\]

- Denote the corresponding subsets of \( T^m \) as \( S_L \) and \( S_R \).
Regression Trees: Which Attribute to Split? (contd.)

❖ We seek for an attribute $j$ and a split point $s$ which minimize the total impurity (=loss, risk):

$$
\min_{c_L} \sum_{(x_i, y_i) \in S_L(j, s)} (y_i - c_L)^2 + \min_{c_R} \sum_{(x_i, y_i) \in S_R(j, s)} (y_i - c_R)^2
$$

for $(x_i, y_i) \in S$ and $S = S_L \cup S_R$

❖ Inner minimizations (region response values) are solved by averaging tree outputs per region:

$$
\hat{c}_L = \frac{1}{|S_L(j, s)|} \sum_{(x_i, y_i) \in S_L(j, s)} y_i \quad \text{and} \quad \hat{c}_R = \frac{1}{|S_R(j, s)|} \sum_{(x_i, y_i) \in S_R(j, s)} y_i
$$

❖ Root node: $S = T^m$
Tree Learning Algorithm

BUILD-TREE($S$)

1. $i = \text{IMPURITY}(S)$ // e.g., the squared loss
2. $\hat{i}, \hat{j}, \hat{s}, \hat{S}_L, \hat{S}_R = 0, 0, \emptyset, \emptyset$ // current best kept in these
3. for $j \in \{1, \ldots, p\}$ // iterate over attributes
4.     for $s \in \text{SPLIT-POINTS}(S, j)$ // iterate over all split points
5.         $S_L, S_R = \text{SPLIT}(S, j, s)$
6.         $i_L = \text{IMPURITY}(S_L)$
7.         $i_R = \text{IMPURITY}(S_R)$
8.     if $i_L + i_R < \hat{i}$ and $|S_L| > 0$ and $|S_R| > 0$
9.         $\hat{i}, \hat{j}, \hat{s}, \hat{S}_L, \hat{S}_R = (i_L + i_R), j, s, S_L, S_R$
10.    if $\hat{i} < i$
11.        $N_L = \text{BUILD-TREE}(\hat{S}_L)$
12.        $N_R = \text{BUILD-TREE}(\hat{S}_R)$
13.    return DECISION-NODE($\hat{j}, \hat{s}, N_L, N_R$)
14.    else return LEAF-NODE($S$)
Bias and Variance of Decision Trees

- Small changes of training data lead to big differences in final trees
- Decision trees grown deep enough have typically:
  - low bias
  - high variance
  ⇒ overfitting
- Idea: *average multiple models* to reduce variance while (happily) not increasing bias much
Averaging Models

- Define *bagging model* $b$ as an average of $K$ component models:

$$b(x) = \frac{1}{K} \sum_{i=1}^{K} h_m^{(i)}(x)$$

trained using a set of i.i.d. datasets of size $m$: $\mathcal{D}_m = \{\mathcal{T}_1^m, \ldots, \mathcal{T}_K^m\}$ so $h_m^{(1)}(x)$ is trained using $\mathcal{T}_1^m$, $h_m^{(2)}(x)$ using $\mathcal{T}_2^m$, etc.

- Note that $b(x)$ approximates the *averaging model*:

$$g_m(x) = \mathbb{E}_{\mathcal{T}_m}[h_m(x)]$$

- We can define the *averaging model* for $b(x)$ as well:

$$g^B_m(x) = \mathbb{E}_{\mathcal{D}_m}[b(x)]$$
Averaging Models: Bias

- Bias remains unchanged for the bagging model compared to any of the component models:

\[
\text{bias}(x)^2 = \left( g_m^B(x) - h^*(x) \right)^2
\]

\[
= \left( \mathbb{E}_{D_m} b(x) - h^*(x) \right)^2
\]

\[
= \left( \mathbb{E}_{D_m} \left[ \frac{1}{K} \sum_{i=1}^{K} h_m(i)(x) \right] - h^*(x) \right)^2
\]

\[
= \left( \frac{1}{K} \sum_{i=1}^{K} \mathbb{E}_{T_i} h_m(i)(x) - h^*(x) \right)^2
\]

\[
= \left( \mathbb{E}_{T_m} h_m(x) - h^*(x) \right)^2 = \left( g_m(x) - h^*(x) \right)^2
\]
Averaging Models: Variance

- For uncorrelated component models $h_m^i(x)$:

\[
\text{Var}_{\mathcal{D}^m}(b(x)) = \text{Var}_{\mathcal{D}^m}\left(\frac{1}{K} \sum_{i=1}^{K} h_m^i(x)\right) = \frac{1}{K^2} \sum_{i=1}^{K} \text{Var}_{\mathcal{T}^m}(h_m^i(x)) = \frac{1}{K} \text{Var}_{\mathcal{T}^m}(h_m(x))
\]

which is a great improvement based on the very strong assumption.

- There is no improvement for maximum correlation, i.e., for all component models equal: $h_m^i(x) = h_m(x)$ for $i = 1, \ldots, K$, we get:

\[
\text{Var}_{\mathcal{D}^m}(b(x)) = \text{Var}_{\mathcal{D}^m}\left(\frac{1}{K} \sum_{i=1}^{K} h_m^i(x)\right) = \text{Var}_{\mathcal{T}^m}(h_m(x))
\]

⇒ we need to train uncorrelated (diverse) component models while keeping their bias reasonably low.
Bootstrapping

- In practice we have only a single training dataset $\mathcal{T}^m$.
- Bootstrapping is a method producing datasets $\mathcal{T}^m_i$ for $i = 1, \ldots, K$ by sampling $\mathcal{T}^m$ uniformly with replacement.
- Bootstrap datasets have the same size as the original dataset $|\mathcal{T}^m_i| = |\mathcal{T}^m|$.
- $\mathcal{T}^m_i$ is expected to have the fraction $1 - \frac{1}{e} \approx 63.2\%$ of unique samples from $\mathcal{T}^m$, others are duplicates (see seminar).
Bagging

- Bagging = Bootstrap AGGregating [Breiman 1994]:
  1. Use bootstrapping to generate $K$ datasets
  2. Train a model $h_m^{(i)}(x)$ on each dataset $T_i^m$
  3. Average the models getting the bagging model $b(x)$

- When decision trees are used as the models $\Rightarrow$ random forests

- Low bias is achieved by growing the trees to maximal depth

- Trees are decorrelated by:
  - training each tree on a different bootstrap dataset
  - randomization of split attribute selection
Random Forest Algorithm

1. For $i = 1 \ldots K$:

   (a) draw a bootstrap dataset $T_i^m$ from $T^m$, $|T_i^m| = |T^m| = m$

   (b) grow a tree $h_{m}^{(i)}$ using $T_i^m$ by recursively repeating the following, until the minimum node size $n_{\text{min}}$ is reached:
   
   i. select $k$ attributes at random from the $p$ attributes
   ii. pick the best attribute and split-point among the $k$
   iii. split the node into two daughter nodes

2. Output ensemble of trees $b(x)$ averaging $h_{m}^{(i)}(x)$ (regression) or selecting a majority vote (classification)

   ◆ Node size $n_{\text{min}}$ is the number of the training dataset samples associated with the node, limits tree depth
Out-of-Bag (OOB) Error

- Cheap way of generalization error assessment for bagging
- Bagging produces bootstrapped sets $\mathcal{T}_1^m, \mathcal{T}_2^m, \ldots, \mathcal{T}_K^m$
- For each $\mathbf{(x}_i, y_i) \in \mathcal{T}_i^m$ select only trees which were not trained on this sample: $H_i = \{ h_m^{(j)} | (\mathbf{x}_i, y_i) \notin \mathcal{T}_j^m \}$
- Average only the OOB trees in $H_i$ when evaluating error for $(\mathbf{x}_i, y_i)$
- Replacement for K-fold cross-validation
Feature Importance

- Random forests allow easy evaluation of feature importances

- Mean Decrease Impurity (MDI):
  - set $f_j = 0$ for all attributes $j = 1, \ldots, p$
  - traverse all trees processing all internal nodes
  - for each node having a split attribute $j$ add its impurity decrease multiplied by the proportion of the node size to $f_j$

- Mean Decrease Accuracy (MDA), permutation importance:
  - evaluate the forest using OOB
  - do the same with permuted values of an attribute $j$
  - watch decrease in accuracy: low decrease means unimportant feature
Random Forest Summary

- Easy to use method: robust w.r.t. parameter settings ($K$, node size)
- While *statistical consistency* is proven for decision trees (both regression and classification) we have only proofs for simplified versions of random forests [Breiman, 1984]
- Related methods: boosted trees
Boosting

- Sequentially train weak learners/predictors *low variance high bias*
- Subsequent predictors fix the mistakes of the previous ones reducing bias
- Methods discussed here:
  - Forward Stagewise Additive Modeling
  - Gradient Boosting Machine
  - Gradient Boosted Trees
  - AdaBoost
Forward Stagewise Additive Modeling (FSAM)

1. Initialize $f_0(x) = 0$

2. For $k = 1$ to $K$:
   
   (a) Find
   
   $$(\beta_k, \theta_k) = \arg\min_{\beta, \theta} \sum_{i=1}^{m} \ell(y_i, f_{k-1}(x_i) + \beta b(x_i; \theta))$$

   where $b(x_i; \theta_k)$ is the basis function and $\beta_k$ the corresponding coefficient

   (b) Set $f_k(x) = f_{k-1}(x) + \beta_k b(x; \theta_k)$

3. Return $h_m(x) = f_K(x)$
FSAM and Gradient Descent

- FSAM update looks very similar to the gradient descent one:

\[ f_k(x) = f_{k-1}(x) + \beta_k b(x; \theta_k) \]

- Just think of
  - \( \beta_k \approx \) step size (learning rate)
  - \( b(x; \theta_k) \approx \) the negative of gradient
FSAM for Squared Loss

- Once again, consider regression with the squared loss:

\[ \ell(y, f(x)) = (y - f(x))^2 \]

- For FSAM we get:

\[ \ell(y_i, f_k(x_i)) = \ell(y_i, f_{k-1}(x_i) + \beta_k b(x_i; \theta_k)) \]

\[ = (y_i - f_{k-1}(x_i) - \beta_k b(x_i; \theta_k))^2 \]

\[ = (r_{ik} - \beta_k b(x_i; \theta_k))^2 \]

where \( r_{ik} = y_i - f_{k-1}(x_i) \) is the residual of the current model for the \( i \)-th sample

- The task of FSAM is to fit the model \( \beta_k b(x_i; \theta_k) \) to match the residuals

- The method is sometimes called the least-squares boosting
Gradient Boosting for Regression

- In case of regression with squared loss we minimize:

  \[ L = \sum_{i=1}^{m} \ell(y_i, f(x_i)) = \sum_{i=1}^{m} \frac{1}{2}(y_i - f(x_i))^2, \]

- We can treat \( f(x_1), f(x_2), \ldots, f(x_m) \) as parameters and take the derivatives:

  \[
  \frac{\partial L}{\partial f(x_i)} = \frac{\partial \left( \sum_{j=1}^{m} \ell(y_j, f(x_j)) \right)}{\partial f(x_i)} = \frac{\partial \ell(y_i, f(x_i))}{\partial f(x_i)}
  \]

  \[ = f(x_i) - y_i = -r_i \]

- The least-squares boosting hence takes steps in the negative gradient direction where \( r_i = -\frac{\partial L}{\partial f(x_i)} \)

- This approach can be generalized for any differentiable loss function!
Gradient Boosting Machine

1. Initialize \( f_0(x) = 0 \) or \( f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{m} \ell(y_i, \gamma) \)

2. For \( k = 1 \) to \( K \):
   (a) Compute:
   \[
g_k = \left[ \frac{\partial \ell(y_i, f_{k-1}(x_i))}{\partial f_{k-1}(x_i)} \right]_{i=1}^{m}
   \]
   (b) Fit a regression model \( b(\cdot; \theta) \) to \(-g_k\) using squared loss:
   \[
   \theta_k = \arg\min_{\theta} \sum_{i=1}^{m} [(-g_k)_i - b(x_i; \theta)]^2
   \]
   (c) Choose a fixed step size \( \beta_k = \beta > 0 \) or use line search:
   \[
   \beta_k = \arg\min_{\beta > 0} \sum_{i=1}^{m} \ell(y_i, f_{k-1}(x_i) + \beta b(x_i; \theta_k))
   \]
   (d) Set \( f_k(x) = f_{k-1}(x) + \beta_k b(x; \theta_k) \)

3. Return \( h_m(x) = f_K(x) \)
Multinominal Classification: Gradient Boosting Machine

- Training examples: $\mathcal{T}^m = \{(x_i, y_i) \in (\mathcal{X} \times \mathcal{Y}) \mid i = 1, \ldots, m\}$, where $\mathcal{Y} = \{1, \ldots, C\}$
- Train one GBM for each of $C$ target classes:
  $$f(x_i) \triangleq [f^c(x_i)]_{c=1}^C$$
- Use softmax to get the probability estimates: $p_{ic} \triangleq \sigma_c(f(x_i))$
- Use multinominal cross-entropy as the loss:
  $$\mathcal{L} = -\sum_{c=1}^{K} y_{ic} \log(p_{ic}),$$
  where $y_{ic} \triangleq [y_i = c]$ holds one-hot encoded target classes
- We then have the following residuals:
  $$\frac{\partial \mathcal{L}}{\partial f^c(x_i)} = p_{ic} - y_{ic}$$
Gradient Boosted Trees

- Gradient Boosting Tree is GBM where all weak learners are decision or regression trees
- Use limit on depth/number of leaves/node size for the weak learners ⇒ high bias
- Often single-level tree: decision stump
- Meta-parameters such as $K$ (number of trees) and $\beta$ (learning rate) have to be found using cross validation
- Model is built sequentially (unlike random forests)
- Highly optimized algorithms based on Gradient Boosting Trees:
  - XGBoost, LightGBM
  - parallelization, scalability, regularization
GBM Example (XGBoost)

- Each $T^m$ is 1000 samples of Rosenbrock function, fixed $\sigma$
- 100 models for each setting
- Learning rate experiment: $K = 100$

Comparison to Random Forest
AdaBoost M1

Binary classifier: $\mathcal{Y} = \{-1, 1\}$

1. Initialize the weights $w_i = 1/m$ for $i = 1, 2, \ldots m$

2. For $k = 1$ to $K$:
   (a) Fit a classifier $G_k(x; \theta_k) \in \{-1, 1\}$ to the training data using loss weighted by $w_i$:

   $$
   \theta_k = \arg\min_{\theta} \sum_{i=1}^{m} w_i [y_i \neq G_k(x_i; \theta)]
   $$

   (b) Compute the weighted error rate

   $$
   \epsilon_k = \frac{\sum_{i=1}^{m} w_i [y_i \neq G_k(x_i; \theta_k)]}{\sum_{i=1}^{m} w_i}
   $$

   (c) Compute the scaling coefficient $\alpha_k = \log((1 - \epsilon_k)/\epsilon_k)$

   (d) Set $w_i \leftarrow w_i \cdot \exp(\alpha_k \cdot [y_i \neq G_k(x_i; \theta_k)])$ for $i = 1, 2, \ldots m$

3. Return $h_m(x) = \text{sign} \left[ \sum_{k=1}^{K} \alpha_k G_k(x; \theta_k) \right]$
AdaBoost is FSAM: the Loss

Claim: AdaBoost is FSAM using the exponential loss

\[ \ell(y, f(x)) = \exp(-yf(x)) \]

For individual classifiers \( G_m(x_i; \theta) \) as basis functions, we get:

\[
(\beta_k, \theta_k) = \arg\min_{\beta, \theta} \sum_{i=1}^m \ell\left(y_i, f_{k-1}(x_i) + \beta G(x_i; \theta)\right)
\]

\[
= \arg\min_{\beta, \theta} \sum_{i=1}^m \exp\left(-y_i\left(f_{k-1}(x_i) + \beta G(x_i; \theta)\right)\right)
\]

\[
= \arg\min_{\beta, \theta} \sum_{i=1}^m w_i^{(k)} \exp\left(-y_i\beta G(x_i; \theta)\right),
\]

where \( w_i^{(k)} \triangleq \exp(-y_if_{k-1}(x_i)) \) does not depend neither on \( \beta \) nor on \( \theta \)
AdaBoost is FSAM II: Fitting the Classifier

We can rearrange further:

\[
(\beta_k, \theta_k) = \arg\min_{\beta, \theta} \sum_{i=1}^{m} w_i^{(k)} \exp \left( -y_i \beta G(x_i; \theta) \right)
\]

\[
= \arg\min_{\beta, \theta} \left[ e^{-\beta} \sum_{y_i = b(x_i; \theta)} w_i^{(k)} + e^\beta \sum_{y_i \neq b(x_i; \theta)} w_i^{(k)} \right]
\]

\[
= \arg\min_{\beta, \theta} \left[ e^{-\beta} \sum_{i=1}^{m} w_i^{(k)} + \left( e^\beta - e^{-\beta} \right) \sum_{i=1}^{m} w_i^{(k)} [y_i \neq G(x_i; \theta)] \right]
\]

For any \( \beta > 0 \) we can minimize \( \theta \) separately:

\[
\theta_k = \arg\min_{\theta} \sum_{i=1}^{m} w_i^{(k)} [y_i \neq G(x_i; \theta)] \quad \text{(same as AdaBoost 2(a))}
\]
AdaBoost is FSAM III: the Weighted Error $\epsilon_k$ and the Scaling Coefficient $\alpha_k$

Let's minimize

$$(e^\beta - e^{-\beta}) \sum_{i=1}^{m} w^{(k)}_i [y_i \neq G(x_i; \theta_k)] + e^{-\beta} \sum_{i=1}^{m} w^{(k)}_i$$

with respect to $\beta$

$$(e^{\beta_k} + e^{-\beta_k}) \sum_{i=1}^{m} w^{(k)}_i [y_i \neq G(x_i; \theta_k)] - e^{-\beta_k} \sum_{i=1}^{m} w^{(k)}_i = 0$$

$$(e^{\beta_k} + e^{-\beta_k}) \epsilon_k - e^{-\beta_k} = 0$$

where $\epsilon_k = \frac{\sum_{i=1}^{m} w_i [y_i \neq G(x_i; \theta_k)]}{\sum_{i=1}^{m} w_i}$ as in AdaBoost 2(b)

Solving for $\beta_k$:

$$\beta_k = \frac{1}{2} \log \frac{1 - \epsilon_k}{\epsilon_k}$$

Define $\alpha_k \triangleq 2\beta_k$ and compare to AdaBoost 2(c)
AdaBoost is FSAM IV: the Weight Update

- We have $w_i^{(k)} = e^{-y_if_{k-1}(x_i)}$ and $f_k(x) = f_{k-1}(x) + \beta_k G(x; \theta_k)$ so:

$$w_i^{(k+1)} = e^{-y_i(f_{k-1}(x_i) + \beta_k G(x_i; \theta_k))} = w_i^{(k)} \cdot e^{-y_i \beta_k G(x_i; \theta_k)}$$

- Finally $-y_i G(x_i; \theta_k) = 2 \cdot [y_i \neq G(x_i; \theta_k)] - 1$ gives the weight update:

$$w_i^{(k+1)} = w_i^{(k)} \cdot e^{\alpha_k [y_i \neq G(x_i; \theta_k)]} \cdot e^{-\beta_k}$$

corresponding to AdaBoost 2(d) up to the factor $e^{-\beta_k}$ which is same for all weights and hence has no effect.
Figure 4.2: Bias-variance decomposition of the expected generalization error for polynomials of degree 1, 5, and 15.

Also, because of low complexity, none of them really fits the trend of the training points, even approximately, which implies that the average model is far from approximating the Bayes model. This results in high bias. On the other hand, polynomials of degree 15 (right) suffer from overfitting. In terms of bias and variance, the situation is the opposite. Predictions have low bias but high variance, as shown in the lower right plot of Figure 4.2. The variability of the predictions is large because the high degree of the polynomials (i.e., the high model complexity) captures noise in the learning set. Indeed, compare the gray line with the blue dots – they almost all intersect. Put otherwise, small changes in the learning set result in large changes in the obtained model and therefore in its predictions. By contrast, the average model is now quite close from the Bayes model, which results in low bias. Finally, polynomials of degree 5 (middle) are neither too simple nor too complex. In terms of bias and variance, the trade-off is well-balanced between the two extreme situations. Bias and variance are neither too low nor too large.

4.1.2 Classification

In direct analogy with the bias-variance decomposition for the squared error loss, similar decompositions have been proposed in the literature for the expected generalization error based on the zero-one loss, i.e., for $\mathbb{E}_{L}\{\mathbb{E}_{Y|X=x}\{1-(L(x))^6=Y\}\} = \mathbb{P}_{L,Y|X=x}(L(x))^6=Y)$. Most note however the Gibbs-like phenomenon resulting in both high variance and high bias at the boundaries of $X$. 

1
FIGURE 9.2. Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel.
FIGURE 9.2. Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows how the corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel.