Week 6: # Chapter 6: Ensemble Methods

- What are "ensemble methods"?
  - get predictions from multiple models
  - aggregate the predictions
- Why use "ensemble methods"?

expected test error = irreducible error + bias^2 + variance

- flexibility of model ↑ bias ↓
- flexibility of model ↑ variance ↑
- Ensemble methods help to reduce bias or variance

There are two main advantages of using ensemble methods.

1. Ensemble methods help to reduce bias or variance.
2. There are a number of models you can use
   - but at the beginning you do not know which ones are good or bad.
   - so you use all of them & aggregate their predictions.
   - on the fly you observe which models are better.

- Ensemble methods
  1. Bagging and Bootstrapping
  2. Random Forest
  3. Boosting

- Bagging & Bootstrapping
  - Given a large dataset, sample a number of (smaller) sub-datasets.
  - Use each sub-dataset to train for a prediction function.
  - Aggregate the predictions via
    - averaging - (for regression) or
    - majority vote (for classification

- Bagging helps reducing variance, while bias is not much affected.
- Out-of-bag (OOB) Prediction and MSE

1. Bagging and Bootstrapping

- Ideal Bagging
  - Ideally, we have B separate training datasets.
  - We use each dataset to train for a prediction function.

Dataset 1 —training→ $\hat{f}_1$

Dataset 2 —training→ $\hat{f}_2$

Dataset 3 —training→ $\hat{f}_3$

- The prediction functions are $\hat{f}^1, \hat{f}^2, \ldots, \hat{f}^B$

To aggregate and make the final prediction,

- For regression problems, we take average:

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x)$$

- For classification problems, we do majority vote. (e.g. when labels are either True or False),

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x)$$

- When each prediction function $\hat{f}^b$ is generated
- by a method with high variance (e.g. decision tree)
- the averaging helps reducing variance.
- Bias is not affected

- Analogue.
  - Given B independent random variables $Y_1, Y_2, \ldots, Y_B$.
  - each with variance σ^2,
  - the variance of the mean is σ^2/B

$$\bar{Y} := \frac{1}{B}\sum_{b=1}^{B} Y_b$$

- Ideally, we have B separate training datasets.
  - But when we says "ideal", usually it does not work in practice.

What's the problem?

- Problem: In reality, data is expensive, so we usually do not have access to multiple training datasets.

Solution: Given a (large) training dataset, we sample a number of sub-datasets.
- How? Bootstrapping

- Bootstrapping

1. • Given: A dataset $\mathcal{D}$ with n observations (datapoints).
2. • Bootstrapping:
   - Generate B different bootstrapped training sets (BTS) independently.
   - To generate each BTS, we sample n observations with replacement.
   - Then train each BTS independently with the algorithm you choose,
   - obtain one prediction function per BTS

3. Sampling n observations with replacement:
- There are n balls in a bag, with numbers $1, \ldots, n$ on them.
- Repeat the following n times:
  - draw a ball uniformly at random,
  - then put the ball back to the bag
  - Thus, a ball can be drawn more than once in the whole process.
  - Also, some balls may not be drawn in the whole process at all.

What is "sampling n observations with replacement?
sample(n, replace = TRUE)

Example:

- Bootstrapped Training:

Dataset 1 — Training → $\hat{f}^1$
Dataset 2 — training → $\hat{f}^2$
Dataset 3 — training → $\hat{f}^3$

Sampling without Replacement

What is "sampling m observations without replacement?

sample (n, m, replace = FALSE)

Example:

How to choose the number B?
- Choose a large enough B such that
  - The test MSE or
  - Test error rate stabilizes.

Next, we discuss a method to estimate
- test MSE or test error rate.

Out-of-Bag Error Estimation

Recall that: in bootstrapping
- Given a dataset with n observations,
- we sample n observations with replacement.

In each BTS, you expect:
- Some observations are sampled more than once.
- Some other observations are not sampled.

The observations which are not sampled are called out-of-bag (OOB) observations.

Example: of OOB Observations.
- When the number of BTS is large, for each observation, you expect:
  - It is sampled in some BTS
  - It is not sampled in other BTS.

Idea: To estimate MSE or error rate, for each observation,
- when it is sampled in a BTS, it is considered as training observation;
- when it is not sampled in a BTS, it is considered as a testing observation.

BTS = bootstrapped training sets.

- For any observation, its OOB prediction is:
  - average of the predictions from those BTS
  - which do not contain the observation.

The OOB MSE is

$$\frac{1}{n}\sum_{j=1}^{n}(\text{label of observation } j - \text{OOB prediction of observation } j)^2$$

- We have a dataset with 8 observations in the form of (x, y):

$D = \{(1,4), (3,3), (4,8), (6,11), (10,16), (11,11), (13,14), (18,24)\}$.

* What is the OOB prediction for $X = 13$?

An issue: we cannot make OOB prediction for $X = 13$,
- because the observation (13, 14) belongs to every BTS.
- This issue rarely happens when n, B are both large.

* What is the OOB MSE?

Summary of Bagging, Bootstrapping & OOB

- You have learnt/seen
  - How to sample a bootstrapped training set (BTS)
    - "sample n observations with replacement"
  - bagging reduces variance, while bias is not really affected.
  - How to determine the number of BTS we need to sample
    - "When the test MSE or test error rate stabilizes"
  - what are out-of-bag (OOB) predictions & OOB MSE, and several concrete examples on how to compute them

Theory of Bootstrapping + OOB

Some Motivating Questions

- In bootstrapping, each BTS is generated by sampling n observations with replacement.

Question 1: Why we sample n observations, not $\frac{n}{2}$ or $2n$?

- In discussions about OOB,
  - we said that when n, B are large,
  - then we can always make OOB prediction for each observation.

Question 2: Why is this claim true?

Plausibility of OOB prediction

- Lemma:

- The probability that an observation is not sampled in a BTS is $(1-\frac{1}{n})^n$.

- when n is large (say $n \geq 100$), the probability is approximately $\frac{1}{e} = 36.8\%$, where, $e = 2.71828\ldots$

- Reason.

- In a BTS we sample n observations with replacement.
- For $j = 1, 2, \ldots, n$ the probability that a particular observation is not chosen at the j^{th} sample is $1 - \frac{1}{n}$.

- Thus, the probability that the particular observation is not chosen is all the n sample is $(1-\frac{1}{n})^n$ which, when n is large, approximately $\frac{1}{e}$.

Corollary 1:
- When B BTS are generated, for any observation,
- there are roughly (B1e) BTS
- which do not sample this observation in expectation.
- In other words, when B is large, it is very likely that we can make OOB prediction for every observation.

Corollary 2:
- In each BTS, there are roughly $n(1-\frac{1}{e}) \approx n.63.2\%$
- of distinct observations being sampled in expectation,
- In other words, each BTS represents more than 60% of the full dataset.

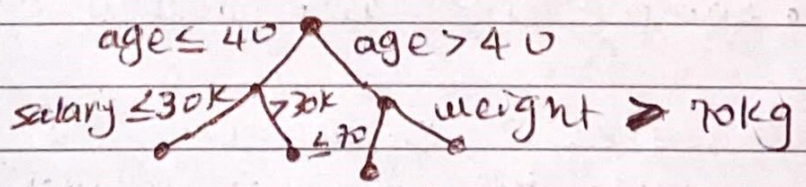
⟹ In sum, we sample n observations with replacement
- to generate each BTS to ensure that
  - it is very likely that we can make OOB prediction for each observation (Corollary 1)
  - and each BTS well-represents the whole dataset (Corollary 2).

Random Forests

- An ensemble method specifically for decision trees.
- Sometimes also called "attribute bagging" or "feature bagging"
- reduce overfitting

Recall: Decision Trees

- When using recursive binary splitting to generate a decision tree,
  - at each node we select the optimal attribute & split value.
    - For regression problem
      - we minimize Residue Sum of Squares (RSS)
    - For classification problem
      - we usually minimize Information Value.



Bagging in Decision Trees

Now, we apply bagging method to generate B decision trees.

- Suppose there are many attributes (age, salary, weight, etc)
- but only a few of them are decisive in the training set (which are called "strong predictors").
- However, these strong predictors may not be strong for test sets.

- Then only these strong predictors will be used in the decision trees,
  i.e. the decision trees are highly correlated.
- This severely inhibits the power of reducing variance by bagging.

Recall:

- Given B independent random variables $Y_1, Y_2, \ldots, Y_B$
- each with variance σ^2, the variance of the mean

$$\bar{Y} := \left(\frac{1}{B}\sum_{b=1}^{B} Y_b\right) \text{ is } \frac{\sigma^2}{B}$$

Random Forests

- To avoid generating decision trees
  - which are highly correlated,
  - instead of considering all attributes,
- now at each node we only consider a small sample of attributes.

- Rules of thumb: if there are p attributes.
  - For classification problem,
    - sample $\sqrt{p}$ attributes at each node;
  - For regression problem
    - sample $p/3$ attributes at each node.

How? Recall sample(p, ceiling(sqrt(p)), replace=FALSE))

Summary of Random Forests

- Random forest method is similar to recursive binary splitting, except (for classification problems):

1. Instead of generating only one decision tree,
   - now you generate B decision trees,
   - while each decision tree is generated using one BTS;
2. Instead of considering all attributes at each node,
   - now you consider only $\sqrt{p}$ random attributes at each node.

3. The prediction is made by
   - a majority vote among the predictions from the B decision trees;

4. You can estimate the error rate
   - via OOB

High-level Pseudocode for Random Forests

1. Load dataset, which has p attributes
2. Factor the categorical/qualitative variables.
3. Specify B, the number of decision trees to be generated.
4. Create a data structure/class for storing a decision tree
5. Create an array of size B to store the decision trees.
6. For i = 1 to B do:
   - Generate a BTS out of the original dataset.
   - Use recursive binary splitting on the BTS
     - to generate a decision tree;
     - but at each node you consider only $\sqrt{p}$ random attributes (round up if $\sqrt{p}$ is not an integer)
   - Save the decision tree to the array created in step 4.
7. After generating the B decision trees,
   - compute the OOB error rate.

Boosting

Ideas of Boosting

- Conceptually, boosting is simple
  - The idea is to make the prediction by

$$\hat{f}(x) = \sum_{b=1}^{B} \eta \cdot \hat{f}^b(x)$$

where

- each $\hat{f}^b$ is a prediction function generated by by a boosting algorithm,
- η is the learning rate

One way to think about it is:

- $\eta \cdot \hat{f}^1$ is the main term,
- $\eta \cdot \hat{f}^2$ is a small residue to $\eta \cdot \hat{f}^1$
- $\eta \cdot \hat{f}^3$ is a small residue to $\eta \cdot \hat{f}^2$, ...

Boosting Algorithm

Input: A dataset, with observation $([x_{i1}, x_{i2}], y_i)$ for $i = 1, 2, \ldots, n$.

Algorithm:

1. Choose a learning rate $\eta > 0$
   e.g. $\eta = 0.01, 0.05, 0.1, 1$
   - The smaller the η, the slower boosting learns.

2. Set $r_i = y_i$ for $i = 1, 2, \ldots, n$ (r_i is called a residue)
3. For $b = 1, 2, \ldots B$ do:
   * Require that B is much larger than $1/\eta$.
   - Train for the prediction function $\hat{f}^b$ using observations $([x_{11}, x_{12}], r_1), ([x_{21}, x_{22}], r_2), \ldots, ([x_{n1}, x_{n2}], r_n)$
   - For $i = 1, 2, \ldots, n$ update $r_i \leftarrow r_i - \eta \cdot \hat{f}^b([x_{i1}, x_{i2}])$
4. Output the boosted prediction function $\hat{f}$:

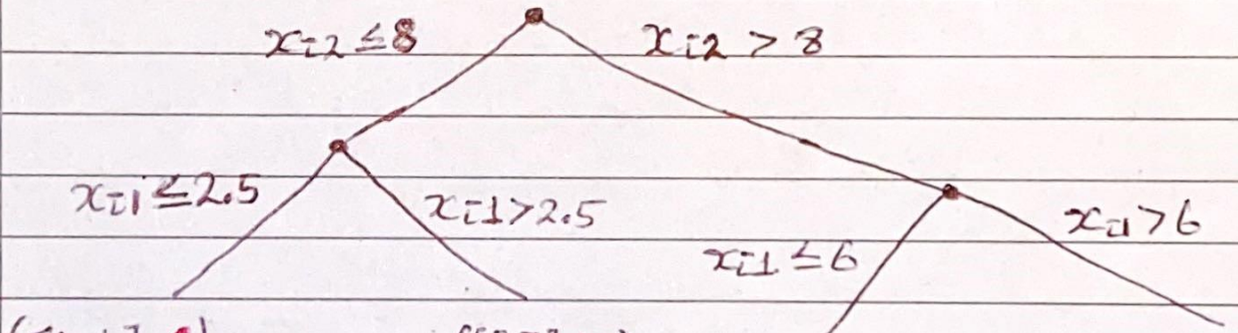
$$\hat{f}(x) = \sum_{b=1}^{B} \eta \cdot \hat{f}^b(x).$$

Boosting Algorithm: Example

Suppose we have 6 observations in the form $([x_{i1}, x_{i2}], y)$:

$([1,6],6), ([2,4],2), ([3,7],10), ([5,10],20), ([7,12],18), ([8,6],12)$

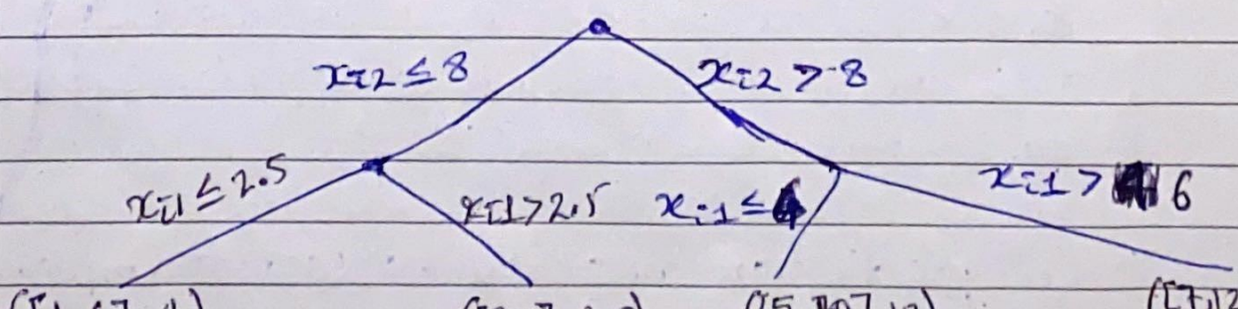
We first use recursive binary splitting to generate a decision tree $\hat{f}_1$ of height 2:



$([1,6],6)$	$([3,7],10)$		
$([2,4],2)$	$([8,6],12)$	$([5,10],20)$	$([7,12],18)$
prediction $= \frac{6+2}{2} = 4$	prediction $= 10+12 = 11$	prediction $= 20$	prediction $= 18$
4	11	20	18

- We apply boosting with $\eta = 0.5$. The first iteration

Observation $([x_{i1}, x_{i2}], r_i)$	prediction $\hat{f}_1([x_{i1}, x_{i2}])$	$r_i - \eta \cdot \hat{f}_1([x_{i1}, x_{i2}])$	updated observation
$([1,6],6)$	4	$6 - 0.5 \times 4 = 6-2 = 4$	$([1,6],4)$
$([2,4],2)$	4	$2 - 0.5 \times 4 = 2-2 = 0$	$([2,4],0)$
$([3,7],10)$	11		$([3,7],4.5)$
$([5,10],20)$	20	$20 - 0.5 \times 20 = 20-10 = 10$	$([5,10],10)$
$([7,12],18)$	18		$([7,12],9)$
$([8,6],12)$	11		$([8,6],6.5)$



$([1,6],4)$	$([3,7],4.5)$	$([5,10],10)$	$([7,12],9)$
$([2,4],0)$			$([8,6],6.5)$
prediction $\frac{4+0}{2} = 2$	4.5	10	$\frac{9+6.5}{2} = 7.75$

Summary of Boosting.

- Boosting
  - is a slow learning process;
  - the smaller the learning rate, the slower the process.
- It generates a sequence of prediction functions that slowly improve the residues.

- The residues get close to zero
  - the training MSE gets close to zero.

- If B is too large,
  - overfitting can occur, albeit slowly.
  - Thus, you need to use testing &/or cross validation to choose the optimal B.

- Each prediction rule can be used to improve $\hat{f}$
  - in areas where it does not fit well.