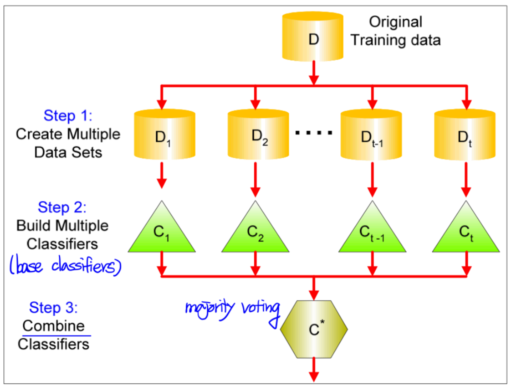
DA-Final-3) Ensembles, Linear, Neural Network model

1. Ensemble model

1-1. Ensemble method 개요

- Construct a set of base classifiers (or weak learner) from the training data
- Predict class label of a testing instance by taking a majority vote on the predictions made by the base classifiers

(다수결)


D
Original Training data
Step 1: Create Multiple Data Sets
D1
D2
Dt-1
Dt
Step 2: Build Multiple Classifiers
(base classifiers)
C1
C2
Ct-1
Ct
Step 3: Combine Classifiers
majority voting
C*

1-2. Ensemble method를 사용하는 이유와 필요 조건

- E.g., suppose there are 25 base classifiers
  - Each classifier has error rate, ε = 0.35
  - If the base classifiers are identical, then the ensemble will misclassify the same examples predicted incorrectly by the base classifiers.
  - Assume classifiers are independent, i.e. their errors are uncorrelated. Then the ensemble makes a wrong prediction only if more than half of the base classifiers predict incorrectly.
  - Probability that the ensemble classifier makes a wrong prediction:

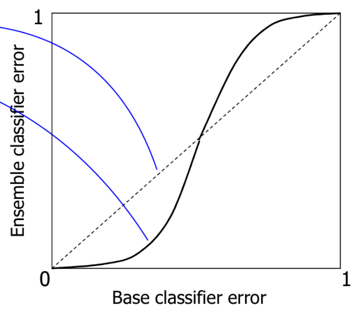
$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06 \quad (\leftarrow 0.35,\ 성능\uparrow)$$

13개의 base classifier가 wrongly predict.
>절반 이상이 misclassifying할 확률

- The diagonal line is when the base classifiers are identical.
- The solid line is when they are independent.

★ **Conditions for Ensembles:**

- *The base classifiers should be independent.*
- *The base classifiers should do better than a classifier that performs random guessing. (error < 0.5)*
- In practice, it is hard to have base classifiers perfectly independent. Nevertheless, improvements have been observed in ensemble methods when they are slightly correlated.



1-3. Ensemble methods

1-3-(1). Bagging

- Bootstrapping, a type of resampling, performs *random sampling with replacement* to produces a dataset of the same size as the original dataset. (복원을 유지.)
- Bagging repeatedly produces bootstrap sample D and builds a classifier on each D.
  - Each bootstrap sample D has the same size as the original data
  - Some instances could appear several times in the same training set while others may be omitted.
  - D will contain approximately 63% of the original data.

암기X
  - Each data object has probability $1-\left(1-\frac{1}{m}\right)^{m}$ of being selected in D where $|D| = m.$

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

①　②

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Decision stump (밑동): one-level binary decision tree, with a test condition x ≤ k. (linear bound)
- Without bagging, the split will be either x ≤ 0.35 or x ≤ 0.75, with accuracy of 70% at most.

10 Bootstrap samples 에 대해 binary classification 적용한 결과
⇓
Voting result

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
y	**1**	**1**	**1**	**-1**	**-1**	**-1**	**-1**	**1**	**1**	**1**
Correct	1	1	1	1	1	1	1	1	1	1

↑ 원본 label과 비교! / 단순 linear bound보다 flexible한 bound 도출!

- The performance of bagging depends on the stability of the base classifier.
  - If a base classifier is unstable, bagging helps to reduce the errors.
  - If a base classifier is stable, bagging may not be able to improve, rather it could degrade the performance.
- Bagging is less susceptible (덜 민감한) to model overfitting when applied to noisy data.

1-3-(2). Boosting

- An iterative procedure/to adaptively change/distribution of training data/by focusing more on previously misclassified records
  - Initially, all m records are assigned equal weights
  - Unlike bagging, weights may change at the end of boosting round
- Weights can be used
  1. As a sampling distribution/to draw a set of bootstrap samples/from the original data
  2. By the base classifier/to learn a model/that is biased toward higher-weight examples
- Records that are wrongly classified/will have their weights increased
- Records that are classified correctly/will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

  - Example 4 is hard to classify 항상 wrongly classified.
  - Its weight is increased,/therefore it is more likely to be chosen again in subsequent rounds
- Boosting algorithms differ in terms of (1) how the weights of the training examples are updated at the end of each round, and (2) how the predictions made by each classifier are combined.

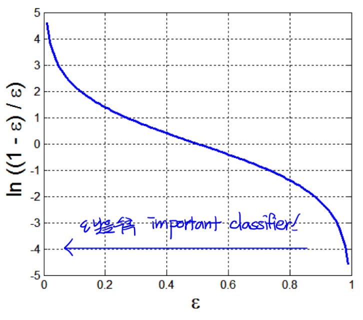
1-3-(2)-a. AdaBoost

- Base classifiers: C1, C2, …
- (델타) δ(p) = 1 if p is true/δ(p) = 0 if p is false
- Error rate of a classifier:

$$\varepsilon_j = \frac{1}{m}\sum_{i=1}^{m} w_i \underbrace{\delta(C_j(x_i) \neq y_i)}_{\text{wrongly classified}}$$

- Importance of a classifier:

$$\alpha_j = \frac{1}{2}\ln\left(\frac{1-\varepsilon_j}{\varepsilon_j}\right)$$



- Classifier of lower error is more important
- Weight update/for each example:

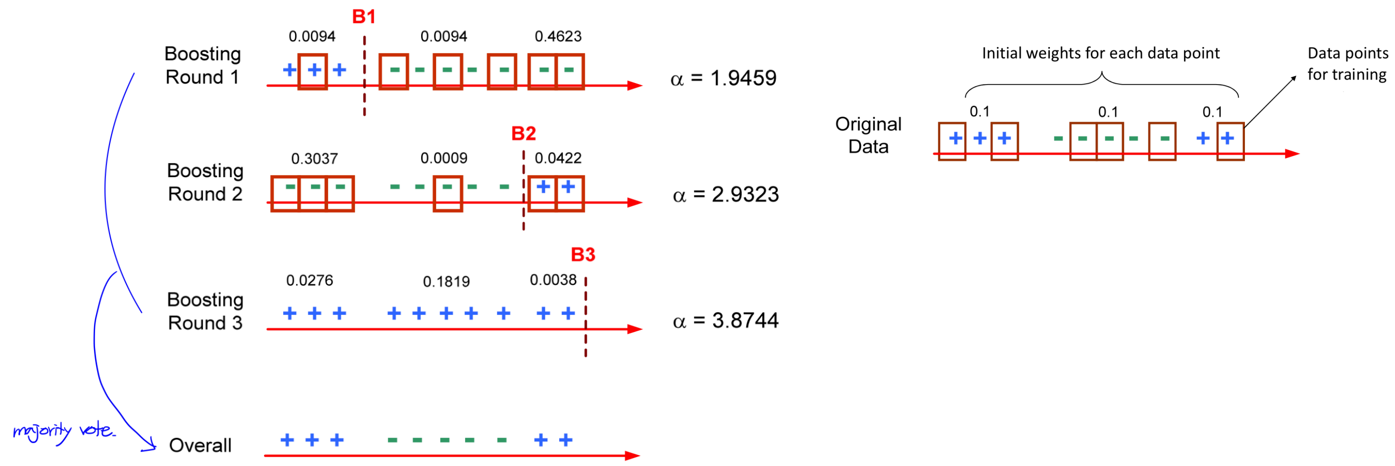
암기X key-idea임!

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \times \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \xrightarrow{\text{weight}} \text{감소!} \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \rightarrow \text{증가!} \end{cases}$$

  - Z_j is the normalization factor/to ensure $\sum_i w_i^{(j+1)} = 1$
  - Misclassified objects will have weight increased

- Classification: $C^*(x) = \arg\max_y \sum_{j=1}^{T} \alpha_j \delta(C_j(x) = y)$

- It penalizes models/that have poor accuracy
- If any (패널티를 주다) intermediate rounds produce error rate higher than 50%,/the weights are reverted (복귀하다) back to 1/n/and the resampling procedure is repeated.



1-3-(2)-b. Gradient Boost

Gradient Boost regression

- Training
  1. Compute the average of the target variable, and build 1st tree that predicts the residuals (= *observed – predicted),* where *predicted* is the average.
  2. Build 2nd tree to predict the residuals (= *observed – predicted)*, where *predicted = the average + learning rate * 1st tree's prediction*.
  3. Build 3rd tree to predict the residuals (= *observed – predicted)*, where *predicted = the average + learning rate * 1st tree's prediction + learning rate * 2nd tree's prediction.*
  4. …
- Predict new data by adding the predictions of all trees. (The learning rate is multiplied to each tree.)

Gradient Boost classification

- Similar to Gradient Boost regression except that the residuals (or classification errors) are computed in a more complicated way using log(odds), sigmoid functions, etc.

1-3-(2)-c. XGBoost (Extreme Gradient Boost)

Similar to Gradient Boost except

- The details about building each regression (or classification) tree
- Lots of optimization techniques like..
  - For huge dataset: approximate greedy algorithm, parallel learning, weighted quantile sketch
  - For missing values: sparsity-aware split finding
  - Hardware-aware optimizations: cache-aware access, blocks for out-of-core computation

1-3-(3). Random forest

≈ bagging + decision tree
└ base learner

- Random forest focuses on producing *diverse* decision trees.
  - Generate a bootstrap sample *D* from the original dataset.
  - Builds a decision tree on *D* with a randomly selected small subset of features (e.g. $\sqrt{\#\text{ of features}}$).
  - Repeat this.
- Predictions are made based on majority voting

Decision tree의 ensemble을 사용한 feature selection

- How to compute feature importance? ← 계산
  - For each node, compute the performance measure (e.g. *Information Gain*) weighted by # of observations the node is responsible for, and average them for all trees
  - Perform wrapper method on OOB (Out-Of-Bag) observations (i.e. test set).

↑ feature selection을 위한 method

2. Linear models

2-1. Linear classifier for binary classification

- A linear function classifies an object (n-dimensional vector) $\mathbf{x} = (x_1, x_2, \dots x_n)$ (or $\mathbf{x} \in \mathbb{R}^n$) based on:

$$f(\mathbf{x}) = b + \sum_{i=1}^{n} w_i x_i = \sum_{i=0}^{n} w_i x_i \ (x_0 = 1) = \mathbf{w}^{\mathrm{T}}\mathbf{x} = \mathbf{w} \cdot \mathbf{x}$$

- w_i is the weight of a feature x_i $w_0 x_0 = w_0$
- b (or w_0) is bias (scalar)
- $\mathbf{x}$ is classified as positive if $f(\mathbf{x}) > 0$
- $\mathbf{x}$ is classified as negative if $f(\mathbf{x}) < 0$

- A binary classifier $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$ defines a hyperplane with normal vector $\mathbf{w}$.

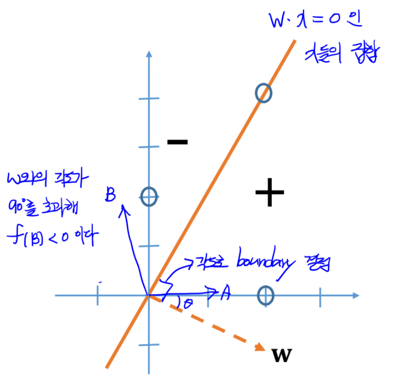
  similar↑, 값이↑

  $\mathrm{W} \cdot \mathrm{X} = 0$

  $\Rightarrow \mathrm{W} \perp \mathrm{X}$

  - $\mathbb{R}^2 \Rightarrow$ ~~*line*~~ *line*
  - $\mathbb{R}^3 \Rightarrow$ ~~*plane*~~ *plane*
  - $\mathbb{R}^{4 \text{ or higher}} \Rightarrow$ *hyperplane*

- Example:
  - $\mathbf{w} = [2, -1]$
  - $\mathbf{x} \in \{[2,0], [0,2], [2,4]\}$



2-2. Linear, Quadratic, Polynomial classifiers

- Linear function: consider weights of individual features

$$f(\mathbf{x}) = b + \sum_{i=1}^{n} w_i x_i$$

- Quadratic function: also consider weights of concatenations of two features ∀ pair of features

$$f(\mathbf{x}) = b + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i x_j$$

P=2 case

- Polynomial function: also consider weights of concatenations of multiple features (*Degree p* denotes the size of concatenations)

$$f(\mathbf{x}) = b + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\, x_i x_j + + \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{j=1}^{n} w_{ij\,k} x_i x_j x_k + \cdots$$

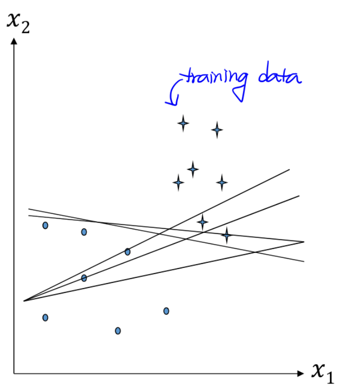
$O(n^p)$: p에 대해 exponential한 계산 복잡도

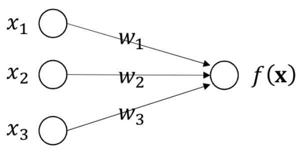
2-3. Perceptron algorithm

- Perceptron: Algorithm to learn a linear function

$$f(\mathbf{x}) = \sum_{i=0}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

- Steps:
  1. Initiate f by randomly setting $\mathbf{w}$
  2. For each training point $\mathbf{x}$,
     if f misclassifies $\mathbf{x}$, adjust $\mathbf{w}$ toward the direction that correctly classifies $\mathbf{x}$
  3. Repeat Step 2





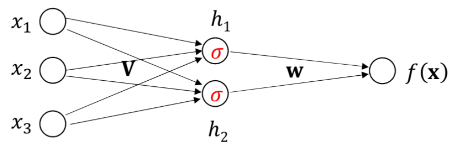
- Nondeterministic
  - For the same training set, it could generate different f depending the data ordering.
  - Several hyper-parameters to tune such as # of iterations, how much adjust $\mathbf{w}$.
- Limited to linear functions
  - # of $\mathbf{w}$ for polynomial function is too large to train in a nondeterministic way.
  - Inherently requires $O(n^p)$ to learn a polynomial function where p is the degree
    (본질적으로)
  - (SVM can learn a polynomial function with an arbitrary degree in $O(n)$.)

$$f(\mathbf{x}) = \sum_{i} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

3. Neural Network models

3-1. Multi-Layer Perceptron (MLP)



$$h_i = \sigma\left(\sum_j v_{ij} x_j\right) = \sigma(\mathbf{v}_i \cdot \mathbf{x}) \text{ where } \sigma(z) = (1 + e^{-z})^{-1}$$

$$f(\mathbf{x}) = \sum_i w_i \sigma(\mathbf{v}_i \cdot \mathbf{x}) = \sum_i w_i h_i = \mathbf{w} \cdot \mathbf{h}$$

- σ is called activation function: Traditionally, sigmoid function was used, but **ReLU (Rectified Linear Unit) σ(z) = max{z,0}** is now most popularly used.
- **V** and **w** are learning parameters (learned by backpropagation).
- Multiple neurons are used for multiclass classification.
- Details will be covered in AI, Machine learning, and Deep learning courses.

3-2. Deep learning / Artificial Neural Network (ANN)

same meaning.
- Multi-Layer Perceptron (MLP), Feed Forward Neural Network (FFNN), Fully connected neural network, Dense layer
- More on deep learning
  - CNN, ResNet, InceptionNet, MobileNet, …
  - RNN, LSTM, GRU, …
  - Transformer and language models
  - …
- For tabular data
  - Tree-based: Random Forest and XGBoost [2]: Good performance without hard tuning
  - DL-based: MLP, TabNet [1], ResNet [3], FT-Transformer [3]: Input optimization is also possible.

  - [1] TabNet: Attentive Interpretable Tabular Learning, arXiv 2019.
  - [2] Tabular Data: Deep Learning is Not All You Need, arXiv 2021
  - [3] Revisiting Deep Learning Models for Tabular Data, NeurIPS 2021