SAMSUNG

Samsung Innovation Campus

Artificial Intelligence Course

Together for Tomorrow!
Enabling People
Education for Future Generations

Chapter 5.

Machine Learning 1
- Supervised Learning

Artificial Intelligence Course

Chapter Description

Chapter objectives

- ✓ Be able to introduce machine learning-based data analysis according to the business objective, strategy, and policy and manage the overall process.
- ✓ Be able to select and apply a machine learning algorithm that is the most suitable to the given problem and perform hyperparameter tuning.
- ✓ Be able to design, maintain, and optimize a machine learning workflow for AI modeling using structured and unstructured data.

Chapter contents

Unit 4.

Decision Tree

4.1. Tree Algorithm

Overview of Decision Tree

Definition

- A classification model that analyzes data collected from the past and expresses the patterns found (characteristics of each category) as a combination of features

Purpose

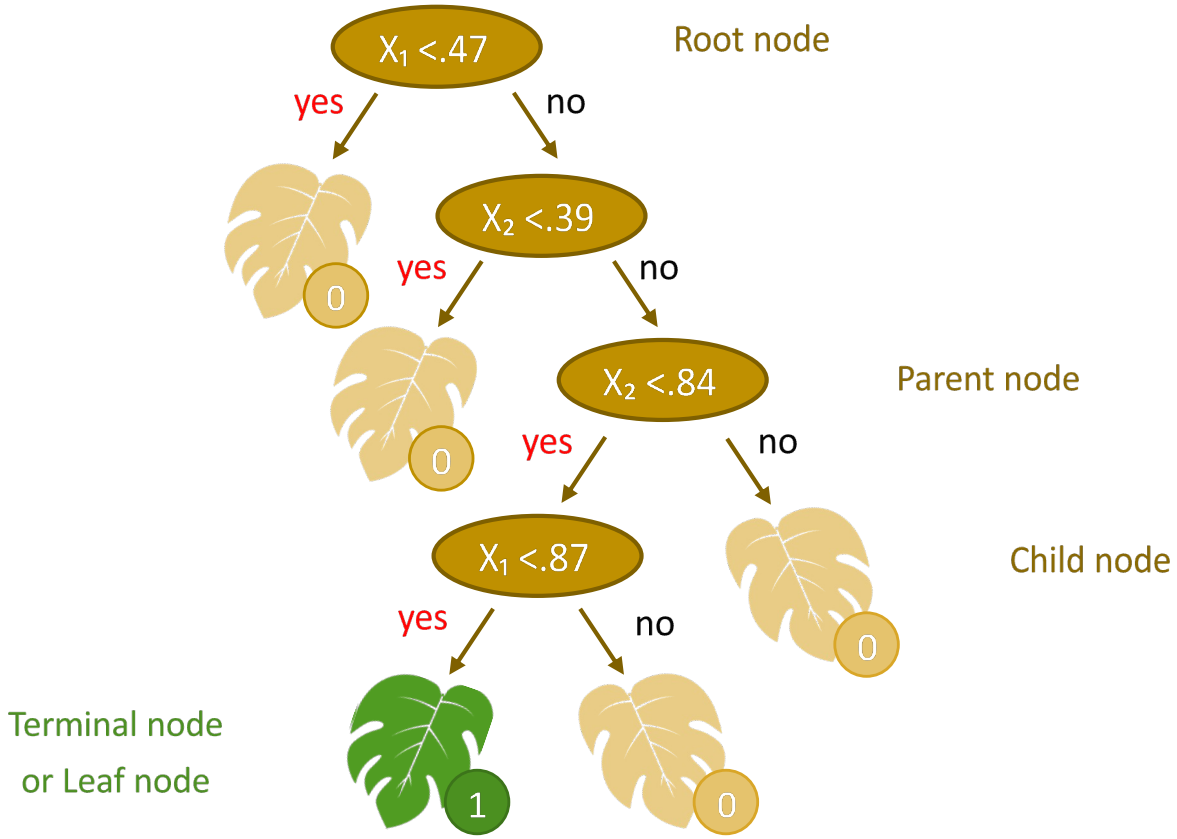
- Classification of unseen data and prediction of categorical values
- Extraction of generalized knowledge in a tree structure from the data

Classification depending on the objective variable types

- Categorical variable: Classification Tree
- Continuous variable: Regression Tree

Composition

- Node, Branch, Depth



How to construct a decision tree model

Construction of a decision tree

Construction of a decision tree	Making a decision tree by designating an appropriate split criterion and stopping rules according to the purpose and data structure of analysis
Branching	Removing branches that have a high risk of error rate or inappropriate rules
Validity evaluation	Evaluation of the decision tree through cross-validation using the gain chart, risk chart, or test data
Interpretation and prediction	Interpretation of the decision tree and setting a prediction model

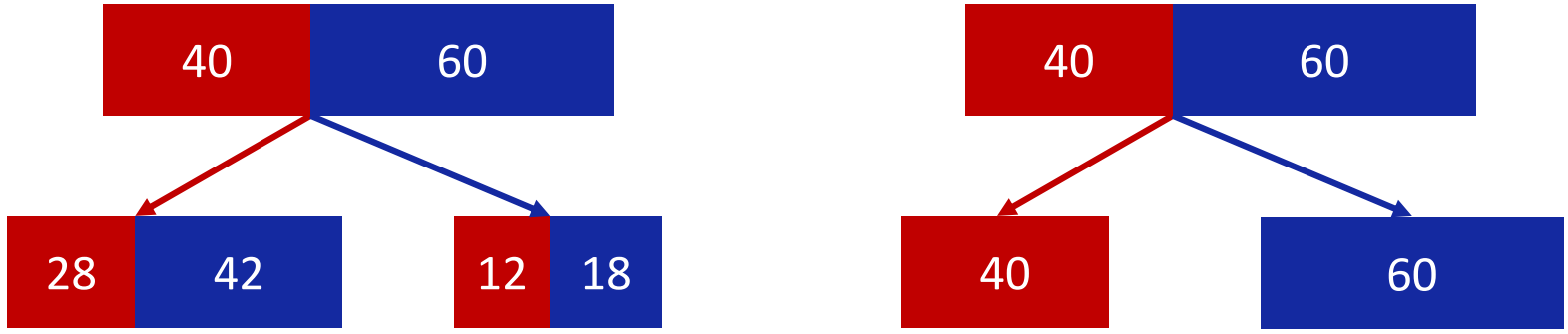
Split criterion of the decision tree

Analysis process of a decision tree

- Repetitive splitting: Repetitive splitting of the dimensional space of independent variables using training data
- Branching: Branching with evaluation data

Split criterion

- Create the classification tree so that the purity of the child node is greater than that of the parent node.



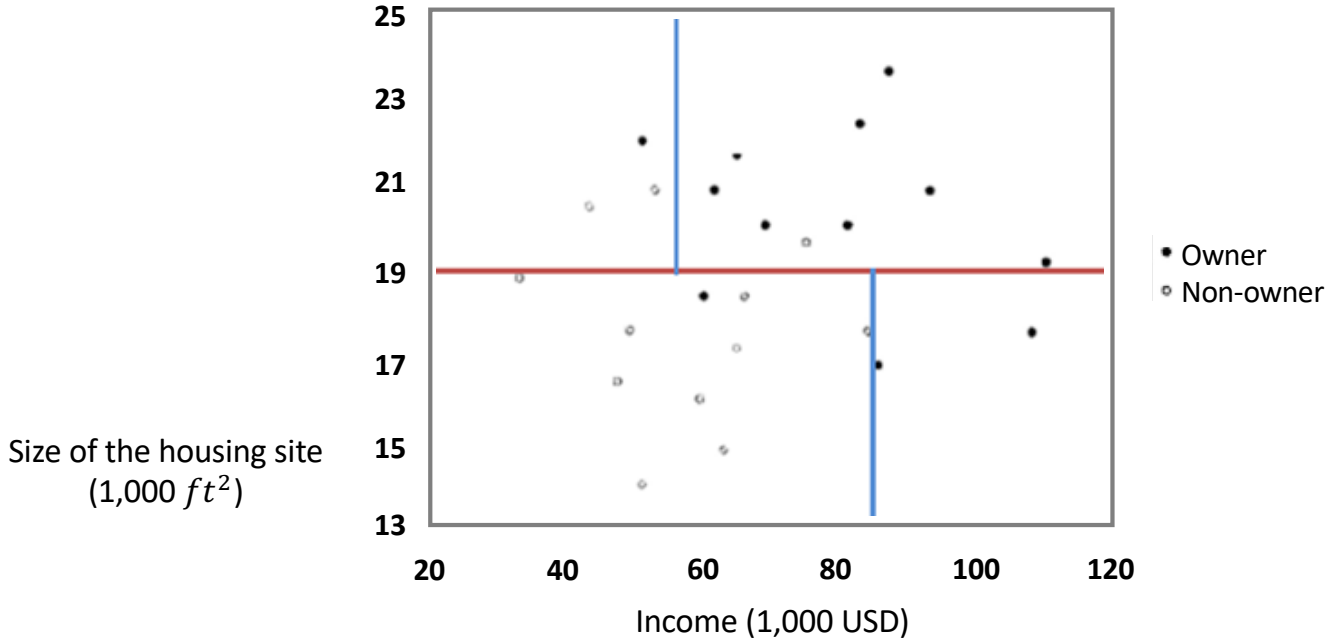
No Improvement

Perfect Split

Repetitive splitting process

Purpose

- Divide the entire space into rectangles and make each as pure or homogeneous as possible.
- Definition of 'pure':
  - Divide the area into pure or homogenous rectangular spaces as much as possible
  - All variables in the final rectangle belong to the same group.



Repetitive splitting process

1) Select xi, one of the variables, and the xi value (si as a split criterion) is designated to split the p-dimension space into two.

2) xi → {xi<=si} ∪ {xi>si}

3) Select a variable again and split it in the same way.

4) Repeat the process until it reaches desired purity.

Split criterion

Discrete objective variables

- Chi squared statistic – p-value: Creates child nodes with a predictor variable with the lowest p-value and the optimal partitioning
- Gini index: Selects child nodes with a predictor variable that reduces the Gini index and the optimal partitioning
- Entropy measure: Creates child nodes with a predictor variable with the lowest entropy measure and the optimal partitioning

Split criterion

Continuous objective variables

- F statistic in ANOVA: Creates child nodes with a predictor variable with the lowest p-value and the optimal partitioning
- Variance reduction: Creates child nodes with the optimal partitioning that maximizes variance reduction

Selection of algorithms and classification variables

	Discrete objective variables	Continuous objective variables
CHAID (multi space partitioning)	Chi squared statistic	ANOVA F statistic
CART (binary space partitioning)	Gini index	Variance reduction
C4.5	Entropy measure	

Impurity measure

Gini index

- Selects child nodes with a predictor variable that reduces the Gini index and the optimal partitioning
- If the T data set is split into k categories and the category performance ratios are p1, …, pk, it is expressed as the following equation.

$$Gini(T) = 1 - \sum_{l=1}^{k} p_l^2$$

high impurity(diversity), low purity

GI = 1 –(3/8)²–(3/8)²–(1/8)²–(3/8)²= .69

low impurity(diversity), high purity

GI = 1 –(6/7)²–(1/7)²= .24

Entropy measure

- In thermodynamics, entropy measures the degree of disorder.
- Creates child nodes with a predictor variable with the lowest entropy measure and the optimal partitioning.
- If the T data set is split into k categories and the category performance ratios are p1, …, pk, it is expressed as the following equation.

$$Entropy(T) = -\sum_{l=1}^{k} p_l \log_2 p_l$$

Ex If 4 categories consist of ratios of 0.25, 0.25, 0.25, 0.25 (T0):

$$Entropy(T_0) = -(0.25 \log_2 0.25) * 4 = 1.39$$

Ex If 4 categories consists of ratios of 0.5, 0.25, 0.25, 0 (T1):

$$Entropy(T_1) = -(0.5 \log_2 0.5 + 0.25 \log_2 0.25 + 0.25 \log_2 0.25) = 1.04$$

Stopping criteria

- A rule to designate the current node as a terminal node without further splitting
  - Designates the depth of the decision tree
  - Designates the minimum number of records in the terminal node

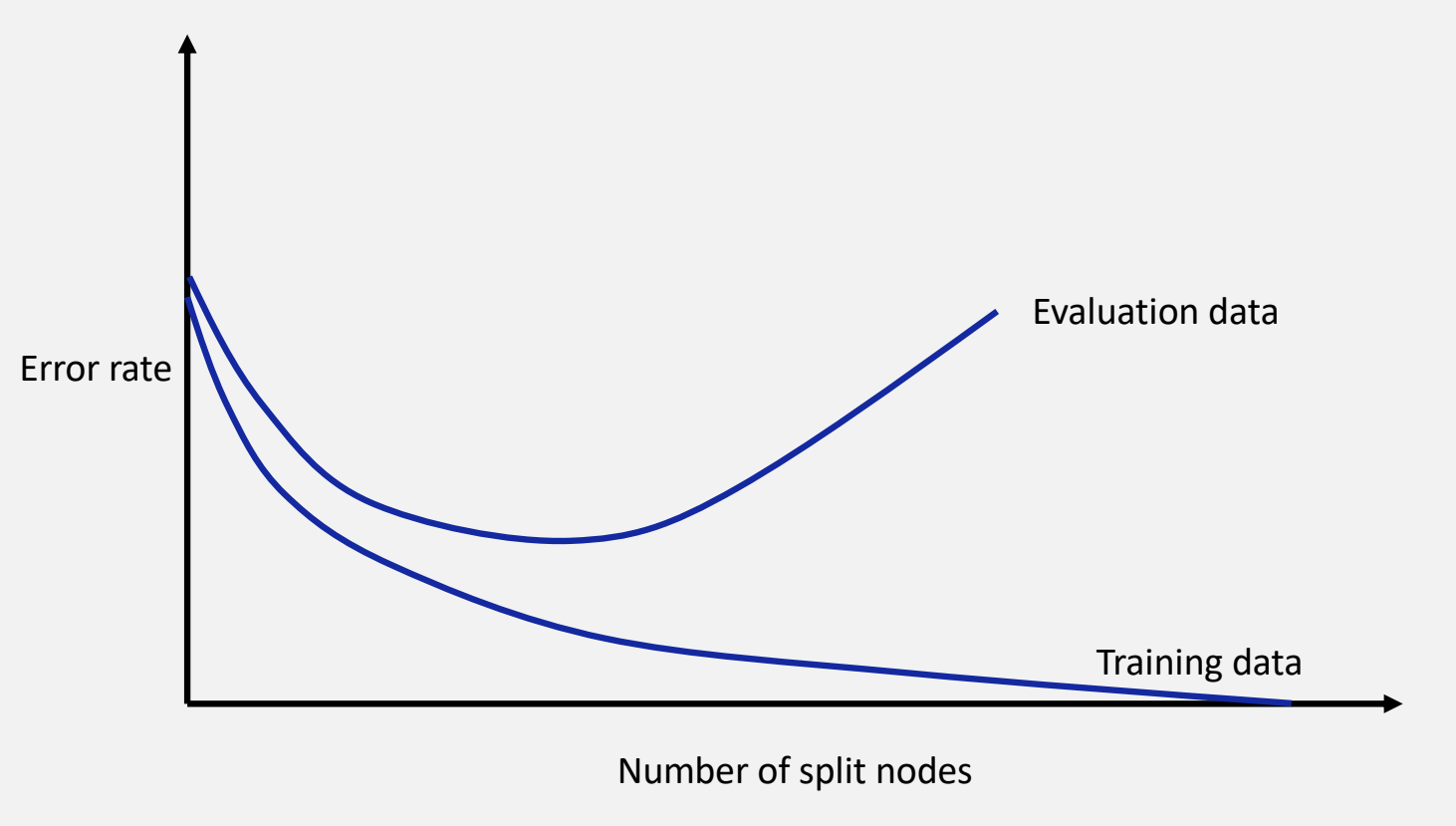
Branching criteria

- Application of test data
  - Application of the test data to the constructed model
  - Reviewing the predictive value of the constructed model through test data
  - Removing the branches that have a high risk of error rate or inappropriate rule of inference

- By an expert
  - An expert reviewing the validity of rules suggested in the constructed model
  - Removing rules without validity

Overfitting problem

Overfitting problem graph



Pros

- Creation of understandable rules (can be expressed with SQL)
- Useful in classification prediction
- Able to work with both continuous and discrete variables
- Shows a more relatively significant variable

Cons

- Not suitable to predict continuous variable values
- Unable to perform time series analysis
- Not stable

Tree Algorithm

Pros

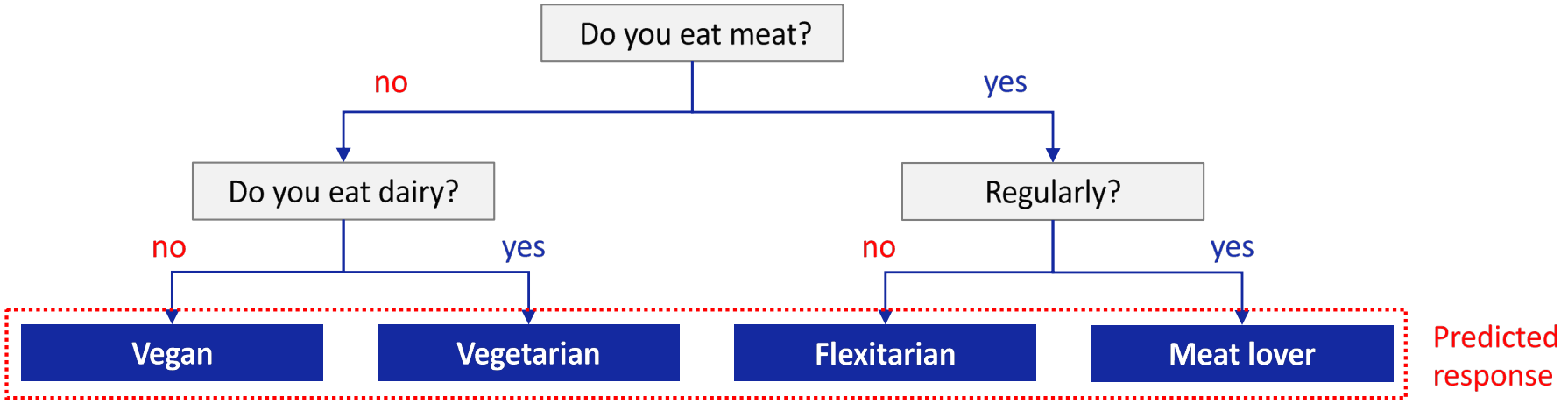
- Intuitive and easy to understand
- No assumptions about the variables
- No need to scale or normalize data
- Not that sensitive to the outliers

Cons

- Not that powerful in the most basic form
- Prone to overfitting. Thus, “pruning” is often required.

Classification Tree

Ex



- Training step creates an inverted tree structure as above.
- Conditions are evaluated at the nodes and then branch out.
- Each leaf node corresponds to a **region** in the configurational space.

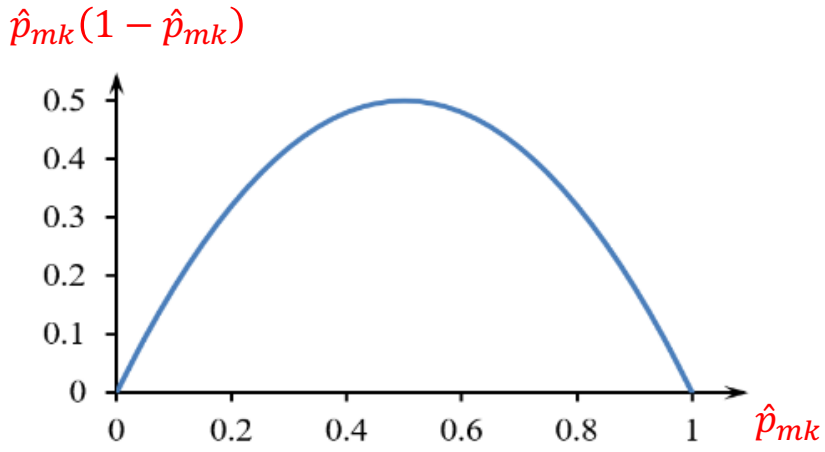
Classification Tree

- The tree structure is trained by minimizing the Gini impurity (or entropy).

$$G_m = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) = 1 - \sum_{k=1}^{K} \hat{p}_{mk}^{\,2}$$

or

$$Entropy_m = -\sum_{k=1}^{K} \hat{p}_{mk} Log(\hat{p}_{mk})$$



- G_m is the Gini impurity in the leaf node m.
- $Entropy_m$ is the entropy in the leaf node m.

} The smaller, the better.

- Here, $\hat{p}_{mk}$ is the proportion of the class k in the leaf node m.
- K is the total number of possible classes.
- The class with the largest proportion is the prediction at that leaf node.

Classification Tree: Procedure

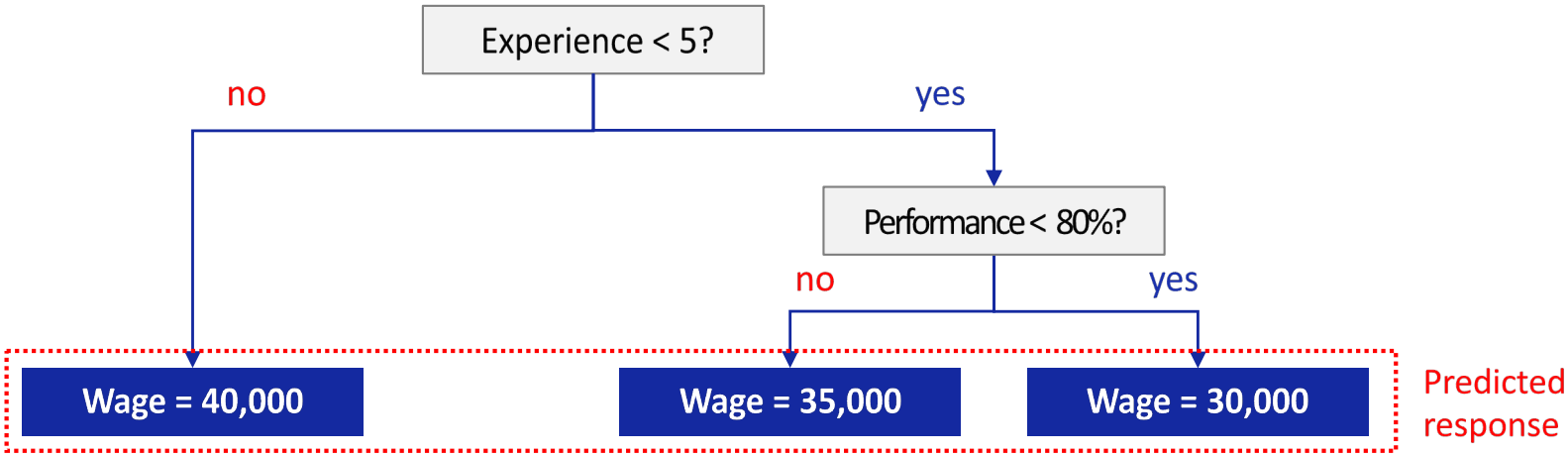
a) Make a basic tree.

b) Prune branches that do not provide better performance.
Pruning can be done during the cross-validation step.

c) Predict with the optimized tree.

Regression Tree

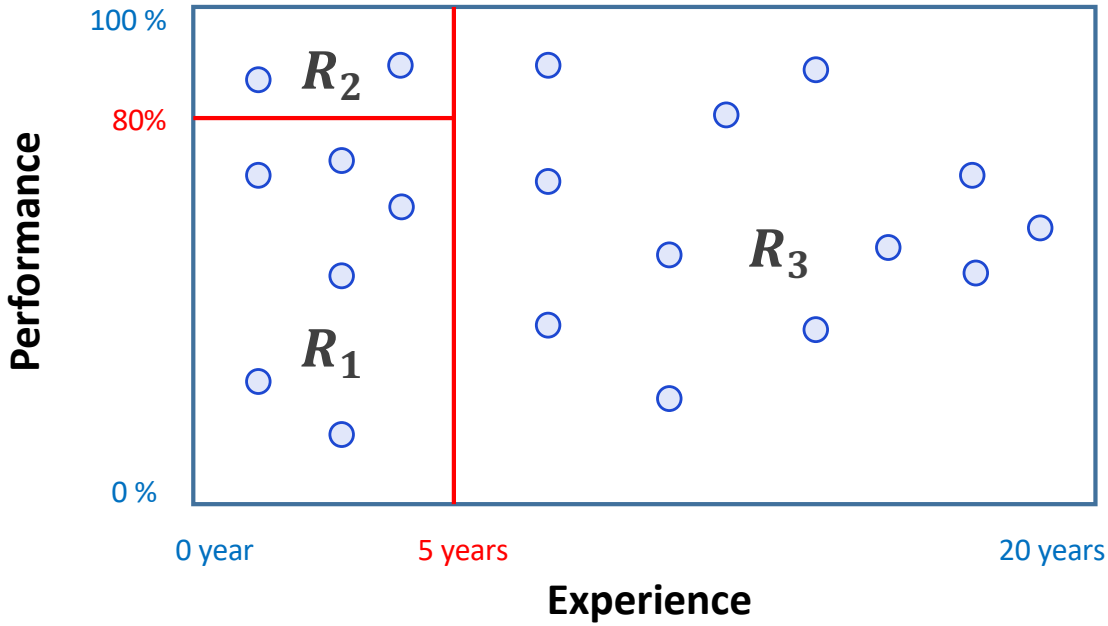
Ex



- Predicts numeric values rather than categories or classes.
- Conditions are evaluated at the nodes and then branch out.
- Each leaf node corresponds to a region in the configurational space.

Regression Tree

Ex



- Each leaf node corresponds to a region in the configurational space.

Regression Tree

- The configurational space is split into regions: $\{R\_1, R\_2, \cdots, R\_J\}$.
- The tree structure is trained by minimizing the RSS (residual sum of squares):

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \hat{y}_{R_j} \right)^2$$

  - J is the total number of leaf nodes.
  - The predicted value in the j-th region $\hat{y}_{R_j}$ is given by the average in that region.
  - For the observations belonging to the region R_j, the same predicted response $\hat{y}_{R_j}$ is assigned analogous to the classification Tree.

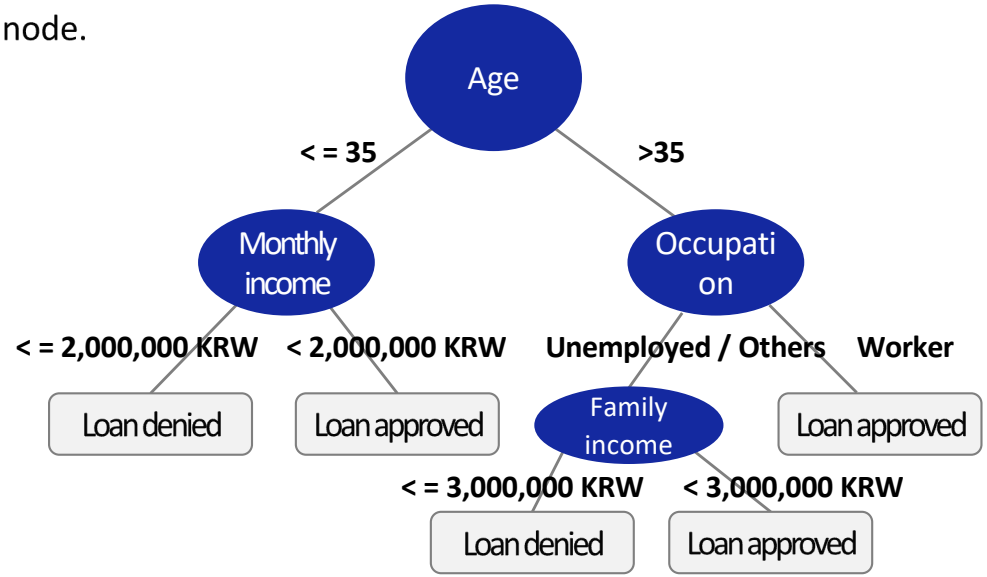
Scikit-Learn DecisionTreeClassifier/Regressor Hyperparameters:

Hyperparameter	Explanation
max_depth	The maximum depth of a tree
min_samples_leaf	The minimum number of sample points required to be at a leaf node
min_samples_split	The minimum number of sample points required to split an internal node
max_features	The number of features to consider when looking for the best split
max_leaf_nodes	The maximum number of leaf nodes in the tree

- Need to be tuned for optimized performance.
- More information can be found at:
  Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
  Regressor: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

Decision Tree

- Decision tree refers to a modeling technique as the shape of a tree branching out from root to leaf nodes. It depends on reference values of independent variables (explanatory or input variables) that affect the classification or prediction of objective variables.
- In the decision tree, each node is split in the form of if-then depending on explanatory variables' characteristics or reference values. When following the tree structure, it is possible to easily understand how the attribute value of data is classified into the category.
- The figure provided below is a typical form of the decision tree. From the example, 'age' is the root. It can be inferred that 'age' is the most significant variable when deciding the loan approval.
- The squared shape node at the end of each branch is the leaf node.



Split Criteria of Decision Tree

- Decision tree gives a question to each node and separates data by branching according to the response.
- To evaluate how well the data is separated, a specific criterion is required. In general, impurity is the evaluation criterion. The impurity becomes higher as various classifications are mixed in the node; it is the lowest when there's only one classification.
- Thus, if each node's impurity is low after node splitting, we can say that the decision tree is well classified.
- Impurity measures include Gini impurity and entropy.

$$\text{Gini } Coff. = 1 - \sum_{i=1}^{K} p_i^2 , \quad Entropy\ Coff. = -\sum_{i=1}^{K} p_i \log_2 p_i$$

- The impurity index is 0 when $p_i = 0$ or $p_i = 1$, and it gets the largest when $p_i = \frac{1}{2}$, thus making a parabola. In other words, the impurity index is the lowest when there's a certain classification in the node or the node is completely free from any classification. In contrast, the impurity becomes the largest when many classifications are found in the same node.

Unit 8.

Ensemble Algorithm

- 8.1. The concept of Ensemble Algorithm and Voting
- 8.2. Bagging & Random Forest
- 8.3. Boosting

Ensemble algorithms

About ensemble algorithms

- Strong predictive model based on the weaker learners.
- **Voting type**:
  - A collection of basic learners that “vote”
  - An ensemble of different kinds of learners **Ex** Combine Tree, KNN, SVM, etc.
- **Bagging type**:
  - A collection of independent weak learners that “vote”
  - An ensemble of the same kind of weak learners **Ex** Random Forest
- **Boosting type**:
  - A series of weak learners that adaptatively learn and predict.
  - The series is grown by adding new learners multiplied by the “boosting weights.”

Ex AdaBoost, GBM, XGBoost, etc.

About ensemble algorithms

- Making a more suitable decision-making by appropriately combining multiple opinions obtained by different experts
- Majority Voting – Analysis using many different classification models with one identical training set
- Bagging (different training set)
  - bootstrap + aggregating
  - Bootstrap + aggregating
  - Bagging (random sampling with replacement (random bootstrap))

    Ex Random Forest
- Boosting
  - Retain 50% of incorrectly classified data or use weight for sample selection.

bootstrap

- Estimation of unknown statistics
- Easy and effective estimation method with the unknown distribution of the model parameter sample
- Process of recalculating the statistics and model for each sample through additional sampling with replacement from the current sample
  - No assumption is required such that the parameters or sample statistics should be in a normal distribution.
- bootstrap sample – Aggregation of observed data (sampling with replacement obtained from the record value and dependent variable)

Contents to be learned in the future

- Resampling – Involves permutation under-sampling without replacement
- Bootstrap aggregation – Making a result from aggregating predicted values obtained from different bootstrap samples

What is Ensemble Learning?

- The term ‘ensemble’ can be defined as follows:
  - In statistical mechanics, an ensemble of a system refers to the collection of equivalent systems.
- In other words, it is an assembly of similar groups.
- Instead of expecting performance results from a single model, ensemble learning draws a better result using the collective intelligence of different models, such as averaging out many different single models or making a decision based on the majority vote.
- There are many different ensemble methods using collective intelligence.
  - Voting – Drawing results through voting
  - Bagging – Bootstrap aggregating (duplicated creation of various samples)
  - Boosting – Weighting by supplementing previous errors
  - Stacking – A meta-model based on different models
- There can be other more different methods since ensemble learning applies a certain technique/methodology.
  Yet, the four methods listed above are the most representative ensemble techniques in the sklearn library.

Voting Ensemble

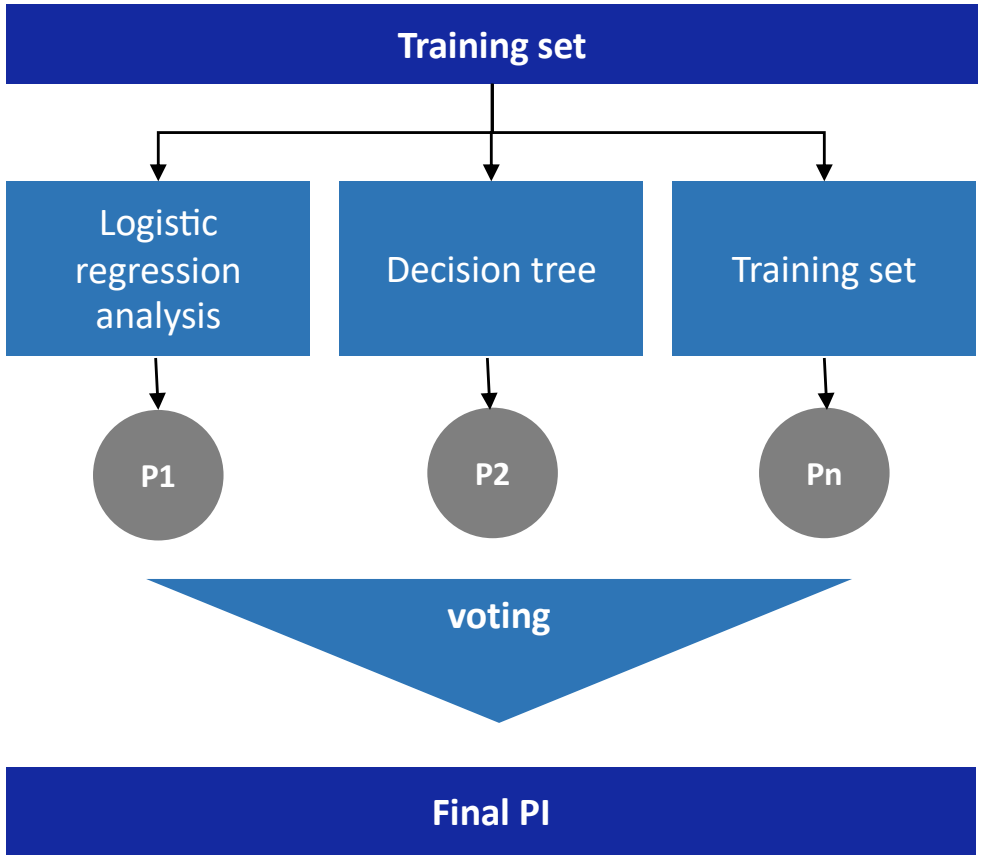
Voting ensemble

- Can be applied to classification and regression.
- The learners that form an ensemble should be of different kinds for a good performance.
- Two voting methods for the classifier:
  - Hard: predicted class label is given by the majority rule voting.
  - Soft: predicted class label is given by the **argmax** of the sum of the predicted probabilities.

Voting

- As the word itself, voting makes a decision through votes. Voting is similar to bagging as it uses a voting method, but they are highly differentiated from each other as follows:
  - Voting: Combines different algorithm models.
  - Bagging: Uses different sample combinations within the same algorithm.
- Voting selects final results by having final voting on results deduced by different algorithms.
- Voting is classified into hard voting and a soft voting.
  - Hard voting: Decides the final value of the result through voting.
  - Soft voting: Draws the final value by adding all the probability values of getting the final result and then calculating every probability of the final result.

Voting



- Use a single training set.
- Use different classification models.
- Predictive value
- If the predictive values of each analysis model are different from voting, choose the result with the most values.
  - hard voting
  - soft voting

  → **Predict the highest class by averaging out the prediction of individual classifier.**

Voting

- Hard Voting
  - Taking classification as an example. Suppose the predictive values for classification are 1,0,0,1,1 since 1 has three votes and 0 has 2 votes. In that case, 1 becomes the final predictive value in the hard voting method.
- Soft Voting
  - Soft voting method calculates the average value of each probability and then determines the one with the highest probability.
  - Suppose the probability of getting class 0 is (0.4, 0.9, 0.9, 0.4, 0.4), and the probability of getting class 1 is (0.6, 0.1, 0.1, 0.6, 0.6). The final probability of getting class 0 is (0.4+0.9+0.9+0.4+0.4) / 5 = 0.44; the final probability of getting class 1 is (0.6+0.1+0.1+0.6+0.6) / 5 = 0.4. Therefore, the selected final value is different from the result of the hard voting above.
- In general, using the soft voting method is considered more reasonable than the hard voting method in competitions because the soft voting method provides a much better actual performance result.

Voting ensemble in Scikit-Learn

- To import the voting ensemble as class:

```
from sklearn.ensemble import VotingClassifier        # For classification
from sklearn.ensemble import VotingRegressor         # For regression
```

- To instantiate an object that implements voting ensemble:

Ex

```
myKNN = KNeighborsClassifier(n_neighbors = 3)
myLL = LogisticRegression()
myVotingEnsemble=VotingClassifier(estimators=[('lr',myLL),('knn',myKNN)],voting='hard')
```

Voting ensemble in Scikit-Learn

- We can train and predict just like any other estimator:

  Ex myVotingEnsemble.fit(X_train, Y_train)

  myVotingEnsemble.predict(X_test)

Scikit-Learn VotingClassifier/Regressor Hyperparameters:

Hyperparameter	Explanation
estimators	The list of basic learner objects
voting	Either 'soft' or 'hard' (for classifier only)

- More information can be found at:

  Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

  Regressor: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingRegressor.html

Unit 8.

Ensemble Algorithm

- 8.1. The concept of Ensemble Algorithm and Voting
- 8.2. Bagging & Random Forest
- 8.3. Boosting

Bagging Ensemble: Random Forest

About Random Forest

- An ensemble algorithm based on trees
- Can be applied to classification and regression

Pros

- Powerful
- Few assumptions
- Little or no concern about the overfitting problem

Cons

- Training is time consuming.

Random Forest algorithm

- Randomly chosen trees will form a forest
- Prediction is decided by the majority vote.

Random Forest algorithm

1) Make trees with randomly selected variables and observations.

2) Keep only those with the lowest Gini impurity (or entropy).

3) Repeat from step 1) a given number of times.

4) Using the trees gathered during the training step, we can make predictions by majority vote.

Scikit-Learn RandomForestClassifier/Regressor Hyperparameters:

Hyperparameter	Explanation
n_estimators	The number of trees in the forest
max_depth	The maximum depth of a tree
min_samples_leaf	The minimum number of sample points required to be at a leaf node
min_samples_split	The minimum number of sample points required to split an internal node
max_features	The number of features to consider when looking for the best split

- Except for "n_estimators," the rest are analogous to those of DecisionTreeClassifier/Regressor.
- More information can be found at:

  Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

  Regressor: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Bagging

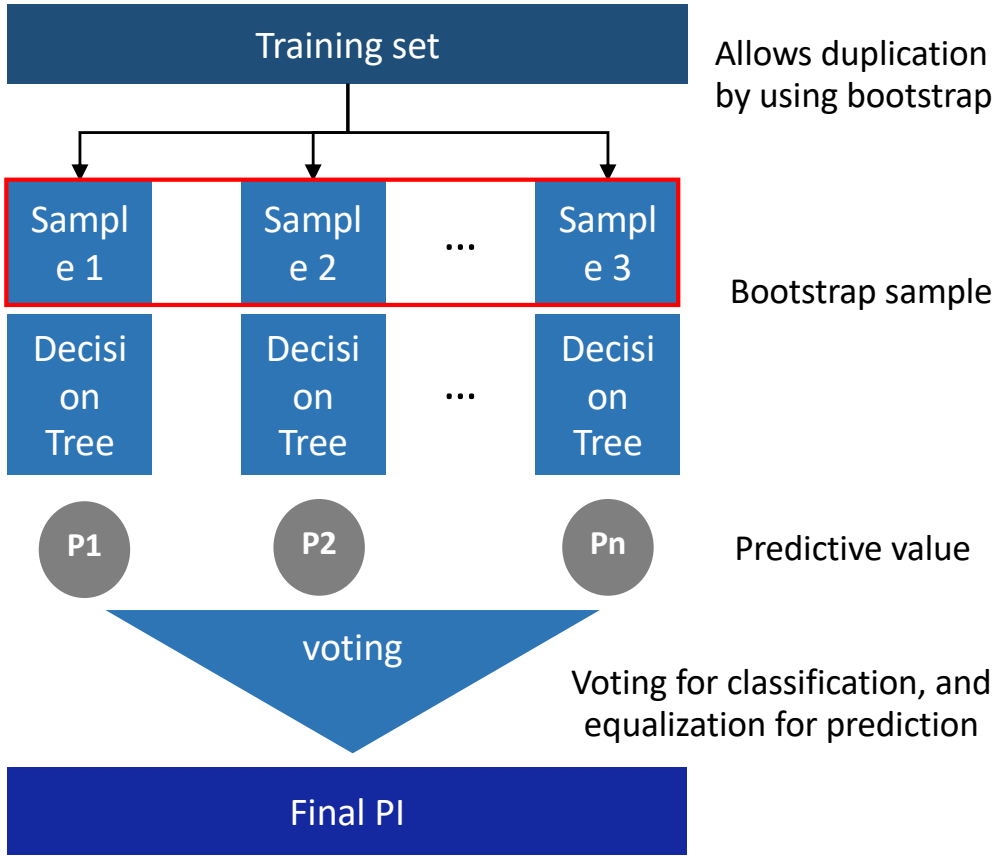
- Bagging-based ensemble method

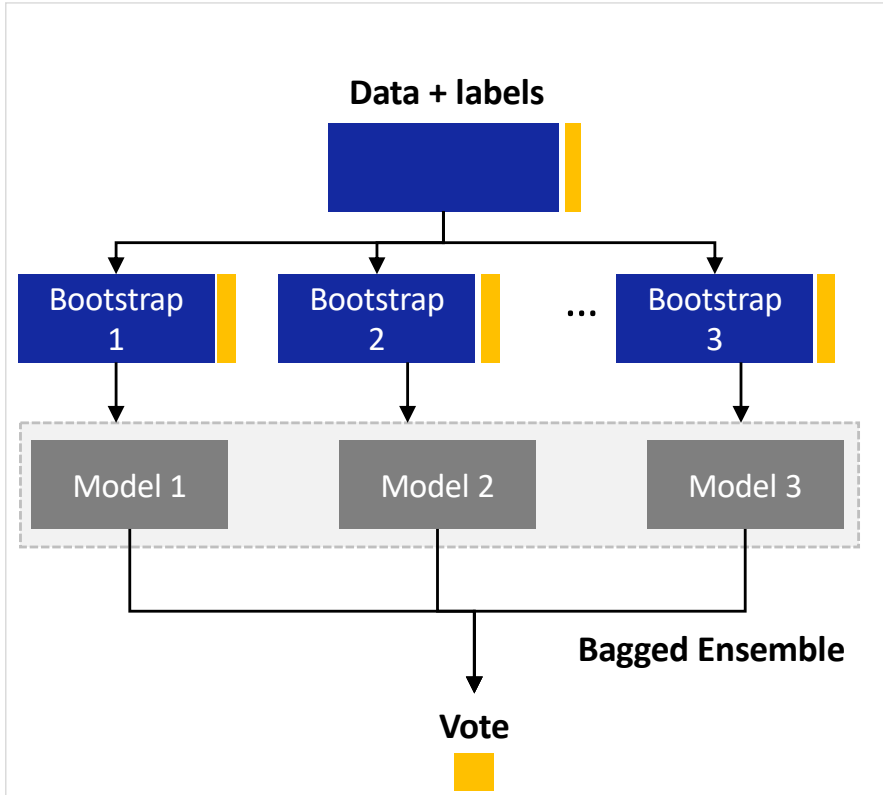
  Ex Random Forest algorithm

  - Easy to use since it is well constructed in the Sklearn library
  - Relatively quick performance speed
  - High performance

- The ensemble method has been widely used as it raises the performance level and is easy to use. The bagging-based ensemble method is commonly found in the high-ranked solutions in Kaggle.

Bagging


Training set
Allows duplication by using bootstrap
Sampl e 1
Sampl e 2
...
Sampl e 3
Bootstrap sample
Decisi on Tree
Decisi on Tree
...
Decisi on Tree
P1
P2
Pn
Predictive value
voting
Voting for classification, and equalization for prediction
Final PI


Data + labels
Bootstrap 1
Bootstrap 2
...
Bootstrap 3
Model 1
Model 2
Model 3
Bagged Ensemble
Vote

Bagging

- Bagging is an abbreviation of Bootstrap Aggregating.
- Bootstrap = Sample
- Aggregating = Adding up
- Bootstrap refers to a method that allows overlapping of different data sets for sampling and splitting.

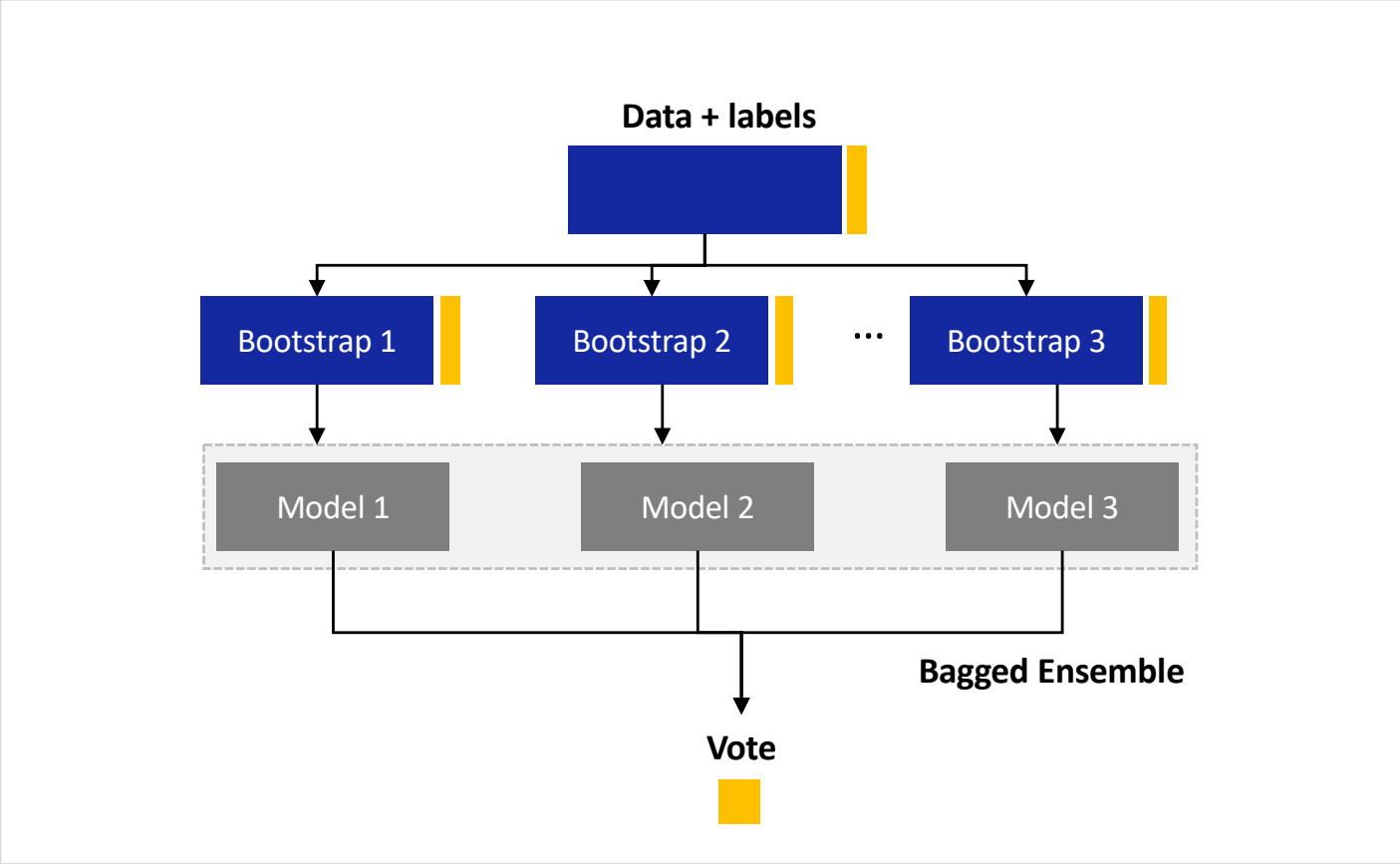
Bagging

Ex Random Forest is a typical bagging method algorithm.

- It creates multiple decision trees and performs sampling of different data sets while allowing overlapped data sets.
- If the data set consists of [1, 2, 3, 4, 5]:
  - Group 1 = [1, 2, 3]
  - Group 2 = [1, 3, 4]
  - Group 3 = [2, 3, 5]
- This is the bootstrap method. In the classification problem, voting is done on each tree trained with different sampling for the final prediction result.
- In regression problems, the average of each obtained value is calculated.

Bagging

- "**Bagging**": **B**ootstrap **AGG**regat**ING**



Differences Between Bagging and Voting

- The greatest difference between the bagging and voting methods is whether to use multiple single algorithms or apply various algorithms to the same sample data set.
- In general, the bagging method is to train a single algorithm with different sampling data sets and perform voting. It has relatively better usability than the voting method because it uses a single algorithm. Thus, what is important is the hyperparameter of the single algorithm.
- Taking the Random Forest algorithm as an example again, it is suitable to obtain the baseline score as it requires simple hyperparameter setting such as how many trees to use (n_estimators), maximum depth (max_depth), and the minimum number of samples for splitting (min_samples_leaf).

Advantages of the Bagging Ensemble

- When using the bagging method, it can reduce variance compared to making a prediction with a single model. There are three major training errors in a model: variance, noise, and bias. (Of course, more significant factors include overfitting/underfitting and many different preprocessing issues, but assume they are already being set.)
- The ensemble method reduces variance, thus enhancing the performance of the final result.

sklearn.ensemble.BaggingClassifer/BaggingRegressor

- The sklearn library package provides the wrapper class called BaggingClassifier/BaggingRegressor.
- When designating the base algorithm to the base_estimator parameter, the BaggingClassifier/BaggingRegressor performs bagging ensemble.

Unit 8.

Ensemble Algorithm

- 8.1. The concept of Ensemble Algorithm and Voting
- 8.2. Bagging & Random Forest
- 8.3. Boosting

Boosting Ensemble: AdaBoost

About AdaBoost

- A sequence of weak learners such as trees
- A weighted ensemble of weak learners
  - One set of weights that increase the importance of the better-performing learners
  - One set of weights that increase the importance of the wrongly classified observations
- Similar pros and cons as the Random Forest

AdaBoost classification algorithm

- Let's suppose n observations for the training step: $\boldsymbol{x}_i$ and y_i.
  We also suppose that $y_i \in \{-1,+1\}$. (binary y)
- We will make a series of weak learners $G_m(\boldsymbol{x})$ with $m = 1,\ldots,M$.
- The ensemble classifier is made up of a linear combination of these weak learners:

$$G_{ensemble}(\boldsymbol{x}) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x})\right)$$

where α_m are the "boosting weights" that need to be calculated.

- Heavier weight is given to a better performing learner.

AdaBoost classification algorithm

1) For the first step (m=1), equal weight is assigned to the observations: $w_i^{(1)}=\frac{1}{n}$

2) For the boost sequence m=1,…, M:

a) Train the learner $G_m(\boldsymbol{x})$ using observations weighted by $w_i^{(m)}$.

b) Calculate the error ratio: $\varepsilon_m = \frac{\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G_m(\boldsymbol{x}_i))}{\sum_{i=1}^{n} w_i^{(m)}}$

⇨ $I(y_i \neq G_m(\boldsymbol{x}_i))$ gives 1 for an incorrect prediction, else 0.

⇨ $0 \leq \varepsilon_m \leq 1$

AdaBoost classification algorithm

3) For the boost sequence m=1,…, M:

c) Calculate the boosting weight: α_m

$$\alpha_m = \frac{1}{2} log\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$$

⇨ As $\varepsilon_m \to 0$, α_m is a large positive number. The learner is given more importance!

⇨ As $\varepsilon_m \cong 0.5$, $\alpha_m \cong 0$.

⇨ As $\varepsilon_m \to 1$, α_m is a large negative number.

d) For the next step, the weights of the **wrongly** predicted observation are rescaled by a factor $e^{a}{}_{m}$.

This can be compactly expressed as:

$$w_i^{(m+1)} = w_i^{(m)} \times e^{\alpha_m \cdot I\left(y_i \neq G_m(x_i)\right)}, \text{ where } i = 1, \ldots, n$$

⇨ In the next sequence step, the wrongly predicted observations receive heavier weight.

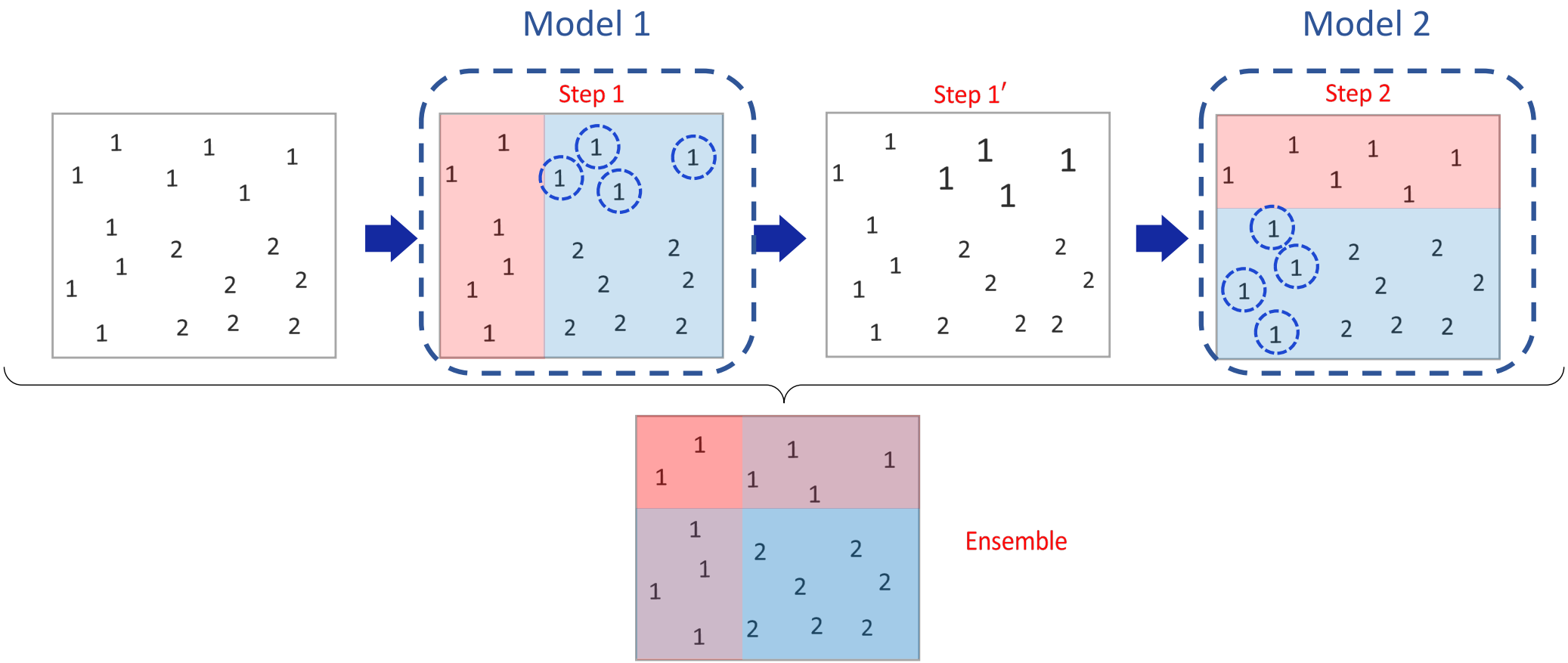
AdaBoost classification algorithm

4) The ensemble classifier is made up of a linear combination of weak learners:

$$G_{ensemble}(\boldsymbol{x}) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x})\right)$$

5) For a new testing condition $\boldsymbol{x}'$, we can predict y' by $G\_ensemble\ (\boldsymbol{x}')$.

AdaBoost classification algorithm



Scikit-Learn AdaBoostClassifier/Regressor Hyperparameters

Hyperparameter	Explanation
base_estimator	The base estimator with which the boosted ensemble is built
n_estimators	The maximum number of estimators at which boosting is terminated
learning_rate	The rate by which the contribution of each learner is shrunken
algorithm	Either 'SAMME' or 'SAMME.R'

- "base_estimator" is by default None, which means DecisionTreeClassifier(max_depth=1).
- More information can be found at:

  Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

  Regressor: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html

Boosting Ensemble: GBM & XGBoost

Gradient Boosting Machine (GBM)

- GBM is also made up of a sequence of weak learners similar to AdaBoost.
- The disagreement between the predicted $\hat{y}$ and the true y can be represented by a “loss” function.
- In the sequence of weak learners, the learner at step m, $G_m(\boldsymbol{x})$ can be obtained by moving the previous step learner $G_{m-1}(\boldsymbol{x})$ towards the direction that decreases the loss as defined above.
- These updating movements are restricted to a set of base learners.

Scikit-Learn GradientBoostingClassifier/Regressor Hyperparameters

Hyperparameter	Explanation
loss	The loss function
n_estimators	The number of boosting steps (weak learners)
learning_rate	The contribution of each weak learner
subsample	The fraction of data that will be used by individual weak learner

- Need to be tuned for optimized performance.
- More information can be found at:

  Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

  Regressor: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

Extreme Gradient Boosting (XGBoost)

- Improves upon GBM in the execution speed.
- More resistant to the overfitting than GBM.
- Not included in the Scikit-Learn library. Requires installation of the “xgboost” library.

XGBClassifier/Regressor Hyperparameters

Hyperparameter	Explanation
booster	gbtree or gblinear
n_estimators	The number of boosting steps (weak learners)
learning_rate	The contribution of each weak learner
subsample	The fraction of data that will be used by individual weak learner

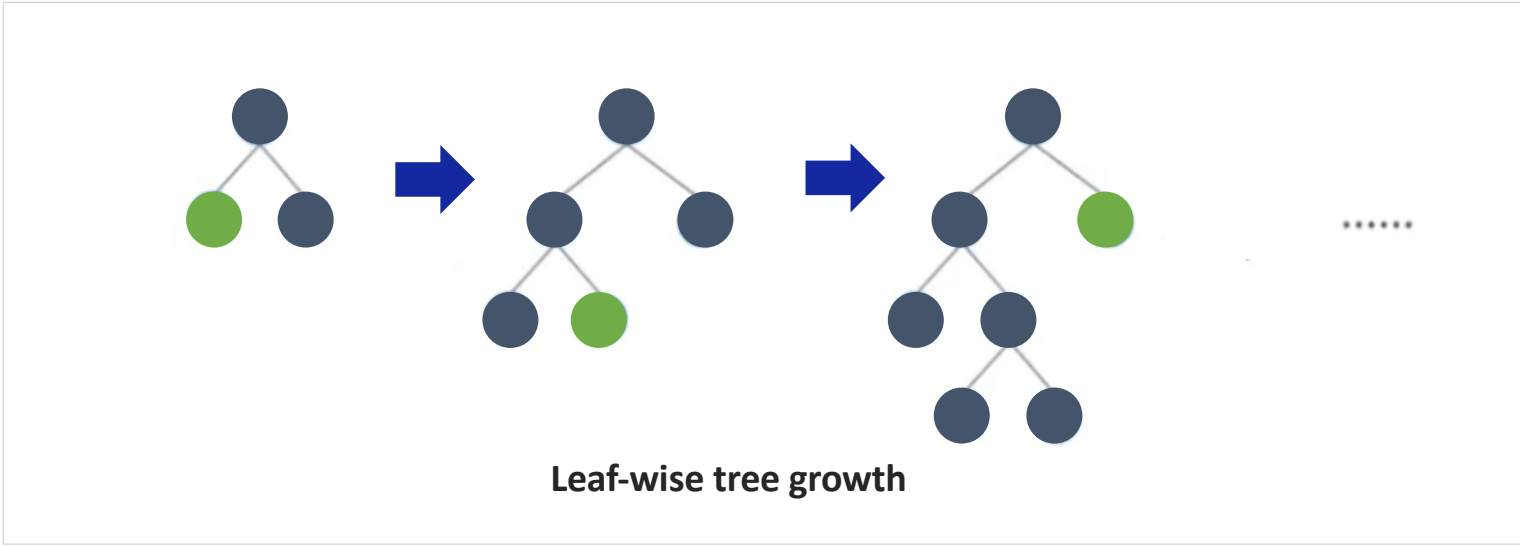
- Need to be tuned for optimized performance.
- More information can be found at: https://xgboost.readthedocs.io/en/latest/python/index.htm

XGBoost

- XGBoost is a library that implements the gradient boosting algorithm to be used in the distributed system.
- It supports both regression and classification problems. This popularly used algorithm features good performance and resource efficiency.
- Gradient boost is a representative algorithm using the boosting method. The XGBoost is the library using this algorithm to support parallel training.
- It has been recently used a lot due to its high performance and computer resource application rate. It has become more popular as it was frequently used by top rankers of Kaggle.

Light GBM

- While the tree is vertically expanded with Light GBM, other algorithms expand the tree horizontally.
  In other words, while the Light GBM is leaf-wise, other algorithms are level-wise.
- For expansion, the leaf with max delta loss is selected. When expanding the same leaf, the leaf-wise algorithm can reduce more loss than the level-wise algorithm.
- The following diagram shows how a boosting algorithm is implemented, which differs from the Light GBM.

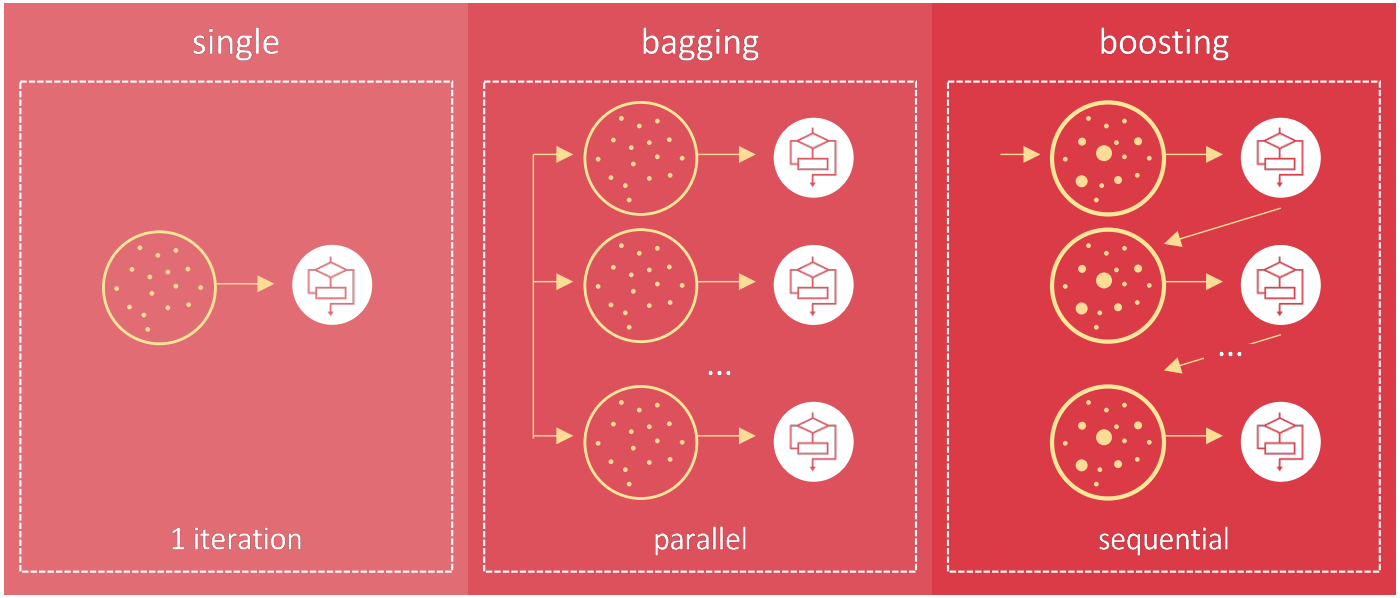


Light GBM operation method

Boosting

- The boosting algorithm is also ensemble learning. After sequentially learning weak learning machines, it supplements errors by adding weight to inaccurately predicted data from the previous learning.
- The difference from other ensemble methods is that it performs sequential learning and supplements errors by adding weight. However, one of the disadvantages is that it is difficult to process parallel due to its sequential property, leading to a longer learning time compared to other ensembles.

Single estimator, Bagging, Boosting



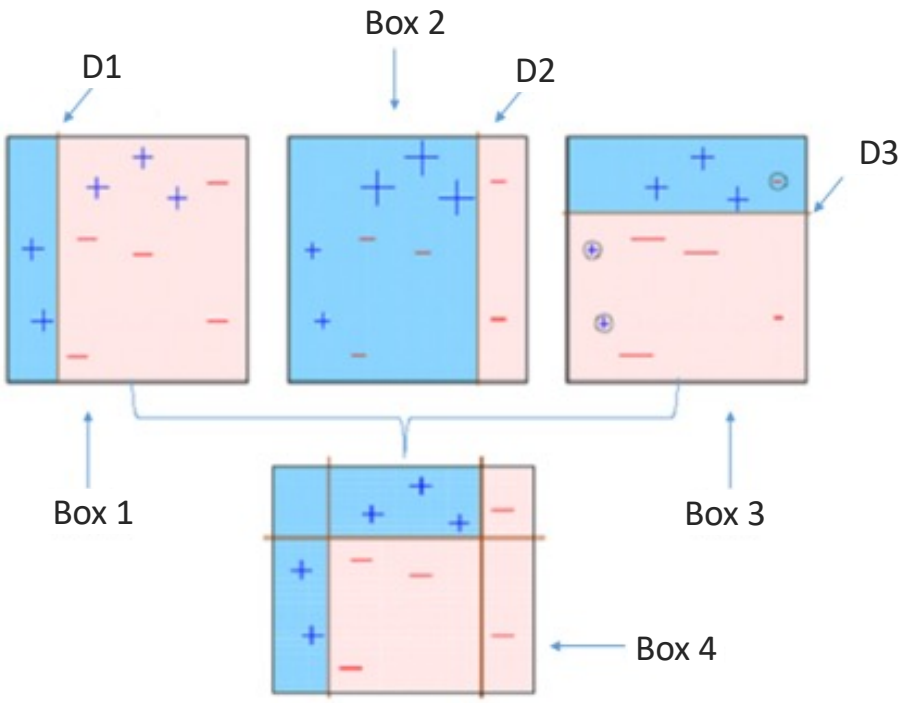
Learning a series of predictors by supplementing previous models

- AdaBoost
  - Train the first distributor (e.g., decision tree) in the training set and make a prediction.
  - Slightly increase the weight of the training sample with an inaccurately classified algorithm.
  - The second distributor uses the updated weight to be trained at the training set and makes a prediction again.
  - The weight is updated again, and the same process is repeated.
- Gradient boosting
  - Similar to AdaBoost, gradient boosting sequentially adds predictors to the ensemble to correct the previous errors.
  - However, instead of modifying the sample weight repeatedly as AdaBoost, it trains a new predictor to residual error created by the previous predictor.
  - For regression problems – gradient boosted regression tree (GBRT)

AdaBoost (Adaptive Boosting)

- AdaBoost (Adaptive Boosting) is an adaptive boosting method, one of the most representative boosting algorithms.
- As explained, the AdaBoost sequentially learns weak learning machines and supplements errors by applying weights to inaccurately predicted data.

Principle of the AdaBoost



- Box 1 results from a weak learning machine classified as the D1 sector. However, the error rate is relatively high because some data sets are expressed in + distributed in the red sector.
- The D2 line in Box 2 moves to the right to supplement the error rate from Box 1. Here, the data sets expressed in – are distributed in the blue sector for better performance, but it is not satisfactory yet.
- The D3 line in Box 3 is horizontally drawn on the top. However, the data set expressed in – is incorrectly classified.
- By training the previous Box 1, 2, and 3, Box 4 finds the most ideal combination. It shows much better performance compared to the previous three individual learning machines.

Principle of the AdaBoost

- How is the weight applied for combination?

Ex Suppose that the following weight will be applied to the performance of Box 1~3.

- Performance of Box 1: weight = 0.2
- Performance of Box 2: weight = 0.5
- Performance of Box 3: weight = 0.6

It can be expressed as the following formula:

```
0.2 * Box 1 + 0.5 + Box 2 + 0.6 * Box 3 = Box 4
```

Gradient Descent

- The key to the boosting method is to supplement errors from the previous learning.
- The AdaBoosting and gradient descent methods are slightly different in how to supplement errors.
- Gradient descent uses differentiation to minimize the difference between the predicted value and actual data.
  - Weight
  - Input_data = feature data (input data)
  - Bias
  - Y_actual = actual data value
  - Y_predict = predicted value
  - Loss = error rate
- Y_predict = weight*input_data+bias
  - The predictive value can be obtained from the above formula.
    Calculating the difference with actual data will result in the total error rate.
- Loss = Y_predict – Y_actual
  - (There are many different functional formulas to define error rate, including root means square error and means absolute error, but the above definition is provided for convenience.)
  - The purpose of gradient descent is to find the weight that makes the loss closest to 0.

Gradient Boosting Machine (GBM)

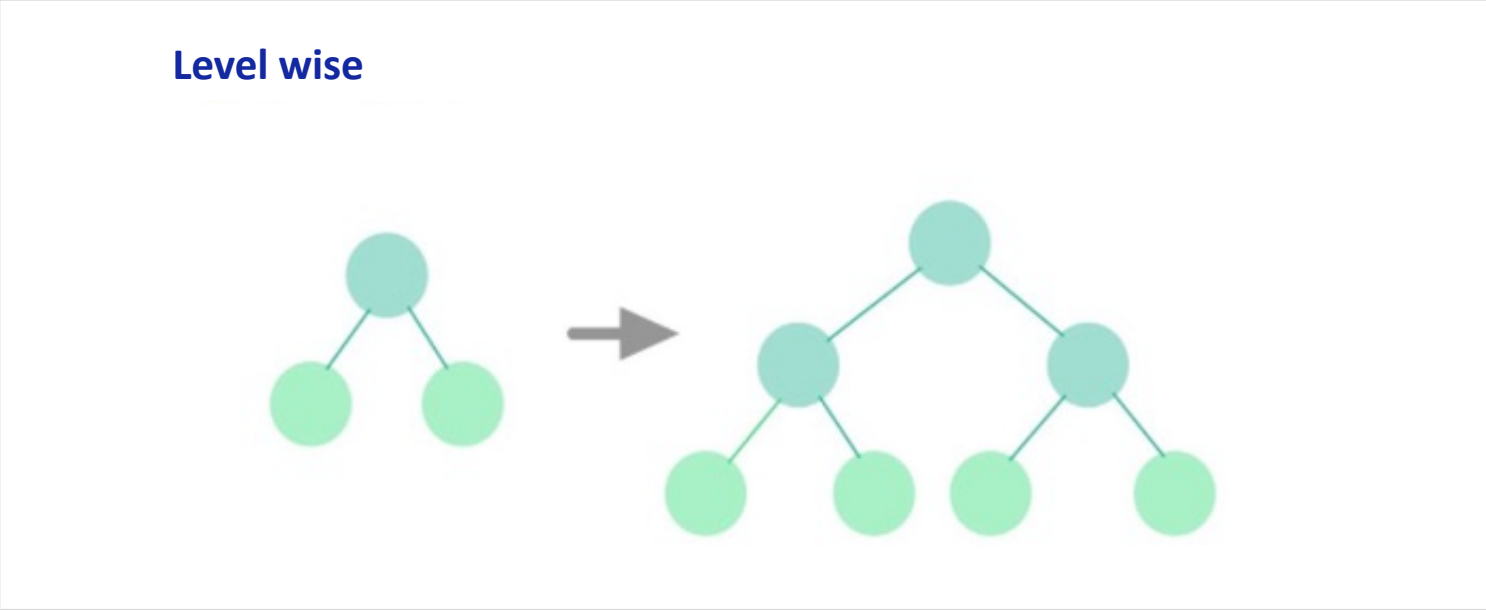
- The boosting algorithm in gradient descent is called gradient boosting machine, abbreviated as GBM. It is provided in the sklearn package and applicable to both classification and regression problems.
  - GradientBoostingClassifier
  - GradientBoostingRegressor
- It is extremely easy to apply in actual problems.
- XGBoost and LightGBM are two major ML packages mostly used in Kaggle.
- XGBoost and LightGBM are not provided in the existing sklearn package. However, the sklearn wrapper class allows an easy application for fit & predict similar to the ML class of the existing sklearn package.

Gradient Boosting Machine (GBM)

- Together with the XGBoost, LightGBM is the most highlighted boosting algorithm. Although XGBoost has an excellent performance, its learning time is too long.
- Advantages of the LightGBM
  - Short learning time
  - Relatively small memory use
  - Automatic conversion and optimal splitting of categorical features

Characteristics of the LightGBM

- The existing tree-based algorithm uses a level-wise method. It splits but maintains a balanced tree as much as possible, so the tree depth becomes minimum. A disadvantage of the level-wise method is that it takes time to make a balanced tree.
- On the other hand, LightGBM uses a leaf-wise method. It does not balance the tree but continuously splits the leaf node with the max data loss. The tree depth becomes greater, and an asymmetrical tree is created. The repetition of the leaf node with the max data loss minimizes predicted error loss than splitting a balanced tree.



Hyperparameter tuning methods

- The increased Num_leaves value raises accuracy. However, it also increases the tree depth and makes the model complex, thus increasing the probability of overfitting.
- Min_data_in_leaf is a significant parameter for overfitting. It depends on the Num_leaves and the size of training data, but in general, it prevents a deeper tree when setting it as a higher value.
- Max_depth constraints the depth size and improves overfitting when combined with the two parameters above.

XGBoost

Parameter	Default value	Description
num_iterations	100	Designates the number of trees for repetitive work. Overfitting occurs if the value is too high.
learning_rate	0.1	Updated when boosting steps are repeatedly conducted. The value is designated between 0~1.
max_depth	1	Identical to the max_depth of tree-based algorithms. No restriction is applied to tree depth when entering a value smaller than 0.
min_data_in_leaf	20	Identical to the min_samples_leaf of a decision tree. Used as a parameter to control overfitting.
num_leaves	3	Signifies the max. The number of leaves for one tree.
boosting	GBDT	
bagging_fraction	1.0	Designates the ratio for data sampling. Used to control overfitting.
feature_fraction	1.0	The ratio of the random feature for individual tree learning
Lambda_l1	0.0	The value for L1 regulation
Lambda_l2	0.0	The value for L2 regulation


End of
Document

SAMSUNG