Popular Machine Learning Methods: Idea, Practice and Math

Part 2, Chapter 2, Section 5: Tree Based Models

Yuxiao Huang

Data Science, Columbian College of Arts & Sciences
George Washington University

Fall 2020

Reference

- This set of slices was largely built on the following 7 wonderful books and a wide range of fabulous papers:

HML	Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition)
PML	Python Machine Learning (3rd Edition)
ESL	The Elements of Statistical Learning (2nd Edition)
PRML	Pattern Recognition and Machine Learning
NND	Neural Network Design (2nd Edition)
LFD	Learning From Data
RL	Reinforcement Learning: An Introduction (2nd Edition)

- For most materials covered in the slides, we will specify their corresponding books and papers for further reference.

Code Example & Case Study

- See related code example in github repository: /p2_c2_s5_tree_based_models/code_example
- See related case study of Kaggle Competition in github repository: /p2_c2_s5_tree_based_models/case_study

Table of Contents

Learning Objectives: Expectation

- It is expected to understand
  - the idea of Decision Tree and Decision Tree Learning algorithm
  - the idea of Information Gain
  - the idea of Gini Impurity and Entropy
  - the idea of Ensemble Learning
  - the idea of Bagging, Random Forest and its learning algorithm
  - the idea of Boosting, Gradient Boosting and its learning aglorithm
  - the idea of Early Stopping
  - the good practice for using sklearn Decision Tree, Random Forest and Gradient Boosting

Learning Objectives: Recommendation

- It is recommended to understand
  - the math of Decision Tree Learning algorithm
  - the math of Information Gain
  - the math of Gini Impurity and Entropy

Toy Dataset

Table 1: Toy dataset.

Day	Weather	Activity
Weekday	Sunny	Work
Weekday	Cloudy	Work
Weekday	Rainy	Work
Weekend	Sunny	Hike
Weekend	Cloudy	Jog
Weekend	Rainy	Read

- The toy dataset has two features:
  - Day, with two values: Weekday and Weekend
  - Weather, with three values: Sunny, Cloudy and Rainy
- The target of the dataset is Activity, which has four classes:
  - Work
  - Hike
  - Jog
  - Read

A Corresponding Decision Tree

Table 1: Toy dataset.

Day	Weather	Activity
Weekday	Sunny	Work
Weekday	Cloudy	Work
Weekday	Rainy	Work
Weekend	Rainy	Read
Weekend	Cloudy	Jog
Weekend	Sunny	Hike

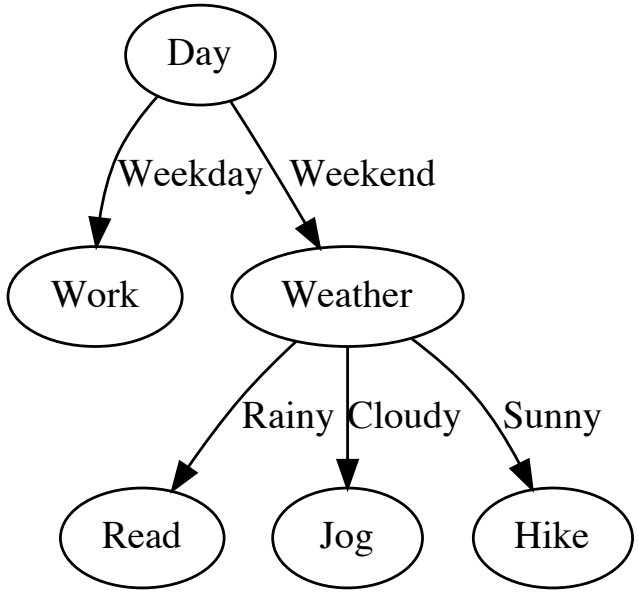


Figure 1: A Corresponding Decision Tree.

- A *Decision Tree* is a flowchart like tree structure.
- In its simplest form:
  - each internal node (i.e., non-leaf node) represents a feature
  - each branch represents a unique value of the feature
  - each leaf node represents a class
- Fig. 1 shows the decision tree corresponding to the toy dataset in table 1.

Prediction

Table 2: A test sample.

Day	Weather	Activity
Weekend	Sunny	?

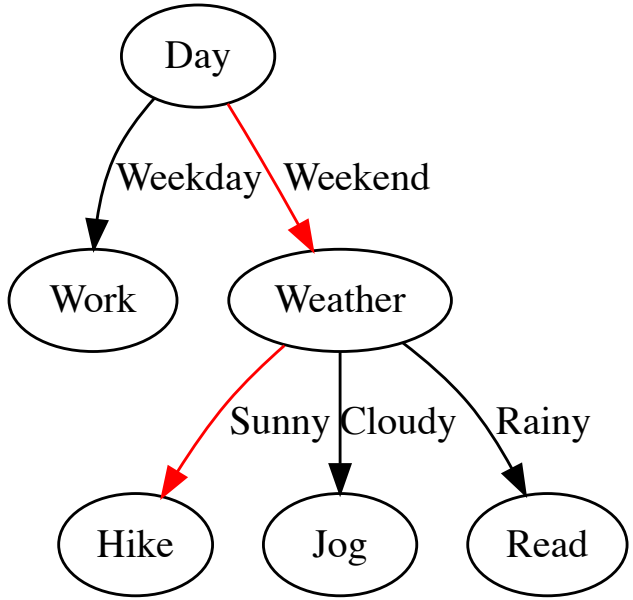


Figure 2: The corresponding decision rule (highlighted in red).

- Once we have the decision tree, we can literally walk along a path (starting from the root node and ending at a leaf node) in the tree to make predictions.
- Fig. 2 shows such a path corresponding to the test sample in table 2.
- Such path is also referred to as *Decision Rule*.
- The decision rule in fig. 2 (highlighted in red) can be written as

$$(\text{Day} = \text{Weekend}) \wedge (\text{Weather} = \text{Sunny}) \rightarrow \text{Activity} = \text{Hike}. \tag{1}$$

Interpretability of Decision Tree

Table 2: A test sample.

Day	Weather	Activity
Weekend	Sunny	?

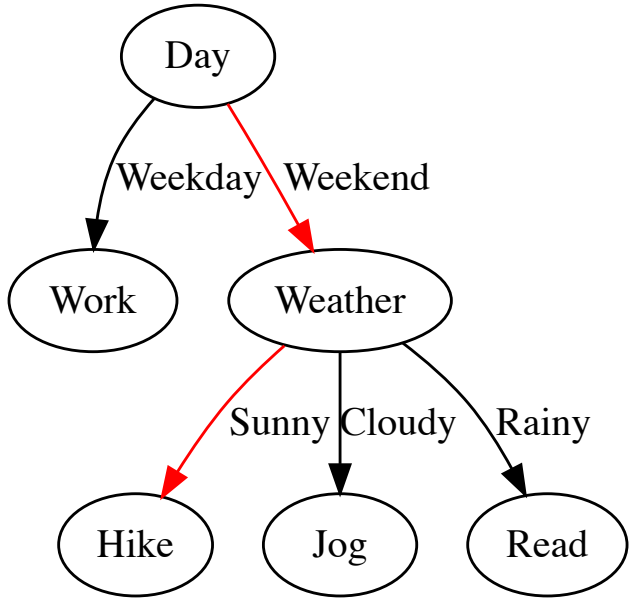


Figure 2: The corresponding decision rule (highlighted in red).

- As shown in fig. 2, a very nice property of decision tree is that, compared to most models (particularly Deep Neural Networks), decision tree is much easier to interpret.
- Because of this, decision tree has been widely used in areas (e.g., law and banking), where interpretability of the model is paramount.

Takeaway

- Decision tree has been widely used in areas (e.g., law and banking) where interpretability of the model is paramount.

Sklearn DecisionTreeClassifier: Code Example

- See /p2_c2_s5_tree_based_models/code_example/:
  1. cells 50 to 53
  2.

Sklearn DecisionTreeClassifier: Decision Tree

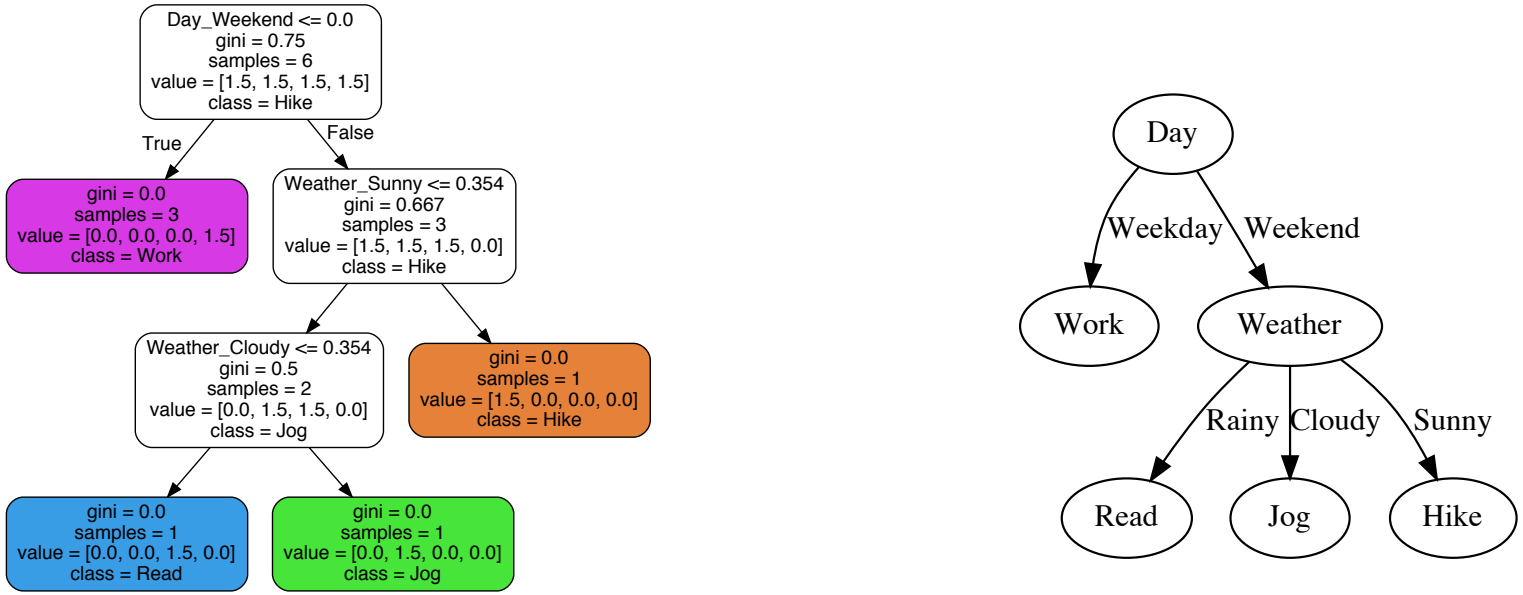


Figure 4: The decision tree produced by sklearn DecisionTreeClassifier and the decision tree in fig. 1.

- The left panel in fig. 4 shows the decision tree produced by sklearn DecisionTreeClassifier:
  - each intermediate node represents a test of the feature-value pair (e.g., Day_Weekend in the root)
  - each edge represents an outcome of a test (i.e., True or False)
- The right panel in fig. 4 shows the decision tree in fig. 1:
  - each intermediate node represents a feature (e.g., Day in the root)
  - each edge represents a value of a feature (e.g., Weekday or Weekend)
- Although the two trees have different structures, they model the exact same decision rules.

Training Decision Tree

- While using decision tree for prediction is straightforward, training decision tree is not that simple.
- There are two widely used methods for training decision tree:
  - CART [Breiman et al., 1984]: this is the method used in sklearn DecisionTreeClassifier
  - C4.5 [Quinlan, 2014]: this has been updated to c5.0
- For illustration purposes, here we will introduce one of the most simplest (hence much less powerful) method, named *Decision Tree Learning* algorithm (DTL).

Decision Tree Learning Algorithm

Decision tree learning

Aim: find a small tree consistent with the training examples

Idea: (recursively) choose "most significant" attribute as root of (sub)tree

function DTL(*examples, attributes, default*) **returns** a decision tree

  if *examples* is empty **then return** *default*
  else if all *examples* have the same classification **then return** the classification
  else if *attributes* is empty **then return** MODE(*examples*)
  else
    $best \leftarrow$ CHOOSE-ATTRIBUTE(*attributes*, *examples*)
    $tree \leftarrow$ a new decision tree with root test *best*
    for each value v_i of *best* **do**
      $examples_i \leftarrow$ {elements of *examples* with $best = v_i$}
      $subtree \leftarrow$ DTL($examples_i$, $attributes - best$, MODE(*examples*))
      add a branch to *tree* with label v_i and subtree *subtree*
    return *tree*

Figure 5: Decision tree learning algorithm. Picture courtesy of *Artificial Intelligence: A Modern Approach (Third edition)*.

Training Decision Tree using DTL

- Assume that when we apply DTL on the toy dataset in table 1, the best feature returned by function `CHOOSE-ATTRIBUTE` is Day.
- **Q:** Can you find the decision tree produced by DTL?

Table 1: Toy dataset.

Day	Weather	Activity
Weekday	Sunny	Work
Weekday	Cloudy	Work
Weekday	Rainy	Work
Weekend	Sunny	Hike
Weekend	Cloudy	Jog
Weekend	Rainy	Read

Training Decision Tree using DTL

- Assume that when we apply DTL on the toy dataset in table 1, the best feature returned by function `CHOOSE-ATTRIBUTE` is Day.
- **Q:** Can you find the decision tree produced by DTL?
- **A:** The decision tree is exactly the same as that in fig. 1.

Table 1: Toy dataset.

Day	Weather	Activity
Weekday	Sunny	Work
Weekday	Cloudy	Work
Weekday	Rainy	Work
Weekend	Sunny	Hike
Weekend	Cloudy	Jog
Weekend	Rainy	Read

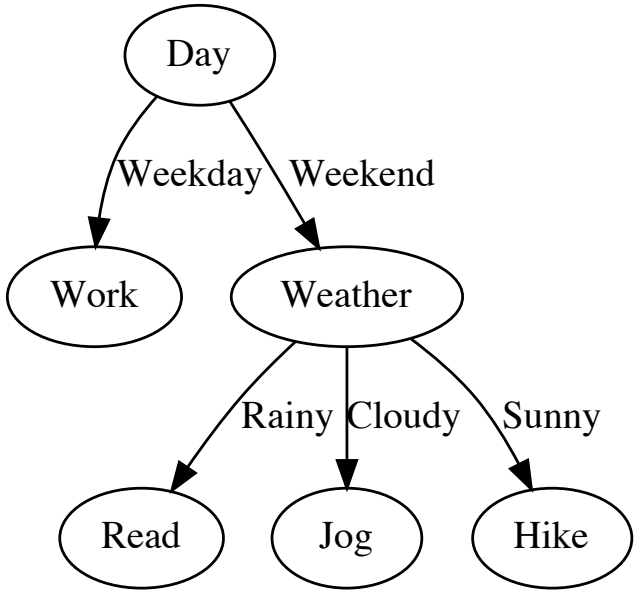


Figure 1: A Corresponding Decision Tree.

The Best Feature

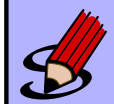
- When applying DTL to train decision tree on the toy dataset, we assumed that the best feature returned by function `CHOOSE-ATTRIBUTE` is Day.
- To see whether this is the case and, more importantly, what makes a feature the best feature, we need to introduce the loss function for decision tree.

The Loss Function

- The loss function with respect to data $\mathbf{D}$ and feature $\mathbf{x}$, $\mathcal{L}(\mathbf{D}, \mathbf{x})$, is defined as

$$\mathcal{L}(\mathbf{D}, \mathbf{x}) = I(\mathbf{D}) - \sum_{i=1}^{m} \frac{|\mathbf{D}_i|}{|\mathbf{D}|} I(\mathbf{D}_i). \tag{2}$$

  Here:
  - $\{x_1, \ldots, x_m\}$ are the unique values of feature $\mathbf{x}$ (with m being the number of unique values)
  - $\mathbf{D}_i$ is the subset of data $\mathbf{D}$ produced by splitting $\mathbf{D}$ using feature $\mathbf{x}$ and value x_i (i.e., $\mathbf{x} = x_i$ in $\mathbf{D}_i$)
  - $|\mathbf{D}|$ and $|\mathbf{D}_i|$ are the number of samples in $\mathbf{D}$ and $\mathbf{D}_i$
  - $I(\mathbf{D})$ and $I(\mathbf{D}_i)$ are the *Impurity* of $\mathbf{D}$ and $\mathbf{D}_i$ (more on this later)
- Eq. (2) says that, the loss is the difference between the impurity of data $\mathbf{D}$ and the weighted sum of the impurity of each subset $\mathbf{D}_i$.
- The loss in eq. (2) is also referred to as the *Information Gain* by splitting data $\mathbf{D}$ using feature $\mathbf{x}$.
- The best feature is then defined as the one that maximizes the information gain.
- To see why this makes sense, we need to introduce the measurement for impurity.



Takeaway

- The best feature in a decision tree is the one that maximizes information gain.

Measurement for Impurity

- There are two widely used methods for measuring the impurity of data:
  - Gini impurity
  - Entropy
- The two measurements are similar in that, both of them aim to capture the imbalance of the data:
  - the more imbalanced the data, the lower the value of the two measurements
  - the less imbalanced the data, the higher the value of the two measurements
  - the two measurements reach the maximum when the data has uniform class distribution (i.e., the data is balanced)
- The two measurements are different in that, they use different ways to capture the imbalance of the data.

Gini impurity

- The gini impurity of data $\mathbf{D}$, $I_g(\mathbf{D})$, is defined as

$$I_g(\mathbf{D}) = \sum_{k=1}^{c} p(y_k|\mathbf{D})(1 - p(y_k|\mathbf{D})) = 1 - \sum_{k=1}^{c} p(y_k|\mathbf{D})^2. \tag{3}$$

  Here:
  - $\{y_1, \ldots, y_c\}$ are the unique classes in data $\mathbf{D}$ (with c being the total number of unique classes)
  - $p(y_k|\mathbf{D})$ is the proportion of class y_k in data $\mathbf{D}$
- Eq. (3) says that, the gini impurity of data $\mathbf{D}$ is 1 minus the sum of the squared proportion across each class.
- Based on eq. (3), we can prove that the gini impurity:
  - reaches the minimum (i.e, 0) if data $\mathbf{D}$ has only 1 class (i.e., $c = 1$)
  - reaches the maximum (i.e., $1 - \frac{1}{c}$) if data $\mathbf{D}$ has uniform class distribution (i.e., the data is balanced)
- For example, in a binary class setting the gini impurity:
  - reaches the minimum (i.e., 0) if $p(0|\mathbf{D}) = 1$ or $p(1|\mathbf{D}) = 1$
  - reaches the maximum (i.e., 0.5) if $p(0|\mathbf{D}) = p(1|\mathbf{D}) = 0.5$

Entropy

- The entropy of data $\mathbf{D}$, $I_h(\mathbf{D})$, is defined as

$$I_h(\mathbf{D}) = -\sum_{k=1}^{c} p(y_k|\mathbf{D}) \log_2 p(y_k|\mathbf{D}). \tag{4}$$

- Here:
  - $\{y_1, \ldots, y_c\}$ are the unique classes in data $\mathbf{D}$ (with c being the total number of unique classes)
  - $p(y_k|\mathbf{D})$ is the proportion of class y_k in data $\mathbf{D}$
- Eq. (4) says that, the entropy of data $\mathbf{D}$ is the multiplicative inverse of the sum of the product of the proportion and log proportion across each class.
- Based on eq. (4), we can prove that the entropy:
  - reaches the minimum (i.e, 0) if data $\mathbf{D}$ has only 1 class (i.e., $c = 1$)
  - reaches the maximum (i.e., $\log c$) if data $\mathbf{D}$ has uniform class distribution (i.e., the data is balanced)
- For example, in a binary class setting the entropy:
  - reaches the minimum (i.e., 0) if $p(0|\mathbf{D}) = 1$ or $p(1|\mathbf{D}) = 1$
  - reaches the maximum (i.e., 1) if $p(0|\mathbf{D}) = p(1|\mathbf{D}) = 0.5$

Gini Impurity VS Entropy

- While gini impurity and entropy measure impurity differently, they usually yield very similar results.
- Scikit learn DecisionTreeClassifier uses gini impurity as the default setting for hyperparameter `criterion`.

Good practice

- Scikit learn DecisionTreeClassifier uses gini impurity as the default setting for hyperparameter `criterion`.

Defining the Best Feature: Revisit

- Earlier we claimed that the best feature is the one that maximizes the information gain, given in eq. (2)

$$\mathcal{L}(\mathbf{D}, \mathbf{x}) = I(\mathbf{D}) - \sum_{i=1}^{m} \frac{|\mathbf{D}_i|}{|\mathbf{D}|} I(\mathbf{D}_i). \tag{2}$$

- Since for different features, the impurity of data D, $I(\mathbf{D})$, is the same, then the best feature is also the one that minimizes the weighted sum of the impurity of each subset $\mathbf{D}_i$, $\sum_{i=1}^{m} \frac{|\mathbf{D}_i|}{|\mathbf{D}|} I(\mathbf{D}_i)$.
- Then the question why maximizing the information gain makes sense becomes why minimizing the weighted sum of the impurity of each subset makes sense.

Defining the Best Feature: Revisit

Table 1: Toy dataset.

Day	Weather	Activity
Weekday	Sunny	Work
Weekday	Cloudy	Work
Weekday	Rainy	Work
Weekend	Sunny	Hike
Weekend	Cloudy	Jog
Weekend	Rainy	Read

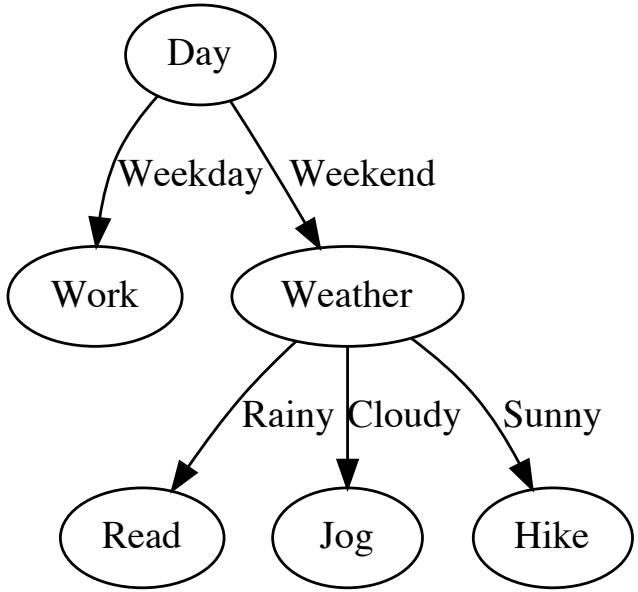
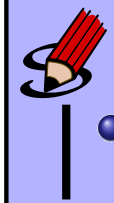


Figure 1: A Corresponding Decision Tree.

- Intuitively, the best feature should be the one that has the strongest predictive power over the target:
  - ideally, when using the best feature (e.g, Day in fig. 1) to split the data (e.g, the toy dataset in table 1), we want each subset to have only one class
  - this is because the feature-value pair can then deterministically predict the class (e.g., Day-Weekday can guarantee class Work)
- As discussed earlier, no matter how we measure impurity (gini or entropy), data with only one class always has the lowest impurity.
- This is why minimizing the weighted sum of the impurity of each subset makes sense.

Feature Importance

- As discussed earlier, a very nice property of decision tree is that, compared to most models, decision tree is much easier to interpret.
- This is partially due to the decision rules modeled by decision tree.
- Another key reason is that, decision tree can detect feature's relative predictive power over the target.
- Such relative predictive power is sometimes referred to as *Feature Importance* (a.k.a., *Variable Importance*).
- In sklearn DecisionTreeClassifier, feature importance is defined as the normalized information gain with respect to the feature.
- Feature importance has been widely used for tasks such as feature selection, where:
  - features with high importance are kept in the data
  - features with low importance are removed from the data



Takeaway

- Decision tree can detect feature importance, which can be used for feature selection.

Feature Importance: Code Example

- See /p2_c2_s5_tree_based_models/code_example/:
  1. cells 54 and 55

Problem of Decision Tree

- Decision tree tends to go too deep, resulting in overfitting.
- A popular way to limit the depth of scikit-learn `DecisionTreeClassifier` is using the following hyperparameters:
  - `max_depth`: the maximum depth of the tree
  - `min_samples_split`: the minimum number of samples required to split an internal node
  - `min_samples_leaf`: the minimum number of samples required to be at a leaf node

Limiting the Depth: Code Example

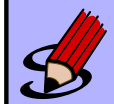
- See /p2_c2_s5_tree_based_models/code_example/:
  1. cell 68

Ensemble

- A group of predictors is called an *Ensemble*.
- The idea of ensemble is that, a group of weak models could be more accurate than a single strong model (possibly inspired by *The Wisdom of Crowds*).
- There are some of the most popular ensemble methods:
  - Bagging
  - Boosting
  - Stacking
- Here we will focus on Bagging and Boosting, since they lead to two of the most widely used shallow models:
  - Random Forest
  - Gradient Boosting
- See a very nice discussion of Stacking in HML: Chap 7.

Bagging

- The goal of bagging is training an ensemble of independent models.
- That is, the result of one model does not depend on the result of the other models.
- Because the models are independent:
  - we can use bagging to address overfitting (as mentioned in /p2_c3_s2_training_shallow_models)
  - we can train bagging in parallel so bagging is usually fast
  - generally the more models we used in bagging the better the performance of bagging
  - as a result, we do not have to use regularization to fine-tune the number of models in bagging



Takeaway

- We can use bagging to address overfitting.
- We can train bagging in parallel so bagging is usually fast.
- Generally the more models we use in bagging the better the performance of bagging.
- We do not have to use regularization to fine-tune the number of models in bagging.

The Power of Bagging

- Here is an example illustrating the power of bagging:
  - assume we have:
    - an ensemble of n independent weak models, each of which with accuracy 51%
    - a single strong decision tree with accuracy 95%
  - if we use the ensemble of n independent weak models for prediction, where the predicted class is obtained by the majority vote (i.e., the mode), then the accuracy of bagging can be calculated as:

$$\text{Bagging accuracy} = \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} \left(\binom{n}{i} \times (51\%)^i \times (49\%)^{(n-i)} \right) \tag{5}$$

  - based on eq. (5) we have the following bagging accuracy:
    - 0.54, when $n = 10^2$
    - 0.73, when $n = 10^3$
    - 0.98, when $n = 10^4$
  - that is:
    - the bagging accuracy increases when n increases
    - the ensemble of independent weak models can be more accurate than the single strong model when the ensemble is large enough

The Power of Bagging

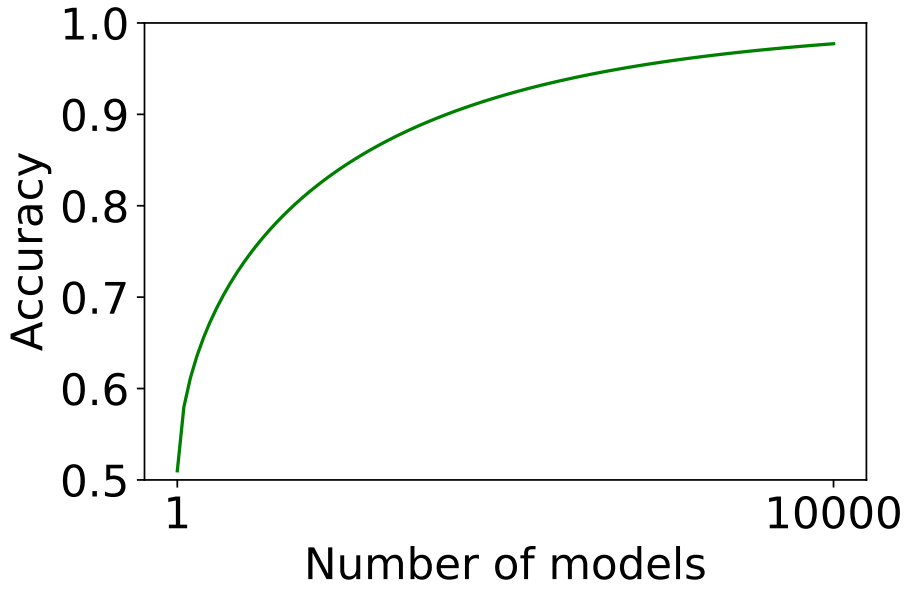


Figure 6: The accuracy of bagging as a function of the number of independent weak models (with accuracy 0.51) in the ensemble.

- As fig. 6 shows, generally the more models we use in bagging the better the performance of bagging.
- This is the reason why we tend to use a relatively large number of models for bagging (when computational cost permits).

Takeaway

- When computational cost permits, it is recommended to use a relatively large number of models for bagging.

Random Forest

- The most widely used bagging is *Random Forest* which, as the name suggests, is an ensemble of k independent decision trees.
- Concretely, we train each of the k decision trees in the following steps:
  1. randomly select a samples from the training set with replacement to train the decision tree
  2. repeat the following steps:
     - 2.1 randomly select b features without replacement
     - 2.1 select the best feature (that maximizes the information gain) from the a samples obtained in step 1 and b features obtained in step 2.1
- Here:
  - the larger the k, the better the performance of random forest, and the higher the computational cost
  - a is usually the same as m (with m being the number of samples in the training data)
  - b is usually the same as $\sqrt{n}$ (with n being the number of features in the training data)
- Once random forest is trained, the independent decision trees make predictions using the mean (for regression) or majority vote (for classification).

Sklearn RandomForestClassifier: Code Example

- See /p2_c2_s5_tree_based_models/code_example/:
  1. cells 56 to 59

Feature Importance

- Similar to decision tree, random forest can also provide feature importance, which captures feature's relative predictive power over the target.
- In sklearn RandomForestClassifier, feature importance is calculated as follows:
  1. calculate the weighted average of the information gain with respect to the feature across all the decision trees that contain the feature, where the weight for each decision tree is the number of samples in the data which was split by the feature in the decision tree
  2. the importance of the feature is then the weighted average calculated in step 1 normalized across all the features (so that the sum of importance across all the features equals 1)
- Similar to the reason why random forest is more accurate than decision tree, the feature importance provided by random forest is more accurate than that provided by decision tree.
- As a result, feature importance provided by random forest is much more widely used than that provided by decision tree.

Takeaway

- Feature importance provided by random forest is much more widely used than that provided by decision tree.

Feature Importance: Code Example

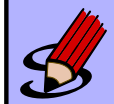
- See /p2_c2_s5_tree_based_models/code_example/:
  1. cells 58 and 59

Further Reading

- Besides decision tree and random forest, many methods have been proposed to provide feature importance.
- One of the most widely used method is Shapley Additive explanations (SHAP) [Lundberg and Lee, 2017], which is a unified framework for interpreting predictions of any (shallow or deep) model.

Boosting

- Unlike bagging which aims to train an ensemble of independent models, the goal of boosting is training an ensemble of dependent models.
- That is, the result of one model depends on the result of the other models.
- Because the models are dependent:
  - we can use boosting to handle underfitting (as mentioned in /p2_c3_s2_training_shallow_models)
  - we cannot train boosting in parallel so boosting can be much slower than bagging
  - generally when the number of models used in boosting increases, the performance of boosting first increases then decreases (due to overfitting)
  - as a result, we should use regularization to fine-tune the number of models in boosting (more on this later)



Takeaway

- We can use boosting to address underfitting.
- We cannot train boosting in parallel so boosting can be much slower than bagging.
- Generally when the number of models used in boosting increases, the performance of boosting first increases then decreases.
- We should use regularization to fine-tune the number of models in boosting.

The Idea of Boosting

- The idea of boosting is adding models sequentially to the ensemble in such a way that, the new models correct the error of the old models.
- There are two of the most popular boosting methods:
  - Adaptive Boosting (a.k.a., AdaBoost)
  - Gradient Boosting
- Here we will focus on Gradient Boosting, since it is much more powerful than AdaBoost and hence much more widely used.
- As a matter of fact, Gradient Boosting is regarded as the most accurate model for shallow learning (e.g., it is the winning model for many Kaggle competations).
- See a nice discussion of AdaBoost in HML: Chap 7.

Gradient Boosting

- *Gradient Boosting* is an ensemble where the k^{th} newly added model fits the *Residual* (i.e., prediction error) of the $(k-1)^{\text{th}}$ added model.
- Concretely, we train gradient boosting in the following steps:
  1. fit the 1st model on feature $\mathbf{X}$ and target $\mathbf{y}$, generate prediction $\widehat{\mathbf{y}}_1$
  2. for k from 2 to n (with n being the number of models in the ensemble):
     - fit the k^{th} model on feature $\mathbf{X}$ and target $\mathbf{y} - \sum_{i=1}^{k-1} \widehat{\mathbf{y}}_i$, generate prediction $\widehat{\mathbf{y}}_k$
- Here:
  - for regression, $\widehat{\mathbf{y}}_1$ is the predicted target value
  - for classification, $\widehat{\mathbf{y}}_1$ is the predicted probability distribution of the classes (using softmax)
- Once gradient boosting is trained, the dependent models make the final prediction, $\widehat{\mathbf{y}}$, using the sum of their individual prediction, $\widehat{\mathbf{y}}_1, \ldots, \widehat{\mathbf{y}}_n$:

$$\widehat{\mathbf{y}} = \sum_{i=1}^{n} \widehat{\mathbf{y}}_i \tag{6}$$

The Power of Gradient Boosting

- Here is an example illustrating the power of gradient boosting:
  - assume we have two kinds of model to predict target value 1:
    - a single strong model whose prediction is 0.95
    - an ensemble of n dependent weak models, where the 1st model randomly picks a number between $[0, 1)$, $\widehat{\mathbf{y}}_1$, and the k^{th} model $(2 \leq k \leq n)$ randomly picks a number between 0 and $\mathbf{y} - \sum_{i=1}^{k-1} \widehat{\mathbf{y}}_i$, $\widehat{\mathbf{y}}_k$
  - if we use the ensemble of n dependent weak models for prediction, where the predicted value is the sum of their individual predicted value, $\widehat{\mathbf{y}}_1, \ldots, \widehat{\mathbf{y}}_n$, then the ensemble may have the following predictive value:
    - 0.37, when $n = 1$
    - 0.97, when $n = 2$
    - 0.99, when $n = 3$
  - that is:
    - the prediction of boosting approaches the target value when n increases
    - when the ensemble is large enough, the prediction of an ensemble of dependent weak models can be much closer to the target value (compared to the prediction of the single strong model)

The Power of Gradient Boosting

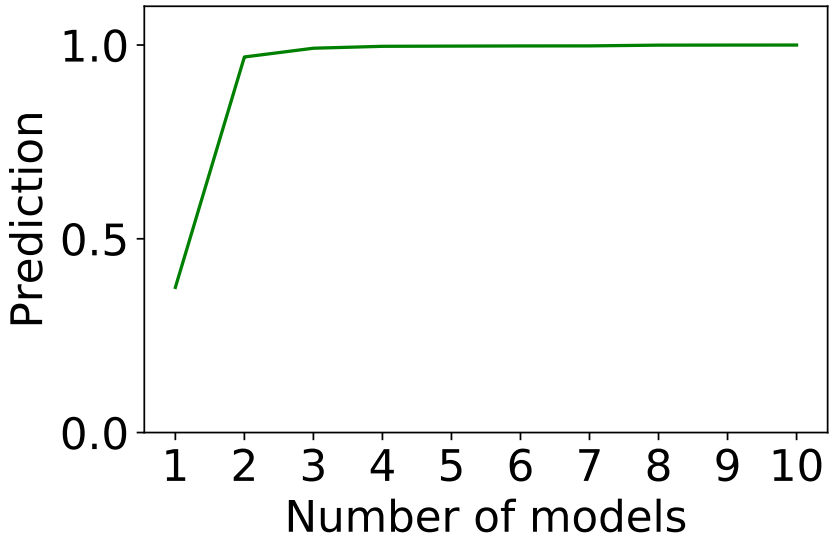


Figure 7: The prediction of boosting as a function of the number of dependent weak models in the ensemble.

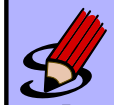
- As fig. 7 shows, generally the more models we use in boosting the closer the prediction to the target value (and hence the smaller the training MSE).
- However, as discussed in /p2_c3_s2_training_shallow_models, a model with low training MSE could be overfitting and, in turn, generalize poorly in reality.

Takeaway

- Unlike bagging, using too many models in boosting could lead to overfitting.

Early Stopping

- As mentioned in /p2_c3_s2_training_shallow_models, besides Lasso, Ridge and Elastic Net, another popular regularization method is *Early Stopping*.
- The idea of early stopping is, as the name suggests, stopping training early to avoid overfitting.
- Concretely, early stopping entails:
  - monitor the validation loss when training proceeds
  - stop training when the validation loss has not improved for a predetermined period
  - call back the model corresponding the lowest validation loss
- For boosting, early stopping allows us to fine-tune the number of models in boosting:
  - monitor the validation loss when adding models to boosting
  - stop training when the validation loss has not improved for a predetermined period
  - call back the boosting corresponding the lowest validation loss



Takeaway

- Early stopping allows us to fine-tune the number of models in boosting.

Sklearn HistGradientBoostingClassifier: Code Example

- See /p2_c2_s5_tree_based_models/code_example/:
  1. cells 60 and 61

Bibliography I


Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984).
Classification and Regression Trees.
Wadsworth and Brooks, Monterey, CA.


Lundberg, S. M. and Lee, S. I. (2017).
A Unified Approach to Interpreting Model Predictions.
In *Advances in Neural Information Processing Systems*, pages 507–514.


Quinlan, J. R. (2014).
C4. 5: Programs for Machine Learning.
Elsevier.