Lecture 21: Combining Different Models for Ensemble Learning - Bagging

Dr. Huiping Cao

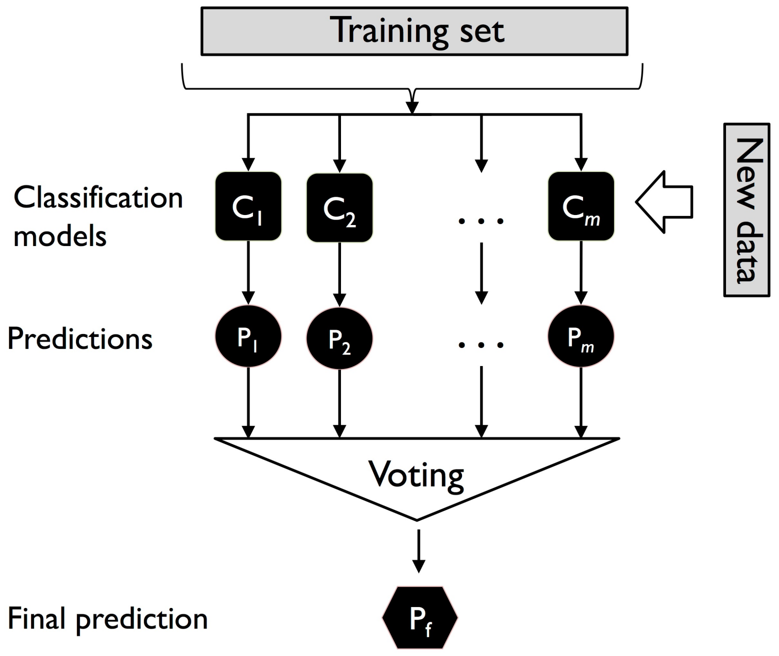
Basics

- The goal of **ensemble methods** is to combine different classifiers to a meta-classifier that has better generalization performance than each individual classifier alone.
- A basic **perception** of how ensembles work
  - Assume that we collected predictions from 10 experts, ensemble methods will strategically combine those predictions by the 10 experts to come up with a prediction that is more accurate and robust than the predictions by each individual expert.

Majority voting principle

- Most popular ensemble methods use the **majority voting** principle.
    - Majority voting means that we select the class label that has been predicted by the majority of classifiers.
    - Majority voting refers to binary class settings.
    - In the multi-class setting, this is called **plurality voting**, in which we select the class label receiving the most votes (mode).
        - The **mode** is the most frequent event or result in a set. E.g., mode{1,2,3,1}=1

General ensemble approach using majority voting



Workflow

- To **predict** a class label using majority voting or plurality voting, we can combine the predicted class labels of each individual classifier and select the class label that receives the most votes.
  - $\hat{y} = mode\{C_1(\mathbf{x}), C_2(\mathbf{x}), \dots, C_m(\mathbf{x})\}$

Why do ensemble methods work better than individual classifiers

- The error probability of an ensemble is always better than the error of an individual base classifier.
    - Two assumptions
        - All m-base classifiers for a binary classification task have an equal error rate ε.
        - The classifiers are independent, and the error rates are not correlated.
    - Error probability of an ensemble of m base classifiers (the probability that the prediction of the ensemble is wrong)

$$\varepsilon_{ensemble} = P(y \geq k) = \sum_{k}^{m} \left\langle \begin{matrix} m \\ k \end{matrix} \right\rangle \varepsilon^{k} (1-\varepsilon)^{m-k}$$

        - $\left\langle \begin{matrix} m \\ k \end{matrix} \right\rangle$ is the binomial coefficient *"m choose k"*
        - k should be bigger than m/2
        - Example: m=11 and ε =0.25

$$P(y \geq k) = \sum_{k=6}^{m} \left\langle \begin{matrix} 11 \\ k \end{matrix} \right\rangle 0.25^{k} (1-0.25)^{11-k} = 0.034$$

Sampling (1)

- **Sampling without replacement**: the number that is sampled is not sampled multiple times. The probabilities of drawing a number in different turns are different.
- Example: given 10 numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and draw 5 numbers
  - In the first round, the chance of drawing a particular number (e.g., 2) is 1/10.
  - In later rounds, the numbers that were drawn in previous rounds are not put back to the set. The probability of drawing a particular number from the set of remaining numbers in the next round depends on the previous rounds.
  - In this example, assume that we draw 2 in the first round. In the second round, the probability to draw a particular number (e.g., 4) becomes 1/9 (NOT 1/10 anymore).
  - The final five number that we draw can be 2, 4, 5, 1, 0

Sampling (2)

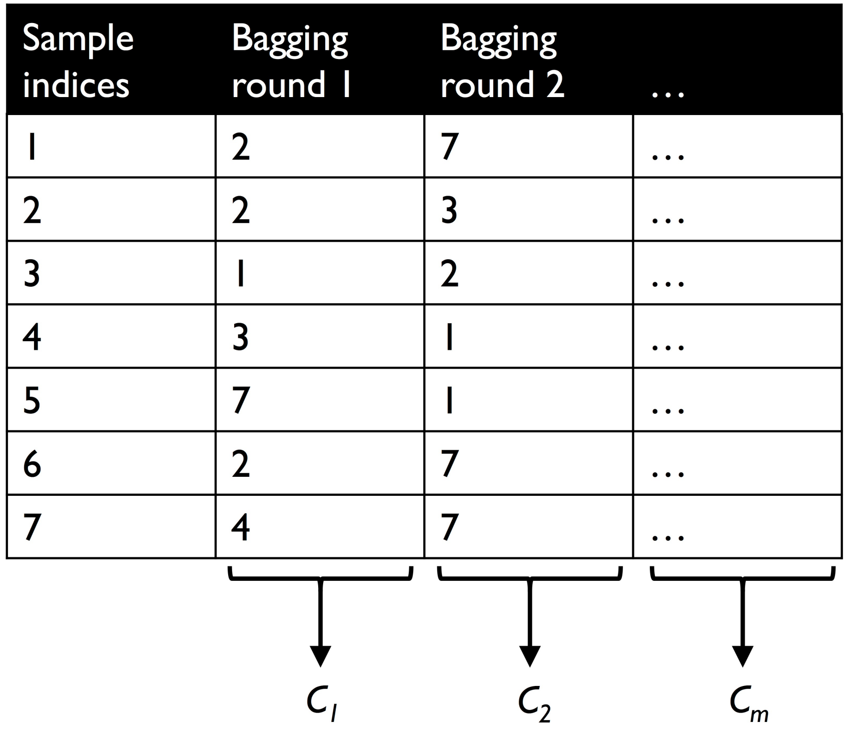
- **Sampling with replacement**: the number that is sampled is put back and can be resampled. The probabilities of drawing a number in different turns are the same.
  - Example, draw 5 numbers from {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. The result can be 2, 4, 2, 5, 1
- **Bootstrap sample**: Randomly choose *n* samples from the training set with replacement.

Bagging – bootstrap **agg**regat**ing**

- Uses majority voting.
- Use bootstrap samples to train individual classifiers. It is also known as **bootstrap aggregating**.
- First proposed by Leo Breiman in a technical report in 1994. *Bagging predictors, L. Breiman, Machine learning, 24(2),:123-140, 1996.*

Bagging

- Example: right figure
  - Train *m* classifiers.
  - Each classifier is trained using bootstrap samples.
  - The individual classifiers are typically an unpruned decision tree.
- Predictions are made based on majority voting.



Sample indices	Bagging round 1	Bagging round 2	…
1	2	7	…
2	2	3	…
3	1	2	…
4	3	1	…
5	7	1	…
6	2	7	…
7	4	7	…

BaggingClassifier algorithm

- Use an unpruned decision tree as the base classifier
- Create an ensemble of 500 decision trees on different bootstrap samples of the training set

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(criterion='entropy',
                              max_depth=None,
                              random_state=1)

bag = BaggingClassifier(base_estimator=tree,
                        n_estimators=500,
                        max_samples=1.0,
                        max_features=1.0,
                        bootstrap=True,
                        bootstrap_features=False,
                        n_jobs=1,
                        random_state=1)
```

Prediction

- On one dataset (wine), we can calculate the accuracy of the prediction using the decision tree and the bagging methods
- Training accuracy =1.0, decision trees have overfitting issue
- Bagging methods have higher prediction accuracy

```
#on wine dataset

from sklearn.metrics import accuracy_score

tree = tree.fit(X_train, y_train)
y_test_pred = tree.predict(X_test)
tree_test = accuracy_score(y_test, y_test_pred)

bag = bag.fit(X_train, y_train)
y_test_pred = bag.predict(X_test)
bag_test = accuracy_score(y_test, y_test_pred)

# tree test: 0.833
# bag test:  0.917
```

Bagging - discussions

- Advantages
  - Trees have lower generalization performance while bagging has higher generalization performance
  - More complex classification tasks and a dataset’s high dimensionality can lead to overfitting in single decision trees. Bagging algorithm can really play to its strengths.
  - Bagging algorithm can be an effective approach to reduce the variance of a model.
- Bagging is not effective in reducing model bias. Thus, we typically perform bagging on an ensemble of classifiers with low bias (e.g., unpruned decision trees).
  - **Bias** is a source of error in a model that causes it to over-generalize and underfit your data.
  - **Variance** is sensitivity to noise in the data that causes a model to overfit.
  - We call it a **tradeoff** because improving one will often make the other metric worse.

Random forests

- Random forests are a special case of bagging. They use random feature subsets when fitting the individual decision trees.
- **Idea**: aggregate multiple deep decision trees.
- **Algorithm**:
    1. Draw a random bootstrap sample of size *n*.
    2. Grow a decision tree from the bootstrap sample. At each node:
        1) Randomly **select *d* features** without replacement.
        2) Split the node using the feature that provides the best split according to the objective function (e.g., maximizing the information gain)
    3. Repeat the above two steps **k** times.
    4. Aggregate the prediction by each tree to assign the class label by majority vote.

Random forests – parameters

- Number of trees
  - Generally, the larger *k*, the better performance, the higher computation cost.
- The size of bootstrap samples
  - Decrease the size of bootstrap samples: the tree may be more random, and it helps to reduce the effect of overfitting. When this size is too small, generally worse performance.
  - Increase the size of bootstrap samples: the overall performance is better, increase the degree of overfitting.
  - In most implementations, the size of the bootstrap sample is chosen to be equal to *n*.
- Number of features *d*: typically smaller than the total number of features in the training set. A reasonable $d = \sqrt{m}$ where m is the number of features.

Random forests - Discussions

- Advantages
  - More robust to noise.
  - No need to prune individual decision trees.
- Disadvantage
  - Do not offer the same level of interpretability as decision trees.

Random forests – scikit learn class

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(criterion='gini',
                                n_estimators=25,
                                random_state=1,
                                n_jobs=2)
forest.fit(X_train, y_train)
```

- **n_estimators** parameter denotes the number of trees in the forest.
- ***n_jobs*** parameter denotes the number of jobs to run in parallel.

Decision trees and random forests

class sklearn.tree.**DecisionTreeClassifier**(*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0*)

class sklearn.ensemble.**RandomForestClassifier**(***n_estimators****=100, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,* ***bootstrap****=True, oob_score=False,* ***n_jobs****=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*)

References

- Chapter 7, Sebastian Raschka and Vahid Mirjalili: Python Machine Learning (Machine learning and deep learning with Python, scikit-learn, and TensorFlow), 3rd Edition.