MP4: CS 412 Introduction to Data Mining

Decision Tree and Random Forest Classification Framework

Sittichok Thanomkulrat (thanomk2)

December, 6 2016

Introduction

In this assignment, we studied and implemented the general data classification framework based on the knowledge of Decision tree and Random forest with gini-index as main measure. The framework was verified against the provided test datasets and the classification result was within expectation. Although in some cases, the random forest classification did not provide significant improvement over that of decision tree; this is due to the nature of the sample dataset itself which we will explain more in the following section. In summary, it was certain that the Random forest method could help reducing the over-fitting problem and enhancing the predicting power of naïve decision tree classification. Python3 was the main language for this assignment.

Decision tree classifier

The general idea of this supervised learning method is that the model will be trained from training dataset by branching at every non-terminal nodes based on pre-specified measure function. Terminal node will store the class label that will be assigned to test dataset later in classification part. Then, datasets will be classified by traversing from root to leaf node of decision tree. As specified in the assignment, Gini-index will be the main measure in this framework.

Gini Index

According to Wikipedia, gini index measures how often the randomly chosen element would be wrongly assigned a label, based on the distribution of labels. The more falsely assign the label, the higher the index value. Hence, we can utilize this index to find the splitting attribute that provides the lowest value of splitting gini index, in which reduces impurity of the split dataset the most. The formula of gini-index is as follow.

$$gini(D) = 1 - sum(p_i^2), \forall i \in class(D)$$

$$where\ p_i = relative\ frequency\ of\ class\ i\ in\ D$$

Data preprocessing

Before we can create the decision tree, we have to preprocess the dataset into a suitable format for implementation. Since the datasets are all in libsvm format, we decide to leave the data unchanged and import them into list.

Decision tree creation

The decision tree can be implemented on standard tree data structure. The implementation of this data structure is simple and intuitive which can be seen in source code. The decision tree creation part is straight-forward as well, and the pseudo-code is as below. The code itself is an adaptation from chapter 8 of Professor Han's textbook.

Tree creation pseudo-code

```
1) First we will create a root node of the tree

2) If the maximum depth has reached, find the majority class; assign it to current node and continue with next test tuple (Pre-pruning)

3) Verify whether all of the leaf nodes have the same label.

    - If they are all identical, return this node with that label.

4) Then, we will check if there are any attributes that can be used as a split point.

    - If there are none, find the majority label in the sample class and return this node

5) Find the optimal splitting attribute from this dataset, and mark this node as a splitting node.

6) Split the dataset into new datasets based on the splitting criteria and remove that attribute to prevent duplicated selection in next row

7) For each dataset,

    a. If it is empty, create a child node with assigned majority label; attach child node to current node and return current node.

    b. Recursively create a new child node with the new dataset; attach child node to current node and return current node.

8) return current node
```

Decision tree classification

After we create the decision tree, the testing data will be classified based on the model. The pseudo-code for classification method is as follow.

Classification pseudo-code

```
1) While there are tuples in test dataset, do
    a. Get the root node of decision tree and one tuple from testing dataset
    b. While the current node is not null, do
        i. If the current node is not leaf, move to the next node based on the splitting
           criteria of that node
        ii. set the label of test tuple to that node label and set the confusion matrix;
            continue with next test tuple
2) Print the confusion matrix
```

Classification result

We decided to implement the pre-pruning method due to its simplicity and computational performance. According to the trial-and-error, we found that the optimal value of tree depth is approximately the dimension of feature vectors. The result can be seen in the table below.

Accuracy	Balance-scale	Nursery	Poker	Led
Train Data	0.852	0.956	0.973	0.86
Test Data	0.733	0.947	0.615	0.858

From the data above, it is certain that the decision tree has been implemented correctly and it can predict the test data, especially in the nursery case. It is noteworthy that the poker's accuracy rate is quite low at around 62%. This is due to the nature of dataset in that the number of attributes is quite higher than others and the sample data does not represent all classes well, comparing to other datasets. In this case, we have to utilize other techniques to pre-process the training dataset before training decision tree, such as rebalancing the sample classes or reduce noises.

Per class matrix

Balance-scale (Train)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Balance	0.0	0.914	0.0	0.0	0.0	0.0	0.0
Left	0.844	0.86	0.924	0.844	0.882	0.906	0.859
Right	0.984	0.737	0.8	0.984	0.882	0.831	0.941

Balance-scale (Test)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Balance	0.0	0.783	0.0	0.0	0.0	0.0	0.0
Left	0.849	0.626	0.699	0.849	0.767	0.725	0.814
Right	0.716	0.742	0.77	0.716	0.742	0.759	0.727

Nursery.data (Train)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Priority	0.93	0.969	0.937	0.93	0.933	0.936	0.931
Very recommend	0.394	0.97	0.975	0.394	0.561	0.753	0.447
Spec_prior	0.982	0.944	0.929	0.982	0.955	0.939	0.971
Not recommend	1.0	0.934	1.0	1.0	1.0	1.0	1.0
Recommend	0.0	0.956	0.0	0.0	0.0	0.0	0.0

Nursery.data (Test)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Priority	0.816	0.877	0.765	0.816	0.79	0.775	0.805
Very recommend	0.0	0.878	0.0	0.0	0.0	0.0	0.0
Spec_prior	0.813	0.876	0.806	0.813	0.81	0.807	0.812
Not recommend	1.0	0.784	1.0	1.0	1.0	1.0	1.0
Recommend(*)	0.0	0.947	0.0	0.0	0.0	0.0	0.0

*There is no this class in testing dataset

Poker (Train)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
One pair	0.988	0.935	0.975	0.988	0.981	0.977	0.985
Three of a kinds	0.935	0.988	0.968	0.935	0.952	0.962	0.942

Poker (Test)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
One pair	0.867	0.646	0.862	0.867	0.864	0.863	0.866
Three of a kinds	0.646	0.867	0.657	0.646	0.652	0.655	0.648

Led (Train)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Three	0.777	0.896	0.767	0.777	0.772	0.769	0.775
Others	0.896	0.777	0.901	0.896	0.898	0.9	0.897

Led (Test)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Three	0.628	0.92	0.776	0.628	0.694	0.741	0.653
Others	0.92	0.628	0.849	0.92	0.883	0.862	0.905

Random Forest

Although the decision tree is a powerful classification method, it is suffered from the classic problem of model over-fitting. The goal of model training is to create a function that maps input data to some class labels based on feature vectors. However, the over-fitting occurs when that function also maps the idiosyncratic noises/outliers of that training dataset as well, leading to the hyper-sensitive model and subpar predictor. This section will show that the random forest method can be implemented to reduce this problem.

Random forest is one of the ensemble learning method that combines the multiple weaker into a strong classifier. As the name implied, this method generates many modified decision trees as a base and ensembles those results into decision. Usually, this method can greatly attenuate the over-fitting issue in exchange with increased overhead workload.

Modification to training dataset

To prevent the correlation of training data, the data bootstrapping method will be implemented. Usually, we will generate the random subset of training data for each tree to learn where each subset represents $\frac{2}{3}$ of original dataset. We select this number since it provides the best accuracy from multiple back-testing. Unless performing this step, generated trees will be highly correlated, leading to the less effective algorithm.

Modification to based decision tree

Not only generating new training subset, the decision tree construction must be modified as well. At every node, we will sample, without replacement, the subset of attribute lists; generally, we will select $\sqrt{n}, \ where\ n = number\ of\ arguments,$ as a sample splitting candidates, selecting from the recommend value in many papers. The reason why this step is important is that some attributes constantly outperform the others in the selection process, resulting in the increasing correlation between trees. We can considerably diminish this effect by doing this step.

Modified tree creation pseudo-code

1) First we will create a root node of the tree and verify whether all of the leaf nodes have the same label.

    - If they are all identical, return this node with that label.

2) Then, we will check if there are any attributes that can be used as a split point.

    - If there are none, find the majority label in the sample class and return this node

3) Sample the random subset of attributes; find the optimal splitting attribute from this list, and mark this node as a splitting node.

4) Split the dataset into new datasets based on the splitting criteria and remove that attribute to prevent duplicated selection in next row

5) For each dataset,

    a. If it is empty, create a child node with assigned majority label; attach child node to current node and return current node.

    b. Recursively create a new child node with the new dataset; attach child node to current node and return current node.

6) return current node

Random forest classification

The classification for this method is almost identical to the previous one; however, the label will be based on the majority voting from every decision tree instead.

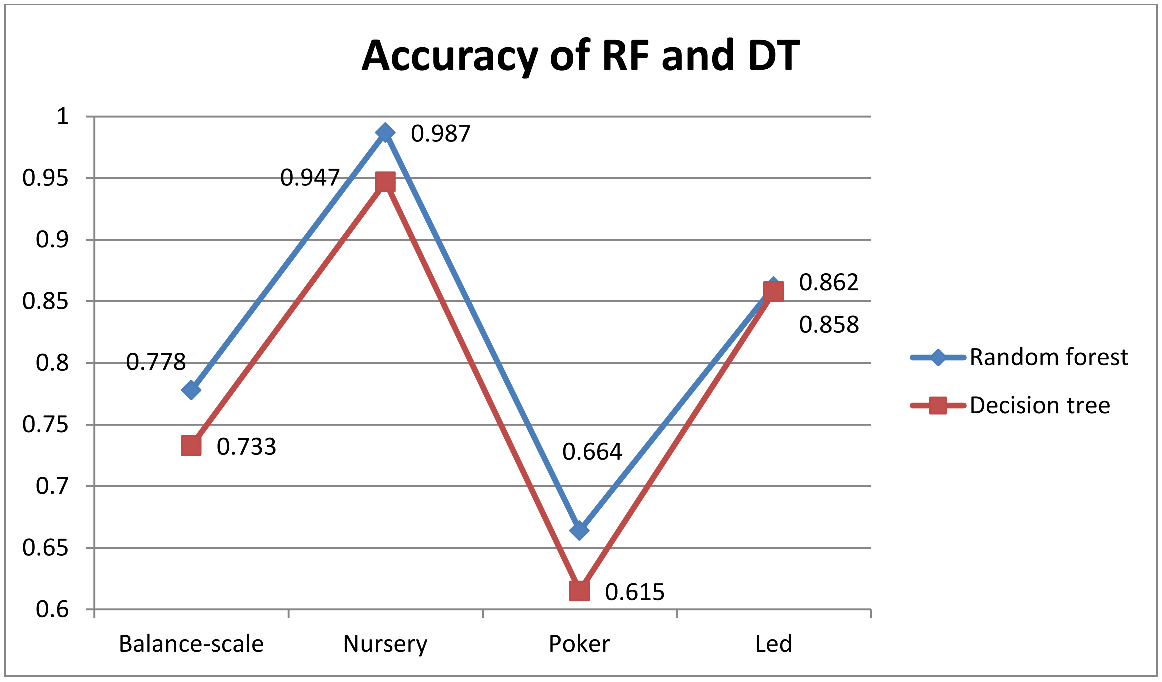
Classification pseudo-code

```
1) While there are tuples in test dataset, do
    a. For every decision tree in forest, do
        i. Perform prediction of current dataset
        ii. Store the predicted class from that tree
    b. Find the majority class from every tree
    c. Assign popular class to that test data
    d. Update confusion matrix
2) Print the confusion matrix
```

Classification result

The random forest was generated based on 100 decision trees. After adjusting this number from 1 to 500 with the sample datasets, it seems that this amount is the optimal value with respect to prediction accuracy and computation workload. Theoretically, the larger the number of trees in the forest, the better the quality of classification will be. However, there is a limit in the increment of this number where the accuracy improvement is not on the same scale with the overhead execution.

Accuracy	Balance-scale	Nursery	Poker	Led
Train data	0.995	1.0(rounding)	0.983	0.859
Test data	0.778	0.987	0.664	0.862



It is clear that the random forest method can improve the overall accuracy, comparing to that of Decision tree. According to the data above, it seems that the randomization that was introduced into the calculation can enhance the separation power by around 3 percentages.

<u>Per class matrix</u>

Balance-scale (Train)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Balance	0.926	1.0	1.0	0.926	0.961	0.984	0.94
Left	1.0	0.991	0.989	1.0	0.995	0.991	0.998
Right	1.0	0.991	1.0	1.0	1.0	1.0	1.0

Balance-scale (Test)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Balance	0.0	0.862	0.0	0.0	0.0	0.0	0.0
Left	0.931	0.65	0.736	0.931	0.822	0.769	0.884
Right	0.792	0.766	0.87	0.792	0.829	0.853	0.806

Nursery data (Train)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Priority+	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Very recommend	0.995	1.0	1.0	0.995	0.997	0.999	0.996
Spec_prior+	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Not recommend+	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Recommend+	1.0	1.0	1.0	1.0	1.0	1.0	1.0

+Rounding

Nursery data (Test)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Priority	0.976	0.992	0.983	0.976	0.98	0.982	0.978
Very recommend	0.892	0.989	1.0	0.892	0.943	0.976	0.912
Spec_prior	0.992	0.984	0.976	0.992	0.984	0.979	0.988
Not recommend	1.0	0.98	1.0	1.0	1.0	1.0	1.0
Recommend(*)	0.0	0.987	0.0	0.0	0.0	0.0	0.0

*There is no this class in testing dataset

Poker (Train)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
One pair	1.0	0.939	0.976	1.0	0.988	0.981	0.995
Three of a kinds	0.939	1.0	1.0	0.939	0.968	0.987	0.950

Poker (Test)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
One pair	0.95	0.064	0.68	0.95	0.793	0.721	0.88
Three of a kinds	0.064	0.95	0.378	0.064	0.109	0.19	0.077

Led (Train)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Three	0.766	0.9	0.771	0.766	0.769	0.77	0.767
Others	0.9	0.766	0.898	0.9	0.899	0.899	0.9

Led (Test)

	sensitivity	specificity	precision	recall	F-1	F-0.5	F-2
Three	0.772	0.902	0.779	0.772	0.775	0.777	0.773
Others	0.902	0.772	0.898	0.902	0.9	0.9	0.9

Source Code

The Python 3 source code for this assignment was as follow.

code/DecisionTree.py – this code contains the implementation of decision tree framework

code/RandomForest.py – this code contains the implementation of random forest framework