Intelligent Systems: Reasoning and Recognition

Yagmur G Cinar, James L Crowley

24 April 2018

MoSIG M1

Second Semester 2017/2018

Contents

1 Decision Trees **1**

1.1 Extracting Rules 3

1.2 Building a Decision Tree 4

1.3 Overfitting of Decision Trees 7

2 Random Forest **8**

3 An Example of Building a Decision Tree **10**

1 Decision Trees

A decision tree is a hierarchical learning model which can used be for both classification and regression. Decision trees enable us to explore complex input output relations without any prior assumptions on the data. We can see a decision tree model as a nonlinear function f which estimates the a map from input space X to target space y: $y = f(X)$. For class classification target space y is categorical ($y \in \{C_1, C_2, \dots, C_k\}$), for regression target space y is numerical ($y \in \mathbb{R}$). We consider a supervised learning case. Hence, we have the target values of y.

A decision tree splits input space X into local regions by using some distance measure and the goal is to learn well separated, homogeneous partitions. We divide into local regions by asking test questions and according to answers to this test questions, we obtain n-ary split. So, while going down the tree, at each node we make some decisions.

Some characteristics of decision trees:

- Decision trees are non-parametric we do not have any prior assumptions on the tree structure.
- Decision trees can work with heterogeneous data, both numerical (Wind is 15 km/h) and categorical (Wind is $\{\text{Strong}, \text{Weak}\}$).
- Decision trees intrinsically implement feature selection, so they can also be used for feature selection.
- It is easy to interpret the decision trees and we can learn rules from a decision tree.

Figure 1 illustrates a simple example of decision tree and the corresponding decisions in the input data space. As we can see in the example tree in Figure

1, a decision tree is composed of internal decision nodes and terminal nodes. Each internal decision node implements a test $q(x_s)$ that splits the data X_s at node s to its child nodes. Each terminal node has a label y representing the local partition of training data that is in this terminal node. In Figure 1 left, the data is in two dimension $\{X1, X2\}$ which is called also variable, attribute or features. It is a binary class classification with square and round classes. It starts at the root node by implementing a test question $(X_1 > w_{20})$ which divides the data into two subsets left and right child nodes. With the first split, we obtain one node with only circle class instances (right child) and one node with both circle ad squares (left child). We make a second split on the left child node by asking the test question $(X_2 > w_{10})$, and we obtain two homogeneous subsets (right and left children).

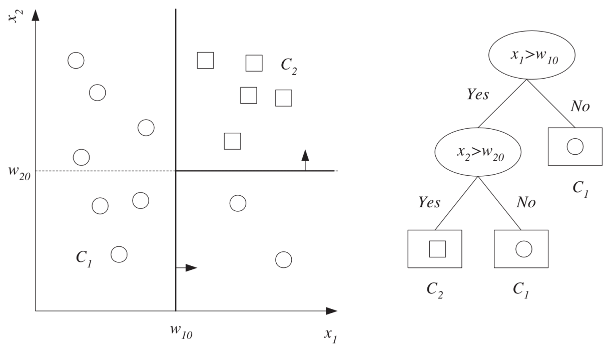


Figure 1: A simple example of data (left) and corresponding decision tree (right) [1]

Figure 2 shows another example of decision tree that predicts if is it good for playing tennis depending on the weather conditions. This is a classification tree, where the task is binary classification with (Yes, No) classes. The variables are categorical compared to the previous example (Figure 1). What the play tennis tree would predict if we are on a day (Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Strong)? When we want to predict for a

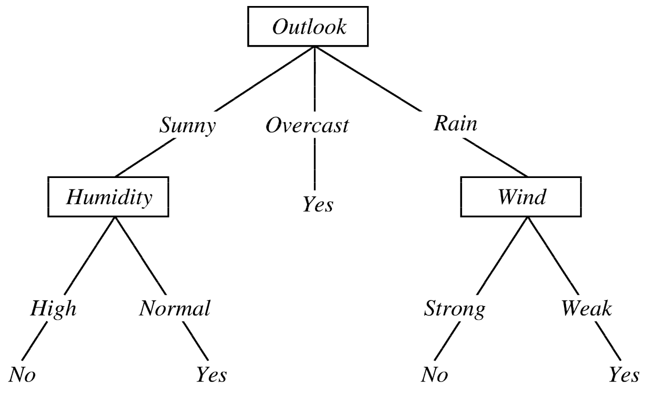


Figure 2: Play Tennis or not? [4]

test example we basically follow from the root to a terminal node, by asking the internal decision node questions iteratively. The terminal node that we end up give us the prediction of the tree. For this test example (Outlook = Sunny, Temperature = Hot, Humidity = Normal, Wind = Strong), we would predict Yes (see Figure 2).

1.1 Extracting Rules

Decision trees are high interpretable. We can easily understand the information on the decision nodes and learn if then rules from a decision tree. Each path from root to terminal is a conjunction of test questions which needs to be satisfied to reach this terminal node. We can convert them to if then rules. We can write the following if then rules for the decision tree example for play tennis (Figure 2).

- **IF** (Outlook = Sunny) **AND** (Humidity = High) **THEN** Play Tennis = No
- **IF** (Outlook = Sunny) **AND** (Humidity = Normal) **THEN** Play Tennis

= Yes

- **IF** (Outlook = Overcast) **THEN** Play Tennis = Yes
- **IF** (Outlook = Rain) **AND** (Wind = Strong) **THEN** Play Tennis = No
- **IF** (Outlook = Rain) **AND** (Wind = Weak) **THEN** Play Tennis = Yes

1.2 Building a Decision Tree

At the root node we start with whole training set and we apply a test question at the root node that splits data into n parts (n children). The aim of splitting is to achieve more homogeneous partitions. For each child we check if we reach a terminal node and we iteratively continue splitting until we obtain good enough partitions.

In real life data, the number of attributes and the number examples are much larger. We need some evaluation measure of goodness of a split which is an impurity measure. We can measure impurity of a node s by using entropy for K class classification problem, given in Equation 1. Figure 3 illustrates entropy function for binary classification. For two class (binary) classification example probability of seeing one class is 0.5 when we have the equal number of examples for both class in one node. The entropy gets the maximum value of 1.0, since we have the maximum impurity (equal number of instances for each class).

$$\text{Entropy(S)} = -\sum_{i=1}^{K} p_s^i \log_2(p_s^i) \tag{1}$$

Information gain(S,A) is the expected amount of decrease in entropy gained by dividing set S into n number of subsets by considering attribute A.

$$\text{Information gain}(S, A) = \text{Entropy}(S) - \sum_{t=1}^{n} \frac{N_{t,s}}{N_s} \text{Entropy}(S^t) \tag{2}$$

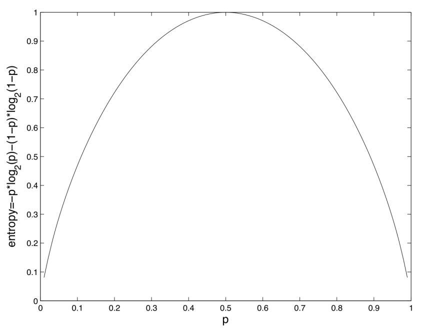


Figure 3: Entropy for binary classification [1]

where N_s is the total number of instances at node s and $N_{t,s}$ is the total number of instances fall into node t after the test at node s.

Here, the probability of class C_i at a node t can be estimated with $P(C_i|x,t) \equiv p_t^i = \frac{N_t^i}{N_t}$. N_t is the number of training instances falls to node t and N_t^i is the number of instances belong to class C_i.

And total impurity after splitting is Entropy$'$, given in Equation 3.

$$\text{Entropy}' = -\sum_{t=1}^{n} \frac{N_{s,t}}{N_s} \sum_{i=1}^{K} p_t^i \log_2(p_t^i) \tag{3}$$

While selecting which attribute to consider at a node, we choose the attribute that gives the highest Information gain. Similarly we can say that we choose the attribute the one leads to minimum Entropy$'$ after splitting (Equation 3).

Pseudo code of decision tree building is given in Figure 4 which is also basis of Classification and Regression Trees (CART) [2], ID3 algorithm [5], C4.5 algorithm [6].

For a categorical attribute we can define the split according to values of that attribute takes. For a numerical attribute we can order the values that attribute takes and calculate the impurity reduction for all possible splits. And choose the best split that gives the lowest impurity measure.

```
GenerateTree($\mathcal{X}$)
    If NodeEntropy($\mathcal{X}$)< $\theta_I$ /* equation 9.3 */
      Create leaf labelled by majority class in $\mathcal{X}$
      Return
    $i$ ← SplitAttribute($\mathcal{X}$)
    For each branch of $\boldsymbol{x}_i$
      Find $\mathcal{X}_i$ falling in branch
      GenerateTree($\mathcal{X}_i$)

SplitAttribute($\mathcal{X}$)
    MinEnt← MAX
    For all attributes $i = 1, \ldots, d$
        If $\boldsymbol{x}_i$ is discrete with $n$ values
          Split $\mathcal{X}$ into $\mathcal{X}_1, \ldots, \mathcal{X}_n$ by $\boldsymbol{x}_i$
          e ← SplitEntropy($\mathcal{X}_1, \ldots, \mathcal{X}_n$) /* equation 9.8 */
          If e<MinEnt MinEnt ← e; bestf ← i
        Else /* $\boldsymbol{x}_i$ is numeric */
          For all possible splits
              Split $\mathcal{X}$ into $\mathcal{X}_1, \mathcal{X}_2$ on $\boldsymbol{x}_i$
              e←SplitEntropy($\mathcal{X}_1, \mathcal{X}_2$)
              If e<MinEnt MinEnt ← e; bestf ← i
    Return bestf
```

Figure 4: Pseudocode of tree building [1].

Gini index is another measure of impurity for classification task, Equation 4.

$$\text{Gini Index} = 1 - \sum_{i=1}^{K} (p_s^i)^2 \tag{4}$$

For classification we can use entropy (Equation 1) or gini index (Equation 4). For regression task we can use mean squared error (MSE) to measure impurity of a node, Equation 5.

$$\text{MSE} = \frac{1}{N_t} \sum_{i=1}^{N_t} (y_i - \hat{y}_i)^2 \tag{5}$$

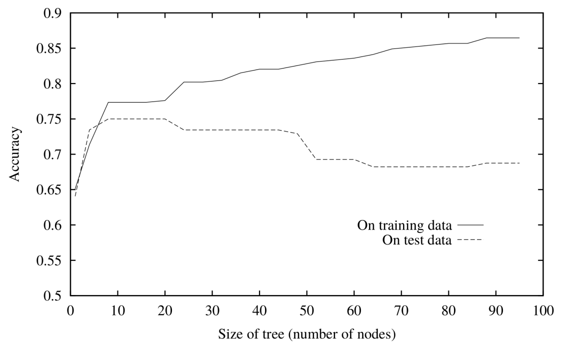


Figure 5: Model complexity versus training and test error [4]

1.3 Overfitting of Decision Trees

Growing a decision tree fully might end up overfitting. Overfitting occurs when we have a model is not well generalized. We can say the model is overfitted when the training error keeps decreasing while validation error no longer improves or get worse, Figure 5.

To avoid overfitting pre-pruning and post-pruning is applied. Pre-pruning is stopping early when we don't improve much by splitting. We can implement pre-pruning according to some pre-defined heuristics (e.g. impurity lower than some threshold, or reaching pre-defined maximum depth, etc..). Post-pruning is letting tree to grow fully and after removing some sub-trees that does not improve the information gain so much. This can be done by using a validation set or applying some significance test (e.g. C4.5). Usually, post-pruning leads to better results since we can observe more. And by pre-pruning we might not be able to observe the full structure of the data. However, post-pruning is computationally more expensive than pre-pruning. If we have some good estimation about the problem or some prior knowledge, we can still obtain good results also with pre-pruning.

2 Random Forest

Random Forest is an ensemble learning approach. We build T decision trees (weak learners) and combine their results.

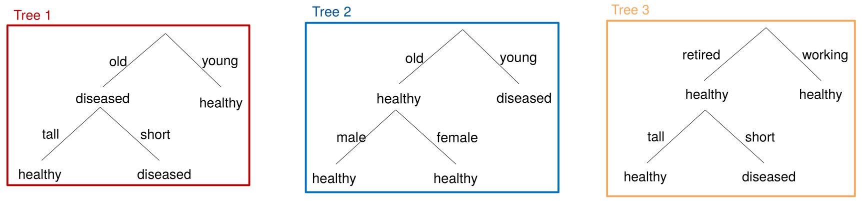


Figure 6: A simple example of a Random Forest [3]

In random forest, while training a single tree we select a subset of the training set sized Z (boostrap resample of data). Boostrap resample of data is drawing with replacement Z samples. Figure 7 illustrates a toy example of resampled subset of data.

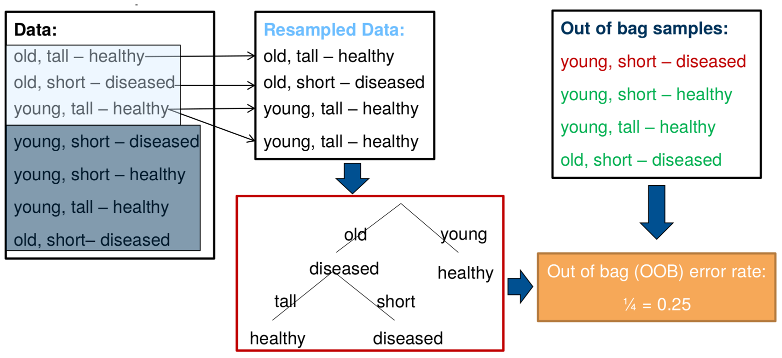


Figure 7: Out of bag [3]

We build T number of trees independently and output their combined results. Since, each tree is independently built, learning random forest can be easily parallelized.

While building the forest, for each tree:

- Draw a boostrap sample Z

- Grow a decision tree using boostrap data by recursively repeating following steps for each node until the minimum node size is reached
  - Draw V variables at random from total A variables
  - Pick the best variable among V variables
  - Split to n-child nodes on variable V

After we draw N samples with replacement, the remainder of the samples are called out of bag samples. These samples can be used to evaluate the generalization of this tree. This error is called out of bag (OOB) error.

Prediction of random forest, for classification is majority vote of predictions of T trees. For regression it is the average target value y of the terminal node. If we have a test example of (old, retired, male, short), our simple random forest example (Figure 6) prediction with majority vote would be diseased.

The number of variables V and the number of trees T are the two main parameters of random forests. Some heuristics to choose number of variables are using the squared root of the total number of variables ($V = \sqrt{A}$) or taking $\log_2$ ($V = \log_2(A)$). A is the total number of attributes. Increasing the number of trees T leads to better predictions, but also more expensive in terms the computation. Random forest algorithm provides low variance and bias, so we obtain a good generalization. However, we loose the interpretability of the model compared to decision trees.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Figure 8: Example data of playing tennis according to weather

3 An Example of Building a Decision Tree

For a data set of 14 instances for playing tennis according to weather condition is given in Figure 8. There are 4 attributes/variables: Outlook (Sunny, Overcast, Rain), Temperature (Hot, Mild, Cool), Humidity (High, Normal), and Wind (Weak, Strong). We would like to build decision tree for classification of playing tennis. It is a binary classification problem with two classes, Play Tennis: (Yes, No). We can start with choosing the attribute of root node. S is the whole training set with 14 instances. We can calculate the information gain of Outlook, Humidity, Wind and Temperature.

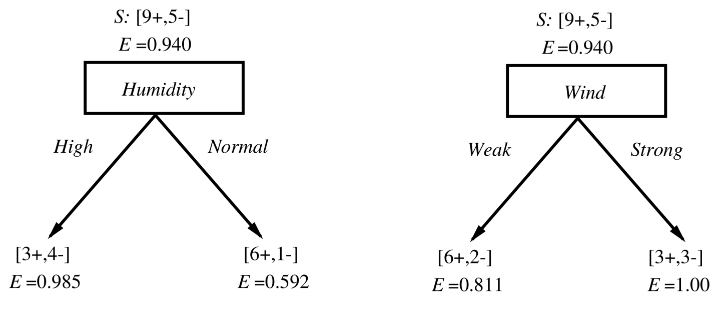
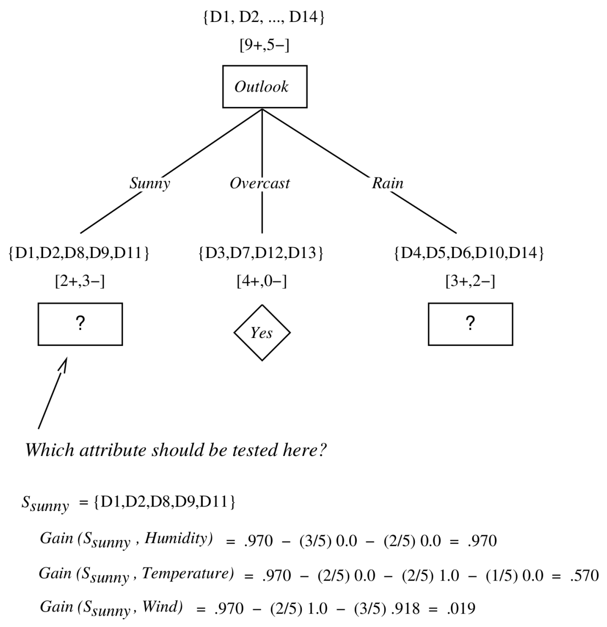


Figure 9: Root node attribute selection

We choose the one gives the maximum information gain. And we repeat it for the child nodes.


{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
{D1,D2,D8,D9,D11}
[2+,3−]
?
{D3,D7,D12,D13}
[4+,0−]
Yes
{D4,D5,D6,D10,D14}
[3+,2−]
?
Which attribute should be tested here?
S_{sunny} = {D1,D2,D8,D9,D11}
Gain (S_{sunny} , Humidity) = .970 − (3/5) 0.0 − (2/5) 0.0 = .970
Gain (S_{sunny} , Temperature) = .970 − (2/5) 0.0 − (2/5) 1.0 − (1/5) 0.0 = .570
Gain (S_{sunny} , Wind) = .970 − (2/5) 1.0 − (3/5) .918 = .019

References

[1] Ethem Alpaydin. *Introduction to Machine Learning.* The MIT Press, 2nd edition, 2010.

[2] Leo Breiman. *Classification and regression trees.* Wadsworth statistics/probability series. Wadsworth International Group, 1984.

[3] Markus Kalisch. Lecture notes in applied multivariate statistics, February 2012.

[4] Thomas M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[5] J. Ross Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.

[6] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.