Decision Tree

Adaptive Software Systems

Name:	**Wolfgang Ofner**
Matrikelnummer:	**1510277013**

Wiener Neustadt, 10.September.2017

Contents

Chapter 1

Related Papers

The algorithm to create the decision tree in the program is the ID3. The ID3 algorithm was first published in [1]. The algorithm will be described in the chapter use of the algorithm.
Related papers point out that there are another two popular classification algorithm: Naive Bayes and NBTree. Naive Bayes is a supervised classification algorithm and is used to prognosticate the likelihood of group members. This algorithm is based on applying the Bayes theorem. The algorithm assumes that a predictor's effect on a class is independent to other predictors. Due to this independence and since the algorithm is based on the Bayes theorem, it is simple to build a Bayesian network, which has no complicated parameter estimation and therefore is useful for a large dataset. Because of its simple design, the Naive Bayes algorithm is widely used. An example for its usage is the intrusion detection. [4]
The formula to calculate the posterior probability of a class is: $P(c|x) = (P(x|c)P(c))/P(x)$

- P(c | x) is the posterior probability of a class.
- P(c) is the prior probability of a class.
- P(x | c) is the prior probability of the predictor. [4]

The NBTree is a mix of the Decision tree and the Nayive Bayes algorithm. The algorithm to create a NBTree is as followed:

1. For each attribute Xi, evaluate the utility, u(Xi), of a split on attribute Xi. For continuous attributes, a threshold is also evaluated at this stage.
2. Let J = AttMax(Ui). The attribute with highest utility (Maximum utility).
3. If Uj is not significantly better than the utility of the current node, create a Naive Bayes classifier for the current node and return.

4. Partition T according to the test on Xj. If Xj is continuous, a threshold split is used; if Xj is discrete, a multi-way split is made for all possible values.
5. For each child, call the algorithm recursively on the portion of T that matches the test leading to the child. [4]

"Scaling Up the Accuracy of Naive-Bayes Classiers: a Decision-tree Hybrid" describes the NBTree. As already mentioned, the NBTree is a hybrid algorithm which suits in scenarios where several attributes are relevant for the classification. This paper shows that the NBTree works greatly with real-world data and also scales up well in accuracy. This paper also shows that the algorithm produces highly accurate classifiers during the training phase. NBTree outperformed the Naive-Bays and C4.5 algorithm for datasets over 10 000 instances and the results show that the running time can scale up. With the NBTree algorithm the segments are easy to understand since the algorithm uses an univariate decision tree. The result of the NBTree needs way less leafs than the C4.5 to represent the result. Since interpretability is an important factor for data mining the leafs are easily understood since every leaf is a Naive-Bays classifier. [2]
"Decision tree classification of land cover from remotely sensed data" measured the effect of increasing the accuracy of a decision tree with boosting. Boosting generates several classifiers in an iterative way and is a general method to improve the quality of a learning algorithm. During this iterative process the algorithm focuses on attributes which are harder to classify and can achieve a better result this way. The result of the experiment in this paper is that with boosting the accuracy could be increased by more than 2% from 84.3% to 86.7%. No matter if boosting is used or not, the training data remains crucial. Without enough correct training data, no correct results can be produced. [3]
As already mentioned, an area to work with classification algorithm is the intrusion detection. "Analyzing NB, DT and NBTree Intrusion Detection Algorithms" conducted a research with the decision tree, Naive Bayes and NBTree algorithm, to work out which one works best. The research showed, that the NBTree has the highest accuracy and least error rate. This result isn't a surprise since the NBTree is a hybrid form of the decision tree and Naive Bayes and therefore achieves the best result. The good result comes at the cost of higher construction and processing time. There the Naive Bayes algorithm is better. [4]

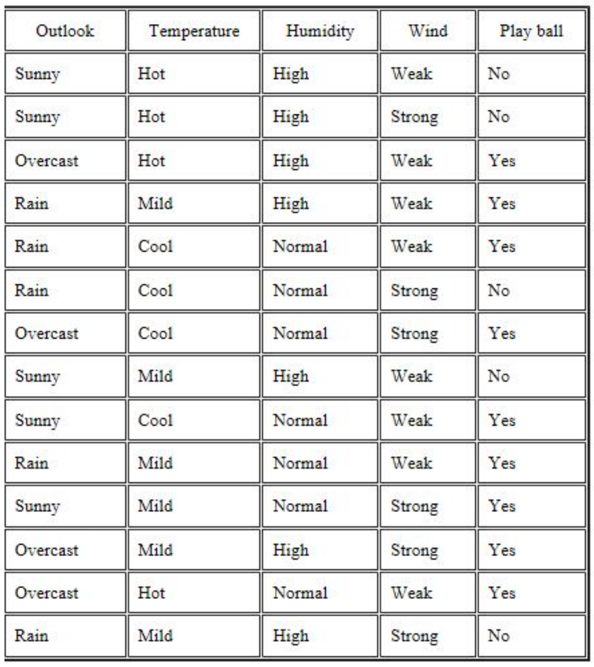
Chapter 2

Description of the idea and algorithm

This chapter will describe how the used algorithm works, then will describe what ideas are in the implementation of the program and also its limitations.

2.1 Use of the algorithm

The program uses the ID3 algorithm which was developed by J. Ross Quinlan and published in [1]. The algorithm builds a decision tree from a sample of data. The training phase results in a decision tree, which will be used to classify future input data. Every leaf represents a value of the result class and a non-leaf node represents a decision node.



Outlook	Temperature	Humidity	Wind	Play ball
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Figure 2.1: Test data [5]

1. The algorithm calculates the Entropy of the dataset. The formula for that is $Entropy(S) = \sum -p(I) log2p(I)$ where $p(I)$ is the proportion of S belonging to class I. $\sum$ is over c. Log2 is log base 2. Figure 2.1 has 9 out of 14 times Yes and 5 out of 14 times No as a result. Therefore the Entropy for the dataset is: $Entropy(S) = -(9/14)Log2(9/14) - (5/14)Log2(5/14) = 0.940$.

2. The next step is to calculate the Information gain of every attribute. In this example the attributes are outlook, humidity, wind and temperature. Every attribute has sub attributes, for example wind has the sub attributes weak and strong. The formula for the information gain is: $Gain(S, A) = Entropy(S) - S((|Sv|/|S|) * Entropy(Sv))$ where: S is each value v of all possible values of attribute A, Sv = subset of S for which attribute A has value v, |Sv| = number of elements in Sv, |S| = number of elements in S. For the wind attribute the formula is the following: $Gain(S, Wind) = Entropy(S) - (8/14) * Entropy(Sweak) - (6/14) * Entropy(Sstrong)$. The entropy formula for Sweak is: $Entropy(Sweak) = -(6/8) * log2(6/8) - (2/8) * log2(2/8) = 0.811$. The result is 6 times Yes with wind weak and 2 times No with wind weak. The entropy for Sstrong follows the same pattern, except that a row is only selected if the wind is strong. The result for Sstrong is: $Entropy(Sstrong) = -(3/6) * log2(3/6) - (3/6) * log2(3/6) = 1.00$.

3. With the Entropy S and the Entropy of the wind attribute the formula for the Information gain is: $0.940 - (8/14) * 0.811 - (6/14) * 1.00 = 0.048$. Therefore the Information gain for wind is 0.048.
4. Repeat the steps 2 and 3 for every remaining attribute.
5. The results of the remaining attributes are outlook = 0.246, temperature = 0.029 and humidity = 0.151.
6. Outlook has the highest Information gain with 0.246, therefore outlook becomes the root node of the decision tree. Outlook has three attributes: sunny, overcast and rain.
7. The next step is to follow one of these attributes until a leaf is reached. The edge leads to a leaf if all the results of this edge have the same result. For example when the outlook is overcast the result is always Yes. Therefore overcast leads to a leaf with yes.
8. Sunny doesn't lead to a leaf. Therefore the information gain with Ssunny must be calculated for the remaining attributes.
9. Humidity has the highest Information gain and becomes the next node.
10. The next step is if the attributes sunny and high lead to a leaf. The answer is yes, these attributes lead to the leaf No since all results are No. Also the attributes sunny and normal lead to the leaf Yes. Since there are no edges left this part of the tree is finished.
11. As already described in 7, overcast leads to the leaf Yes.
12. The last remaining edge of the root node is rain. Since it doesn't lead to a leaf the information gain of the next node must be calculated.
13. The remaining attributes are temperature and wind where wind has the higher information gain. Therefore wind will be placed as a next node.
14. With the attributes rain and strong the result is always No and with rain and weak the result is always Yes. Therefore both edges of wind lead to a leaf and the decision tree is finished. [5] [1]

The process of finding the next node continues until no attributes are left or until a certain depth is reached. This implementation of the algorithm doesn't stop at a certain depth. The algorithm creates the tree until all attributes are placed as a node. The result column must have two different values, otherwise an error message will be displayed.

2.2 Overview of the ideas behind the implementation

The main idea behind the implementation is to take the dataset and then break it into smaller pieces until the problem is small enough to be solved. This section will describe how this was achieved and will give an overview about the flow in the code.
At the start of the program, the user can either enter data by hand or import a .csv file with the training data. Importing the data is pretty straight forward. The first row holds the column title and the last column contains the outcome.
Entering the data by hand works accordingly. First the user has to declare the amount of columns. This number must be an integer and greater than 1. Next the user is prompted to enter the title of every column. Then the user can enter the values for the columns.
After the program has received the data for training, either by hand or by importing a .csv file, the program creates the decision tree. This process is called training or learning. Once the learning process is complete the user has different options on how to interact with the program. The user can either print the created tree, export entered data or let the program predict the result. Printing the tree and exporting the test data are pretty straight forward. The print function will be explained in more detail in the next section.
If the user wants to get a prediction, the program prompts the user to enter a value for every previously defined column. After that the program will predict the outcome and print which route it took to iterate through the tree. If no valid route was found, an error message will be printed.

2.3 Details of the ideas behind the implementation

After the data input is completed the CreateTreeAndHandleUserOperation method is called. This method creates the tree and afterwards handles the user input. This means that this method is responsible for rendering the menu and handle the user interaction to work with the created tree. This can be for example printing the tree or exporting the test data. To start the learning process the CreateTreeAndHandleUserOperation method calls the Learn method.
The first step of the learning process is figuring out which column is the root node of the tree. To do that the Learn method calls the GetRootNode method, which will return the root node of the tree. To calculate the root node the method gets all available attributes for each column. Afterwards the information gain is calculated. The calculation works as described in the section Use of the algorithm. After the

calculation is completed, a new tree node will be created with the information of the column with the highest information gain. This tree node will be the root node of the decision tree.
After the root node is set the algorithm checks if this node is a leaf. This doesn't make sense at the first step, but it is necessary because it is the break condition of the recursion to create the tree. If the tree is not a leaf, the dataset will be reduced. This idea was already mentioned earlier. After each step the dataset will be reduced until it cant be reduced any further. To reduce the dataset the method CreateSmallerTable is called. This method returns a datatable which contains the dataset minus the column which was just set as a node. With this new, smaller dataset the Learn method is called again. The recursive call finds the root node for the smaller dataset and sets this new node as child of the previously set node. This process will be repeated until all columns are put into the tree.
To check whether a node is a leaf or not the algorithm gets all result values of the current edge and puts them into a list. Then the algorithm checks if all entries in the list have the same value. If they have the same value, this node is a leaf and will be added to the tree and marked as a leaf. As example: the edge "sunny" is currently checked. The algorithm gets all result values (from the last column of the dataset) for this edge. Then it checks if they all have the same value. If all result values are the same, for example yes, then a new tree node will be created and the boolean IsLeaf will be set. The value of the result node, in this example yes, will also be added to the node.
After the tree is created the user has different options to interact with the program. One option is printing the whole tree. This doesn't print a tree but every possible route through the tree. This behavior will be discussed in more detail in the next section. The root node is printed in magenta, an edge in lower case letters in yellow, nodes in uppercase letters in cyan and decision nodes in green and also in uppercase letters. To make it easier for the user a legend is also printed, explaining the meanings of the different colors. The separation between a node and an edge is "–" and between an edge and a node is "–>".
Another interaction is calculating a route. The user will be prompted to enter a value for every column. The user input is stored in a dictionary with the input and the column name as key value pair. This dictionary is necessary to check if the entered value is part of the currently checked node. For example you have the node humidity and want to check the edge high. Without the dictionary the algorithm would only check if high leads to another node or leaf. This would make it possible to enter high as attribute for another column and the algorithm would choose it. This would lead to a wrong route calculation. After a value for every column is entered, the algorithm tries to calculate the route. To calculate the outcome, the algorithm iterates through the tree and returns the found route as a string. Then the string is printed. If no route is found, an error message is returned.

2.4 Limitations of the implementation

As already described in the previous sections, the implementation has some limitations. The first limitation is that the last column of the training data, the result column, can only contain two different values. These values should be Yes and No. If the result values are two different values, but not Yes and No, the program will work but the leafs are not identified as such and therefore printed in a wrong color. The second limitation affects the ability to print the tree. Due to the nature of the console it is not possible to create a print function which can print all sizes and variations of a tree in a user friendly way. Therefore the approach with different colors, separators and casings was chosen.

List of Figures

Bibliography

[1] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986. [Online]. Available: http://dx.doi.org/10.1023/A:1022643204877

[2] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 202–207. [Online]. Available: http://dl.acm.org/citation.cfm?id=3001460.3001502

[3] M. Friedl and C. Brodley, "Decision tree classification of land cover from remotely sensed data," *Remote Sensing of Environment*, vol. 61, no. 3, pp. 399 – 409, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0034425797000497

[4] D. Mahmood and M. Hussein, "Analyzing nb, dt and nbtree intrusion detection algorithms," vol. 16, pp. 87–94, 03 2014.

[5] [Online]. Available: https://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/2.htm