CREDIT RISK ASSESSMENT USING DECISION AND EXPLORATION TREES

PETR BERKA[1,2]

[1] University of Finance and Administration,
Department of Computer Science and Mathematics,
Estonská 500, Prague, Czech Republic

[2] University of Economics, Prague, Faculty of Informatics and Statistics,
Department of Information and Knowledge Engineering,
W. Churchill Sq. 4, Prague, Czech Republic
e-mail: berka@vse.cz

Abstract

Credit risk assessment, credit scoring or loan applications approval are one of typical classification tasks, in which the final decision can be either a crisp yes/no decision about the loan or a numeric score expressing the financial standing of the applicant. A corresponding classifier can be created from data about past decisions. Beside logistic regression that constitutes a de-facto banking industry standard and a benchmark algorithm, a number of data mining and machine learning algorithms can be used as well. In the paper we focus on tree building algorithms. Due to their understandability, trees can be used not only for classification but also for concept description. Another advantage is that trees can be created also from data with missing values. We present the basic concept of learning trees from data, describe our method for creating exploration trees and discuss its difference with algorithms for creating decision trees. We also compare our method with standard tree learning algorithms C4.5 and CART on some data from the loan application domain.

Key words: *data mining, decision trees, exploration trees, loan application data.*

1. Introduction

Credit risk assessment, credit scoring or loan applications approval are one of the typical tasks that can be solved using expert systems or machine learning. In the first case, the knowledge to be used is formulated by domain experts, in the second case, the knowledge to be used is inferred from data about past decisions. These data usually consists off socio-demographic characteristics (e.g. age, sex, region, job), economic characteristics (e.g. income, deposit), the characteristics of the loan (e.g. purpose, amount, monthly payments) and, of course, the loan approval decision.

From the machine learning perspective, loan applications evaluation is a classification task, in which the final decision can be either a crisp yes/no decision about the loan or a numeric score expressing the financial standing of the applicant. To create a classifier, different methods and algorithms can be used: decision trees, decision rules, neural networks, SVM, Bayesian networks, instance-based classifiers. The common underlying idea of all these methods is:

- similarity-based assumption (examples that belong to the same class have similar characteristics),

- generalization ability (the models created from training data will perform sufficiently well on unseen examples).

We present the basic concept of learning trees from data, describe our method for creating exploration trees and discuss its difference with algorithms for creating decision trees. We also compare our method with a standard tree learning algorithms C4.5 and CART on some data from the loan application domain.

2. Decision and Exploration Trees

The TDIDT (top-down induction of decision trees) family of algorithms recursively partitions the attribute space to build rectangular regions that contain examples of one class. All these algorithms are based on greedy top-down search in the space of possible trees. Once a splitting attribute is chosen, the algorithm proceeds further and no backtracking (and changing the splitting attribute) is possible. A sketch of a generic TDIDT algorithm is shown in Figure 1.

Figure 1: Generic TDIDT algorithm

TDIDT algorithm

1. select the best splitting attribute as a root of the current (sub)tree,
2. divide data in this node into subsets according to the values of the selected attribute and add new node for each this subset,
3. if there is an added node, for which the data do not belong to the same class, goto step 1.

Source: the author.

This algorithm will process only categorical attributes (at a given node, data are split according to every value of the attribute chosen for branching) and data without noise (the growing of the tree stops if all leaves contain examples of the same class). However both these limitations can be solved. Numeric attributes can be discretized (either in advance during the data preprocessing step or within the tree induction algorithm), and the stopping criterion can be relaxed to allow to analyze noisy data and to overcome the problem of overfitting. The result of the algorithm is thus a single tree. The decision trees are usually used for classification. In this case, a new example is propagated down the tree at each splitting node selecting the branch that corresponds to the value of the splitting attribute. The leaf of the tree then shows the class for this example. The accuracy of the decision tree is evaluated using the confusion matrix. This matrix shows the number of correctly and wrongly classified examples. In the most simple case of classification into two classes, the numbers in the matrix are *TP* (the number of examples correctly classified to the first class), *TN* (the number of examples correctly classified to the second class), *FP* (the number of examples wrongly classified to the first class) and *FN* (the number of examples wrongly classified to the second class). A number of quality measures can be computed from these numbers. The basic criterion used to evaluate the tree is the classification accuracy defined as

$$\frac{TP+TN}{TP+FP+TN+FN}. \tag{1}$$

As this criterion does not work well for imbalanced classes, we will use a modified version, where classification accuracy is evaluated for each class separately

$$\frac{TP}{TP+FP}\ ,\quad \frac{TN}{TN+FN}\ . \tag{2}$$

But simultaneously, a decision tree can be interpreted as a partition of the given data set into subsets, that are homogeneous w.r.t. class attribute, i.e. as a description of the concept behind each class.

2.1 Standard Algorithms for Creating Decision Trees

The TDIDT method, also known as "divide and conquer" has been implemented in a number of algorithms like ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993), CART (Breiman et al., 1984) or CHAID (Biggs et al., 1991). The main differences between the particular algorithms are:

- the form of the tree,
- the criterion used when choosing the splitting attribute,
- the stopping criterion that prevents tree from further growing.

The ID3 corresponds to the generic algorithm shown in Figure 1; the entropy is used to find a split in step 1 of the algorithm:

$$H(A)=\sum_{i=1}^{R}\frac{r_i}{n}\left(-\sum_{j=1}^{S}\frac{a_{ij}}{r_i}\log_2\frac{a_{ij}}{r_i}\right). \tag{2}$$

The C4.5 uses information gain to find a split:

$$InfoGain(A)=-\sum_{j=1}^{S}\frac{s_j}{n}\log_2\frac{s_j}{n}-H(A), \tag{3}$$

where $H(A)$ is defined in the same way as in formula (2).This algorithm uses post-pruning of created tree to reduce its size. A binarization of numeric attributes (discretization into two intervals) is incorporated into the algorithm – C4.5 thus can directly handle numeric attributes. C4.5 is used by the machine learning community as a de-facto standard.

CHAID algorithm uses χ^2 statistics

$$\chi^2(A)=\sum_{i=1}^{R}\sum_{j=1}^{S}\frac{\left(a_{ij}-\frac{r_i\cdot s_j}{n}\right)^2}{\frac{r_i\cdot s_j}{n}}\ . \tag{4}$$

Unlike the abovementioned algorithms, CART creates a binary tree (the values of a categorial attribute are thus grouped into two groups). Gini index is used to decide about a split:

$$Gini(A)=\sum_{i=1}^{R}\frac{r_i}{n}\left(1-\sum_{j=1}^{S}\left(\frac{a_{ij}}{r_i}\right)^2\right). \tag{5}$$

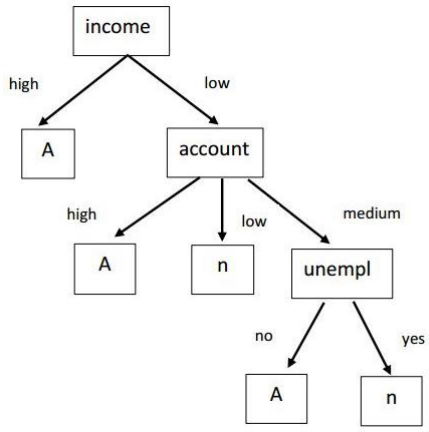
In all formulae above a_{ij} is the number of examples that have the i-th value of the attribute A (this attribute has R values) and the j-th value of the target attribute (this attribute has S values), r_i is the number of examples that have the i-th value of the attribute A, s_j is the number of examples that have the j-th value of the target attribute and n is the number of all

examples. Using entropy and Gini index, we prefer the attribute with the lowest value, using info gain or χ^2, we prefer the attribute with the highest value. Nevertheless, the criteria tend to select similar attributes. The best attribute will be the attribute that perfectly splits the data into subsets of examples belonging to the same class, the worst attribute will be the attribute where each value uniformly represents examples of every class.

To illustrate the differences between these algorithms, we use a demo data set containing 12 examples described by four input attributes: income (with possible values low, high), account (with values low, medium, high), sex (with values male, female) and unemployed (with values yes, no) and a class attribute loan (with values A, n). The resulting trees created by different algorithms are shown in Figures 2 – 4. The trees are different but the root attribute is always the attribute income. Figure 2 right part illustrates how to understand the tree written in textual form (left part). The numbers assigned to each leaf show the number of examples in the leaf (first number) and eventually the number of examples wrongly classified by the leaf (second number).

Figure 2: Result of the ID3 algorithm as implemented in Weka

```
income = high: A (5.0)
income = low
|  account = high: A (2.0)
|  account = low: n (2.0)
|  account = medium
|  |  unemployed = no: A (1.0)
|  |  unemployed = yes: n (2.0)
```



Source: the author.

Figure 3: Result of the C4.5 and CHAID algorithms as implemented in Weka

```
income = high: A (5.0)
income = low
|   account = high: A (2.0)
|   account = low: n (2.0)
|   account = medium: n (3.0/1.0)
```

Source: the author.

Figure 4: Result of the CART algorithm as implemented in Weka

```
income = high: A(5.0)
income = low
|  account = high: A(2.0)
|  account = (low OR medium)
|  |  unemployed = yes : n(3.0)
|  |  unemployed = no
|  |  |  account = low: n(1.0)
|  |  |  account = medium: A(1.0)
```

Source: the author.

2.2 Exploration Trees

Our proposed algorithm (Berka, 2011) for exploration trees is based on an extension of the TDIDT approach. Instead of selecting single best attribute to make a split of the data, we select more suitable attributes. The result thus will be a set of trees (forest) where each tree represents one model applicable for both description and classification. Figure 5 illustrates this fact on some trees created for the same data as used to create trees shown in Figures 2 – 4. In the case of the ETree algorithm, every tree has a different splitting attribute in the root.

To decide about the suitability of an attribute to make a split, we use the χ^2 criterion shown in formula (4). This criterion not only ranks the attributes but also evaluates the "strengths" of the relation between the evaluated and class attributes. So we can consider only significant splitting attributes. The significance testing allows also to reduce the number of generated trees; if the best splitting attribute is not significantly related with the class, we can immediately stop growing tree at this node and need not to consider other splitting possibilities. We can of course use only the best splitting attribute, in this case we get the classical TDIDT algorithm.

We consider several stopping criteria to terminate the tree growing. Beside the standard node impurity (the fraction of examples of the majority class) we can use also the node frequency (number of examples in a node), the depth of the tree and the above mentioned χ^2 test of significance.

Figure 5: Results of the ETree algorithm

```
income = high: A (5.0)
income = low
|  account = high: A (2.0)
|  account = low: n (2.0)
|  account = medium
|  |  unemployed = no: A (1.0)
|  |  unemployed = yes: n (2.0)
```

```
account = high: A (4.0)
account = low
|  income = high: A (2.0)
|  income = low: n (2.0)
account = medium
|  income = high: A (1.0)
|  income = low: n
|  |  unemployed = no: A (1.0)
|  |  unemployed = yes: n (2.0)
```

```
sex = female
|  income = high: A (3.0)
|  income = low
|  |  account = high: A (1.0)
|  |  account = low: n (1.0)
|  |  account = medium: n (1.0)
sex = male
|  income = high: A (2.0)
|  income = low
|  |  account = high: A (1.0)
|  |  account = low: n (1.0)
|  |  account = medium
|  |  |  unemployed = no: A (1.0)
|  |  |  unemployed = yes: n (1.0)
```

```
unemployed = no: A (7.0/1.0)
unemployed = yes
|  account = high: A (2.0)
|  account = medium: n (2.0)
|  account = low
|  |  sex = male:  A (1.0)
|  |  sex = female
|  |  |  income = high: A (1.0)
|  |  |  income = low: n (1.0)
```

Source: the author.

The available parameters of the ETree algorithm can be summarized in three areas:

1. for tree growing:
   - class attribute that divides data into classes,
   - input attributes used for the data mining task,
   - max. number of "best" attributes considered for every split,
2. stopping criterion:
   - max. depth of the tree,
   - max. node impurity,
   - min. node frequency,
   - testing the significance of a split,
3. for tree saving:
   - min. tree quality,
   - only tree with full depth.

The quality of the exploration tree is evaluated using the classification accuracy on training data. This quantity, like the confidence of an association rule, shows how good the tree corresponds to the given data (and only trees that satisfy the given lower bound are produced at the output of the algorithm). So we are primary interested in description not classification; in this situation, each tree should be inspected and interpreted by the domain expert. The last parameter mentioned above allows to decide whether to save only full generated trees or to save also their sub-trees starting from root.

3. Experiments in the Loan Application Domain

We used the state-of-the-art C4.5 and CART algorithms implemented in weka, a widely used publicly available data mining software (Hall et al., 2009) and our implementation of ETree to analyze several data sets from the credit risk assessment domain. Their characteristics are shown in Table 1. We give for each data set the number of examples, the number of attributes and the percentage of examples within each class. As can be seen, all data sets can be used for binary classification tasks. The first three data sets comes from the UCI Machine Learning Repository that contains a number of benchmark data sets used by the machine learning community to evaluate various machine learning and data mining algorithms (Lichman, 2013). Australian credit data concerns credit card applications. The dataset contains a mixture of continuous, nominal with small numbers of values, and nominal with larger numbers of values attributes. All attribute names and values have been changed to meaningless symbols to protect confidentiality of the data. This data set has been for the first time used by Quinlan in 1987. German credit data were provided by prof. Hofmann from University Hamburg, this data consists of 7 numerical and 13 categorical input attributes. Japan credit data set represents consumer loans of a specific purpose (e.g car, or PC); this data set was prepared by Chiharu Sano in 1992. The Discovery Challenge data set was prepared for the Discovery Challenge workshop held at the European conference of Principles and Practice on Knowledge Discovery PKDD 1999 (Berka, 1999), the analyzed table (6181 examples, 7 input attributes) is an excerpt from the whole database used in the workshop.

Table 1: Basic characteristics of the used data sets

Name	Number of examples	Number of input attributes	Percentage of examples within each class
Australian credit	690	15	56/44
German credit	1000	20	70/30
Japan credit	125	10	68/32
Discovery Challenge	6181	7	88/12

Source: the author.

The goal of the experiments was to test the ETree algorithm when building description trees. We thus evaluate the results only on the training data. We run two types of experiments. Our aim in experiment 1 was to create single, most accurate tree. Tree growing in ETree was thus not restricted by any of the parameters min. node impurity, min. node frequency. In C4.5 and CART we disable pruning of the tree. The results are given in Table 2. It can be seen that the results of all three algorithms are almost the same in terms of classification accuracy on training data computed for each class separately (the numbers in Table 2 give for each used algorithm the classification accuracy for the majority class and the classification accuracy for the minority class). The goal in experiment 2 was to build more exploration trees. We set the minimal number of examples in a leaf to 2 for the Australian and Japan data, to 15 for the German data and to 10 for Challenge data in all three algorithms. In ETree, we further set the max. number of best attributes considered for splitting to 2 and the depth of the tree corresponding to the depth of the tree generated in C4.5. The results for this experiment (Table 3, the meaning of the numbers is the same as in Table 2) show, that ETree always found a tree that was better than that generated by C4.5 or CART.

Table 2: Summary of the results for Experiment 1

Data	C4.5	CART	ETree
Australian credit	0.9896/0.9934	0.9846/1.0000	0.9846/1.0000
German credit	1.0000/1.0000	1.0000/1.0000	1.0000/1.0000
Japan credit	1.0000/1.0000	1.0000/1.0000	1.0000/1.0000
Discovery Challenge	0.9886/1.0000	0.9886/1.0000	0.9886/1.0000

Source: author

Table 3: Summary of the results for Experiment 2

Data	C4.5	CART	ETree
Australian credit	0.9208/0.8907	0.9301/0.7867	0.9565/0.9143
German credit	0.7865/0.6482	0.7977/0.6384	0.8640/0.6733
Japan credit	0.8791/0.8530	0.8511/0.8387	0.9360/0.8870
Discovery Challenge	0.9213/0.8800	0.9248/0.8645	0.9443/0.9503

Source: author

4. Related Work

The exploration trees, especially when used for classification, can be understood as a kind of tree-based ensembles. From this point of view, we can find some related work in the area of combining classifiers. AdaBoost creates a sequence of models (trees) where every model in

the sequence focuses on examples wrongly classified by its predecessors. New examples are then classified by weighted voting of the models (Freund and Schapire, 1996). Random forest algorithm builds a set of trees by randomly splitting the data into training subsets and by randomly selecting a subset of input attributes to build an individual tree. New example is classified by majority vote of the trees (Breiman, 2001). The Option Trees algorithm creates a single tree that beside "regular" branching nodes contains also so called option nodes that include more attributes proposed to make a split. This tree thus represents a set of trees that differ in the splitting attribute used in the option node (Kohavi and Kunz, 1997). The ADTree (alternating decision trees) algorithm builds a sequence of trees with increasing depth. These trees are represented in a compact form of the tree with the maximal depth where each branching node is replaced by a pair [classifying node, branching node], where classifying node is a leaf node at the position of branching (Freund and Mason, 1999). All these algorithms focus on building ensemble classifiers, so the individual trees are not presented to the user.

5. Conclusion

The paper presents ETree, a novel algorithm for building so called exploration trees. Its main difference to standard TDIDT algorithms is the possibility to use more attributes to create a split and thus the possibility to build more trees that describe given data. The resulting trees can be used for both classification and description (segmentation).

We empirically evaluated our algorithm by comparing its classification accuracy (computed for the training data) on some benchmark data with the state-of-the-art algorithms C4.5 and CART. The first experiment shows that when giving no restrictions that will reduce the tree growing, the “best” tree generated by all three algorithms is the same. The second experiment shows, that when building more trees with setting that correspond to a pruned version of the tree generated by C4.5, our algorithm creates (among the trees generated for given data) a better tree than C4.5 and CART. The reason can be twofold: (1) C4.5 and CART are optimized to perform well on unseen data and thus do not cover the training data as precise as possible, and (2) the greedy search need not to find the best tree (especially if the selection of the splitting attribute is based on few examples, that is typical for branching nodes far from the root).

Acknowledgements

The paper was processed with contribution of long term institutional support of research activities by Faculty of Informatics and Statistics, University of Economics, Prague.

References

[1] BERKA, P. 1999. Workshop notes on discovery challenge. 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases. Prague : University of Economics. 1999. 64 pp.

[2] BERKA, P. 2011. ETree miner: a new GUHA procedure for building exploration trees. In KRYSZKIEWICZ et al., (eds.) Foundations of intelligent systems. 19th Int. Symposium on Methodologies for Intelligent Systems ISMIS 2011. New York : Springer, 2011, pp. 96-101.

[3] BIGGS, D., DEVILLE, B., SUEN, E. 1991. A method of choosing multiway partitions for classification and decision trees. In Journal of Applied Statistics, 1991, vol. 18, iss. 1, pp. 49-62.

[4] BREIMAN, L. 2001. Random forests. In Machine Learning, 2001, vol. 45, iss. 1, pp. 5-32.

[5] BREIMAN, L. et al. 1984. Classification and regression trees. Boca Raton : Chapman and Hall/CRC, 1984.

[6] FREUND, Y., MASON, L. 1999. The alternating decision tree learning algorithm. In Proceedings of the 16th International conference on machine learning, Morgan Kaufman, 1999, pp. 124-133.

[7] FREUND, Y., SCHAPIRE R. E. 1996. Experiments with a new boosting algorithm. In proceedings of the 13th International conference on machine learning, Morgan Kaufman, 1996, pp. 148-156.

[8] HALL, M. et al. 2009. The WEKA data mining software: an update. In Association for computing machinery, Special interest group on knowledge discovery in data explorations newsletter, 2009, vol. 11, iss. 1, pp. 10-18.

[9] KOHAVI, R., KUNZ, C. 1997. Option decision trees with majority notes. In Proceedings of the 14th International conference on machine learning, Morgan Kaufman, 1997, pp. 161-169.

[10] LICHMAN, M. 2013. UCI machine learning repository. Irvine : University of California, School of information and computer science, http://archive.ics.uci.edu/ml.

[11] QUINLAN, J. R. 1986. Induction of decision trees. In Machine Learning, 1986, vol. 1, iss.1, pp. 81-106.

[12] QUINLAN, J. R. 1993. C4. 5: programs for machine learning. Morgan Kaufman, 1993.