An Approach to Mining Data with Continuous Decision Values

Hung Son Nguyen[1,2], Marta Łuksza[1,2], Ewa Mąkosa[1,2], and Jan Komorowski[1]

[1] The Linnaeus Centre for Bioinformatics, Uppsala University
Husargatan, 3, Uppsala, SE-751 24, Sweden.
[2] Faculty of Mathematics, Informatics and Mechanics, Warsaw University,
Banacha 2, 02-097, Warsaw, Poland

Abstract. We propose a novel approach to discover useful patterns from ill-defined decision tables with a real value decision and nominal conditional attributes. The proposed solution is based on a two-layered learning algorithm. In the first layer the preference relation between objects is approximated from the data. In the second layer the approximated preference relation is used to create three applications: (1) to learn a ranking order on a collection of combinations, (2) to predict the real decision value, (3) to optimize the process of searching for the combination with maximal decision.

1 Introduction

Mining data sets with continuous decision attribute is one of the most challenging problems in KDD research. The most well known approach to this problem is a regression method which constructs approximation of the decision attribute (unknown variable) with a function(linear, quadratics or other) of the conditional attributes (known variables).

Many data mining problems (e.g. in bioinformatics) force us to deal with ill-defined data, i.e., data sets with few objects but a large number of attributes. In addition, attributes may be of different types: some of them are nominal and the other continuous. The regression method can deliver a solution for the prediction task, but the output model is very hard to interpret and to draw conclusions. Moreover, in statistical terms, the small number of examples makes the regression model less significant. Furthermore, in many applications, the description task is even more important than the prediction task, since it helps discover patterns (rules) revealing the real mechanisms hidden behind the data.

In this paper we propose an alternative method designed to manage with these problems. Our method is based on a layered learning idea and decision rule techniques. Instead of searching for the direct description of decision attribute, we decompose this task into several learning subtasks. The first subtask is to approximate the preference relation between objects from the data. Using approximate preference relation we solve other subtasks such as,

for instance, learning ranking order, prediction of continuous decision value, or minimization of the cost of maximal decision value searching process.

2 Basic notions

We use the notion of decision table to describe the data. Intuitively, a decision table is a rectangular data table with rows containing description of objects. Usually columns are called attributes (or features or variables) among which one distinguished column is called decision.

Therefore, decision table consists of a collection of condition attributes $A = \{a_1, ...a_k\}$ and one decision attribute *dec*. All of them are determined on a finite set of objects U. Formally, decision table (see [2]) is a pair $\mathbb{S} = (U, A \cup \{dec\})$, where U is a non-empty, finite set of *objects* and A is a non-empty, finite set, of *attributes*. Each $a \in A \cup \{dec\}$ corresponds to a function $a : U \to V_a$, where V_a is called the *value set* of a.

In this paper we consider a special type of decision tables with nominal condition attributes and continuous decision attribute. This kind of decision tables is hard to deal with and has proven to be too difficult for many data mining methods. For example, regression methods requires conversion of nominal attributes in to continuous ones, while rule-based methods require a discretization of the decision attribute.

3 A rule-based approach to continuous decision

Layered learning is a well known idea in machine learning study and has many successful applications, see [4]. The main principle of layer learning is based on a decomposition of the complex learning task into subtasks and construction of their solutions. Usually, these subtasks are located in a hierarchical tree, and solutions (outputs) of subtasks in the lower layers are used to resolve the subtasks in the higher layers.

The proposed solution to the problem of data with continuous decision is based on layered learning. Instead of searching for a direct description of the decision attribute, we decompose this task into several learning subtasks.

3.1 Learning the preference relation

Let a decision table $\mathbb{S} = (U, A \cup \{dec\})$ with continuous decision, i.e., $V_{dec} = \mathbb{R}^+$, be given.

As it is in the standard classification problem setting, the value of the decision attribute *dec* is determined only on a subset of the universe $\mathcal{X}$ of all possible condition attribute values combinations.

We define a parameterized relation $PREF_\varepsilon \in \mathcal{X} \times \mathcal{X}$ as follows:

$$(x, y) \in PREF_\varepsilon \Leftrightarrow dec(x) - dec(y) > \varepsilon$$

where ε is called a tolerance parameter. We call the relation $PREF_\varepsilon$ a *preference relation*, since $x\ PREF_\varepsilon\ y$ means x is more preferred than y. Let us define a function $\theta_\varepsilon : \mathcal{X} \times \mathcal{X} \to \{-1, 0, 1\}$ as follows:

$$\theta_\varepsilon(x,y) = \begin{cases} 1 & \text{if } dec(x) - dec(y) > \varepsilon \\ 0 & \text{if } |dec(x) - dec(y)| \leq \varepsilon \\ -1 & \text{if } dec(x) - dec(y) < -\varepsilon \end{cases}$$

Then the preference relation $PREF_\varepsilon$ can be defined by the function θ_ε by

$$(x,y) \in PREF_\varepsilon \Leftrightarrow \theta_\varepsilon(x,y) = 1$$

Our learning algorithm for preference relation is performed on a new decision table $\mathbb{S}^* = (U^*, A^* \cup \{d^*\})$. This new table, called *the difference table*, is constructed from the given decision table $\mathbb{S} = (U, \{a_1, ..., a_k, dec\})$ as follows:

$$\begin{aligned} U^* =& U \times U = \{(x,y) : x, y \in U\} \\ A^* =& \{a_j^* : U \times U \to V_{a_j} \times V_{a_j} \text{ for } j = 1, ...k\} \\ & a_j^*(x,y) = (a_j(x), a_j(y)) \text{ for any pair of objects } x, y \in U \\ & d^* : U \times U \to \{-1, 0, 1\} \\ & d^*(x,y) = \theta_\varepsilon(x,y) \text{ for any pair of objects } x, y \in U \end{aligned}$$

One can apply any learning algorithm to the difference table. Our experiments were done by using decision rule (based on rough set algorithms), Naive Bayes algorithm, nearest neighbors, decision tree, and boosting algorithms for nearest neighbors and decision tree. The results are obtained by using Rosseta [1] and WEKA [5] systems.

If the original decision table contains n objects, then the difference table will have n^2 objects. We considered this transformation a way of dealing with the small sized input sets. Using the same learning algorithm, one can expect a higher statistical significance of results obtained from difference table than of those obtained by the original decision table.

Another very interesting problem is how to evaluate the quality of the preference learning algorithm. Based on the difference table, every learning algorithm constructs a classification algorithm, called a classifier. Usually, classifiers are more general than the function d^*, i.e., they are determined also for those pairs (x, y) which not necessarily belong to $U \times U$. In our consideration, such classifiers can be treated as approximations of the preference relation $PREF_\varepsilon$.

Let us denote by $\pi_{\mathbf{L},U}$ the classifier extracted from difference table $\mathbb{S}^*$ by using learning algorithm $\mathbf{L}$. Such classifier is a function

$$\pi_{\mathbf{L},U} : \mathcal{X} \times \mathcal{X} \to \{-1, 0, 1, unknown\}$$

determined for any pair of objects from $\mathcal{X}$.

Let $t \in \mathcal{X}$ be a test case, the accuracy of learning algorithm $\mathbf{L}$ on the object t is defined by

$$accuracy_{\mathbf{L},U}(t) = \frac{|\{u \in U : \pi_{\mathbf{L}}(u,t) = \theta_{\varepsilon}(u,t)\}|}{|U|}$$

The accuracy of the learning algorithm $\mathbf{L}$ on the test set $V \subset \mathcal{X}$ is computed as an average accuracy on test objects:

$$accuracy_{\mathbf{L},U}(V) = \frac{1}{|V|}\sum_{t \in V} accuracy_{\mathbf{L},U}(t)$$

3.2 Ranking learning

Ranking learning can be understood as a problem of reconstruction of the correct ranking list of a set of objects. In this section we present a simple algorithm for reconstruction of a ranking list using approximated preference relation, which has been described in the previous section.

Let us assume that $\mathbb{S} = (U, A \cup \{dec\})$ is a training data set and $(u_1, ..., u_n)$ is an ordered sequence of objects from U according to dec, i.e.,

$$dec(u_1) \leq dec(u_2) \leq ... \leq dec(u_n).$$

The problem is to reconstruct the ranking list of objects from a test data set $\mathbb{S}' = (V, A \cup \{dec\})$ without using decision attribute dec.

Our algorithm is based on the round robin tournament system which is carried out on the set of objects $U \cup V$. Similarly to football leagues, every object from V playing the tournament obtains a total score which summarizes its all played games. The objects from V are sorted with respect to their scores.

Assume that we have applied the learning algorithm $\mathbf{L}$ on the difference table constructed from training data set $\mathbb{S} = (U, A \cup \{dec\})$, and let $\pi_{\mathbf{L},U}$ be the output classifier of this algorithm. The classifier can be treated as a referee in the match between two objects in our tournament system. The *total score* of an object $x \in V$ is computed by

$$Score(x) = \sum_{y \in U \cup V} w(y) \cdot \pi_{\mathbf{L},U}(x,y)$$

where $w(y)$ is a weighting parameter that measures the importance of the object y in our ranking algorithm. We propose to define those weights by

$$w(y) = \begin{cases} 1 & \text{if } y \text{ is a test object, i.e., } y \in V; \\ 1 + \frac{i}{n} & \text{if } y = u_i \in U. \end{cases}$$

This ranking algorithm gives a higher score to those objects that recorded most victories over high valued objects.

The algorithm can be applied for all the objects from $U \cup V$. In such situation we say that objects from V are *embedded* in the ordered sequence $(u_1, u_2, ..., u_n)$.

To measure the quality of a ranking algorithm it is necessary to have a method to express the agreement between two ranking lists (one original and one computed by our algorithm) for the same set of objects. There are several well known "compatibility tests" for this problem, e.g., Spearman R, Kendall τ, or Gamma coefficients, see [3]. If the proper ranking list of V is denoted by $X = (x_1, x_2..., x_k)$, then the ranking list obtained from our algorithm can be treated as a permutation of elements of V, and represented by $Y = (x_{\sigma(1)}, x_{\sigma(2)}, ..., x_{\sigma(k)})$, for some permutation $\sigma : \{1, ..., k\} \to \{1, ..., k\}$. The Spearman coefficient is computed by using the formula:

$$R = 1 - \frac{6 \sum_{i=1}^{k} (\sigma(i) - i)^2}{k(k-1)(k+1)} \tag{1}$$

The Spearman coefficient can take any value of the interval $[-1; 1]$.

3.3 Prediction Problem

In this section we present a decision prediction algorithm which uses the approximate preference relation and ranking learning algorithm as components.

Let the training set of objects $U = \{u_1, ...u_n\}$ be given. The prediction algorithm computes the decision value of the test object $x \notin U$ in two steps:

Algorithm 1 Prediction algorithm

Input: The set of labeled objects U and unlabeled object x;
parameters: learning algorithm $\mathbf{L}$;

Output: A predicted decision for x;

1: Embed the object x into the sequence $(u_1, u_2, ..., u_n)$ by applying ranking algorithm for objects from $\{x\} \cup U$ using $\mathbf{L}$ and decision table for U;
2: Let us assume that x is embedded between u_i and u_{i+1};
3: Return $prediction(x) = \frac{dec(u_i) + dec(u_{i+1})}{2}$ as a result of prediction.

The error rate of the prediction algorithm on the set of testing objects is measured by

$$error(V) = \frac{1}{card(V)} \sum_{x \in V} |dec(x) - prediction(x)|$$

It is obvious that the smaller the error rate, the better the prediction algorithm is.

3.4 Dynamic ranking algorithm

The quality of ranking algorithm can be low due to the small number of objects. In many applications the number of training objects is increasing in time, but it is connected with certain cost of examination. As an example may serve biological experiment data, where every training object needs to be confirmed in expensive laboratory tests.

In this section we treat a ranking problem as an optimization problem, in which we wish to get the highest value element using as low as possible the number of requests, i.e., to minimize the number of examinations and the cost of the whole process.

We propose another ranking method based on active learning approach. Instead of using a randomly selected test set, the learning algorithm has access to the set of unlabeled objects and can request the labels for some of them. In the biological research example, unlabeled objects correspond to the samples that were not examined yet; a request for a label responds to performing single laboratory test that gives the decision value of the requested sample. In the *dynamic ranking algorithm*, the unlabeled objects are requested according to the actual ranking list. Then, after obtaining the labels of new objects, the ranking list will be corrected.

Algorithm 2 presents the main framework of our idea. The notion of STOP CONDITION is defined differently for each optimization problem and refers to cost limits for the considered process.

Algorithm 2 The dynamic ranking algorithm

Input: The set of labeled objects U and unlabeled objects V;
parameters: learning algorithm **L** and positive integer *request_size*;
Output: A list of objects to be requested; Ranking of elements in the U_2 in the *RankList*;

1: $U_1 \leftarrow U$; $U_2 \leftarrow V$;
2: *RankList* $\leftarrow$ [ ]; *//the empty list*
3: **while** *not* STOP CONDITION **do**
4: Rank elements of U_2 by using **L** and decision table for U_1; Let this ranking list be: $(x_1, x_2, ...)$;
5: **for** $i = 1$ **to** *request_size* **do**
6: *RankList.append*(x_i)
7: $U_1 \leftarrow U_1 \cup \{x_i\}$; $U_2 \leftarrow U_2 \setminus \{x_i\}$;
8: **end for**
9: **end while**

4 Experimental results

Our method was tested on an artificially generated decision table. The table consisted of six nominal condition attributes of values from the set $\{1, 2\}$ and

a real decision attribute. As each of the condition attributes had two possible values, the whole space of objects was of size $2^6 = 64$. Values of the decision attribute were calculated from the function:

$$dec = e^{a_1{}^{a_2}} + (a_1 + a_2 + a_3 + a_4 + a_5) * a_6/a_3 + \sin(a_4) + \ln(a_5) + noise$$

where $a_1, a_2, a_3, a_4, a_5, a_6$ are the attributes' values, and *noise* is a random value from the interval $[-1, 1]$ obtained using the uniform distribution on this interval. All the resulting decision values were in the interval $[5.772, 32.732]$. The tolerance parameter ε was set to 0.7.

Several different algorithms used in the model training step were compared in the tests:

- Decision rule method based on the rough set approach (see [1]) with the SAVgenetic reducer. In the result tables referred to as "Decision Rules".
- Naive Bayes algorithm from the Rosetta system (referred to as "Naive bayes")
- Nearest neighbor like algorithm using non-nested generalized exemplars, implemented in the WEKA system. It is referred to as "Nnge" in Table 1.
- Multi-boosting of the Nnge algorithm in the WEKA system. Multi-boosting is an extension to the AdaBoost technique for forming decision committees. Multi-boosting can be viewed as combining AdaBoost with wagging. It is designed to harness both AdaBoost's high bias and variance reduction with wagging's variance reduction. For more information, see [6]. In the result tables referred to as "MultiBoosting Nnge".
- C4.5 decision tree, impemented in WEKA. For more information, see [7]. Referred to as "C4.5".
- Multi boosting of C4.5 in the WEKA system. ("MultiBoosting C4.5")

We tested the quality of the rankings obtained by computing the Spearman coefficient of the correlation with the true ranking using Equation 1. In every iteration of experiment, the original data set was partitioned with proportion (50%:50%) into the training set U and test set V. The mean Spearman coefficient value and mean model accuracy for every learning algorithm are computed as average results over 10 iterations of experiments and reported in Table 1.

To test the real decision value prediction accuracy there were 7-fold cross validation tests performed on the original data set. The mean absolute error is reported.

The quality of the dynamic ranking algorithm is measured by the position of the best candidate in the resulting ranking list. The lower the resulting position, the better. Random shuffle of elements would result in expected position of this element in the middle index. The original data set was split in half to form the train data set U and the test data set V. Table 1 shows the positions of the best candidates in the ranking list constructed by our

approach. The experiment with every learning algorithm was repeated 10 times and we return the average position of the best candidate within those 10 repetitions. The tests were performed with *request_size* value set to 2. In order to compare the dynamic ranking approach with the static one, we calculated the Spearman coefficient.

Table 1. Experiment results achieved using six different learning algorithms. The first column reports the results ranking quality tests: Spearman coefficient and model accuracy on change table; second column reports the Pearson correlation coefficient and prediction mean absolute error; third column summarizes the results obtained while testing the dynamic ranking algorithm, average position and final ranking Spearman coefficient.

Learning	Ranking		Prediction		Dynamic Ranking	
algorithm	Spearman	acc.(%)	Pearson	pred.error	pos.	Spearman coef.
Decision Rules	0.893	83.28%	0.9653	1.4547	1.3	0.9501
Naive Bayes	0.7984	78.52%	0.5948	3.8336	1.3	0.8540
Nnge	0.777	77.19%	0.9178	1.8245	2.5	0.9165
MultiBoosting Nnge	0.8318	80.27%	0.9184	1.6244	1.6	0.9433
C45	0.7159	75.7%	0.8372	2.2108	2.7	0.8736
MultiBoosting C45	0.8536	80.74%	0.9661	1.3483	1.6	0.9475

5 Conclusions

We presented a novel approach to mining ill-defined data with continuous decision. Experimental results seem to be very interesting and promising. We plan to make more experiments on other types of data, especially on those coming from the proteochemometrics study, to confirm the power of the proposed method.

Acknowledgements:

The authors would like to thank Helena Strömbergsson and Peteris Prusis for valuable discussions and suggestions during preparation of this paper.

The research has been done during the visit of Dr Nguyen Hung Son at the The Linnaeus Centre for Bioinformatics, Uppsala University. This research was partially supported by the ARI project, the grant 3T11C00226 from Ministry of Scientific Research and Information Technology of the Republic of Poland, and a grant from the Knut and Alice Wallenberg foundation.

References

1. Komorowski, J. Skowron, A. and Ohrn, A. 2000. The ROSETTA system, Klosgen W and Zytkow J, Handbook of Data Mining and Knowledge Discovery, Oxford University Press, ch. D.2.3.

2. Pawlak Z.: *Rough sets: Theoretical aspects of reasoning about data*, Kluwer Dordrecht, 1991.
3. S. Siegel, N.J. Castellan (1988). Nonparametric statistics for the behavioral sciences (2nd ed.) New York: McGraw-Hill.
4. Stone, P.: Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer. The MIT Press, Cambridge, MA (2000)
5. I. H. Witten, E. Frank. Data Mining: Practical machine learning tools with Java implementations. Morgan Kaufmann, San Francisco, 1999.
6. Geoffrey I. Webb (2000). MultiBoosting: A Technique for Combining Boosting and Wagging. Machine Learning, 40(2): 159-196, Kluwer Academic Publishers, Boston
7. Ross Quinlan (1993). C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, San Mateo, CA.