2010 International Conference on Pattern Recognition

Feature Extraction from Discrete Attributes

Olcay Taner Yıldız
Department of Computer Engineering, Işık University, TR-34980, Istanbul, Turkey
olcaytaner@isikun.edu.tr

Abstract

In many pattern recognition applications, first decision trees are used due to their simplicity and easily interpretable nature. In this paper, we extract new features by combining k discrete attributes, where for each subset of size k of the attributes, we generate all orderings of values of those attributes exhaustively. We then apply the usual univariate decision tree classifier using these orderings as the new attributes. Our simulation results on 16 datasets from UCI repository [2] show that the novel decision tree classifier performs better than the proper in terms of error rate and tree complexity. The same idea can also be applied to other univariate rule learning algorithms such as C4.5Rules [7] and Ripper [3].

1. Introduction

In pattern recognition the knowledge is extracted as patterns from a training sample for future prediction. Most pattern recognition algorithms such as neural networks [1] or support vector machines [8] make accurate predictions but are not interpretable, on the other hand decision trees are simple and easily comprehensible. They are robust to noisy data and can learn disjunctive expressions. Surveys of work on constructing and simplifying decision trees can be found in [5] and [9].

Decision trees are tree-based structures where each internal node implements a decision function, $f_m(\boldsymbol{x})$, each branch of an internal node corresponds to one outcome of the decision, and each leaf corresponds to a class. In a univariate decision tree [7], the decision at internal node m uses only one attribute, i.e., one dimension of $\boldsymbol{x}$, x_j. If that attribute is discrete, there will be L children (branches) of each internal node corresponding to the L different outcomes of the decision. ID3 is one of the best known univariate decision tree algorithm with discrete features [6].

One of the drawbacks of the L-ary splits is that the training examples are separated into small subsets, which in turn gives us a poor predictor for the unseen test instances. One can convert discrete features having $L > 2$ different values to L binary features using 1-of-L encoding, this will result in a larger tree than the former. Although there are alternative approaches to handle the selection bias that favors the attributes having many values over those with few values [6], [4], those approaches can not help if the attributes have nearly similar number of values. Another drawback of the univariate decision trees with discrete attributes is that there is only a single possible split for each discrete attribute. On the other hand, there are $n - 1$ di ff er ent possible splits for a continuous attribute, where n represents the number of distinct values of that continuous attribute.

To increase the number of distinct splits for datasets with discrete attributes, we define k-ordering, where for each subset (size k) of the attributes, we combine them by generating all possible orderings of the values of those attributes exhaustively. Our approach as feature extraction can be connected to PCA with three important differences, (i) we use discrete features, PCA uses continuous features, (ii) we use a subset of features, PCA uses all of them (iii) we increase the dimensionality, PCA reduces it. Then we apply the usual univariate decision tree algorithm using these orderings as the new attributes. In this way, the number of distinct possible splits for each extracted feature will be $n_1 \times n_2 \times \ldots \times n_k$, where n_i represents the number of distinct values of the selected feature i.

This paper is organized as follows: We give the definition of k-ordering in Section 2, its application to univariate decision trees in Section 3, give experimental results in Section 4, discuss possible extensions and conclude in Section 5.

1051-4651/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPR.2010.952

2. k-ordering

Let $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, ..., $\boldsymbol{x}_d$ be d discrete attributes of a dataset D. Each attribute $\boldsymbol{x}_i$ can have n_i distinct values which can be represented as v_{i1}, v_{i2}, ..., v_{in_i}.

Definition 1. *For each k attributes $\boldsymbol{x}_{s_1}$, $\boldsymbol{x}_{s_2}$, ..., $\boldsymbol{x}_{s_k}$ from a d dimensional dataset D, k-ordering is defined as the permutation list (p_1, p_2, ..., p_k), where p_i is a permutation of values of the feature s_i (a permutation of v_{s_i1}, v_{s_i2}, ..., $v_{s_in_{s_i}}$).*

For example, ((red, blue, green), (yes, no), (large, extralarge, small, medium)) is a 3-ordering, where selected $k = 3$ features have three, two and four distinct values respectively. Note that we can use both categorical discrete attributes (such as red, green, blue) as well as nominal discrete attributes (such as cold, neutral, warm, very warm) in defining the k-ordering. There are $\prod_{s_1,s_2,\ldots,s_k \in 1,2,\ldots,d} n_{s_1}! n_{s_2}! \ldots n_{s_k}!$ distinct k-orderings of a d dimensional dataset. As an example, given a dataset with two dimensions having values x_1 = (red, green, blue) and x_2 = (yes, no), there are 8 (3! + 2!) and 24 (3! 2! + 2! 3!) distinct 1 and 2-orderings respectively.

We now define a relational operator between two instances. Given a k-ordering of a dataset D and two instances (x_1, x_2, ..., x_d) and (y_1, y_2, ..., y_d) from this dataset;

Definition 2. *(x_{s_1}, x_{s_2}, ..., x_{s_k}) $\prec$ (y_{s_1}, y_{s_2}, ..., y_{s_k}) if and only if $x_{s_i} = y_{s_i}$ for i = 1, ..., $t \geq 1$, and $x_{s_{t+1}}$ comes before $y_{s_{t+1}}$ in permutation p_{t+1}.*

For example, given the 2-ordering ((red, blue, green), (no, yes)), all possible values of the instances can be sorted as (red, no) $\prec$ (red, yes) $\prec$ (blue, no) $\prec$ (blue, yes) $\prec$ (green, no) $\prec$ (green, yes).

3. Application to univariate decision trees

We now apply k-ordering definition to find k-ordered splits in the univariate decision tree algorithm. The idea is as follows: At each decision node, we generate all possible k-orderings. Since each k-ordering defines a new attribute, $\prod_{s_1,s_2,\ldots,s_k \in 1,2,\ldots,d} n_{s_1}! n_{s_2}! \ldots n_{s_k}!$ new attributes are generated. For each new attribute there are $n_{s_1} \ldots n_{s_k}$ distinct split points. For all attributes and for all split points of those attributes, we calculate impurity and choose one that has the minimum entropy. As usual univariate decision tree algorithm, tree construction continues recursively.

```
Best-k-OrderingandSplit(m, k)
1  bestImpurity = ∞
2  bestSplit = bestOrdering = nil
3  for each feature permutation (s_1, ..., s_k)
4     foreach permutation p_1 of values of x_{s_1}
5        ...
6        foreach permutation p_k of values of x_{s_k}
7           foreach split point sp = (v_1, ..., v_k)
8              if Impurity(sp, m) < bestImpurity
9                 bestImpurity = Impurity(sp, m)
10                bestSplit = (v_1, ..., v_k)
11                bestOrdering = (p_1, ..., p_k)
12 return bestSplit, bestOrdering
```

Figure 1. The pseudocode of the exhaustive k-ordered best split search algorithm: m: Decision node, k: parameter in the k-ordering

The pseudocode for finding the best k-ordered split at a decision node is given in Figure 1. First we initialize best split and best k-ordering (Line 2). For each k-permutation of the attributes (Line 3) we generate all possible k-orderings. Instead of generating all possible permutations of the values of the attributes beforehand, we traverse them iteratively (Lines 4-6). As explained above, each possible k-ordering defines a new attribute, so now we can search the best split point for that new attribute (Line 7). If a split (k-ordering and the split point for that ordering) has a smaller impurity than the best impurity so far, we update best split, ordering and impurity (Lines 9-11).

For example, given the 2-ordering ((long, short), (no, yes)), all possible split points will be (long, no) $\prec$ (long, yes) $\prec$ (short, no) $\prec$ (short, yes). Let say there are 5, 3, 2, 10 instances of class C_1 and 2, 4, 5, 4 instances of class C_2 having those values respectively. Then the split point (long, yes) will divide the instances into two where the left subtree will have $5 + 3 = 8$, $2 + 4 = 6$ instances from classes C_1 and C_2 respectively and the right subtree will have $2 + 10 = 12$, $5 + 4 = 9$ instances from classes C_1 and C_2 respectively.

Since we know which attributes are selected for the k-ordering after step 3, we can exhaustively make a table of counts for each k-ordering, where there are counts for a split point $v_1, v_2, \ldots, v_k$. Each count represents the number of instances who has values v_1 for feature s_1, v_2 for feature s_2, etc.

Table 1. Details of the datasets. d: Number of attributes, C: Number of classes, N: Sample size, n/v: n attributes of the dataset has v distinct values

Dataset	d	C	N	n/v
acceptors	88	2	3889	88/4
artificial	10	2	320	10/2
balance	4	3	625	4/5
car	6	4	1728	3/3, 2/4, 1/5
connect4	42	3	67557	42/3
donors	13	2	6246	13/4
hayesroth	4	3	160	1/3, 3/4
krvskp	36	2	3196	35/2, 1/3
monks	6	2	432	1/2, 3/3, 1/4
nursery	8	5	12960	1/2, 4/3, 2/4, 1/5
promoters	57	2	106	57/4
spect	22	2	267	22/2
splice	60	3	3175	60/4
tictactoe	9	2	958	9/3
titanic	3	2	2201	2/2, 1/4
vote	16	2	435	16/2

After filling this table, instead of counting which instances fell to the left (or right) of a split point (needed to calculate the impurity of a split point), we just add the counts.

The computational complexity of the algorithm is $\mathcal{O}(d^k(N + n_{s_1}!n_{s_2}!\ldots n_{s_k}!n_{s_1}\ldots n_{s_k}))$. Here $\mathcal{O}(d^k N)$ is the time required to fill the table of counts and $\mathcal{O}(d^k n_{s_1}!n_{s_2}!\ldots n_{s_k}!n_{s_1}\ldots n_{s_k})$ is the time required to search the best split using the table of counts.

4. Experiments

In this section, we compare the performance of our proposed decision tree algorithm (K-tree) with C4.5 in terms of generalization error and model complexity as measured by the number of nodes in the decision tree. We run our proposed algorithm for $k = 1, 2, 3$, and 4. Since the time complexity of the algorithm changes exponentially with the number of distinct values of an attribute (n_i), we run the algorithm for the datasets with $n_i \leq 5$. We use a total of 16 data sets where 14 of them are from UCI repository [2] and 2 are (*acceptors and donors*) bioinformatics datasets (see Table 1).

Our methodology in generating train, validation and test sets is as follows: A data set is first divided into two parts, with 1/3 as the test set, *test*, and 2/3 as the training set. The training set is resampled using 2×5 cross-validation to generate ten training and validation folds, tra_i, val_i, $i = 1, \ldots, 10$. tra_i are used to train the decision trees and val_i are used to prune the decision trees using cross-validation based postpruning. *test* is used to estimate the expected error of the decision trees.

Table 2 shows the average and standard deviations of error rates of decision trees generated using C4.5, and K-tree algorithms with $k = 1, 2, 3, 4$. We see from the results that, K-tree is significantly better than C4.5 in terms of error rate. In *krvskp*, *car*, *tictactoe*, *nursery* datasets, K-tree has 3, 6, 5, 10 times better error rate compared to C4.5. In some datasets such as *acceptors*, *balance*, *monks*, *splice*, *tictactoe* increasing k also increases accuracy. On the other hand, in other datasets such as *hayesroth* increasing k may cause overfitting (although there is pruning).

Table 3 shows the average and standard deviations of number of nodes of decision trees generated using C4.5, and K-tree algorithms with $k = 1, 2, 3, 4$. Similar to the results above, in all datasets except *balance* and *spect*, K-tree generates significantly smaller trees compared to C4.5. Especially in *donors*, *monks*, *nursery*, and *splice* the K-trees are at least 5 times smaller than the C4.5's trees. Without notably exceptions, as we increase k, the tree complexity decreases. Note that, if you only store the index of the new feature and the split point as integers, the decision nodes of K-tree's are equally complex as the decision nodes of C4.5.

5. Conclusions

In this paper, we propose a new schema to order a subset of k discrete attributes in decision trees. Using all orderings of values of those k attributes as new extracted features, we propose a novel univariate decision tree classifier. Our simulation results on 16 datasets show that our proposed algorithm performs better than C4.5 in terms of error rate and tree complexity.

The same idea can also be applied to other univariate rule learning algorithms such as C4.5Rules [7] and Ripper [3].

References

[1] E. Alpaydın. *Introduction to Machine Learning*. The MIT Press, 2010.

[2] C. Blake and C. Merz. UCI repository of machine learning databases, 2000.

Table 2. The average and standard deviations of error rates of decision trees generated using C4.5, and K-tree algorithms with k = 1, 2, 3, 4.

Dataset	C4.5	1-tree	2-tree	3-tree	4-tree
acceptors	15.66±1.74	12.65±0.82	12.04±0.94	-	-
artificial	0.74±1.56	0.74±1.56	0.74±1.56	0.74±1.56	0.00±0.00
balance	40.53±2.70	27.13±3.39	24.88±2.32	-	-
car	12.46±1.92	3.97±0.55	2.36±0.54	3.14±1.15	-
connect4	26.01±0.45	24.88±0.42	23.66±0.37	-	-
donors	7.76±0.69	6.22±0.34	6.68±0.43	-	-
hayesroth	26.73±1.50	21.45±4.09	27.82±4.78	25.82±4.83	28.00±5.30
krvskp	1.23±0.40	0.96±0.37	0.94±0.41	0.46±0.20	0.60±0.25
monks	14.72±6.12	12.71±5.79	0.00±0.00	0.00±0.00	0.00±0.00
nursery	5.50±0.45	1.66±0.25	0.43±0.23	0.60±0.17	-
promoters	24.44±10.29	27.22±12.20	20.56±3.97	-	-
spect	20.33±2.46	20.33±2.46	20.89±0.70	20.11±2.43	19.78±2.86
splice	9.76±0.87	7.11±0.83	6.11±0.56	-	-
tictactoe	22.78±1.65	10.53±3.67	8.97±2.67	4.59±2.47	5.56±3.19
titanic	21.80±0.54	21.72±0.53	21.32±0.41	21.29±0.44	-
vote	5.10±0.36	5.10±0.36	5.03±0.33	5.38±0.29	5.38±1.16

Table 3. The average and standard deviations of number of nodes of decision trees generated using C4.5, and K-tree algorithms with k = 1, 2, 3, 4.

Dataset	C4.5	1-tree	2-tree	3-tree	4-tree
acceptors	31.60±20.73	9.80±4.83	8.20±3.97	-	-
artificial	5.60±0.84	5.60±0.84	3.80±0.42	2.80±0.42	3.00±0.00
balance	9.80±7.00	13.80±4.57	11.60±2.63	-	-
car	62.70±8.55	32.60±3.34	19.50±2.12	16.20±0.92	-
connect4	887.00±115.99	575.70±65.82	314.10±58.83	-	-
donors	73.60±19.12	20.30±6.36	9.50±4.55	-	-
hayesroth	16.60±2.37	8.90±1.37	5.50±0.71	5.30±1.57	4.40±0.70
krvskp	29.80±4.44	26.90±3.14	17.20±1.69	12.00±1.33	8.70±1.34
monks	31.10±7.03	22.80±7.89	5.00±0.00	5.00±0.00	5.00±0.00
nursery	243.90±20.79	122.90±6.14	35.30±2.50	36.60±2.32	-
promoters	5.80±3.22	2.20±0.92	2.50±0.71	-	-
spect	1.80±2.53	1.80±2.53	1.50±1.58	1.80±1.32	2.10±1.85
splice	64.90±7.49	16.90±3.11	9.20±1.81	-	-
tictactoe	48.20±9.44	24.70±3.89	15.50±4.25	12.20±1.87	12.10±1.10
titanic	7.70±1.64	5.80±1.03	3.50±1.43	3.30±0.95	-
vote	2.90±1.20	2.90±1.20	3.00±1.76	2.20±0.63	2.00±0.00

[3] W. W. Cohen. Fast effective rule induction. In *The Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[4] R. L. de Mantaras. A distance -based attribute selection measure for decision tree induction. *Machine Learning*, 6:81–92, 1986.

[5] S. K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.

[6] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[7] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Meteo, CA, 1993.

[8] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

[9] O. T. Yıldız and E. Alpaydın. Linear discriminant trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(3):323–353, 2005.