Detecting Complex Dependencies in Categorical Data

Tim Oates, Dawn Gregory, and Paul R. Cohen
Computer Science Department, LGRC
University of Massachusetts
Box 34610
Amherst, MA 01003-4610
{oates,gregory,cohen}@cs.umass.edu

Abstract

Locating and evaluating relationships among values in multiple streams of data is a difficult and important task. Consider the data flowing from monitors in an intensive care unit. Readings from various subsets of the monitors are indicative and predictive of certain aspects of the patient's state. We present an algorithm that facilitates discovery and assessment of the strength of such predictive relationships called *Multi-stream Dependency Detection* (MSDD).

We use heuristic search to guide our exploration of the space of potentially interesting dependencies to uncover those that are significant. We begin by reviewing the dependency detection technique described in [3], and extend it to the multiple stream case, describing in detail our heuristic search over the space of possible dependencies. Quantitative evidence for the utility of our approach is provided through a series of experiments with artificially-generated data. In addition, we present results from the application of our algorithm to two real problem domains: feature-based classification and prediction of pathologies in a simulated shipping network.

1 Dependency Detection

A *dependency* is an unexpectedly frequent or infrequent co-occurrence of events over time. Our goal is to find dependencies between tokens contained in multiple streams. A stream is sequence of values produced over time, and a token is one of the finite set of values that a stream can produce. Dependencies across multiple streams may take many forms: perhaps token A in stream 1 predicts token B in stream 2, or perhaps token A in stream 1 *and* token C in stream 2 predict token B in stream 2. In general, if stream j contains t_j distinct tokens, there are $[\prod_{j=1}^{n} t_j + 1]^2$ possible dependencies between two items.

The dependency detection technique in [3] uses contingency tables to assess the significance of dependencies in a *single* stream of data. Let (t_p, t_s, δ) denote a dependency. Each dependency *rule* states that when the precursor token, t_p, occurs at time step i in the stream, the successor token, t_s, will occur at time step $i + \delta$ in the stream with some probability. When this probability is high, the dependency is strong.

Consider the stream ACBABACCBAABACBBACBA. Of all 19 pairs of tokens at lag 1 (e.g. AC, CB, BA, ...) 7 pairs have B as the precursor; 6 of those have A as the successor, and one has something other than A (denoted $\overline{\text{A}}$), as the successor. The following contingency table represents this information:

$Table(\text{B},\text{A},1) =$

		A	$\overline{\text{A}}$	*total*
	B	6	1	7
	$\overline{\text{B}}$	1	11	12
	total	7	12	19

It appears that A depends strongly on B because it almost always follows B and almost never follows anything else ($\overline{\text{B}}$). We can determine the significance of each dependency by computing a G statistic for

its contingency table:

$$G\begin{pmatrix} n_1 & n_2 & r_1 \\ n_3 & n_4 & r_2 \\ c_1 & c_2 & t \end{pmatrix} = 2\left[n_1 \log \frac{n_1 t}{r_1 c_1} + n_2 \log \frac{n_2 t}{r_1 c_2} + n_3 \log \frac{n_3 t}{r_2 c_1} + n_4 \log \frac{n_4 t}{r_2 c_2}\right]$$

For example, the contingency table shown above has a G value of 12.38, which is significant at the .001 level, so we reject the null hypothesis that A and B are independent and conclude that (B,A,1) is a real dependency.

We extend this technique to the multiple stream case by introducing the concept of a *multi-token*. A multi-token represents the value of any or all streams at any given time i. For a series with n streams, all multi-tokens will have the form $< x_1, \ldots, x_n >$, where x_j indicates the value in stream j. In order to support the "any or all" requirement, we add a special wildcard symbol, *, to the set of values that may appear in each stream. Thus we can indicate a "don't care" condition by placing an * in the appropriate stream.

For a multi-stream example, consider the following streams:

ACBABACCBAABACBACBAC
BACACABACBABABCABCAB

The dependency (<B,C>,<A,A>,1) indicated in boldface is significant at the .01 level with a G value of 7.21. The corresponding contingency table is:

$Table$(B,A,1) =

	<A,A>	$\overline{\text{<A,A>}}$	*total*
<B,C>	4	1	5
$\overline{\text{<B,C>}}$	2	12	14
total	6	13	19

We now have both syntax and semantics for multi-stream dependencies. Syntactically, a dependency can be expressed as a triple containing two multi-tokens (a precursor and a successor) and an integer (the lag). For each of the n streams, the multi-tokens contain either a token that may appear in the stream or a wildcard. Dependencies can also be expressed in the form $x \rightarrow_\delta y$ where x and y are multi-tokens. Semantically, this says the occurrence of x is indicative of or predicts the occurrence of y, δ time steps in the future.

2 Searching for Dependencies

The problem of finding significant two-item dependencies can be framed in terms of search. A node in the search space consists of a precursor/successor pair, a predictive rule. The goal is to find predictive rules that are "good" in the sense that they apply often and are accurate. The root of the search space is a pair of multi-tokens with the wildcard in all n positions. The children of a node are generated by replacing (instantiating) a single wildcard in the parent, in either the precursor or successor, with a token that may appear in the appropriate stream. For example, the node < A,* >→< *,X > has both < A,Y >→< *,X > and < A,* >→< B,X > as children.

The rule corresponding to a node is always more specific than the rules of its ancestors and less specific than any of its descendants. This fact can be exploited in the search process by noting that as we move down any branch in the search space, the value in the top left cell of the contingency table (n_1) can only remain the same or get smaller. This leads to a powerful pruning heuristic. Since rules based on infrequently co-occurring pairs of multi-tokens (those with small n_1) are likely to be spurious, we can establish a minimum size for n_1 and prune the search space at any node for which n_1 falls below that cutoff. In practice, this heuristic dramatically reduces the size of the search space that needs to be considered.

Our implementation of the search process makes use of *best first search* with a heuristic evaluation function. That function strikes a tunable balance between the expected number of hits and false positives for the predictive rules when they are applied to previously unseen data from the same source. We define *aggressiveness* as a parameter, $0 \leq a \leq 1$, that specifies the value assigned to hits relative to the cost associated with false positives. For a given node (rule) and its contingency table, let n_1 be the size of the top left cell, let n_2 be the size of the top right cell, and let t_S be the number of non-wildcards in the successor multi-token. The value assigned to each node in the search space is $S = t_S(an_1 - (1-a)n_2)$. High values of aggressiveness favor large n_1 and thus maximize hits without regard to false positives. Low aggressiveness favors small n_2 and thus minimizes false positives with a potential loss of hits. Since the size of the search space is enormous, we typically impose a limit on the number of nodes expanded. The output of the search is simply a list of the nodes, and thus predictive rules, generated.

3 Empirical Evaluation

In this section we evaluate the performance of the algorithm on artificially-generated data sets. The goal is to answer a variety of questions regarding the behavior of the algorithm over its domain of applications. Artificial data simplifies this task since the "real" dependencies are known, providing means for distinguishing structure in the data from noise.

Artificial data sets are generated by random sampling and applying a set of probabilistic *structure rules*: $R = \{(P, Pr_P, S, Pr_S)\}$. Each series is initialized by generating n streams of length l, sampled randomly from the token set T. Values for n, l, T, and R are determined by the experiment protocol. Default values are $n = 5$, $l = 100$, $T = \{$A,B,C,D,E$\}$, and $R = \{(<$ A,A,*,*,* $>, .1, <$ C,D,D,*,* $>, .8), (<$ * C,C,*,* $>, .1, <$ *,A,A,B,* $>, .8), (<$ *,*,D,D,* $>, .1, <$ *,*,D,C,B $>, .8)\}$.

Structure is then introduced into this random series in two phases: first, seed the precursors P into each time-slice with probability Pr_P; then, whenever a time-slice i matches the precursor of a rule r, insert the successor into time-slice $i + \delta$ with probability $Pr_S(r)$. For analysis, we can partition the resulting series into noise and structure by determining which components are predicted by the dependency rules $(P(r), S(r), \delta)$ for each structure rule $r \in R$.

In each experiment, we run one or more iterations of the search algorithm for each experiment condition. Unless different values are specified by the experiment protocol, we gather 5000 predictive rules with aggressiveness set to 0.5. These rules are post-processed as described below, and used to make *predictions* in ten new data sets generated from the same structure rules. The results are evaluated with respect to two factors: *predictive power* (the total number of predictions made) and *accuracy* (the percentage of the predictions that were correct). These factors are considered separately for the structure and noise portions of the data set.

3.1 Selecting the Best Dependency Rules

The MSDD search algorithm generates a large set of dependencies, from which we would like to select the most accurate and predictive rules. Since all our experiments depend on the quality of this selection process, the first question we wish to answer is, "what post-processing strategy will select the best predictive rules?" Although more sophisticated techniques may be needed to resolve redundancy, the simplest approach is to *filter* and *sort* the rules, first discarding rules that do not conform to certain criteria, and then prioritizing them according to some precedence function.

In this experiment, four different filter criteria are combined with six different sort functions for a total of 24 experiment conditions. The filter options discard rules under the following conditions: (1) never; (2) G not significant at the 0.05 level ($G < 3.84$); (3) $n_1 < 5$; and (4) $n_1 < n_2$. The remaining rules are then sorted according to one of these six functions: (1) a randomly selected number; (2) the G statistic (computed over the training data); (3) the number of true instances n_1; (4) the approximate

number of true predictions $n_1 \times t_S$; (5) the percentage of instances that are true $\frac{n_1}{n_1+n_2}$; and (6) the approximate percentage of predictions that are true, $\frac{n_1}{n_1+n_2} \times t_S$.

We ran five iterations of each condition on data sets with default structure. The results indicate that the highest predictive power and accuracy are achieved when discarding rules with less than 5 true instances (filter condition 3), and sorting them according to the G statistic (sort condition 2). This result is as expected: the rules that remain are unlikely to be spurious dependencies, and they are applied in order of their significance.

3.2 Comparison of Search Heuristics

Now that we know how to effectively use the output of MSDD, we can address important issues regarding the performance of the algorithm. In this experiment, we compare the performance of the S heuristic to other heuristics and across different levels of aggressiveness.

All the search heuristics used in this experiment are based on contingency table analysis of the dependency rules. In addition to the S heuristic, we also use:

1. A normalized S value $\frac{S}{(n_1+n_2)(n_1+n_3)}$, where S is normalized by its *expected count.*
2. The aggressiveness-weighted ratio of hits to false-positives, $\frac{an_1}{(1-a)n_2}$.
3. The agressiveness-weighted fraction of the instances that are hits, $\frac{an_1}{n_1+n_2}$.

The results (which are not included here due to space constraints) confirm that S is the best of these heuristics: it produces good accuracy and predictive power while allowing the user to tune the performance with the aggressiveness parameter; the other heuristics are not affected by tuning. As expected, high aggressiveness favors predictive power while low values favor accuracy.

3.3 Effects of Inherent Structure

Perhaps the most important question to be resolved is: How strong must a dependency be in order for it to be found by the algorithm? In practical terms, this involves two issues: how frequently a dependency occurs and how often the precursor multitoken appears but the successor multitoken does not. In this experiment, we generated 243 data sets of default size, with 1, 3, or 5 structure rules spanning all combinations of: precursor size $t_P \in \{1, 3, 5\}$, precursor probability $Pr_P \in \{.1, .2, .3\}$, successor size $t_S \in \{1, 3, 5\}$, successor probability $Pr_S \in \{.1, .5, .9\}$.

The results of this experiment are very encouraging. They indicate that the successor probability is the only limitation on the accuracy of the algorithm, even though the number of rules, the size and probability of the precursor patterns determine the amount of structure that is available to be predicted. Further exploration is required to confirm these results.

3.4 Effects of Problem Size

The final issue to be resolved is the influence of the problem size on the performance of the algorithm. In this experiment, we are primarily concerned with the level of performance attained for a given number of predictive rules as the problem size increases. Ideally, we can bound performance as a polynomial function of the input size.

In this experiment, we generate 27 data sets spanning all combinations of: number of streams $n \in \{5, 10, 20\}$, stream length $l \in \{100, 1000, 5000\}$, and number of tokens $\mid T \mid \in \{5, 10, 20\}$. For each data set, we let MSDD generate 1000, 5000, 10000, and 20000 predictive rules, with aggressiveness set to 0.5.

This experiment has several interesting results. First, performance actually *improves* as the number of tokens increases; intuitively, this is due to the probability of each token decreasing as their numbers

increase. Second, the accuracy of the algorithm is basically constant as the stream length increases. This is due to the probability distributions remaining constant as the length increases. The time requirement of the algorithm does increase with stream length. Finally, it appears that MSDD need only generate $n \times 1000$ search nodes to discover the significant dependencies; this is a very strong claim that needs to be supported by further experimentation.

4 Applications

4.1 Feature Based Classification

In the interest of generality, we applied MSDD to a task for which it was not explicitly designed: feature-based classification. We present results for twelve datasets from the UC Irvine collection. Eleven of those datasets were selected from a list of thirteen presented in [8] as being a minimal representative set that covers several important features that distinguish problem domains. The precursor multi-tokens were n-ary feature vectors and the successor "multi-tokens" contained only the class label. These pairs of multi-tokens serve as input to the MSDD algorithm. The results are presented below in Table 1. The accuracy shown in the table is the mean obtained over ten trials where the data was randomly split on each trial into a training set containing 2/3 of the instances and a test set containing the remaining 1/3. The exceptions are NetTalk (training data was generated from a list of the 1000 most common English words, and accuracy was tested on the full 20,008 word corpus), Monks-2 (a single trial with 169 training instances and 432 test instances to facilitate comparison with results contained in [6]), and Mushroom (500 training instances and 7624 test instances). We compared MSDD's performance with other published results for each dataset [2,6,7]. On ten datasets for which we had multiple published results, MSDD performance exceeds half of the reported results on six datasets. In no case did it perform badly, and it often performed extremely well. For a more complete comparison, refer to [4].

Data Set	Mean Accuracy	Search Nodes
Breast Cancer	95.15%	10,000
Diabetes	71.33%	10,000
Heart Disease	79.21%	20,000
Hepatitis	80.77%	10,000
LED-7	70.54%	5,000
LED-24	71.28%	5,000
Lymphography	78.16%	15,000
NetTalk	70.11%	50,000
Monks-2	79.17%	5,000
Mushroom	99.49%	30,000
Thyroid	95.46%	20,000
Waveform-40	73.02%	15,000

Table 1: Performance of MSDD as a feature-based classifier on 11 datasets from the UC Irvine collection

4.2 Pathology Prediction

We applied MSDD to the task of predicting pathologies in a simulated shipping network called TransSim. When several ships attempt to dock at a single port at the same time, most will be queued to await a free dock, resulting in a *bottleneck*. We built a pathology demon that predicts the potential

for bottlenecks before they actually form, and we built an agent that modifies the shipping schedule in an effort to keep predicted pathologies from materializing. Using the demon as an oracle, we gathered data from a single run of the simulator and used MSDD to generate rules to predict bottlenecks. To assess the utility of the previously generated rules, we ran ten simulations in each of two conditions; one with the existing demon and another with the demon replaced by the rules. We used t tests to determine whether or not the means of various costs associated with each simulation were lower in the rule condition as compared to the demon condition. The results are presented below in Table 2. Note that the number of pathologies predicted (PP) by the demon is almost twice the number predicted by the rules and, therefore, the agent made about twice as many schedule modifications (SM). However, of the five cost measures (QL, IC, CT, SU, and SD) only SD was significantly lower in the demon condition when compared to the rule condition. That is, even though the agent is taking a much more active role, performance is not significantly better. Inspection of execution traces shows that the demon is much more likely than the rule set to predict short-lived pathologies. The rules are good at forecasting substantial pathologies, ones that will not go away of their own accord, but miss the more fleeting pathologies. Said differently, MSDD rules are not misled by small, noisy fluctuations in the state of the simulation. This behavior is beneficial when we view disruption to the original schedule as a cost that we want to minimize.

Cost	Demon Mean	Rule Mean	p Value
PP	184.2	94.6	0.0001
CT	2289.3	2377.9	0.0689
IC	1149.8	1202.1	0.1844
QL	637.7	640.5	0.9177
SD	131.1	141.6	0.0019
SU	188.8	202.2	0.3475
SM	21.6	9.2	0.0001

Table 2: Comparison of simulation costs using demon and MSDD rules for pathology prediction

This experiment points to the fact that MSDD is capable of discovering indicators of pathological states in TransSim from high level domain information. MSDD can identify relevant state information to emulate the objective function of an external oracle. One limitation of this approach, as compared with the demon, is that an initial run of the simulator is required to gather data to drive the rule generation process. However, the domain knowledge supplied to the MSDD algorithm was minimal in comparison to the demon.

5 Conclusion

In this paper we described how the problem of finding significant dependencies between the tokens in multiple streams of data can be framed in terms of search. The notion of dependencies between pairs of tokens introduced in [3] was extended to pairs of multi-tokens, where a multi-token describes the contents of several streams rather than just one. We introduced the Multi-stream Dependency Detection (MSDD) algorithm that performs a general-to-specific best-first search over the exponentially sized space of possible dependencies between multi-tokens. The search heuristic employed by MSDD strikes a tunable balance between the expected number of hits and false positives for the dependencies discovered when they are applied as predictive rules to previously unseen data from the same source. We presented results from an empirical evaluation of MSDD's performance over a wide range of artificially generated data. In addition, we applied MSDD to the task of pathology prediction in a simulated

shipping network and to a number of classification problems from the UC Irvine collection. The results that we obtained are very encouraging.

We are currently working on an incremental version of MSDD that can identify dependencies by processing data as it is generated, and that adapts to changing probability distributions. Also, we are working to remove the need for a fixed sized multi-token and a fixed time interval between multi-tokens.

Acknowledgments

This work is supported by ARPA/Rome Laboratory under contract #'s F30602-91-C-0076 and F30602-93-C-0010. The U.S. government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

The heart disease data was gathered by Andras Janosi, M.D. at the Hungarian Institute of Cardiology, Budapest; William Steinbrunn, M.D. at the University Hospital, Zurich, Switzerland; Matthias Pfisterer, M.D. at the University Hospital, Basel, Switzerland; Robert Detrano, M.D., Ph.D., V.A. Medical Center, Long Beach and Cleveland Clinic Foundation. The lymphography data was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia from M. Zwitter and M. Soklic. The breast cancer data was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg [1].

References

[1] Bennett, K. P. and Mangasarian, O. L. "Robust linear programming discrimination of two linearly inseparable sets", Optimization Methods and Software 1, 1992, 23-34 (Gordon and Breach Science Publishers).

[2] Holte, Robert C. Very simple classification rules perform well on most commonly used datasets. In *Machine Learning*, (11), pp. 63-91, 1993.

[3] Howe, Adele E. *Accepting the inevitable: The role of failure recovery in the design of planners.* Ph.D. thesis, Department of Computer Science, University of Massachusetts, Amherst.

[4] Oates, Tim. MSDD as a Tool for Classification. Memo 94-29, Experimental Knowledge Systems Laboratory, Department of Computer Science, University of Massachusetts, Amherst, 1994. Available via the WWW at http://eksl-www.cs.umass.edu/papers/msdd-classification.ps.

[5] Oates, Tim and Cohen, Paul R. Toward a plan steering agent: experiments with schedule maintenance. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pp. 134-139, 1994. Also Department of Computer Science Technical Report 94-02, University of Massachusetts, Amherst. Available via the WWW at ftp://ftp.cs.umass.edu/pub/eksl/tech-reports/94-02.ps.

[6] Thrun, S.B. The MONK's problems: A performance comparison of different learning algorithms. Carnegie Mellon University, CMU-CS-91-197.

[7] Wirth, J. and Catlett, J. Experiments on the costs and benefits of windowing in ID3. In *Proceedings of the Fifth International Conference on Machine Learning*, pp. 87-99, 1988.

[8] Zheng, Zijian. A benchmark for classifier learning. Basser Department of Computer Science, University of Sydney, NSW.