Learning from Examples in a Single Graph

Joseph T. Potts, Diane J. Cook, and Lawrence B. Holder

University of Texas at Arlington
416 Yates
Arlington, TX 76019 USA
{potts, cook, holder}@cse.uta.edu

Abstract

Of all of the existing learning systems, few are capable of accepting graphs as input. Yet graphs are a powerful data representation capable of efficiently conveying relationships in the data to those who use them, both machine and human. But even among the systems capable of reading graph-based data, most require the examples for each class to be in disjoint graphs. We introduce a learner that can use a single, connected graph with the training examples embedded therein. We propose a new metric to determine the value of a classification. Finally we present the results of a learning experiment on sea surface temperature data.

Introduction

Learning systems capable of utilizing graph-based data have typically required disjoint graphs for the training examples. In some cases training examples may be individual disjoint graphs, each of which is an example of one of n classes. There might even be only one graph for each class. In either case, the goal is to learn one or more concepts which allow the user to determine to which class new (previously unseen) graphs belong.

If training examples are actually contained in a single graph, one is very likely going to encounter some problems in preparing the data for input into systems such as those above. If one has to have individual graphs for each example, then one can excise each example along with some amount of surrounding data to create a disconnected graph containing that example. If the examples are close enough to each other in the original graph, then this surrounding data may overlap with the surrounding data of another example or even the example itself. This will result in some data appearing in more than one example graph. There is also some risk of taking the wrong amount of surrounding data, either too large a region around the example causing extra data to be handled, or too small a region resulting in the loss of potentially vital information. And of course it may be impossible to determine the "shape" of the area that should be excised. Since processing graph-based data is very resource intensive, any redundant information can have a drastic effect on performance.

Our goal was to develop a learner that allowed the original graph, containing all the training examples for all classes to be input with a minimum of preprocessing and a minimum of added or redundant information. We developed a program that achieved this goal by starting with some of the core routines of the Subdue graph-based relational learning system (Cook and Holder 1994; 2000). We then modified the search strategy, added new evaluation criteria, and added the capability to apply the learned concepts to novel graphs to measure their effectiveness.

The result is that the only preprocessing required on the graph is to add a vertex for each example identifying its class, and adding edges to connect those vertices to each vertex of the examples. The program then reads the augmented graph and produces a series of substructures which can be used to determine the class of marked examples in a novel graph.

We will first provide some background information on Subdue. Then we will discuss our evaluation metric and other extensions that resulted in Subdue-EC, the embedded classification version of Subdue, designed to learn classifying substructures from training examples contained in a single, connected graph. We then show the system learning to determine whether any point on a global grid will experience a decrease, an increase or no change in its sea surface temperature over one month. This will be followed by conclusions and ideas for future work.

Subdue

Subdue was originally written to discover interesting patterns in structural data. Subdue is not restricted to any domain as long as the data can be represented by a graph. The graph contains vertices and edges, both of which have labels. The edges may be directed or undirected. Vertices may have multiple edges and self edges (a vertex connected to itself). There is no restriction on the number of graphs that may be input at one time, though they will be processed as if there were only one.

In accordance with the MDL principle (Rissanen 1989), the substructure that Subdue deems to be the most "interesting" is the one that results in a small descriptive length when it is used to compress the graph. This concept comes from information theory. If we wish to transmit the graph to an agent, then we must somehow encode it in a binary string. The length of that string in bits is the *Descriptive Length* or DL of the graph. Now if we find a substructure that appears frequently enough in the graph and if it is small enough, then we may be able to encode the graph in a shorter string. First we substitute a new, special vertex for each occurrence of the substructure giving us a new, smaller graph. Then we encode both the new graph and the substructure and send them to the agent. The agent substitutes the substructure back into the graph for every occurrence of the special vertex and will then have a correct and complete copy of the original graph. If the DL of the new graph plus the DL of the substructure is less than the DL of the original graph, then we have reduced the message length required to transmit the graph. We have, in effect, compressed the graph. The substructure that compresses the graph the most gives us the Minimum Description Length or MDL. The amount of compression of a graph G by a subgraph S is calculated as:

$$Compression = \frac{DL(S) + DL(G \mid S)}{DL(G)}$$

where DL(G) is the description length of the input graph, DL(S) is the description length of the subgraph and DL(G|S) is the description length of the input graph compressed by the subgraph. For the sake of convention, the search algorithm tries to maximize the reciprocal of the compression referred to as the *value* of the substructure.

The initial state of the search is the set of single vertex substructures composed of a single vertex subgraph for each unique vertex label along with all of its instances. Each substructure in the current state is extended by one edge. This may mean a new vertex is also added or it may simply be the addition of an edge connecting two vertices already in the substructure. The new substructure is then evaluated and the value used to insert the substructure in a value ordered queue of a limited size specified by the user (beam width). After all substructures in the current state have been extended and evaluated, the queue of extended substructures becomes the current state. The process is repeated with the current state until there are no more substructures to extend or the total number of substructures evaluated reaches a user-specified limit (limit).

If the user has requested multiple iterations, then all instances of the best substructure are replaced by a vertex labeled "Sn" and the entire process is repeated with this new graph as input. It is possible to continue discovering substructures and using them to compress the graph until the graph consists of a single vertex. The discovered substructures form a hierarchical description of the structural data in the original graph. Each substructure can be described in terms of substructures discovered earlier in the process

Figure 1 shows a simple input graph. Subdue discovers three instances of the substructure A->B. Figure 2 shows the graph after being compressed by the substitution of vertex S1 for each instance of substructure A->B.

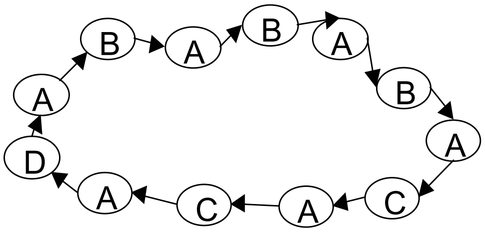

Figure 1. A simple graph.

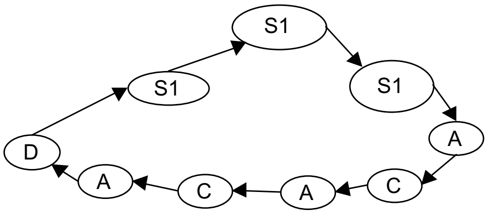

Figure 2. Graph after compression.

Subdue initially operated only in unsupervised discovery mode as described above. However, through a series of extensions such as Subdue-CL (Gonzales, Holder, and Cook 2001) capability to perform concept learning was added. For this application, all graphs that are positive examples are kept distinct from the graphs that are negative examples. The MDL principle is still used although the goal is to compress the positive graphs and NOT compress the negative graphs. The substructure which does this best represents the concept that distinguishes the positive graph s from the negative ones. It was found, though, that for some data a structure might provide a high degree of compression for one or a few of the positive graphs but not all. This situation would mislead the search for a discriminating concept. As a result, a second evaluation method was added based on set covering. With this evaluation, the best structure occurs at least once in a large number of the positive graphs and occurs rarely or not at all in the negative graphs. Other graph-based learners use different algorithms, but like Subdue-CL, most other concept learners were still not able to accommodate a single input graph containing all of the examples.

Subdue-EC

Although Subdue-CL used compression as a metric, if one is performing inductive learning on a single graph containing both positive and negative examples, compression of the input graph provides no guidance in selecting discriminating substructures. Set covering was initially considered for the new task, but did not have the same appeal as a new approach inspired by a discussion on Minimum Descriptive Length in (Mitchell 1997). To allow the MDL principle to guide us in classification, we have to look not at the graph, but at the classification itself. That is, we assume that our receiving agent already has the graph and all of the examples it contains. What we need to encode and send is the *classification* of those examples. The straightforward way to do that is simply to send the class number for each example. Since the examples are in the same order in the agent's copy of the graph as they are in ours, we can just send the class number for example, 1 to n and the agent will be able to classify each example in its copy of the graph. It takes $n\log_2$ (number of classes) bits to send this information.

An alternative to just telling the agent what the class is for each example, is to provide one or more substructures each with an associated class. If an instance of the substructure contains one or more vertices of the example, then the class associated with that substructure is assigned to that example. Of course, it is possible that this may misclassify some examples or leave some of them unclassified. In this case we must inform the agent of the correct classification for those examples. Thus the descriptive length of our alternative message is the sum of the descriptive lengths of each substructure plus a class number for each substructure plus all of the exceptions for each class. Now we can define *classification compression* as the quotient of the DL of the "naïve" classification message and the substructure based classification message :

Classification Compression =

$(n_e*\log_2 \; n_c) \; / \; (\Sigma DL_s+n_s*\log_2 \; n_c+(n_m+n_u)\log_2 n_e)$

where n_e number of examples
n_c number of classes
n_s number of substructures
n_m number of examples misclassified
n_u number of examples unclassified

The reciprocal of this number as before will be referred to as *value*. We will chose a set of substructures which maximize value.

Now that we have a metric, the other issue is identifying the examples and their classes. This is accomplished by the addition of a vertex for each example labeled with the class name for that example. This additional vertex is connected to each vertex of its example by an edge. We do not need to mark the edges of the example since Subdue-EC will include them in the classifying substructure if they improve the value of the substructure. To increase efficiency in the search algorithm, the labels on these class vertices are replaced as the graph is read by the label "_EXAMPLE". This means that the initial state is much, much smaller than in traditional Subdue searches. In fact, there is only one substructure in that initial state. This "focuses" the search immediately on the right places in the graph. In addition, the search algorithm is modified so that Subdue-EC never adds an "_EXAMPLE" node during extension.

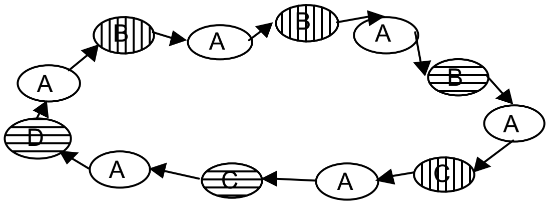

Figure 3. A graph containing vertices from two classes.

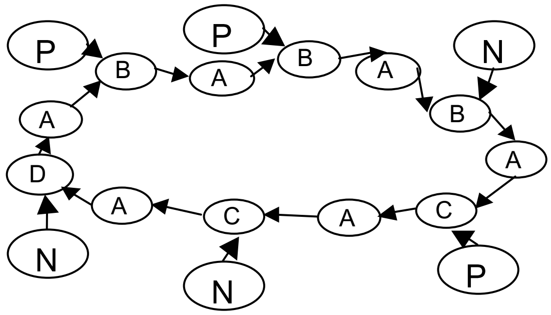

Figure 4. Input graph augmented with class vertices.

When the example in figure 3 is processed by Subdue-EC,the following five classifying substructures are discovered:

D negative
B>A>C positive
C negative
B>A>B positive
B is negative

Two things should be noted. First, the order in which the classifying substructures are discovered is the order in which they must be applied. If we searched a novel graph for the last substructure first, we would classify all B vertices as negative. This would clearly be incorrect. Instead, for each example marked in the novel graph, we must work our way down through the sequence of classifying substructures trying to find an instance of that substructure which contains a vertex of the target example. The second thing to note is that each substructure has one vertex underlined. This designates where the example vertex occurs in the substructure. Thus, if we applied this sequence of substructures to a new graph, Y>B>A>B>X, in order to classify the two vertices labeled B, B>A>B classifies the leftmost B as positive, and B classifies the rightmost B as negative.

Experimental Results

For the initial test of Subdue-EC on real data, we chose a simple classification task on a large set of data. We obtained sea surface temperature (SST) data from NASA (JPL 2000). This data is averaged over a five day period and placed on a one degree global grid. The data contains a fill value for grid points for which the SST is not available such as points on land or due to missing information.

We first determined for each grid point whether the temperature INCREASED, DECREASED or stayed the SAME from January 8, 1990, to February 7, 1990. We then placed the non-fill temperature values into one of 9 equal width bins. We created a graph composed of one vertex for each grid point (labeled “JAN”). Each JAN vertex was connected to its neighbor on the west by a directed edge labeled “W” and its neighbor on the north by one labeled “N”. The westerly edges continued in a circle around the entire globe. The northerly edges ended at latitude 89.5 N. The grid would thus look like a mesh cylinder. Each grid point also was connected to a unique vertex containing its temperature bin or the fill value by a directed edge labeled TEMP and to another unique vertex labeled N or S by an edge labeled HEMI. This value was based on the node’s latitude. We then created a training graph by randomly selecting 90% of the nodes to which we attached each node’s class vertex. This vertex was labeled either INCREASE, DECREASE or SAME depending on whether the temperature at that node was higher, lower or unchanged after one month.

This created a graph of 259,200 vertices and 323,640 edges. We duplicated the original graph, and attached class vertices to the remaining10% of the grid nodes to create a novel graph for testing. Ten sets of training/test graphs were created and tested with the results shown in Table 1.

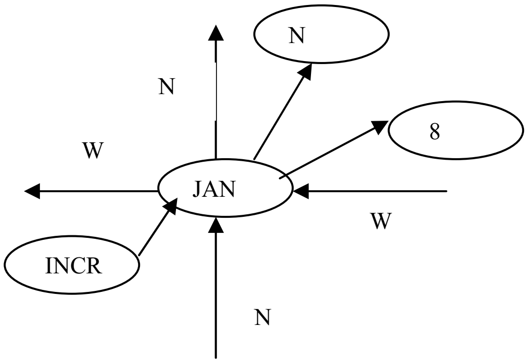

Figure 3. Representative SST grid node.

The first column of % correct applies to the training data. The second column is applying the learned sequence of classifying substructures to the ten percent of the data held back (6,480 grid points). The accuracy was consistent with the accuracy of the training data.

run #	subs	secs.	% correct	
0	106	52822	86.07%	85.31%
1	104	49669	85.81%	85.26%
2	109	76336	85.57%	85.25%
3	100	71679	85.81%	85.32%
4	104	78874	85.66%	85.69%
5	111	80388	85.84%	86.22%
6	108	73174	85.84%	85.00%
7	112	77236	85.81%	85.39%
8	99	75392	85.64%	84.57%
9	108	80497	85.76%	84.10%
Min	99	49669	85.57%	84.10%
Max	112	80497	86.07%	86.22%
Avg.	106.1	71607	85.78%	85.21%

Table 1. Ten-fold cross validation.

We also conducted some tests varying Subdue parameters such as beam size and limit (the number of substructures extended and evaluated). These tests were conducted on 100% of the examples. That is, class vertices were attached to all 64,800 grid points. Surprisingly, the accuracy did not change a lot even when the number of substructures decreased substantially. This is due to the tradeoff in the numerator of classification compression between substructure size and misclassifications. For this data adding one more vertex adds about 16 bits to the size of the substructure. Since there are about 64,000 examples, the penalty for one misclassification is about 16 bits (that is how many bits it takes to tell the sender the example number of the misclassified example. Thus eliminating two misclassifications more than pays for adding one vertex to the substructure.

This tends to drive substructure growth larger and larger until the algorithm is terminated by the limit parameter. These larger substructures have fewer misclassifications, but they also leave more examples unclassified. However, the unclassified examples are then classified on a subsequent iteration. So there are more substructures and larger substructures as limit is increased, but the accuracy does not get that much better (see Table 2).

The next test was to train on 100% of the January 1990 data and then apply that learned sequence of classifying substructures to the 1991 data. That is, create a similar graph for the same dates in the following year January 8, 1991 to February 7, 1991, determine the correct classification for all grid points, and apply the classifying substructures learned from the 1990 graph to this new, previously unseen 1991 graph. As shown in Table 2, using the classifying substructures learned from the January/February 1990 graph, we were 81.98% accurate in predicting the SST direction of change for January/February 1991, one year later. Table 2 shows how we classified each of the 64,800 grid points.

Actual	Called	Called	Called	Incr.
	Same	Decr.	Total	
Same	25473	1076	1079	27628
Decr	1474	9737	2487	13698
Incr	1633	3925	17916	23474
Total	28580	14738	21482	64800
	Called	Right:	53126	81.98%

Table 1. Classifying following year.

The learned substructures are what one might intuitively expect. The first in the sequence addresses the large number of SAME examples. These are primarily land which is still land 30 days later and therefore still has a fill value for the temperature and is in the SAME class. Otherwise the Northern hemisphere tends to gets colder in winter and the Southern hemisphere gets warmer. One of the more interesting classifying substructures is TempBin 0 classifies as SAME. Does this mean it's just too cold to get warm? And Southern hemisphere grid points North of nodes with TempBin 6 are classified as DECREASE. These may be right on the equator and therefore starting to cool off as winter drags on and they get less sun. Finally, it should be noted that none of the tests ever have any unclassified. On this data there always seems to be value to a catchall classification substructure at the end of the sequence. This data has enough correct classifications in the catchall to "pay" for its misclassifications.

As a final test, we created a third graph from the July/August 1990 SST data. We would NOT expect our classifying substructures to perform well on this data since it is for a time period six months later when the Northern hemisphere is heating up and the Southern, cooling. Table 3 shows that indeed, we did very poorly, correctly classifying only 46.69% of the grid points, most of which were in the SAME class.

Although we have not yet done so, we are confident Subdue-EC can also learn subsequences for classifying the July/August temperature shift as well as it learned the January/February one.

Actual	Called	Called	Called	Incr.
	Same	Decr.	Total	
Same	22759	547	2730	26036
Decr	1753	3007	17532	22292
Incr	3316	10611	2545	16472
Total	27826	14165	22807	64800
	Called	Right:	28311	43.69%

Table 2. Classifying following summer.

Conclusions

The learner we have developed accepts a single, connected graph containing all examples for all classes. This learner performed reasonably well on real world data in the Earth Science domain.

The approach is effective for problems with more than just two classes. Our successful classification of the SST data demonstrates its use on a three class problem.

Our representation of the class labels in the graph is concise and quite amenable to randomized sampling without disturbing the original structure of the complete input graph.

The heuristic used in the new learner, classification compression, is somewhat sensitive to graph size and number of examples.

Future Directions

Subdue-EC needs to be tested on data from additional domains. We are particularly interested in how it might perform on social networks in a terrorist detection domain.

The classification compression heuristic may need some tuning to reduce its sensitivity to small numbers of examples. If other types of data show this same tendency to make more and more substructures but gain little in accuracy or change in classification compression, then the calculation may need to be done slightly differently.

Finally, we believe that this approach to learning can be used for graphs representing temporal data. The major issue here is that if all examples are in a single graph, the learner must be prevented from looking forward in time. into the future as it were.

Acknowledgements

This work was supported by NASA Headquarters under the Earth Systems Science Fellowship Grant NGT5 and by NSF grant IIS-0097517.

References

Cook, D. J., and Holder, L. B. 2000. Graph-Based Data Mining. *IEEE Intelligent Systems*, 15(2):32-41.

Cook, D. J., and Holder, L. B., Substructure Discovery Using Minimum Descriptive Length and Background Knowledge. *Journal of Artificial Intelligence Research* 1:231-255, 1994

Gonzalez, J, Holder, L. B., and Cook, D. J. 2001. Graph-Based Concept Learning. In *Proceedings of the Florida Artificial Intelligence Research Symposium,*377-381.

Jonyer, Holder, L. B., and Cook, D. J. 2000. Graph-Based Hierarchical Conceptual Clustering. In *Proceedings of the Florida AI Research Symposium*,91—95.

JPL, Physical Oceanography DACC, WOCE Global Data, V2.0,Satellite Data, Sea Surface Temperature, July 2000

Mitchell, Tom M., *Machine Intelligence,* McGraw-Hill, New York, New York, 1997

Rissanen, J. 1989. Stochastic Complexity in Statistical Inquiry. World Scientific Publishing Company.