An Evolutionary Approach to Constructive Induction for Link Discovery

Tim Weninger, William H. Hsu, Jing Xia, Waleed Aljandal
234 Nichols Hall
Kansas State University
Manhattan, KS 66506
{weninger, bhsu, xiajing, waleed}@ksu.edu

ABSTRACT
This paper presents a genetic programming-based symbolic regression approach to the construction of relational features in link analysis applications. Specifically, we consider the problems of predicting, classifying and annotating friends relations in friends networks, based upon features constructed from network structure and user profile data. We first document a data model for the blog service LiveJournal, and define a set of machine learning problems such as predicting existing links and estimating inter-pair distance. Next, we explain how the problem of classifying a user pair in a social network, as directly connected or not, poses the problem of selecting and constructing relevant features. We use genetic programming to construct features, represented by multiple symbol trees with base features as their leaves. In this manner, the genetic program selects and constructs features that may not have been originally considered, but possess better predictive properties than the base features. Finally, we present classification results and compare these results with those of the control and similar approaches.

General Terms
Design, Experimentation

Categories and Subject Descriptors
I.2.6 [Computing Methodologies]: Learning; I.1.1 [Computing Methodologies]: Expressions and Their Representation—symbolic regression

Keywords
machine learning, classification, genetic programming

1. INTRODUCTION
Traditional data mining tasks such as association rule mining or market basket analysis attempt to find patterns in a dataset. According to Getoor, “This is consistent with the classical statistical inference problem of trying to identify a model given a random sample from a common underlying distribution” [7]. However, it is important to also mine datasets that are relational, semi-structured or otherwise consist of links between various entities. These links can be explicit, such as an anchor tag in a web page, or implied such as a join operation in a relational database. As shown by the PageRank algorithm used by the popular search engine Google, link existence can be exploited to improve the predictive accuracy of learned models [4]. Intuitively, attributes of linked objects are often more closely related than those of unlinked objects, and links are more likely to exist between objects that share common attributes [7].

Feature construction in the multi-relational setting is also possible. Traditionally, the attributes of an object provide the basic description of the object. However, by leveraging the information contained in the relationships of objects, more information about the object can be gleaned providing the learning algorithm an appropriate context for a better induction model.

This work focuses mainly on the construction of features in order to improve link discovery, i.e., predicting the existence of links between objects. In order to discover links not previously known to exist, a genetic programming approach is used to construct features that appropriately leverage the knowledge contained in the presence or absence of links. A brief overview of related material is given in the following sections, however, we refer readers unfamiliar with one or more of the following topics: constructive induction, social networks, and link mining, to [27, 20, 8, 26, 7].

1.1 Constructive Induction
We consider a genetic programming approach to theory formation that consists of relational feature construction. Synthetic features represented by GP expression trees in our framework are produced using classification fitness in a link existence prediction domain. This approach constitutes a kind of descriptive generalization [18], the formulated theory characterizes a collection of entities. In the case of link mining, the primitive features of the theory are functions of candidate pairs, e.g., the number of mutual friends and interests shared by two users in a social network. The role of GP is to synthesize functions of these relational primitive features that are validated using inductive learning algorithms, producing a fitness value for each synthetic feature.

Most types of machine learning problems (including link mining) can be viewed as search problems [19] involving a large hypothesis space, that is, the space consisting of all possible theories under consideration. In this paradigm the goal of the search is to find the best theory with respect to the available training examples.

Constructive induction is the process of improving the attribute vector of a learning algorithm in order to make the
problem more easily learned for a particular learning algorithm [16]. Constructive induction is often used when the explicitly selected features do not effectively represent the stated problem. For instance, given a machine learning algorithm \( L \), constructive induction would be appropriate if the training set contains all of the relevant information for the induction of the target function but this information cannot be extracted by \( L \) [23]. Moreover, constructive induction is used to deal with learning algorithms, such as feed forward neural networks trained with back propagation and classification and regression tree (CART) algorithms [1, 13]. Constructive induction generally consists of two parts: (1) for the construction of new features, and (2) for generating theories. After being constructed, the new, synthetic features are treated the same way as the initial, primitive features that were used to construct the synthetic features.

The construction of new features is essentially the application of a set of constructive operators to the primitive features; this results in the combination, and therefore the construction, of one or more new features [16]. For common learning problems the number of possible constructive operators, such as mathematical operators, set operators, logical operators, etc. and the number of possible constructive operands for each operator is very large, so it is not feasible to search through all possible combinations. The GP approach presented in this work does not need to search through all possible combinations.

Common operators include conjunction (\( \land \)), disjunction (\( \lor \)) and negation (\( \lnot \)). Take, for example, the decision tree depicted in Fig. 1. This decision tree shows a fundamental limitation of the selective induction procedure involved in the creation of decision trees called the replication problem [21]. Because a decision tree divides each feature space into mutually exclusive regions it is possible to derive duplicate subtrees, such as those shown in the grey regions. If a subtree is replicated many times then more training examples are needed in order to grow the size of the tree. This often leads to either an overly complex final decision tree, inaccurate pruning of the decision tree, or a premature termination of the learning algorithm. In other words, the replication of subtrees degrades the prediction accuracy of a decision tree learning algorithm.

Because the structure of the theory language is fixed, theory learning in the new feature space is expected to be easier than in the original feature space. By constructing new attributes (\( A \lor B \)) and (\( C \lor D \)), a decision tree can be built for the constructed features (\( (A \lor B) \land (C \lor D) \)) whose representation, shown in Fig. 2, is dramatically simplified making more complex concepts easier to learn. Therefore, from this point of view, the newly constructed features are more representationally powerful than the primitive features.

![Figure 2: Decision tree from Fig. 1 with two constructed features replacing duplicate subtrees](image)

Constructive induction can be realized in several ways. One commonly used algorithm is greedy search. Greedy search is easily applied to constructive induction tasks on decision trees; the algorithm generates new feature at each decision node based on original features or previously constructed features. To construct a new feature, the algorithm searches greedily through the instance space using a pre-specified set of constructive operators. Starting from an empty set of decision nodes, the algorithm systematically adds and/or deletes decision nodes until the instance space has been searched or the algorithm is forced to terminate. To evaluate candidate decision trees standard metrics, such as class entropy and model complexity, can be considered in lieu of test data, or classification accuracy can be used to evaluate the performance of the candidate decision tree in the presence of test data. Constructive induction can also be achieved using evolutionary computation approaches such as genetic algorithms or genetic programming as shown in later sections.

### 1.2 Link Mining in Social Networks

Analysis of friends networks provides a basis for understanding the web of influence [11] in social media. In particular, the problems of determining the existence of links and of classifying and annotating known links are first steps toward identifying potential relationships. This inferred information can in turn be used to introduce new, potential friends to one another, make basic recommendations such as community recruits or moderator candidates, or identify whole cliques and communities.

In 2006, Hsu et al. introduced a link prediction problem for LiveJournal: given a graph in which the existence of a candidate link is hidden (elided if it exists), classify it as present or absent given all other attributes of the graph and of the endpoints. Hsu's initial approach to link identification consisted of dividing friends network features into graph features and interest-based features. The link mining aspect of our research is an extension of Hsu's work. [9, 10, 27]

Furthermore, our work explores the application of the evolutionary computation approach to constructive induction for the purposes on link mining. The related research section also includes a brief survey of similar, iterative machine learning techniques which serve as a motivation for this work. Next we discuss the methodology of our approach by
first describing how a large social network is crawled and locally stored as a graph. Then we describe the setup and execution of the experiments to be performed and the means by which results are gathered and presented. Next, the results section presents the outcome of the experiments described in the methodology section. The results section also discusses the statistical significance of our results as part of a comparative analysis between our results, the control, and other approaches. This culminates in the final section, which contains the conclusions that can be drawn from these experiments and suggestions for future work.

2. RELATED WORK

2.1 Meta-learning

The broader topic of meta-learning studies how learning algorithms and other systems can increase in efficiency with experience. The goal of meta-learning is to understand how learning occurs and then use that information to improve the learner [25]. Traditional learning algorithms differ from the meta-learning approach in that a meta-learner discovers the learning bias dynamically by searching for the best learning strategy as the algorithm progresses [24].

The proper meta-learning system would begin with a certain base learner, i.e., a traditional learning algorithm with a fixed bias. Once the system is started and meta-information about the state of the learner begins to accumulate, the meta-learner is able to change its bias by switching to another base learner [29]. A more “granular” [25] approach consists of selecting a learning algorithm for each individual training example. That is, if meta-data is gathered that can be applied to discriminate different classes of examples then the best base-learner can be chosen and applied to each particular example [17]. The algorithm selection is done based on its performance on each class of examples.

Meta-learning is similar to constructive induction using genetic programming because they both leverage information about the current state/progress of the system in order to make adjustments to the learning process. Meta-learning does this in the manner described above, while genetic programming does this by constructing new features based on previous features.

Despite the current research efforts and promising results thereof, meta-learning is not a candidate for this work because it will not necessarily result in a generalized theory. This is because the different base learners will all individually construct their own theories based on their inherent biases. Specifically we compare our GP results to the Bagging [2], Boosting [6], and Random Forests [9] meta-learning algorithms.

2.2 Krawiec’s Wrapper Approach

In Krawiec’s work [12], a genetic program (GP) is used to change the representation of the input data (i.e., training and test examples) for machine learning algorithms. Specifically, the author first proposes the general framework for GP-based feature construction. The author also proposes an extended approach that preserves useful features from the constructed features may not adequately represent the problem space then the classifier’s performance will suffer as a result. Previous studies [9, 10, 5] all use a similar link mining problem and use a similar feature set. This feature set includes features that are (1) user-dependent, (2) link-dependent, and (3) pair-dependent.

Krawiec uses the ECJ software package [14], and the function set included +, −, ×, ÷ (protected), log (protected), <, >, =, and an approximate equality operator. The terminal set contained the primitive features. Individuals’ fitnesses were evaluated by running 5 independent 2-fold cross validation runs on the training set. The WEKA 3.5.7 [28] implementation of the decision tree inducer C4.5 [22] is the learner used from training and testing.

3. METHODOLOGY

Our approach for link mining constructive induction using genetic programming is described in distinct parts. First, we briefly describe our data collection technique. Second, we define base features from the gathered data. Third, we generate three sets of candidate pairs. Fourth, we describe the genetic program which takes as input the aforementioned primitive features, constructs synthetic features and computes instances based on these new features. Finally, we discuss the experimental design for this research with descriptions of the standard evaluation metrics.

In order to focus on studying the impact of GP-based feature synthesis in this link mining domain, GP is used to synthesize features for classical inductive learning algorithms only, rather than for the entire link existence task. Such a holistic view, while perhaps feasible, would not necessarily make clear what the impact constructed features have. We therefore consider it beyond the scope of those work and a topic of future research.

3.1 Crawling a Social Network

Before learning can begin an appropriately sized and realistically distributed example set needs to be obtained. For this work, we chose to crawl the social network LiveJournal for user information and relationships. User information is kept in two forms on LiveJournal: (1) in HTML profile pages and (2) in friend of a friend (FOAF) files. Profile pages are available from $http://username.livejournal.com/profile?mode=full$. FOAF pages are available as XML documents from $http://username.livejournal.com/data/foaf$. A crawl of LiveJournal was executed once at 6:41pm CST on August 13, 2008, and was let run until 9:10pm CST on a Beowulf cluster. As a result of this crawl, 39,024 users were crawled and 770,595 users were queued but not crawled giving an average of 4.09 users per second (39,024 users in 159 minutes). Also, in that time, 372,931 unique interests were discovered and users declared a total of 2,151,090 interests, giving a mean of 55.12 interests per user and a mean of 5.77 unique interests per user. From the users crawled, 2,992,607 directed links were found, giving an average of 76.69 links per user.

3.2 Primitive Feature Selection

Primitive feature selection is an essential part of the overall approach. Without a broad set of primitive features, the constructed features may not adequately represent the search space. If the constructed features under-represent the problem space then the classifier’s performance will suffer as a result. Previous studies [9, 10, 5] all use a similar link mining problem and use a similar feature set. This feature set includes features that are (1) user-dependent, (2) pair-dependent, and (3) graph-dependent.

1. User-Dependent
   (a) Number of interests listed by u: $|I_u|
   (b) Number of interests listed by v: $|I_v|
2. Pair-Dependent
(a) Interests of u and v: \( I_u, I_v, (I_u \cap I_v), (I_u \cup I_v) \)
(b) Interest popularity of u and v: \( P_u, P_v, (P_u \cap P_v), (P_u \cup P_v) \)
(c) Friends of u and v: \( F_u, F_v, (F_u \cap F_v), (F_u \cup F_v) \)
(d) Friend age of u and v: \( A_u, A_v, (A_u \cap A_v), (A_u \cup A_v) \)

3. Graph-Dependent
(a) Indegree of u, i.e. \( d_i(u) \): popularity of the user
(b) Indegree of v, i.e. \( d_i(v) \): popularity of the candidate
(c) Outdegree of u, i.e. \( d_o(u) \): number of other friends besides the candidate; saturation of friends list
(d) Outdegree of v, i.e. \( d_o(v) \): number of existing friends of the candidate besides the user; correlates loosely with likelihood of a reciprocal link
(e) Forward deleted distance: minimum alternative distance from u to v in \( G_u \) without arc \( \langle u, v \rangle \)
(f) Backward deleted distance: minimum alternative distance from v to u in \( G_v \) without arc \( \langle v, u \rangle \)

This list of features is by no means exhaustive. Rather it is a sample of features from other works that we find useful for this particular problem. More features have been crawled and gathered, but in the interest of brevity and maintaining a reasonable scope they are omitted. Nevertheless, from this data 6,000 total candidate pairs are generated (2000 for training, 2000 for testing, and 2000 for validation).

3.3 Genetic Feature Construction
A Genetic Program is used to change the representation of the inputs for machine learners. In this work we use standard symbolic regression with multiple symbolic regression trees to construct features.

From our base-feature set there are 2 possible user-dependent base features. There are 5 set operations \( (U, V, (U \cap V), (U \cup V), (U \setminus V)) \) that can be applied to 4 types of pair-dependent bases features resulting in 4 potential sets; from these sets 5 statistical operations \( (\text{sum, mean, min, max, count}) \) can be applied resulting in 100 potential features. In addition, there are 6 graph dependent features. In total there are 108 potential features that can be constructed. These operations are performed by the GP and are always prior to the addition of mathematical symbols.

To optimize individual symbolic trees per feature, 10 regression trees are genetically computed on the 108 base features. Afterwards, the newly constructed features were used to train and cross validate a classifier. In some cases the constructed features might not make any logical sense and an inappropriate conclusion may be drawn from the intricate details of the specific sampling. Therefore, these experiments are repeated 100 times, each with a random seed.

3.4 Experiment Design
This section aims to test the ability of a genetic program to construct features in order to improve the accuracy of current link mining algorithms. In order to do so, different sets of experiments were performed on identical training data, namely the social networks data described early in this section.

Three independent example sets are considered by the GP: (1) Training data with 2000 randomly-selected instances including 50% positive examples and 50% negative examples [15]. (2) Test data with 2000 randomly-selected instances of the original distribution (about 1.5% positive examples). (3) Because the test data is used to reinforce the genetic program, it can be argued that the metrics from the evaluation of the test data is an inconclusive indicator of the algorithm’s performance. Therefore, holdout validation data with 2000 examples of the original distribution is used to validate the performance of the various learning algorithms. Metrics gleaned from validation data are presented in the results section.

First, as a baseline, the primitive features were used to train and cross-validate four types of learning algorithms: J48, which is a tree learning algorithm based on ID3, Naïve-Bayes, which is a probabilistic learning algorithm which uses Bayes’ theories of probability, and logistic regression, which is a function finding method similar to symbolic regression, and OneR, which simply picks the best possible single rule.

Second, the genetic program was invoked to learn a classifier based on the application of multiple-trees symbolic regression. Again, the fitness function used by genetic program was based on the AUC metric of the classifier, and the multiple-tree symbolic regression experiment was run 100 times for each classifier. In addition to the set and statistical operators described in Section 3.2, 4 binary mathematical operators \( (+, -, \times, \div) \) are used. In a separate test, 5 additional operators \( (\text{sin}, \text{cos}, \text{exp}, \text{ln}, \text{IFLTE}) \) are considered. In some instances these operators may return incomputable numbers, such as \( \infty \); in these instances the result is set to 1.

Finally, the genetic program performance results were compared to those of the WEKA implementations of meta-learning algorithms such as bagging, boosting, etc. described in Section 2. The meta-learning algorithms are also trained (example set #1) and tested (example set #2) in the same manner as the GP, except that the validation task (example set #3) is handled internally by the WEKA implementation.

The fitness function for all GP experiments is 1-AUC. The inverse on the AUC is used because the ideal AUC is 1, whereas the ideal fitness in the ECJ system is 0.

4. RESULTS
This section presents three sets of results for each of the experiments described in Section 3. The standard metrics used will often be accompanied by a more descriptive analysis of the results. The final subsection will review the results in a comparative paradigm setting up the final section containing conclusions that can be drawn from these experiments.

4.1 Traditional Learning with Primitive Features
This section looks at the performance of traditional machine learning techniques using only the primitive features.

In each experiment the classifier was trained on the 2000 instances and a holdout data set with a 1.5% positive example rate (roughly matching that of the original distribution)
was applied to test the performance of the classifier. For the purposes of this research, the learning algorithm which produces a classifier that maximizes area under the ROC (AUC) was declared the winner.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>AUC</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>OneR</td>
<td>86.7</td>
<td>93.95</td>
<td>16.7/99.7</td>
<td>79.3/94.2</td>
</tr>
<tr>
<td>J48</td>
<td>88.7</td>
<td>90.05</td>
<td>12.1/99.9</td>
<td>93.1/90.0</td>
</tr>
<tr>
<td>IB1</td>
<td>77.4</td>
<td>82.15</td>
<td>5.7/99.5</td>
<td>72.4/82.3</td>
</tr>
<tr>
<td>Logistic</td>
<td>98.0</td>
<td>94.05</td>
<td>19.2/99.9</td>
<td>96.6/94.0</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>91.4</td>
<td>93.95</td>
<td>12.2/99.6</td>
<td>75.9/91.9</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>97.7</td>
<td>89.40</td>
<td>11.4/99.9</td>
<td>93.1/89.3</td>
</tr>
<tr>
<td>Bagging</td>
<td>95.4</td>
<td>92.05</td>
<td>15.7/99.9</td>
<td>93.1/92.6</td>
</tr>
<tr>
<td>RandForest</td>
<td>96.4</td>
<td>90.85</td>
<td>13.3/99.9</td>
<td>96.6/90.8</td>
</tr>
</tbody>
</table>

Table 1 presents results from the various learning algorithms including the AUC, accuracy, precision and recall statistics. Note that the precision and recall statistics give two percentages: the first for the positively labeled instances, and the second for the negatively labeled instances. Classifiers based on meta learning techniques (bottom three in Table 1) generally performed better than the traditional classifiers. However, the Logistic classifier resulted in the largest AUC score and is considered the winner.

4.2 Feature Construction using GP with Multiple Symbol Trees

The results of the major contributions of this work are presented in this section. In each experiment the total number of constructed features was exactly 10, the genetic program’s population was exactly 100 individuals, the number of generations was set to 50, and the probability of mutation, crossover, etc. were set to the ECJ defaults. As explained in earlier sections, the learning algorithm was trained with 2000 examples of a 50/50 distribution and tested on an independent set of 2000 examples of the original distribution; because of this “wrapper” approach, the test examples influenced the learning algorithm, therefore another independent validation set of 2000 examples of the original distribution was used to score the final performance of each algorithm. The scores reported in this section are from the holdout validation data. Because these tests were repeated 100 times, the fitness scores were averaged with other scores of the same generation.

Figure 3 shows the mean fitness progression for all populations in each generation. Almost immediately the populations began to converge. After the final generation, the population evolved with the Logistic classifier resulted in the highest overall fitness with 98.00% AUC, followed by NaiveBayes (97.62%), J48 (97.48%), and IB1 (90.94%) while the OneR classifier performed the worst with (87.40%). The IB1 classifier resulted in the greatest gain in fitness climbing from 69.55% in the first generation to 90.94% in the final generation.

A two-tailed paired T-test was performed in order to gauge the statistical significance of these results in comparison with results of the non-GP classifiers (control). The test list was comprised of the inverse fitness (AUC) scores of the best individual for each of the 100 repetitions. The control list was a list of the base scores (from Table 1), wherein the base scores was enumerated 100 times. Table 2 shows the results of the T-test for each classifier. The results of all classifiers, except Logistic, were shown to have performed significantly better than the control.

<table>
<thead>
<tr>
<th>Learning algorithm</th>
<th>Mean</th>
<th>Variance</th>
<th>Paired t-test</th>
</tr>
</thead>
<tbody>
<tr>
<td>OneR</td>
<td>87.47</td>
<td>0.03</td>
<td>$2.03 \times 10^{-5}$</td>
</tr>
<tr>
<td>J48</td>
<td>97.43</td>
<td>0.01</td>
<td>$3.60 \times 10^{-92}$</td>
</tr>
<tr>
<td>IB1</td>
<td>90.36</td>
<td>0.03</td>
<td>$7.23 \times 10^{-46}$</td>
</tr>
<tr>
<td>Logistic</td>
<td>98.0</td>
<td>5.55 \times 10^{-6}</td>
<td>0.9844</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>97.62</td>
<td>5.60 \times 10^{-5}</td>
<td>$2.07 \times 10^{-93}$</td>
</tr>
</tbody>
</table>

A final comparison of the results is shown in Figure 4, which graphically displays results from Tables 1-2.

5. CONCLUSIONS

In this paper we consider the problem of using a GP to enhance a learning algorithm’s ability to train a classifier. We have demonstrated the ability to crawl the popular social network LiveJournal to ascertain a sufficiently large friends network. We have also identified the need to construct and select features based on information from the friends network. To that end, we employed a genetic program capable of evolving new features from primitive features. Finally, we show that classifiers that have been constructed by the GP perform significantly better than classifiers constructed in lieu of the GP. Moreover, GP-evolved classifiers generally performed better than meta-learning techniques although tests for significance were not attempted.

Finally, one limitation of this work is the expressiveness of the constructed features. One avenue for future research is to allow the GP more degrees of freedom in its construction of features. This could be done by adjusting the evolution parameters in system, or by using alternative fitness measures. This is likely a very difficult task due to the type...
differences inherent in set, statistical and mathematical operations.

6. ACKNOWLEDGEMENTS
This research was partially funded by a grant from the Defense Intelligence Agency.

7. REFERENCES