Using Program Synthesis for Social Recommendations

Alvin Cheung Armando Solar-Lezama Samuel Madden
MIT CSAIL
{akcheung, asolar, madden}@csail.mit.edu

ABSTRACT

This paper presents a new approach to select events of interest to users in a social media setting where events are generated from mobile devices. We argue that the problem is best solved by inductive learning, where the goal is to first generalize from the users' expressed "likes" and "dislikes" of specific events, then to produce a program that can be used to collect only data of interest.

The key contribution of this paper is a new algorithm that combines machine learning techniques with program synthesis technology to learn users' preferences. We show that when compared with the more standard approaches, our new algorithm provides up to order-of-magnitude reductions in model training time, and significantly higher prediction accuracies for our target application. [1]

Categories and Subject Descriptors
H.2.8 [**Database Applications**]: Data Mining; I.2.2 [**Automatic Programming**]: Program synthesis
Keywords
recommender systems, social networking applications, program synthesis

1. INTRODUCTION

At a high level, the problem of selecting interesting events in a social media setting appears similar to recommendation problems in other environments, such as offering movie recommendations on Netflix. In each of these, a user's previously expressed preferences are used to infer new items of interest; every time the user interacts with the site, the system builds a more accurate picture of what she likes and dislikes and uses it to improve recommendations. Social media, however, poses some unique challenges which demand a different approach from standard *collaborative filtering*, where other users' preferences are used to infer what the user will like [10].

To illustrate some of the new challenges that recommendation systems face in this domain, we focus on an application called LifeJoin [5]. We designed this application to model the future of social networking, where a person's profile is continuously updated by an automatically generated event stream from the user's mobile devices, including her location and activities. The system attempts to discover interesting co-occurrences in friends' event streams, such as a meeting of two of the user's friends in a nearby pub. In order to deal with the data deluge, the system gives the user the ability to "like" and "dislike" both individual and combinations of events. LifeJoin uses the expressed likes and dislikes to infer what kinds of events are of interest to the user, which can then be used to auto-populate the user's newsfeed. Collecting events through a mobile device can consume a lot of energy [6], so LifeJoin uses the inferred user's interest to drive subsequent event acquisition. For instance, if LifeJoin infers that Mary's friends are only interested in the places she goes for a jog, then the system will save power on Mary's device by turning off data collection when she is not jogging. Our initial experiments have shown that implementing the data collection scheme in the scenario above can extend the phone battery life by up to 40% [5]. Thus, the more accurately we can detect the users' real interests, the more energy we can save on data collection.

Inferring interests in LifeJoin poses 4 challenges:

1. **Decomposable Models:** For applications such as LifeJoin, models must be decomposable into simple classifiers that can be pushed down to the individual devices to drive event acquisition. One simple way to ensure a model is decomposable is to limit it to only contain boolean combinations of simple predicates over the input features, which can be decomposed in a straightforward way to indicate the required data from phones. Such models are also useful because they allow users to give explicit feedback about whether the system actually understands their true interests, and to manually tune the models to better suit their preferences as discussed in [10]. By contrast, many existing preference learning algorithms produce black box classifiers that are difficult to decompose and understand.

2. **Active Learning:** Given the large number of incoming events and the large number of ways in which they could be combined, it is unreasonable to ask the user to rate any meaningful fraction of them. Thus, the learner needs to intelligently choose a subset of incoming events that can most improve the current model.

3. **Skewed Data:** Since each user's definition of "interesting" is different, it is difficult to make generalizations about the statistical properties such as the anticipated degree of skew in users preferences. In fact, this is currently an active research topic [3].

4. **Personalized Events:** Unlike typical recommendation systems such as those for books or movies, where all users rate a common set of items, the events in LifeJoin tend to be highly personalized. For instance, a user might like an event because it involves her best friend Peter, but the same event would be totally meaningless if it is shown to another user who does not know Peter. Thus, we believe it is easier to learn a model for each user individually rather than trying to discover the relationships between users and design a model that is applicable to all.

These requirements preclude the use of collaborative filtering (CF) techniques which have been successful in other recommendation systems. In particular, these techniques tend to generate

[1] The full paper is published as tech report MIT-CSAIL-TR-2012-025.

CIKM'12, October 29 – November 2, 2012, Maui, HI, USA.

models that are not easy to factor (req 1). For instance, a neighborhood model-based approach might attribute a new rating based on a set of previously rated events that are deemed similar, but it is unclear how the system can easily generalize from the set of similar events to determine what new events to collect. Furthermore, CF techniques require a similarity measure between users or events. It is unclear how that can be done in a setting where events are highly personalized to a small set of users (req 4); this is an active research topic [11, 12], and the proposed solutions require explicitly modeling all social relationships between users.

We avoid the above issues by viewing the problem as inductive learning with an active learning component: given a set of labeled examples, the goal is to learn a set of rules that represents an individual user's preferences, and to choose new events for the user to rate. Unfortunately, standard inductive learning algorithms such as those based on entropy measures (e.g., decision trees) are known have issues with skewed data (req 3) [4], and it is not clear how active learning can be applied,

Recently, the programming languages community has been exploring inductive learning problems in the context of software synthesis in programming-by-example systems [8], where the goal is to infer a program from a set of sample behaviors. Unfortunately, the learning problem in Lifejoin is different enough that none of the previous techniques can be applied out of the box. In particular, the active learning problem has not been sufficiently addressed. Nevertheless, these techniques provide a new set of tools that can be leveraged to attack the problem.

In this paper, we present a new algorithm to infer users' interests; the algorithm combines new techniques in program synthesis with traditional machine learning approaches to satisfy the requirements listed above. Specifically, we make the following contributions:

1. We show that both the classical machine learning approach and an approach based purely on program synthesis do not adequately address the problem.

2. We describe a hybrid approach that employs program synthesis to generate a number of classifying functions, and subsequently asks an SVM to assign weights to the features in each generated functions. When compared to pure machine learning or synthesis approaches, this hybrid technique takes up to an order of magnitude less time to encode the training data into a feature space representation, and improves upon traditional learning algorithms by 30% in overall classification accuracy.

3. We show that we can use a program synthesizer to produce more decomposable and human-understandable models than those generated by traditional machine learning techniques, and provide empirical evidence that the generated models are comparable to the original intentions that the user has in her mind.

We have implemented the learning technique in the context of the LifeJoin application. However, we believe that our approach is applicable to other social networking applications as well, where large amounts of data are collected from users. In the next section we discuss the learning task in LifeJoin, and illustrate our approach with an example.

2. OVERVIEW

In this section, we illustrate the recommendation problem with a concrete example and present our solution. Consider the LifeJoin event stream, which contains large numbers of events about the activities of a user's friends and family. Out of this event stream, suppose the user is interested in events where her friend Joe is away from home either late at night or early in the morning:

$(user = Joe) \wedge (location \neq Home) \wedge (time < 9am \vee time > 9pm)$

The goal of the system is to infer this interest function from the users' rating of events. We want the interest function in the form above because that helps ensure the decomposability described earlier. When expressed in this form, the function can be decomposed into predicates and pushed down to individual users' phones to optimize the data acquisition process as described. Such expressions are also comprehensible by users, and can be manually adjusted to tune the results the user sees. We are not aware of any statistically-based methods that can directly generate such models.

Without knowing the expected distribution of the events, the naïve approach to generate an interest function is to exhaustively explore the space of all possible predicates until a set is found that matches all the previously labeled events. The obvious problem with this approach is that the space of possible predicates is enormous—on the order of 10^{40} in some of our experiments. A deeper problem with the naïve approach is that the predicates found cannot be expected to have much generalization power—that is, they are unlikely to correctly classify new events. Also, they will not be of much use in determining the next event to present to the user for labeling. In the following, we describe two approaches based on combinatorial synthesis that address these problems, which allows us to find matching interest function in a few seconds.

2.1 Synthesis-Based Approaches

The synthesis problem can be seen as a generalization of traditional curve fitting, where a space of possible curves—say, all polynomials of degree less than k—is explored in search of one that satisfies a given set of requirements. Modern synthesis systems go several steps beyond simple curve-fitting by providing rich languages for describing requirements and spaces of candidate programs. The search for a correct solution in this space is performed symbolically; i.e., the space of candidate programs is described through a set of equations which are solved through a combination of inductive and deductive methods by a specialized solver. LifeJoin uses a synthesis system called SKETCH [19], where the space of possible predicates is specified in a grammar, and the labeled events are provided as requirements that the generated interest function needs to fit. Fig. 1 shows a sample of the labeled data and a few interest functions that were found this way.

Unfortunately, the interest functions generated by the synthesizer cannot be used for active learning purposes. To address this, we rely on the idea of boosting [17], where we first generate a number of interest functions using the synthesizer that are consistent with the data. After the synthesis algorithm has found K interest functions f_i, each of these functions can be treated as a weak base learner, and the group forms an ensemble.

The standard way to form an ensemble is to learn a linear function $F(e) = \Sigma w_i \cdot f_i(e)$, where an event is classified as interesting if $F(e) > 0$. The ensemble allows us to follow a standard approach for active learning, namely, to select those events that are closest to the boundary where $F(e) = 0$ [21]. Normally, the weights w_i are computed using the training data, but in our case, all f_i were selected to agree on all the training events, thus each f_i will have equal weight. That reduces the ensemble to a majority vote, and active learning reduces to selecting the event that causes the maximum level of disagreement among all the candidate interest functions. We refer to this pure synthesis based algorithm as the "ensemble" approach. As we will see in Sec. 3, such an approach already outperforms many standard learning techniques, but we can do better.

When defining the space of candidate interest functions, we require them to be in disjunctive normal form. This means that every function f_i can be seen as a disjunction of individual predicates $p_{i,j}$. We exploit this structure when building the ensemble; instead of an ensemble $F(e) = \Sigma w_i \cdot f_i(e)$, we build an ensemble of the form

user	location	time	preference
Joe	Office	10am	✗
Bill	Home	3pm	✗
Joe	Office	11pm	✓
Joe	Bar	6am	✓

Each line below denotes a potential classifier

$(User = Joe) \wedge (location = Office \vee location = Bar) \wedge (time < 7am \vee time > 10pm)$

$(User \neq Bill) \wedge (time > 10pm \vee location = Bar)$

$(User = Joe) \wedge (time < 9am \vee time > 11am)$

Figure 1: Learning example with labeled data (left) and candidate classifiers that are consistent with the labeled data (right)

$F'(e) = \Sigma w_{i,j} \cdot p_{i,j}(e)$. Finding weights for each predicate is no longer trivial. We use an SVM to find weights for the function, which has the additional benefit that the weights will be set in such a way that the resulting classifier will be maximum-margin one.

We call this combination of program synthesis and machine learning techniques the "hybrid" approach, and our experiments show that defining the ensemble in this way significantly improves active learning. One remaining issue is that the SVM may find fractional values for the weights, so the function $F'(e)$ will no longer be a well-formed boolean predicate. Once again, we use synthesis technology to find a well-formed predicate $P(e)$ that is closest to the linear function $F'(e)$ using the support vectors that are found by the SVM. Such predicates have the decomposability property we desire as our experiments show next.

3. EXPERIMENTS

For experiments we use two event streams collected by LifeJoin; one about the location of users, and the other about users' activities. The **unary** features describe those that involve only one comparison operation, otherwise they are **conjunctive** ones. **Full** refers to both unary and conjunctive features together.

We implemented eight different learners. **L1** and **MI** are both classical machine learners based on an SVM classifier. L1 uses the LASSO algorithm for feature selection and a linear SVM for classification. MI performs feature selection by computing the mutual information between each of the features and the output label, and picks the features with the highest scores for subsequent classification using a linear SVM. Both methods enumerate the full feature set before feature selection. The **ensemble** and **hybrid** learners represent the program synthesis approach described in Sec. 2.1.

The **tree** learner learns a decision tree using the C4.5 [16] algorithm in the Weka toolkit [1], and creates features by extracting the path(s) from the root node that leads to the leaves that classify the event as interesting, as in [20]. To avoid degenerate trees, we lowered the support for splitting and did not prune the generated tree. We have also experimented with random trees and the results are similar. The resulting features are then used to train an SVM for classification.

Full is an SVM learner that uses no feature selection on the full set of conjunctive features, **unary** is an SVM learner that has no conjunctive features, **poly** uses the same set of non-conjunctive features as unary except that the features are passed through a polynomial kernel. We did not consider other types of kernels such as radial basis kernel as they combine the input features in a way that does not produce decomposable models. For the learners that involve SVMs, we tuned the parameters (e.g., amount of regularization) using crossfold validation, and we set the degree of the polynomial kernel to be 6 after trying all kernels of degree 2 to 8. We applied the polynomial kernel to other learners (full, L1, MI, ensemble, tree) as well, but that did not improve the results.

3.1 Experiment Setup

We generated a synthetic data set which models 5 users, randomly and uniformly selecting one of 5 location to visit. Each user remains at the location for a random period of time from 1 to 10 hours, and randomly and uniformly selects one of 5 activities to perform at the location. This is meant to model the type of input data that LifeJoin produces. The experiments were run on a server with 32 cores and 30GB of RAM. We chose to evaluate our methods on synthetic data rather than actual data since no publicly available large data set is available, and using synthetic data decouples us from the potential errors in data collection on the phones. The data set is generated randomly and does not favor any particular learner.

Learner	Pred 1	Pred 2	Pred 3	Pred 4	Pred 5	Pred 6
full	100%	100%	85.5%	79.5%	77.5%	81%
unary	75%	75%	75%	75%	75%	75%
poly	76%	76.5%	76.5%	76%	75%	75%
L1	100%	100%	92.5%	88%	74.5%	81.5%
MI	100%	100%	83%	82.5%	78%	82%
tree	100%	100%	91.5%	89.5%	91%	81.5%
ensemble	100%	100%	98%	96.5%	93.5%	92.5%
hybrid	100%	100%	97.5%	95%	94%	93.5%

full	43k	43k	43k	43k	43k	43k
unary / poly / tree	344	344	344	344	344	344
L1	61.5	118.6	32.6	265.5	482.9	604.9
MI	6708.5	6906.5	6990	6696.7	6430.8	6650.8
ensemble	26.5	40.1	50.2	45.2	40.2	46.7
hybrid	27.9	43.8	43.2	39.4	39.5	41.8

Figure 2: (a) Cross validation accuracies (top) and (b) feature set sizes (bottom)

In addition to a data set, we need a way to generate user interests. To do this, we manually created 6 interest functions of increasing complexity and class imbalance in the input training data. For instance, the first interest function labels about 40% of the events to be positive, whereas the last (most complicated) interest function labels only about 10% of the events as positive. This is to model how class distribution can vary drastically among different user interests.

For all the experiments we allow Sketch to learn a maximum of 14 different interests, and allow each interest to consist of a maximum of 7 different conjuncts. The numbers were picked from initial sampling of 5 users. Limiting to 7 conjuncts is more than needed in order to learn the actual predicates, but we used that setting as we believe this level of interest complexity is a reasonable approximation of the maximum complexity of interests a user might have.

To generate training data, we labeled data points in our data set using each of the interest functions, assigning a positive label to the event for a given interest function if the interest function evaluates to true.

3.2 Cross Validation Experiments

In the first set of experiments, we evaluate the accuracies of the different schemes using cross validation. The goal is to evaluate learner performance in the absence of any performance anomalies the active learning methods may introduce.

For each of the predicates we first generated a dataset of 100 positively and 300 negatively labeled events. The events are uniformly sampled from a domain consisting of 5 users, 5 locations, and 5 different types of activities. We ran 10-fold validation on the dataset, where we divide the positive and negative events into 10 partitions. Figure 2(a) shows the average accuracies and Fig. 2(b) shows the number of features that are used for classification.

The results show that our hybrid learner has similar accuracy as compared to standard machine learning techniques, but does not require using the full feature set as in learners such as LI or MI. This is particularly important when comparing the number of features that are used for classification. To achieve the same overall

accuracy, the number of features used by the hybrid and ensemble learners are an order of magnitude smaller as compared to others, as shown in Fig. 2(b).

3.3 Active Learning Experiments

Next, we evaluate the learners in the actual usage setting, where the user is asked to label a few new data points each time she visits her newsfeed. At the end of each round the learner is given the newly rated events along with the previously rated ones to refine its model about the user. The goal of the learner is to select the list of events to present in each round so as to maximize the accuracy of the model, and to do so with as few rounds as possible.

3.3.1 Basic Setting

We generate 100 positive and 300 negative events as a training set to be presented during active learning. The events are generated using the same settings as in the cross validation experiments. We then generate an additional 10k events and ratings (which are not given to the learners) to use as the test set. The events in the test set are generated randomly without regards to the ratio of positive and negative events. Initially, the learners are given 1 positive and 1 negative event to learn an initial model. During each iteration, the learners choose 5 events from the training pool to query for their ratings to rebuild the model. We measure the accuracy of the model at the end of each round for 20 rounds. Figure 3(a) shows the results on the most complex interest function averaged over 10 runs. The results for the others are similar.

The focus of these results is the learning rate, i.e., the rate at which the accuracy increases. While the learners that use classical feature selection mechanisms (L1 and MI) do have higher learning rates as compared to those that do not (full and unary), our hybrid and ensemble learners have a significantly higher learning rate than others, since they pick features with higher predictive power.

Figure 3(b) shows the number of features that are used for classification in each round for the learners. While they all increase as the number of rounds increases as expected, the growth rate for the hybrid and ensemble learners that use Sketch for feature selection is much slower than the others.

3.3.2 Making Use of Extended Labels

One of the advantages of the hybrid learner over the ensemble learner is that the SVM in the hybrid learner is able to make use of extended labels. This is because extended labels simply change the problem from classification to regression, where instead of a binary label (e.g., "like" or "dislike"), the goal is to predict ratings on continuous a scale from -1 to +1. We repeat the same experiment as in the basic setting but with extended labels for events. For events that are of interest, the label remains as +1 as before. For those that are not, the label is negative, but its value is computed in the following way. Given the user's interest expressed as N disjuncts $\vee d_i$, where each d_i is a conjunction of predicates, if the event e fails all disjuncts, the value of its label is computed as $\min(\#failed(d_i, e)/\#(d_i, e))$, where $\#failed(d_i, e)$ is the number of predicates that e has failed within d_i, and $\#(d_i, e)$ is the total number of conjuncts in d_i. We choose to pick the minimum since this represents the minimal number of changes in e that would make the user happy. We present the accuracy results in Fig. 3(c) for running on function 6, and they show that the learning rate for the hybrid-regression learner is faster as compared to the ensemble and original hybrid-binary learners. This makes sense since the regression learner is able to make use of the extended information that is embedded within the "near miss" cases in selecting better samples during each round of active learning.

3.3.3 Large Domain

Next, we increase the number of users and the number of locations from 5 to 50, and the number of activities from 5 to 10. This is to model a larger group of users. We generated the 100 positive and 300 negative training events from the new domain using uniform sampling, and an additional 10k events for the test set. We execute the same active learning experiment as before. Figure 4(a) shows the results.

On the outset, it seems that all learners achieve high overall accuracies on the test data, but close examination proves the contrary. Unlike previous experiments where the ratio of positively and negatively rated events is not heavily skewed, in this case, due to the large domain size, only around 3% of the events in the test set are positively rated, so the learners quickly learn to assign negative to most test events to maximize overall accuracy. This results in high precision on the negatively rated events but very low precision on the positively rated ones. The decision tree based classifier, however, decided rather to generalize on the positively labeled events and classifies almost all events as interesting. Thus, it achieves high accuracy on the positive events and poorly on the negative ones, resulting in low overall accuracy. To illustrate this, Figure 4(b) show the accuracy results on just the positive events. The results show that although the overall accuracies of the learners are comparable, the hybrid and ensemble approaches actually perform much better than the other learners on the positive events.

This experiment raises two important points when comparing among the learners. First, all of the learners except for hybrid, ensemble, poly, and tree require enumeration of the full feature set for all events. In this large domain case, this takes a substantial amount of time (2 hours for conversion into the feature representation) and disk space (300 MB needed to encode 10k events), as compared to the synthesis-based feature selection approach used in the hybrid and ensemble learners, which takes much less time (10 min to finish the Sketch runs and seconds to convert the chosen features into feature-space representation) and negligible disk space (600 kB to encode 10k events). Secondly, the fact that the classical machine learning based learners assign negative labels to most events means that they will very likely not be able to identify any interesting events for the user, which is the ultimate goal.

3.4 Model Explanation Experiments

In these experiments we test the effectiveness of using a program synthesizer at producing a decomposable model. As discussed in Sec. 2.1, we took the support vectors after model generation and fed them into Sketch. We used the data from the cross validation experiments to learn 6 interest functions. The model for the most complex function consists of 198 support vectors, and all the models inferred by Sketch are very similar to the original interest functions.

4. RELATED WORK

Many probabilistic modeling approaches have been proposed that can also be applied to the learning problem discussed in this paper, including Bayesian networks [9], statistical relational learning [7], and probabilistic logic [14]. There are also work in building probabilistic models predicting user behavior [22, 10, 2]. However, as with SVMs, models learned using such techniques tend not to generate decomposable models.

Other inductive learning techniques, such as inductive logic programming [15, 13], learn formulas from the training data and can produce decomposable models. However, such tools still assume the input data to have certain class distribution, and it is unclear how feature selection can be done for such techniques.

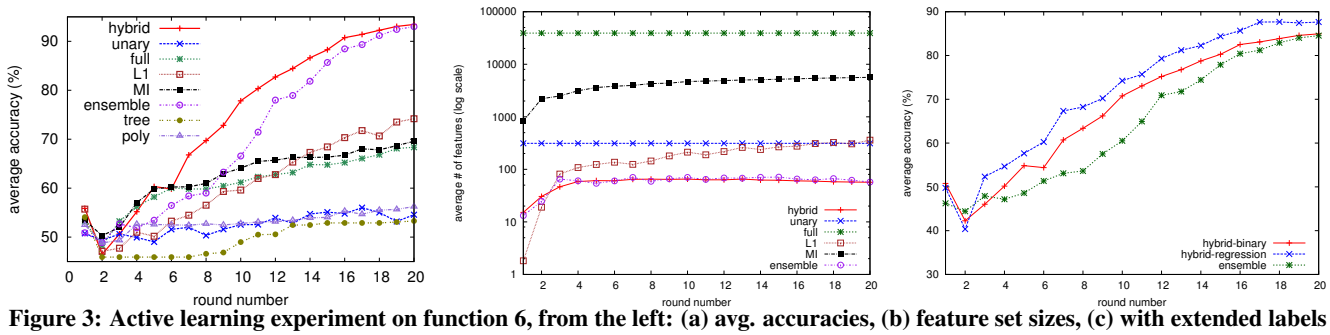

Figure 3: Active learning experiment on function 6, from the left: (a) avg. accuracies, (b) feature set sizes, (c) with extended labels

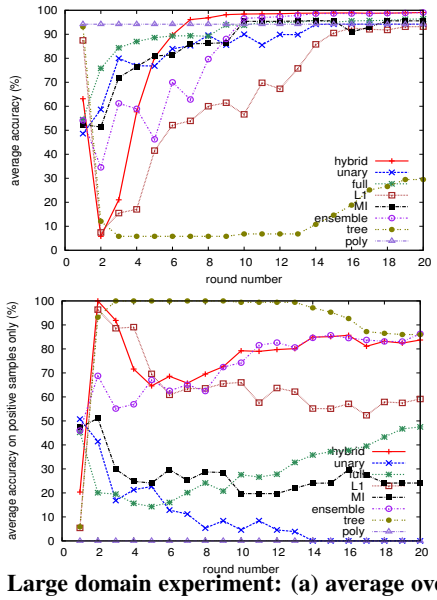

Figure 4: Large domain experiment: (a) average overall accuracies (top) and (b) positive events only (bottom)

There are many feature selection algorithms in addition to mutual information and LASSO. However, our synthesis-based approach differs from classification techniques in that most feature selection techniques focus on the statistical properties of the training data, e.g., approximating the probability distribution of a feature from the training data. Such schemes perform well when fed with a sufficiently large amount of training data, as evident in our cross validation experiments, but do not do so well in cases when the training data size is small, as in our active learning scenarios.

In recent years, the programming languages community has been working on programming-by-example problems to synthesize different types of programs [8, 18]. Our work differs in that we require a feature selection mechanism in place in order to provide reasonable results. The work of Gulwani in [8] queries the user to provide differentiating outputs when the synthesizer cannot decide between multiple programs that satisfy the same input constraints. We generalize this concept and propose the ensemble learning scheme, and show that a scheme that combines synthesis-based feature selection with an SVM for classification can provide excellent performance for social networking applications.

5. CONCLUSIONS

We presented a learning algorithm that combines the machine learning techniques with program synthesis tools and focuses on personalized social recommendation applications. The experiments show that the hybrid approach can significantly outperform traditional classification schemes on synthetic data, but an important next step is to validate the results on real-world data. Similarly, more research is needed in analyzing the generalization properties of the synthesis-based approach. Understanding its theoretical connections with classical machine learning-based techniques with help develop further algorithms that leverage the advantages of the two in improving results.

6. REFERENCES

[1] Weka toolkit. `http://www.cs.waikato.ac.nz/ml/weka/`.

[2] R. M. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *proc. KDD*, pages 95–104, 2007.

[3] H. Cao, E. Chen, J. Yang, and H. Xiong. Enhancing recommender systems under volatile user interest drifts. In *proc. CIKM*, 2009.

[4] N. V. Chawla, N. Japkowicz, and A. Kotcz. Editorial: special issue on learning from imbalanced data sets. *SIGKDD Explorations*, 6(1):1–6, 2004.

[5] A. Cheung, A. Thiagarajan, and S. Madden. Automatically generating interesting events with lifejoin. In *proc. Sensys*, pages 411–412, 2011.

[6] M. Dong and L. Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *proc. MobiSys*, pages 335–348, 2011.

[7] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

[8] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *proc. POPL*, pages 317–330, 2011.

[9] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[10] Y. Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *TKDD*, 4(1), 2010.

[11] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg. Predicting positive and negative links in online social networks. In *proc. WWW*, pages 641–650, 2010.

[12] K. Liu and L. Tang. Large-scale behavioral targeting with a social twist. In *proc. CIKM*, pages 1815–1824, 2011.

[13] S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13(3&4):245–286, 1995.

[14] N. J. Nilsson. Probabilistic logic. *Artif. Intell.*, 28(1):71–87, 1986.

[15] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, Sept. 1990.

[16] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[17] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, July 1990.

[18] R. Singh and A. Solar-Lezama. Synthesizing data structure manipulations from storyboards. In *proc. FSE*, pages 289–299, 2011.

[19] A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, and V. Saraswat. Combinatorial sketching for finite programs. *SIGOPS Oper. Syst. Rev.*, 40:404–415, October 2006.

[20] V. Sugumaran, V. Muralidharan, and K. Ramachandran. Feature selection using decision tree and classification through proximal support vector machine for fault diagnostics of roller bearing. *Mechanical Systems and Signal Processing*, 21(2):930 – 942, 2007.

[21] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In *Journal of Machine Learning Research*, pages 999–1006, 2000.

[22] D. Yin, Z. Xue, L. Hong, and B. D. Davison. A probabilistic model for personalized tag prediction. In *proc. KDD*, pages 959–968, 2010.