Machine learning is one of the most important applications of computers today, but, like the Internet in the mid-90's, its potential is far from being realized. This is because designing a machine learning solution to most problems is still very difficult, and the results are highly variable. Companies like Google have managed to use machine learning technologies to great success, but few organizations can afford to invest that much money in research. In order to reap the greatest benefits from machine learning, we must make it ubiquitous, and in order to make it ubiquitous, we must make it more powerful and less difficult to apply.

1 Overview

The first requirement for machine learning is something to learn from, either data or knowledge (which can be thought of as a particularly rich source of data). In this respect things look good: While some data remains too sensitive to share, the amount of freely available information continues to explode through the low-cost hard drives and sensors and higher bandwidth on the Internet. Each year, we collect more and more information, and much of that information is publicly available for study.

Where we fall behind is in methods to make proper use of this information. Most machine learning algorithms were designed to handle vector data, ignoring the relational structure present in most real-world problems. Some algorithms work with sequences or clusters, and NLP methods work with trees, but many problems fall outside of these simple frameworks. Consider the case of entity resolution, determining which observations refer to the same actual object. A traditional approach is to determine which pairs refer to the same object and take the transitive closure. This casts entity resolution as a similarity function. However, by splitting the problem into independent decisions, we throw away global information that could be exploited through joint inference. A better approach is to answer all interrelated questions at once, so that their respective constraints can all be taken into account. This is already done with sequence models, such as hidden Markov models and conditional random fields, but is rarely done in general. As a second example, consider the task of labeling web pages by topic. In addition to the words on the web page, the labels of linked web pages tend to be highly informative. Therefore, the best way to label a group of linked web pages is to label all of them at once.

This approach of "doing everything at once" is often the most effective solution, but developing statistical models (including learning and inference algorithms for them) is difficult. Ideally, we would like to use a single, simple class of models to handle the full richness of these problems. Rich relational structure is traditionally handled with first-order logic; uncertain data (such as that generated by stochastic processes, corrupted by noise, or incompletely understood) is typically modeled using probability. The goal of statistical relational learning is to combine first-order logic and probability, providing a more powerful and general framework for machine learning. Instead of reducing complex problems to vectors, a statistical relational approach allows us to work directly with the natural problem structure, often yielding better results with less work.

However, in their current state, statistical relational models are far from a panacea: being based on probabilistic graphical models, they inherit computationally intractable inference problems on even relatively simple models. This is made even more difficult given the large size of most statistical relational models, which could have millions of interrelated variables. One solution to this is to develop better approximate inference algorithms. The ideal algorithm would trade-off time and accuracy, giving exact results on small problems or with large amounts of time, and giving the best possible approximation when given larger problems or a small amount of time. It is often possible to exploit the structure of an inference problem to solve it efficiently (exactly or approximately); it is reasonable to think that the rich structures in statistical relational models could sometimes help us find efficient inference procedures. An alternate approach is to use inference cost as a learning bias. Instead of learning a model that has high theoretical accuracy but is intractable in practice, we focus on learning the most accurate model possible within the constraint that inference must be tractable. This can lead to models that are more efficient and more accurate in practice, since they avoid the errors introduced by approximate inference. Whether we can find an efficient and robust approximate inference algorithm or whether it works best to do exact inference on a tractable model, taking the guesswork out of inference is very important. Currently, we are faced with a large number of algorithms – Gibbs sampling, belief propagation, variable elimination, etc. – each with its own subtleties and compromises. Using the wrong algorithm or the right algorithm with the wrong parameters can lead to very bad results. Similarly, we need to reduce the amount of guesswork involved in applying a learning algorithm. Statistical relational learning helps by directly representing relations and easing the integration of knowledge (through first-order logic) and naturally handling uncertainty (through probability), but much can still go wrong: minor differences in how a model is described can still lead to large differences in how hard it is to learn and reason with.

I am interested in developing powerful and practical machine learning representations and algorithms. While it may always be possible to create specially engineered solutions that outperform generic methods, we are likely to see the biggest gains from improving the performance and reliability of our best "generic" methods. A good analogy is microprocessors: while specialized chips still have their purpose, generic microprocessors (x86, ARM, etc.) offer an effective and low-cost solution for many problems. For machine learning and artificial intelligence in general, the best general-purpose solution to date is Markov logic, a language for statistical relational representation, learning, and reasoning. In its simplest form, Markov logic just attaches a weight to each formula in a first-order knowledge base. Given a set of constants for a domain, the sum of the weights of satisfied formulas determines the relative probability of a possible world. In the infinite-weight limit, Markov logic reduces to first-order logic. With finite weights, worlds that violate one or more formulas are not impossible, just less likely. This permits the use of unreliable or even inconsistent knowledge bases. Weights and formulas can be learned or revised from data using ideas from convex optimization and inductive logic programming. Already, Markov logic has been used to build state-of-the-art systems for computational biology, natural language processing, entity resolution, and more.

2 Statistical Relational Learning

My work to date has improved and extended Markov logic in two key ways. First, I developed a multi-layer generalization of Markov logic called recursive random fields (RRFs). RRFs resolve a key inconsistency in the Markov logic representation, represent many probability distributions exponentially more compactly than an MLN, and allow for new, more flexible approaches to learning. One can view a logical knowledge base as a single formula tree, in which each leaf is a variable and each interior node is a conjunction or disjunction, or an existential or universal quantifier. Markov logic "softens" this knowledge base by replacing the top-level universal quantifiers and conjunctions with a noisy AND, but all lower levels remain purely logical. Recursive random fields make every level of the formula tree probabilistic, using multiplication to replace conjunctions (as already done by Markov logic) and using negative weights to replace logical negation. Since by De Morgan's laws negating a conjunction or universal quantifier yields a disjunction or existential quantifier, respectively, this allows us to create softened disjunctions and existentials. This is useful for representing "m-of-n" or "m-of-all" concepts, e.g., a certain diagnosis is likely if at least four of ten symptoms are present, or we might say that popularity depends on having at least 20 friends. Furthermore, by attaching weights to every level of the tree, we can in principle revise the knowledge base itself through weight learning. For example, we expect subformulas not supported by the data to end up with weights of zero, which is equivalent to removing them from the knowledge base. If we add a general set of other links with low weight, then we might discover new formulas without directly searching through the space of formulas. This is similar to backpropagation neural networks, which are capable of representing and learning many different functions starting from a simple structure and simple learning procedure. In fact, I showed how a variant of backpropagation can be used to learn RRF weights.

Unfortunately, using recursive random fields in practice is still quite difficult. I found that learning was very sensitive to the initial weights, and inference was significantly slower than for MLNs. So I put my work on RRFs aside to focus on making MLNs more practical, in hopes of eventually applying those ideas to RRFs. I still believe that RRFs and other generalized representations are important, but sometimes the best approach is to solve a hard problem in a simpler context, such as Markov logic or probabilistic graphical models, and later extend the methods to the more general representation. At the time, the weight learning algorithms for Markov logic were still rather crude and frequently failed on hard problems. Other researchers had been forced to simplify their models or set weights via various hacks. I adapted several convex optimization methods to cope with the particular challenges of Markov logic, including an intractable objective function and a gradient that can only be approximated. My algorithms are now the standard approach to Markov logic weight learning.

3 Efficient Inference

One of the important remaining challenges for Markov logic, and in fact, for graphical models in general, is efficient inference. Exact inference computes the most accurate probabilities, but is often intractable since probabilistic inference is #P-complete. Approximation algorithms range from Markov chain Monte Carlo (MCMC) sampling to message passing, each with its distinct set of advantages and disadvantages. I have worked to address the problem of efficient inference for the case of propositional graphical models, with the hope of eventually extending these results to Markov logic or a similar representation. Rather than trying to solve the hard inference problem directly, I developed a novel technique to avoid it by only learning models that allow for efficient inference. The traditional way to avoid difficult inference problems is to use a model class for which inference is guaranteed to be efficient, such as certain mixture models. I showed that one of the simplest models for probability estimation, naive Bayes mixture models, can be competitive with state-of-the-art Bayesian networks by conducting experiments on 50 datasets. In other words, by choosing a model class such as naive Bayes, you get guaranteed efficient inference and can still achieve similar accuracy to a Bayesian network. For some problems, Bayesian networks remain a better representation. For these cases, we can still avoid hard inference problems by using inference cost as a learning bias. I developed an algorithm that uses the exact same search space as a state-of-the-art Bayesian network structure learning algorithm, but changes the objective function to look at both accuracy and inference cost. We measure inference cost by looking at the size of the

Bayesian network when compiled into an arithmetic circuit, since inference is linear in the size of such a circuit. Arithmetic circuits are able to do inference in some probability distributions very efficiently by exploiting context-specific independence. By looking at circuit size as we search for a good Bayesian network, we are guaranteed to end up with a compact circuit and hence efficient inference, without artificially restricing our model class at all. We found that networks trained to have small circuits were only slightly less accurate than those that purely optimized accuracy. However, taking into account the errors introduced by approximate inference, the networks that optimized accuracy were less accurate and orders of magnitude slower. By combining learning and inference, we were able to learn models that appeared very complex (treewidth ¿ 100) but still allowed for very efficient inference (¡ 100ms).

4 Current and Future Work

I am currently working on exploring what else can be done with arithmetic circuits. For Markov logic networks, the structure of the graphical model is given by the formulas and may not allow for efficient, exact inference. In such a case, we may need to use a form of approximate compilation, in which we build an arithmetic circuit of a limited size to best approximate the probability distribution. Markov logic networks are a good candidate for efficient compilation because the formulas have a great deal of structure that could be exploited. When the necessary conditional independencies do not exist, we should be able to introduce them as necessary, favoring the structures with greatest accuracy and efficiency, as in my work on learning Bayesian networks.

For the future, I plan to continue exploring rich representations and efficient algorithms. People who want to apply machine learning should be able to express their knowledge and data naturally, learn a good model automatically, and use the model to make predictions effficiently. Machine learning has come a long way towards this goal for the subproblem of classification using vector data. Statistical relational learning brings greater challenges, but also greater opportunities to use our data to solve problems in bioinformatics, natural language processing, social network analysis, user interface design, and more. One of the things that excites me about machine learning is its power to solve such a wide range of problems in different fields. But to have the greatest impact, machine learning must be more flexible and require less knowledge of machine learning. That is what I want to help accomplish.

My approach to research is to start with the simplest possible method and use it as a baseline. Often, these baselines work surprisingly well, sometimes better than much more sophisticated methods. I believe in doing thorough empirical evaluations on real-world data with standard metrics and methodologies. Many ideas that sound interesting turn out to be useless on real problems; experimentation is the only way to determine this. However, I also believe in performing whatever theoretical analysis is feasible; theory is one of the most effective ways to predict whether or not an idea will translate well to other problems. Finally, I believe in making data and source code available whenever possible. This is crucial for reproducible results, which is one of the hallmarks of scientific research. It can also catalyze follow-up research, which is necessary for further progress, and industry adoption, which is eventually necessary for having an impact in the real world. I am proud to have contributed to several open-source machine learning software projects, including Alchemy, a toolkit for working with Markov logic.

5 Other Research

I have also worked on various applications of machine learning, including collaborative filtering, spam filtering, and desktop activity recognition. My work on collaborative filtering was to empirically compare a variety of published algorithms and determine which ones actually perform best. I found that well-cited algorithms sometimes performed worse than the simple algorithms they claimed to beat. Some of this was due to misimplementing baseline algorithms in subtle ways. This demonstrates the importance of careful experimental evaluation and that simpler algorithms are sometimes the best.

The work on spam filtering I conducted while interning at Microsoft Research is already well-cited, and led to invited talks at Oregon State University and the University of Cagliari, Italy. In this work, I looked at the adversarial problem of defeating a spam filter by adjusting the content of an email message. I developed simple but novel ways to reverse engineer spam filters by sending test messages, as well as a way to defeat spam filters with no prior knowledge. These practical attacks offer insights into what make spam filters weak, as well as methods for evaluating spam filter vulnerability. I also formulated the adversarial classifier reverse engineering (ACRE) learning problem so that the hardness of attacking different classifiers can be formally analyzed.

Most recently, my work on desktop activity recognition shows that relatively simple classifiers can be effective when given the right set of features. I developed novel feature sets that make a standard classifier aware of the sequence of recent events by representing salience. For example, if the word "paper" most recently appeared in an activity associated with the Job Application project, then this knowledge might override the fact that paper is more commonly associated with the Office Supplies project. This model can be formulated relationally, but due to real-time usability constraints, I did not use Markov logic. I would like to see a full, relational approach eventually applied to such problems.