Oxford Handbook of Computational Linguistics,
R. Mitkov (Ed.), Oxford University Press, pp. 376-394, 2003

20. Machine Learning

Raymond J. Mooney

Abstract

This chapter introduces symbolic machine learning in which decision trees, rules, or case-based classifiers are induced from supervised training examples. It describes the representation of knowledge assumed by each of these approaches and reviews basic algorithms for inducing such representations from annotated training examples and using the acquired knowledge to classify future instances. These techniques can be applied to learn knowledge required for a variety of problems in computational linguistics ranging from part-of-speech tagging and syntactic parsing to word-sense disambiguation and anaphora resolution. Applications to a variety of these problems are reviewed.

20.1 Introduction

Broadly interpreted, **machine learning** is the study of computational systems that improve performance on some task with experience (Mitchell, 1997; Langley, 1996). However, the term is frequently used to refer specifically to methods that represent learned knowledge in a declarative, symbolic form as opposed to more numerically-oriented statistical or neural-network training methods (see chapter 19). In particular, it concerns methods which represent learned knowledge in the form of interpretable decision-trees, logical rules, and stored instances. **Decision trees** are classification functions represented as trees in which the nodes are attribute tests, the branches are attribute values, and the leaves are class labels. Rules are implications in either propositional or predicate logic used to draw deductive inferences from data. A variety of algorithms exist for inducing knowledge in both of these forms from training examples. In contrast, instance-based (case-based, memory-based) methods simply remember past training instances and make a decision about a new case based on its similarity to specific past examples. This chapter reviews basic methods for each of these three approaches to symbolic machine learning. Specifically, we review top-down induction of decision trees, **rule induction** (including inductive logic programming), and nearest-neighbor **instance-based learning** methods.

As described in previous chapters, understanding natural language requires a large amount of knowledge about morphology, syntax, semantics, and pragmatics as well as general knowledge about the world. Acquiring and encoding all of this knowledge is one of the fundamental impediments to developing effective and robust language-processing systems. Like the statistical methods described in the previous chapter, machine learning methods offer the promise of automating the acquisition of this knowledge from annotated or unannotated language corpora. A potential advantage of symbolic learning methods over statistical methods is that the acquired knowledge is represented in a form that is more easily interpreted by human developers and more similar to representations used in manually developed systems. Such interpretable knowledge potentially allows for greater scientific insight into linguistic phenomena, improvement of learned knowledge through human editing, and easier integration with manually developed systems. Each of the machine learning methods we review has been applied to a variety of problems in computational linguistics including morphological generation and analysis, part-of-speech tagging, syntactic parsing, word-sense disambiguation, semantic analysis, information extraction, and anaphora resolution. We briefly survey some of these applications and summarize the current state of the art in the application of symbolic machine learning to computational linguistics.

20.2 Learning for Categorization

Most machine learning methods concern the task of categorizing examples described by a set of features. It is generally assumed that a fixed set of n discrete-valued or real-valued features, $\{f_1, \ldots, f_n\}$, are used to describe examples, and that the task is to assign an example to one of m disjoint categories $\{c_1, \ldots, c_m\}$. For example, consider the task of deciding which of the following three sense categories is the correct interpretation of the semantically ambiguous English noun "interest" given a full sentence in which it appears as context (see Chapter 13).

1. c_1: readiness to give attention

2. c_2: advantage, advancement, or favor

3. c_3: money paid for the use of money

The following might be a reasonable set of features for this problem:

- W+i: the word appearing i positions after "interest" for $i = 1, 2, 3$
- W$-i$: the word appearing i positions before "interest" for $i = 1, 2, 3$
- K_i: a binary-valued feature for a selected keyword for $1 = 1, \ldots, k$, where K_i is true if the ith keyword appears anywhere in the current sentence. Relevant keywords for "interest" might be, "attracted," "expressed," "payments," "bank" etc..

A learning system is given a set of supervised training examples for which the correct category is given. For example:

1. c_1: "John expressed a strong interest in computers."
2. c_2: "Acme Bank charges very high interest."
3. c_3: "War in East Timor is not in the interest of the nation."

In this case, the values of the relevant features must first be determined in a straightforward manner from the text of the sentence. From these examples, the system must produce a procedure for accurately categorizing future examples.

Categorization systems are typically evaluated on the accuracy of their predictions as measured by the percentage of examples that are correctly classified. Experiments for estimating this accuracy for a particular task are performed by randomly splitting a representative set of labeled examples into two sets, a training set used to induce a classifier, and an independent and disjoint test set used to measure its classification accuracy. Averages over multiple splits of the data into training and test sets provide more accurate estimates and give information on the variation in performance

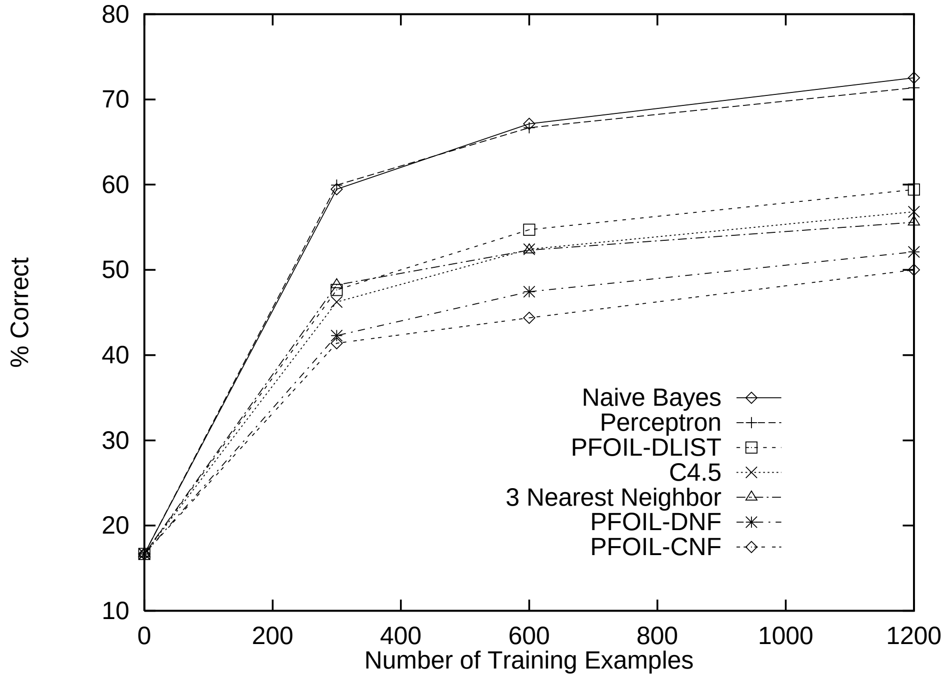

Figure 1: Learning Curves for Disambiguating "Line"

across training and test samples. Since labeling large amounts of training data can be a time-consuming task, it is also useful to look at *learning curves* in which the accuracy is measured repeatedly as the size of the training set is increased, providing information on how well the system generalizes from various amounts of training data. Figure 1 shows sample learning curves for a variety of systems on a related task of semantically disambiguating the word "line" into one of six possible senses (Mooney, 1996). Mitchell (1997) provides a basic introduction to machine learning including discussion on experimental evaluation.

20.3 Decision Tree Induction

Decision trees are classification functions represented as trees in which the nodes are feature tests, the branches are feature values, and the leaves are class labels. An example is classified by starting at the root and recursively traversing the tree to a leaf by following the path dictated by its feature values. A sample tree for the "interest" problem is shown in Figure 2. For simplicity, assume that all of the unseen extra branches for W+1 and W+2 are leaves labeled c_1. This tree

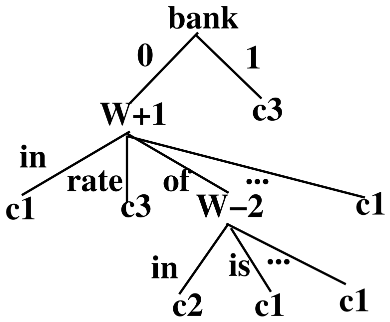

Figure 2: Sample Decision Tree for Disambiguating "interest"

can be paraphrased as follows: If the word "bank" appears anywhere in the sentence assign sense c_3; otherwise if the word following "interest" is "rate," assign sense c_3, but if the following word is "of" and the word two before is "in" (as in "...in the interest of..."), then assign sense c_2; in all other cases assign sense c_1.

The goal of learning is to induce a decision tree that is consistent with the training data. Since there are many trees consistent with a given training set, most approaches follow "Occam's razor" and try to induce the simplest tree according to some complexity measure such as the number of leaves, or the depth of the tree. Since computing a minimal tree according to such measures is an NP-hard problem (i.e. a computational problem for which there is no known efficent, polynomial-time algorithm), most algorithms perform a fairly simple greedy search to efficiently find an approximately minimal tree. The standard approach is a divide-and-conquer algorithm that constructs the tree top-down, first picking a feature for the root of the tree and then recursively creating subtrees for each value of the selected splitting feature. Pseudocode for such an algorithm is shown in Figure 3.

The size of the constructed tree critically depends on the heuristic used to pick the splitting feature. A standard approach is to pick the feature that maximizes the expected reduction in the entropy, or disorder, of the data with respect to category (Quinlan, 1986). The entropy of a set of

InduceTree(*Examples*, *Features*)
  Create a node *Root* for the tree
  If all the examples are in the same category
    then return *Root* labeled with this category as a leaf.
  If features are empty
    then return *Root* labeled with the most common category in *Examples* as a leaf.
  Else pick the best splitting feature f_i and use it to label *Root*.
  For each possible value v_{ij} of f_i
    Add a branch to *Root* for the value v_{ij}.
    Let $Examples_{ij}$ be the subset of *Examples* with value v_j for f_i.
    If $Examples_{ij}$ is empty
      then below the branch add a leaf labeled with the most common category in *Examples*.
      else below the branch add the subtree InduceTree($Examples_{ij}$, $Features - \{f_i\}$).
  Return *Root*.

Figure 3: Decision Tree Induction Algorithm

data, S, with respect to category is defined as:

$$Entropy(S) = \sum_{i=1}^{m} -\frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} \tag{1}$$

where S_i is the subset of S in category i $(1 \leq i \leq m)$. The closer the data is to consisting purely of examples in a single category, the lower the entropy. A good splitting feature fractions the data into subsets with low entropy. This is because the closer the subsets are to containing examples in only a single category, the closer they are to terminating in a leaf, and the smaller will be the resulting subtree. Therefore, the best split is selected as the feature, f_i, that results in the greatest *information gain* defined as:

$$Gain(S, f_i) = Entropy(S) - \sum_j \frac{|S_{ij}|}{|S|} Entropy(S_{ij}) \tag{2}$$

where j ranges over the possible values v_{ij} of f_i, and S_{ij} is the subset of S with value v_{ij} for feature f_i. The expected entropy of the resulting subsets is computed by weighting their entropies by their relative size $|S_{ij}|/|S|$.

The resulting algorithm is computationally very efficient (linear in the number of examples) and in practice can quickly construct relatively small trees from large amounts of data. The basic algo-

bank $\rightarrow c_3$
$\neg$ bank $\wedge$ W+1=in $\rightarrow c_1$
W+1=rate $\rightarrow c_3$
$\neg$ bank $\wedge$ W+1=of $\wedge$ W$-$2=in $\rightarrow c_2$

Figure 4: Sample Rules for Disambiguating "interest"

rithm has been enhanced to handle many practical problems that arise when processing real data, such as noisy data, missing feature values, and real-valued features (Quinlan, 1993). Consequently, decision-tree methods are widely used in *data mining* applications where very large amounts of data need to be processed (Fayyad, Piatetsky-Shapiro, & Smyth, 1996). The most effective recent improvement to decision-tree algorithms have been methods for constructing multiple alternative decision trees from the same training data, and then classifying new instances based on a weighted vote of these multiple hypotheses (Quinlan, 1996).

20.4 Rule Induction

Classification functions can also be symbolically represented by a set of rules, or logical implications. This is equivalent to representing each category in *disjunctive normal form* (DNF), i.e. a disjunction of conjunctions of feature-value pairs, where each rule is a conjunction corresponding to a disjunct in the formula for a given category. For example, the decision tree in Figure 2 can also be represented by the rules in Figure 4, assuming that c_1 is the *default category* that is assigned if none of the rules apply.

Decision trees can be translated into a set of rules by creating a separate rule for each path from the root to a leaf in the tree (Quinlan, 1993). However, rules can also be directly induced from training data using a variety of algorithms (Mitchell, 1997; Langley, 1996). The general goal is to construct the smallest rule-set (the one with the least number of symbols) that is consistent with the training data. Again, the problem of learning the minimally-complex hypothesis is NP-hard, and therefore heuristic search is typically used to induce only an approximately-minimal definition.

```
InduceRules(Examples, Features)
  Let S = ∅
  For each category c_i do
    Let P be the subset of Examples in c_i.
    Let N be the subset of Examples not in c_i.
    Until P is empty do
      Let R = ConstructRule(P,N,Features)
      Let S = S ∪ {R → c_i}
      Let C be the subset of P covered by R.
      Let P = P − C
  Return S
```

Figure 5: Rule-Induction Covering Algorithm

The standard approach is to use a form of greedy set-covering, where at each iteration, a new rule is learned that attempts to cover the largest set of examples of a particular category without covering examples of other categories. These examples are then removed, and additional rules are learned to cover the remaining examples of the category. Pseudocode for this process is shown in Figure 5, where ConstructRule(P, N, $Features$) attempts to learn a conjunction covering as many of the positive examples in P as possible without covering any of the negative examples in N.

There are two basic approaches to implementing ConstructRule. *Top-down* (*general-to-specific*) approaches start with the most general "empty" rule ($True \rightarrow c_i$), and repeatedly specialize it until it no longer covers any of the negative examples in N. *Bottom-up* (*specific-to-general*) approaches start with a very specific rule whose antecedent consists of the complete description of one of the positive examples in P, and repeatedly generalize it until it begins covering negative examples in N. Since top-down approaches tend to construct simpler (more general) rules, they are generally more popular. Figure 6 presents a top-down algorithm based on the approach used in the FOIL system (Quinlan, 1990). At each step, a new condition, $f_i = v_{ij}$, is added to the rule and the examples that fail to satisfy this condition are removed. The best specializing feature-value pair is selected based on preferring to retain as many positive examples as possible while removing as many negatives as possible. A gain heuristic analogous to the one used in decision trees can be

ConstructRule(P, N, $Features$)
  Let $A = \emptyset$
  Until N is empty do
    For each feature-value pair $f_i = v_{ij}$
      Let P_{ij} be the subset of P with value v_{ij} for feature f_i
      Let N_{ij} be the subset of N with value v_{ij} for feature f_i
    Given P, N, P_{ij}, and N_{ij}, pick the best specializing feature-value pair, $f_i = v_{ij}$
    Let $A = A \cup \{f_i = v_{ij}\}$
    Let $P = P_{ij}$
    Let $N = N_{ij}$
  Return the conjunction of feature-value pairs in A

Figure 6: Top-Down Rule Construction Algorithm

defined as follows:

$$Gain(f_i = v_{ij}, P, N) = |P_{ij}|(\log_2(\frac{|P_{ij}|}{|P_{ij}| + |N_{ij}|}) - \log_2(\frac{|P|}{|P| + |N|})) \tag{3}$$

where P_{ij} and N_{ij} are as defined in Figure 6. The first term, $|P_{ij}|$, encourages coverage of a large number of positives and the second term encourages an increase in the *percentage* of covered examples that are positive (decrease in the percentage of covered examples that are negative).

This and similar rule learning algorithms have been demonstrated to efficiently induce small and accurate rule sets from large amounts of realistic data. Like decision-tree methods, rule-learning algorithms have also been enhanced to handle noisy data and real-valued features (Clark & Niblett, 1989; Cohen, 1995). More significantly, they also have been extended to learn rules in first-order predicate logic, a much richer representation language. Predicate logic allows for quantified variables and relations and can represent concepts that are not expressible using examples described as feature vectors. For example, the following rules, written in Prolog syntax (where the conclusion appears first), define the relational concept of an uncle:

```
uncle(X,Y) :- brother(X,Z), parent(Z,Y).
uncle(X,Y) :- husband(X,Z), sister(Z,W), parent(W,Y).
```

The goal of *inductive logic programming* (ILP) or *relational learning* is to infer rules of this sort given a database of background facts and logical definitions of other relations (Lavrac & Dzeroski, 1994). For example, an ILP system can learn the above rules for uncle (the *target predicate*) given a set of positive and negative examples of uncle relationships and a set of facts for the relations parent, brother, sister, and husband (the *background predicates*) for the members of a given extended family, such as:

uncle(Tom,Frank),uncle(Bob,John), ¬uncle(Tom,Cindy), ¬uncle(Bob,Tom)

parent(Bob,Frank), parent(Cindy,Frank), parent(Alice,John), parent(Tom,John),

brother(Tom,Cindy), sister(Cindy,Tom), husband(Tom,Alice), husband(Bob,Cindy).

Alternatively, logical definitions for brother and sister could be supplied and these relations could be inferred from a more complete set of facts about only the "basic" predicates: parent, spouse, and gender.

The rule construction algorithm in Figure 6 is actually a simplification of the method used in the FOIL ILP system (Quinlan, 1990). In the case of predicate logic, FOIL starts with an empty rule for the target predicate ($P(X_1, \ldots, X_r) :-.$) and repeatedly specializes it by adding conditions to the antecedent of the rule chosen from the space of all possible literals of the following forms:

- $Q_i(V_1, \ldots, V_r)$
- $not(Q_i(V_1, \ldots, V_r))$
- $X_i = X_j$
- $not(X_i = X_j)$

where Q_i are the background predicates, X_i are the existing variables used in the current rule, and $V_1, \ldots, V_r$ are a set of variables where at least one is an existing variable (one of the X_i) but the others can be newly introduced. A slight modification of the *Gain* heuristic in equation 3 is used to select the best literal.

ILP systems have been used to successfully acquire interesting and comprehensible rules for a number of realistic problems in engineering and molecular biology, such as determining the cancer-causing potential of various chemical structures (Bratko & Muggleton, 1995). Unlike most methods which require "feature engineering" to reformat examples into a fixed list of features, ILP methods can induce rules directly from unbounded data structures such as strings, stacks, and trees (which are easily represented in predicate logic). However, since they are searching a much larger space of possible rules in a more expressive language, they are computationally more demanding and therefore are currently restricted to processing a few thousand examples compared to the millions of examples that can be potentially handled by feature-based systems.

20.5 Instance-Based Categorization

Unlike most approaches to learning for categorization, **instance-based learning** methods (also called *case-based* or *memory-based* methods) do not construct an abstract function definition, but rather categorize new examples based on their similarity to one or more specific training examples (Aha, Kibler, & Albert, 1991; Stanfill & Waltz, 1986). Training generally requires just storing the training examples in a database, although it may also require indexing the examples to allow for efficient retrieval. Categorizing new test instances is performed by determining the closest examples in the database according to some distance metric.

For real-valued features, the standard approach is to use Euclidian distance, where the distance between two examples is defined as:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (f_i(x) - f_i(y))^2} \tag{4}$$

where $f_i(x)$ is the value of the feature f_i for example x. For discrete-valued features, the difference, $(f_i(x) - f_i(y))$, is generally defined to be 1 if they have the same value for f_i and 0 otherwise (i.e. the *Hamming distance*). In order to compensate for differences in scale between different features,

```
KNN(Example, TrainingExamples, k)
    For each TrainingExample in TrainingExamples
        Compute d(Example,TrainingExample)
    Let Neighbors be the k TrainingExamples with the smallest value for d
    Let c be the most common category of the examples in Neighbors.
    Return c
```

Figure 7: K-Nearest-Neighbor Categorization Algorithm

the values of all features are frequently rescaled to the interval [0,1]. Intuitively, such distance measures are intended to measure the dissimilarity of two examples.

A standard algorithm for categorizing new instances is the *k-nearest neighbor* method (Cover & Hart, 1967). The k closest examples to the test example according to the distance metric are found, and the example is assigned to the majority class for these examples. Pseudocode for this process is shown in Figure 7. The reason for picking k examples instead of just the closest one is to make the method robust by basing decisions on more evidence than just one example, which could be noisy. To avoid ties, an odd value for k is normally used, typical values are 3 and 5.

The basic nearest-neighbor method has been enhanced with techniques for weighting features in order to emphasize attributes that are most useful for categorization, and for selecting a subset of examples for storage in order to reduce the memory requirements of retaining all training examples (Stanfill & Waltz, 1986; Aha et al., 1991; Cost & Salzberg, 1993).

20.6 Applications to Computational Linguistics

Decision tree learning, rule induction, and instance-based categorization have been applied to a range of problems in computational linguistics. This sections surveys applications to a variety of problems in language-processing starting with morphological and lexical problems and ending with discourse-level tasks.

20.6.1 Morphology

Symbolic learning has been applied to several problems in morphology (see Chapter 2). In particular, decision-tree and ILP methods have been applied to the problem of generating the past tense of an English verb, a task frequently studied in cognitive science and neural networks as a touchstone problem in language acquisition. In fact, there has been significant debate whether or not rule-learning is an adequate cognitive model of how children learn this task (Rumelhart & McClelland, 1986; Pinker & Prince, 1988; MacWhinney & Leinbach, 1991). Typically, the problem is studied in its phonetic form, in which a string of phonemes for the present tense is mapped to a string of phonemes for the past tense. The problem is interesting since one must learn the regular transformation of adding "ed," as well as particular irregular patterns such as that illustrated by the examples "sing" → sang," "ring" → "rang," and "spring" → "sprang."

Decision-tree algorithms were applied to this task and found to significantly outperform previous neural-network models in terms of producing correct past-tense forms for independent test words (Ling & Marinov, 1993; Ling, 1994). In this study, verbs were restricted to 15 phonemes encoded using the UNIBET ASCII standard, and 15 separate trees were induced, one for producing each of the output phoneme positions using all 15 of the input phonemes as features. Below is the encoding for the mapping "act" → "acted," where underscore is used to represent a blank.

```
&,k,t,_,_,_,_,_,_,_,_,_,_,_,_ => &,k,t,I,d,_,_,_,_,_,_,_,_,_,_
```

ILP rule-learning algorithms have also been applied to this problem and shown to outperform decision trees (Mooney & Califf, 1995). In this case, a definition for the predicate `Past(X,Y)`, was learned for mapping an unbounded list of UNIBET phonemes to a corresponding list for the past tense (e.g. `Past([&,k,t],[&,k,t,I,d])`) using a predicate for appending lists as part of the background. A definition was learned in the form of a *decision list* in which rules are ordered and the first rule that applies is chosen. This allows first checking for exceptional cases and falling through to a default rule if none apply. The ILP system learns a very concise and comprehensible definition for the past-tense transformation using this approach. Similar ILP methods have also been applied to learning morphology in other European languages (Manandhar, Dzeroski, & Erjavec, 1998; Kazakov & Manandhar, 1998).

20.6.2 Part-of-Speech Tagging

Tagging each word with its appropriate part-of-speech (POS) based on context is a useful first step in syntactic analysis (see chapter 11). In addition to statistical methods that have been successfully applied to this task, decision tree induction (Marquez, Padro, & Rodriguez, 1999), rule induction (Brill, 1995), and instance-based categorization (Daelemans, Zavrel, Berck, & Gillis, 1996) have also been successfully used to learn POS taggers.

The features used to determine the POS of a word generally include the POS tags in a window of 2 to 3 words on either side. Since during testing these tags must also be determined by the classifier, either only the previous tags are used, or an iterative procedure is used to repeatedly update all tags until convergence is reached. For known words, a dictionary provides a set of possible POS categories. For unknown words, all POS categories are possible but additional morphological features such as the last few characters of the word and whether or not it is capitalized are typically used as additional input features. Using such techniques, symbolic learning systems can obtain high accuracies similar to those obtained by other POS tagging methods, i.e. in the range of 96–97%.

20.6.3 Word-Sense Disambiguation

As illustrated by the "interest" problem introduced earlier, machine learning methods can be applied to determining the sense of an ambiguous word based on context (see Chapter 13). As also illustrated by this example, a variety of features can be used as helpful cues for this task. In particular, *collocational* features that specify words that appear in specific locations a few words before or after the ambiguous word are useful features, as are binary features indicating the presence of particular words anywhere in the current or previous sentence. Other potentially useful features include the parts of speech of nearby words, and general syntactic information, such as whether an ambiguous noun appears as a subject, direct object, or indirect object.

Instance-based methods have been applied to disambiguating a variety of words using a combination of all of these types of features (Ng & Lee, 1996). A feature-weighted version of nearest neighbor was used to disambiguate 121 different nouns and 70 verbs chosen for being both frequent and highly-ambiguous. Fine-grained senses from WordNet were used, resulting in an average of 7.8 senses for the nouns and 12 senses for the verbs. The training set consisted of 192,800 instances of these words found in text sampled from the Brown corpus and the Wall Street Journal and labeled with correct senses. Testing on an independent set of 14,139 examples from the Wall Street Journal gave an accuracy of 68.6% compared to an accuracy of 63.7% from choosing the most common sense, a standard baseline for comparison. Since WordNet is known for making fine sense distinctions, these results may seem somewhat low. For some easier problems the results were more impressive, such as disambiguating "interest" into one of 6 fairly fine-grained senses with an accuracy of 90%.

Decision-tree and rule induction have also been applied to sense disambiguation. Figure 1 shows results for disambiguating the word "line" into one of six senses using only binary features representing the presence or absence of all known words in the current and previous sentence (Mooney, 1996). Tree learning (C4.5), rule learning (PFOIL), and nearest neighbor perform comparably on this task and somewhat worse than simple neural network (Perceptron) and statistical (Naive Bayes) methods. A more recent project presents results on learning decision trees to disambiguate all content words in a financial corpus with an average accuracy of 77% (Paliouras, Karkaletsis, &

Spyropoulos, 1999).

20.6.4 Syntactic Parsing

Perhaps the most well-studied problem in computational linguistics is the syntactic analysis of sentences (see Chapters 4 and 12). In addition to statistical methods that have been successfully applied to this task, decision tree induction (Magerman, 1995; Hermjakob & Mooney, 1997; Haruno, Shirai, & Ooyama, 1999), rule induction (Brill, 1993), and instance-based categorization (Cardie, 1993; Argamon, Dagan, & Krymolowski, 1998) have also been successfully employed to learn syntactic parsers.

One of the first learning methods applied to parsing the Wall Street Journal (WSJ) corpus of the Penn treebank (Marcus, Santorini, & Marcinkiewicz, 1993) employed statistical decision trees (Magerman, 1995). Using a set of features describing the local syntactic context including the POS tags of nearby words and the node labels of neighboring (previously constructed) constituents, decision trees were induced for determining the next parsing operation. Instead of growing the tree to completely fit the training data, *pruning* was used to create leaves for subsets that still contained a mixture of classes. These leaves were then labeled with class-probability distributions estimated from the subset of the training data reaching that leaf. During testing, the system performs a search for the highest probability parse, where the probability of a parse is estimated by the product of the probabilities of its individual parsing actions (as determined by the decision tree). After training on approximately 40,000 WSJ sentences and testing on 1,920 additional ones, the system obtained a *labeled precision* (percentage of constructed constituents that were correct) and *labeled recall* (percentage of correct constituents that were found) of 84%.

20.6.5 Semantic Parsing

Learning methods have also been applied to mapping sentences directly into logical form (see Chapter 5) by inducing a parser from training examples consisting of sentences paired with semantic representations. Below is a sample training pair from an application involving English queries about a database of U.S. geography:

> What is the capital of the state with the highest population?
>
> `answer(C, (capital(S,C), largest(P, (state(S), population(S,P))))).`

Unfortunately, since constructing useful semantic representations for sentences is very difficult unless restricted to a fairly specific application, there is a noticeable lack of large corpora annotated with detailed semantic representations.

However, ILP has been used to induce domain-specific semantic parsers written in Prolog from examples of natural-language questions paired with logical Prolog queries (Zelle & Mooney, 1996; Ng & Zelle, 1997). In this project, parser induction is treated as a problem of learning rules to control the actions of a generic shift-reduce parser. During parsing, the current context is maintained in a stack and a buffer containing the remaining input. When parsing is complete, the stack contains the representation of the input sentence. There are three types of operators used to construct logical forms. One is the introduction onto the stack of a predicate needed in the sentence representation due to the appearance a word or phrase at the front of the input buffer. A second type of operator unifies variables appearing in different stack items. Finally, an item on the stack may be embedded as an argument of another one. ILP is used to learn conditions under which each of these operators should be applied, using the complete stack and input buffer as context, so that the resulting parser deterministically produces the correct semantic output for all of the training examples.

Posting from Newsgroup

```
Telecommunications. SOLARIS Systems Administrator. 38-44K. Immediate need

Leading telecommunications firm in need of an energetic individual to fill the
following position in the Atlanta office:

  SOLARIS SYSTEMS ADMINISTRATOR
  Salary: 38-44K with full benefits
  Location: Atlanta Georgia, no relocation assistance provided
```

Filled Template

```
computer_science_job
title: SOLARIS Systems Administrator
salary: 38-44K
state: Georgia
city: Atlanta
platform: SOLARIS
area: telecommunications
```

Figure 8: Information Extraction Example

This technique has been used to induce natural-language interfaces to several database-query systems, such as the U.S. geography application illustrated above. In one experiment using a corpus of 250 queries annotated with logical form, after training on 225 examples, the system was able to answer an average of 70% of novel queries correctly compared to 57% for an interface developed by a human programmer. Similar results were obtained for semantic parsing of other languages after translating the corpus into Spanish, Turkish, and Japanese (Thompson & Mooney, 1999).

20.6.6 Information Extraction

Information extraction is a form of shallow text processing that locates a specified set of relevant items in a natural-language document (see Chapter 30). Figure 8 shows an example of extracting values for a set of labeled slots from a job announcement posted to an Internet newsgroup. Information extraction systems require significant domain-specific knowledge and are time-consuming and difficult to build by hand, making them a good application for machine learning.

A number of rule-induction methods have recently been applied to learning patterns for in-

Pre-filler:	Filler:	Post-filler:
1) tag: {nn,nnp}	1) word: undisclosed	1) sem: price
2) list: length 2	tag: jj	

Figure 9: Sample Learned Information-Extraction Rule

formation extraction (Soderland, 1999; Freitag, 1998; Califf & Mooney, 1999). Given training examples of texts paired with filled templates, such as that shown in Figure 8, these systems learn pattern-matching rules for extracting the appropriate slot fillers from text. Some systems assume that the text has been preprocessed by a POS tagger or a syntactic parser, others use only patterns based on unprocessed text. Figure 9 shows a sample rule constructed for extracting the transaction amount from a newswire article about corporate acquisition (Califf & Mooney, 1999). This rule extracts the value "undisclosed" from phrases such as "sold to the bank for an undisclosed amount" or "paid Honeywell an undisclosed price". The pre-filler pattern consists of two pattern elements: 1) a word whose POS is noun or proper noun, and 2) a list of at most two unconstrained words. The filler pattern requires the word "undisclosed" tagged as an adjective. The post-filler pattern requires a word in the WordNet semantic category "price".

Such systems have acquired extraction rules for a variety of domains, including apartment ads, university web pages, seminar announcements, terrorist news stories, and job announcements. After training on a couple of hundred examples, such systems are generally able to learn rules as accurate as those resulting from a time-consuming human development effort. The standard metrics for evaluating information extraction are *precision*, the percentage of extracted fillers that are correct, and *recall*, the percentage of correct fillers that are successfully extracted. On most tasks that have been studied, current systems are generally able to achieve precisions in the mid 80% range and recalls in the mid 60% range.

20.6.7 Anaphora Resolution

Resolving anaphora, or identifying multiple phrases that refer to the same entity, is another difficult language-processing problem (see chapter 14). Anaphora resolution can be treated as a categorization problem by classifying pairs of phrases as either co-referring or not. Given a corpus of texts tagged with co-referring phrases, positive examples can be generated as all co-referring phrase pairs and negative examples as all phrase pairs within the same document that are not marked as co-referring. Both decision-tree (Aone & Bennett, 1995; McCarthy & Lehnert, 1995) and instance-based methods (Cardie, 1992) have been successfully applied to resolving various types of anaphora.

In particular, decision-tree induction has been used to construct systems for general noun-phrase co-reference resolution. Examples are described using features of both of the individual phrases, such as the semantic and syntactic category of the head noun; as well as features describing the relationship between the two phrases, such as whether the first phrase proceeds the second and whether the semantic class of the first phrase subsumes that of the second. In one experiment (Aone & Bennett, 1995), after training on 1,971 anaphora from 295 texts and testing on 1,359 anaphora from an additional 200 texts, the learned decision tree obtained a precision (percentage of co-references found that were correct) of 86.7% and a recall (percentage of true co-references that were found) of 69.7%. These results were superior to those obtained using a previous, hand-built co-reference procedure (precision 72.9%, recall 66.5%).

20.7 Further Reading and Relevant Resources

Introductory textbooks on machine learning include Mitchell (1997) and Langley (1996). Also useful are the collection of early papers assembled by Shavlik and Dietterich (1990). A recent special issue on natural language learning of the *Machine Learning* journal is presented by Cardie and Mooney (1999). Some websites with useful machine-learning pointers are:

http://www.aic.nrl.navy.mil/˜ aha/research/machine-learning.html and

http://www.ai.univie.ac.at/oefai/ml/ml-resources.html. Also see the Association of Computational Linguistics' special interest group on Natural Language Learning at http://ilk.kub.nl/˜ signll/.

References

Aha, D. W., Kibler, D. F., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, *6*(1), 37–66.

Aone, C., & Bennett, S. W. (1995). Evaluating automated and manual acquisition of anaphora resolution strategies. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95)*, pp. 122–129 Cambridge, MA.

Argamon, S., Dagan, I., & Krymolowski, Y. (1998). A memory-based approach to learning shallow natural language patterns. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and COLING-98 (ACL/COLING-98)*, pp. 67–73 Montreal, Quebec.

Bratko, I., & Muggleton, S. (1995). Applications of inductive logic programming. *Communications of the Association for Computing Machinery*, *38*(11), 65–70.

Brill, E. (1993). Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL-93)*, pp. 259–265 Columbus, Ohio.

Brill, E. (1995). Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, *21*(4), 543–565.

Califf, M. E., & Mooney, R. J. (1999). Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 328–334 Orlando, FL.

Cardie, C. (1992). Learning to disambiguate relative pronouns. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 38–43 San Jose, CA.

Cardie, C. (1993). A case-based apprach to knowledge acquisition for domain-specific sentence analysis. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 798–803 Washington, D.C.

Cardie, C., & Mooney, R. J. (1999). Machine learning and natural language (Introduction to special issue on natural language learning). *Machine Learning*, *34*, 5–9.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*, 261–284.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pp. 115–123 San Francisco, CA.

Cost, S., & Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, *10*(1), 57–78.

Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *13*, 21–27.

Daelemans, W., Zavrel, J., Berck, P., & Gillis, S. (1996). MBT: A memory-based part of speech tagger-generator. In *Proceedings of the Fourth Workshop on Very Large Corpora*, pp. 14–27. ACL SIGDAT.

Fayyad, U. M., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery. In Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., & Uthurusamy, R. (Eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 1–34. MIT Press, Cambridge, Mass.

Freitag, D. (1998). Toward general-purpose learning for information extraction. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and COLING-98 (ACL/COLING-98)*, pp. 404–408 Montreal, Quebec.

Haruno, M., Shirai, S., & Ooyama, Y. (1999). Using decision trees to construct a practical parser. *Machine Learning*, *34*, 131–150.

Hermjakob, U., & Mooney, R. J. (1997). Learning parse and translation decisions from examples with rich context. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL-97)*, pp. 482–489 Madrid, Spain.

Kazakov, D., & Manandhar, S. (1998). A hybrid approach to word segmentation. In *Proceedings of the 9th International Workshop on Inductive Logic Programming (ILP-99)*, pp. 125–134. Springer.

Langley, P. (1996). *Elements of Machine Learning.* Morgan Kaufmann, San Francisco, CA.

Lavrac, N., & Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood.

Ling, C. X. (1994). Learning the past tense of English verbs: The symbolic pattern associator vs. connectionist models. *Journal of Artificial Intelligence Research*, *1*, 209–229.

Ling, C. X., & Marinov, M. (1993). Answering the connectionist challenge: A symbolic model of learning the past tense of English verbs. *Cognition*, *49*(3), 235–290.

MacWhinney, B., & Leinbach, J. (1991). Implementations are not conceptualizations: Revising the verb model. *Cognition*, *40*, 291–296.

Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95)*, pp. 276–283 Cambridge, MA.

Manandhar, S., Dzeroski, S., & Erjavec, T. (1998). Learning multilingual morphology with CLOG. In *Inductive Logic Programming: Proceedings of the 8th International Conference (ILP-98)*, pp. 135–144. Springer.

Marcus, M., Santorini, B., & Marcinkiewicz, M. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, *19*(2), 313–330.

Marquez, L., Padro, L., & Rodriguez, H. (1999). A machine learning approach to POS tagging. *Machine Learning*, *39*(1), 59–91.

McCarthy, J., & Lehnert, W. (1995). Using decision trees for coreference resolution. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 1050–1055.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill, New York, NY.

Mooney, R. J. (1996). Comparative experiments on disambiguating word senses: An illustration of the role of bias in machine learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-96)*, pp. 82–91 Philadelphia, PA.

Mooney, R. J., & Califf, M. E. (1995). Induction of first-order decision lists: Results on learning the past tense of English verbs. *Journal of Artificial Intelligence Research*, *3*, 1–24.

Ng, H. T., & Lee, H. B. (1996). Integrating multiple knowledge sources to disambiguate word sense: An exemplar-based approach. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pp. 40–47 Santa Cruz, CA.

Ng, H. T., & Zelle, J. (1997). Corpus-based approaches to semantic interpretation in natural language processing. *AI Magazine*, *18*(4), 45–64.

Paliouras, G., Karkaletsis, V., & Spyropoulos, C. D. (1999). Learning rules for large vocabulary word sense disambiguation. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 674–679 Stockholm, Sweden.

Pinker, S., & Prince, A. (1988). On language and connectionism: Analysis of a parallel distributed model of language acquisition. In Pinker, S., & Mehler, J. A. (Eds.), *Connections and Symbols*, pp. 73–193. MIT Press, Cambridge, MA.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.

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

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo,CA.

Quinlan, J. R. (1996). Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pp. 725–730 Portland, OR.

Rumelhart, D. E., & McClelland, J. L. (1986). On learning the past tense of English verbs. In Rumelhart, D. E., & McClelland, J. L. (Eds.), *Parallel Distributed Processing, Vol. II*, pp. 216–271. MIT Press, Cambridge, MA.

Shavlik, J. W., & Dietterich, T. G. (Eds.). (1990). *Readings in Machine Learning.* Morgan Kaufmann, San Mateo,CA.

Soderland, S. (1999). Learning information extraction rules for semi-structured and free text. *Machine Learning*, *34*, 233–272.

Stanfill, C., & Waltz, D. L. (1986). Toward memory-based reasoning. *Communications of the Association for Computing Machinery*, *29*, 1213–1228.

Thompson, C. A., & Mooney, R. J. (1999). Automatic construction of semantic lexicons for learning natural language interfaces. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 487–493 Orlando, FL.

Zelle, J. M., & Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pp. 1050–1055 Portland, OR.