Machine Learning, , 1–44 ()

Learning Information Extraction Rules for Semi-structured and Free Text

STEPHEN SODERLAND soderlan@cs.washington.edu
Dept. Computer Science & Engineering, University of Washington, Seattle, WA 98195-2350

Editor: Claire Cardie and Raymond Mooney

Abstract. A wealth of on-line text information can be made available to automatic processing by information extraction (IE) systems. Each IE application needs a separate set of rules tuned to the domain and writing style. WHISK helps to overcome this knowledge-engineering bottleneck by learning text extraction rules automatically.

WHISK is designed to handle text styles ranging from highly structured to free text, including text that is neither rigidly formatted nor composed of grammatical sentences. Such semi-structured text has largely been beyond the scope of previous systems. When used in conjunction with a syntactic analyzer and semantic tagging, WHISK can also handle extraction from free text such as news stories.

Keywords: natural language processing, information extraction, rule learning

1. Information extraction

As more and more text becomes available on-line, there is a growing need for systems that extract information automatically from text data. An information extraction (IE) system can serve as a front end for high precision information retrieval or text routing, as a first step in knowledge discovery systems that look for trends in massive amounts of text data, or as input to an intelligent agent whose actions depend on understanding the content of text-based information.

IE systems have been developed for writing styles ranging from structured text with tabular information to free text such as news stories. A key element of such systems is a set of text extraction rules that identify relevant information to be extracted.

For structured text, the rules specify a fixed order of relevant information and the labels or HTML tags that delimit strings to be extracted. For free text, an IE system needs several steps in addition to text extraction rules. These include syntactic analysis, semantic tagging, recognizers for domain objects such as person and company names, and discourse processing that makes inferences across sentence boundaries. Extraction rules for free text are typically based on patterns involving syntactic relations between words or semantic classes of words.

1.1. Semi-structured text

A useful class of text that falls between these extremes has been largely inaccessible to IE systems. Such semi-structured text[1] is ungrammatical and often telegraphic in style, but does not follow any rigid format. Examples are certain medical records, equipment maintenance logs, and much useful information on the World Wide Web such as the apartment rental ad shown in Figure 1, which will serve as a running example of semi-structured text.

```
Capitol Hill - 1 br twnhme. fplc D/W W/D. Undrgrnd pkg
incl $675. 3 BR, upper flr of turn of ctry HOME. incl gar,
grt N. Hill loc $995. (206) 999-9999 <br>
<i> <font size=-2> (This ad last ran on 08/03/97.)
</font> </i> <hr>
```

Figure 1. An example of semi-structured text from an on-line rental ad

Tucked away among the telegraphic abbreviations is information about two rental units: a 1 bedroom unit for $675 in the Capitol Hill neighborhood, and a 3 bedroom unit for $995 in the same neighborhood. This is represented by the two case frames shown in Figure 2.

```
Rental:
    Neighborhood:   Capitol Hill
    Bedrooms:       1
    Price:          675

Rental:
    Neighborhood:   Capitol Hill
    Bedrooms:       3
    Price:          995
```

Figure 2. Output from the rental ad is two case frames

The methods appropriate for grammatical text will not work for the example in Figure 1. A parser would look in vain for subjects, verbs, or objects in this type of telegraphic text. Neither would the simple rules used for rigidly structured text be adequate.

1.2. Free text

A typical example of extraction from free text is shown in the sentence in Figure 3. This is from the "Management Succession" domain used in the Sixth Message Understanding Conference (MUC-6 1995), one of a series of ARPA-sponsored conferences that has promoted research in free text IE. The relevant information for this domain are corporate officers moving into a new position (PersonIn) or leaving a position (PersonOut), the corporation name (Org), and position title (Post).

Input text:
C. Vincent Protho, chairman and chief executive officer of
this maker of semiconductors, was named to the additional post
of president, succeeding John W. Smith, who resigned to
pursue other interests.

Succession event

PersonIn:	C. Vincent Protho
PersonOut:	John W. Smith
Post:	president

Figure 3. Management Succession domain: An input sentence and the output case frame

Subtle inferences are required to extract information from free text. Domain objects such as person names, company names, and positions must be identified. The syntactic relationship between these objects determines the role each plays, if any, in an output case frame. A domain object may participate in multiple case frames. Consider the sentence "Mr. A, former president of X Corp., was named chairman and CEO of Y Corp., succeeding Mr. B, who became president of Z Corp." Mr. A is PersonOut as president of X and in another case frame is PersonIn as chairman and CEO of Y. Mr. B is PersonOut as chairman and CEO of Y and PersonIn as president of Z.

Figure 4 shows the sample input text after syntactic analysis and automatic tagging of person names, company names, and corporate posts. This example was processed by the University of Massachusetts BADGER syntactic analyzer (Fisher *et al.* 1995). The sentence is segmented into subject, verb, prepositional phrase, and an ad hoc field REL_V for the relative clause attached to the verb. Verb roots are inserted after each head verb, marked with the prefix "@". Person names are bracketed by "@PN[...]PN", posts by "@PS[...]PS", and company names by "@CN[...]CN" (none found in this example).

The full MUC-6 task involves merging information across sentences, including resolving pronouns and generic references (e.g. "this maker of semiconductors") to actual person or company names. In this paper, we focus only on the step of extracting information at the sentence level.

1.3. Single-slot vs. multi-slot extraction

A characteristic of IE systems that cuts across writing genres is whether the system extracts isolated facts from the text or links together related information into multi-slot case frames. The examples given thus far are of multi-slot extraction. The

```
@S[
  {SUBJ   @PN[ C. Vincent Protho ]PN , @PS[chairman and chief
          executive officer ]PS of this maker of semiconductors, }
  {VB     @Passive was named @nam }
  {PP     to the additional post of @PS[ president ]PS , }
  {REL_V  succeeding @succeed @PN[ John W. Smith ]PN ,
          who resigned @resign to pursue @pursu other interests. }
]@S 8910130051-1
```

Figure 4. Syntactically analyzed input sentence

sample rental ad has a price of $675 associated with a 1 bedroom unit and a price of $995 associated with a 3 bedroom unit. Vital information is lost if the system simply reports Price=675, Price=995, Bedrooms=1, Bedrooms=3 without showing which price and bedrooms are related.

In the Management Succession domain multiple events are often mentioned in a single sentence. "Mr. Adams, former president of X Corp., was named CEO of Y Inc." The output from this sentence is the two case frames shown in Figure 5. A system that is only capable of single-slot extraction would report this as PersonOut=Adams, PersonIn=Adams, Post=president, Post=CEO, Org=X Corp., Org=Y Inc. This has limited utility unless a later module assembles these isolated facts into the proper events.

```
Succession event
    PersonOut:   Adams
    Post:        president
    Org:         X Corp.

Succession event
    PersonIn:    Adams
    Post:        CEO
    Org:         Y Inc.
```

Figure 5. Multi-slot extraction is vital when multiple case frames are output for a single sentence.

There are some domains where multi-slot extraction is essential. A web page may list dozens of product names with associated prices, descriptions, and URL pointing to further information. Unless the name, price, description, and URL are reported as a set, the output is a useless jumble.

In other domains, however, single-slot extraction is perfectly adequate. If there is always no more than one event per text, slots can be identified separately, and then all extractions merged into a single case frame for the text. This is the case for two domains discussed later in this paper, a collection of on-line seminar announcements and a collection of on-line job listings. All speakers, locations, start-times and end-times from a seminar announcement can be extracted separately, since they (presumably) all refer to a single seminar. Similarly a variety of slots related to the job listings can be extracted separately, then merged as slots of a single case frame for each document.

Even when a system is capable of multi-slot extraction, related information may be widely scattered throughout a document, particularly in the case of free text. In such cases the slots will be extracted by rules that operate locally in the text and later processing is needed to link together the related information, which is not an easy problem.

1.4. WHISK and other IE machine learning systems

IE systems require a separate set of rules for each domain, whether extracting from structured, semi-structured, or free text. This makes machine learning an attractive option for knowledge acquisition.

Information extraction is a relatively new field, and there have been few systems that apply machine learning to the task. These systems differ in how the IE problem is characterized and in the style of text that they handle. The chart shown in Table 1 lists the capabilities of existing systems and compares them with WHISK, the system presented in this paper. These systems are described in more detail later.

Some systems operate only on rigidly structured text such as Wrapper Induction (Kushmerick *et al.* 1997) (WI in Table 1), or on semi-structured text such as RAPIER (Califf and Mooney 1997) or SRV (Freitag 1998). Wrapper Induction learns rules to extract multiple slots of a case frame at once, but SRV and RAPIER extract single slots in isolation.

The next systems shown in Table 1 can handle free text such as news stories: AutoSlog (Riloff 1993), CRYSTAL (Soderland *et al.* 1995, Soderland 1997), and LIEP (Huffman 1996). AutoSlog does single-slot extraction, LIEP does *only* multi-slot extraction, and CRYSTAL handles either. Each of these systems requires syntactic preprocessing of the text and semantic tagging. Given preprocessed input, CRYSTAL can be extended to handle semi-structured text (Soderland 1997a).

LIEP identifies the desired phrase exactly, while AutoSlog and CRYSTAL do not identify the target phrase directly, but identify the syntactic field that *contains* the target phrase. Because of this AutoSlog and CRYSTAL require later processing to trim extraneous words from the extracted phrase. For example, AutoSlog or CRYSTAL would report that a PersonIn is to be found somewhere in the subject field, "C. Vincent Protho, chairman and chief executive officer of this maker of semiconductors," without specifying which phrase is the actual PersonIn.

WHISK is a new system with broader capabilities than the previous systems. WHISK learns rules in the form of regular expressions that can extract either single slots or multiple slots. This means that, unlike other systems designed for semi-structured text, WHISK learns to identify the relationship between isolated

Table 1. A comparison of WHISK with other systems that learn IE rules

System name	Text style	Multi-slot	Need syntax	Need trim
WI	struct	yes	no	no
SRV	semi	no	no	no
RAPIER	semi	no	no	no
AutoSlog	free	no	yes	yes
CRYSTAL	free	yes	yes	yes
CRYSTAL	semi	yes	yes	yes
LIEP	free	only	yes	no
WHISK	struct	yes	no	no
WHISK	semi	yes	no	no
WHISK	free	yes	yes	no

facts. When applied to structured or semi-structured text WHISK does not require prior syntactic analysis. For free text such as news stories, WHISK works best with input has been annotated by a syntactic analyzer and semantic tagger.

WHISK's rules learn not only the context that makes a phrase or set of phrases relevant to a domain, but also learns the exact delimiters of those phrases. This goes beyond some of the other systems that handle free text, since WHISK requires no later step to trim away extraneous words. Like all modules that perform sentence-level extraction, this does not handle the full job for most free text IE tasks, and leaves co-reference resolution and merging across sentences to later discourse modules.

Performance of WHISK is comparable to other IE rule learners. For structured or semi-structured text, high or even perfect accuracy can often be achieved from a modest amount of training examples. The state-of-the-art in information extraction from free text falls considerably short of human capability, but still provides useful results. This is true whether the rules are hand coded or automatically learned.

WHISK uses supervised learning, inducing a set of rules from hand-tagged training examples. To minimize the human effort involved, WHISK interleaves the learning and the annotation of training data. Rather than presenting arbitrary examples to be hand-tagged, WHISK presents instances that are likely to be near the decision boundaries of the rules generated so far. This process is iterated as rules are further refined with new examples.

1.5. Organization of the paper

The next section presents WHISK's rule representation and the implementation of its regular expression matching. The WHISK algorithm is then described, followed by a discussion of WHISK's strategy for minimizing the number of training examples to be hand-tagged. The following section gives empirical results for three classes of text: structured, semi-structured, and free text. Related work and conclusions are then discussed.

ID:: 1
Pattern:: * (*Digit*) ' BR' * '$' (*Number*)
Output:: Rental {Bedrooms $1} {Price $2}

Figure 6. A WHISK rule to extract number of bedrooms and price

2. WHISK rule representation

This section begins by discussing rules to handle structured and semi-structured text, then introduce extensions that handle free text.

2.1. Rules for structured and semi-structured text

WHISK rules are based on a form of regular expression patterns that identify the context of relevant phrases and the exact delimiters of those phrases. A rule that applies to the Rental Ad example from Figure 1 is shown in Figure 6.

This rule looks for number of bedrooms and associated price. The wildcard "*" means to skip any number of characters until the *next* occurrence of the following term in the pattern. In this case the pattern skips until it reaches the first digit, which is "1" in our example. Tokens within single quotes indicate a literal to be matched exactly, without regard to case. The token *Digit* matches a single digit, the token *Number* matches a possibly multi-digit number.

Parentheses, unless within single quotes, indicate a phrase to be extracted. The phrase within the first set of parentheses is bound to the variable $1 in the output portion of the rule, the second to the variable $2, and so forth, much as in the Perl pattern matching language. If the entire pattern matches, a case frame is created with slots filled as labeled in the output portion of the rule.

If part of the input remains after a rule has succeeded, the rule is re-applied starting from the last character matched by the prior application of the rule. This single rule will extract the following two case frames from the example text, as shown in Figure 7.

A major difference between WHISK rules and true regular expressions is that the wildcard "*" is limited to enforce proximity between matched tokens. In Figure 6 the wildcard between ' BR' and '$' does not match all possible strings between those literals. That would associate the *Bedroom* slot with every following dollar amount in the text, rather than taking only the nearest as the *Price* associated with that *Bedroom* slot. Allowing an unlimited number of matches on each wildcard would also make the pattern matching unacceptably slow for an on-line system, as multiple wildcards would be applied in an exponential number of combinations.

As each term of the pattern is matched to a portion of the input text, the matched characters are consumed. If the entire pattern has been matched successfully and a portion of the input remains, the rule is re-applied to the unconsumed portion.

```
Rental:
    Bedrooms:     1
    Price:        675

Rental:
    Bedrooms:     3
    Price:        995
```

Figure 7. Output from the WHISK rule for *Bedroom* and *Price*

If a pattern fails, phrases that have been bound to an output slot are retained and the rule restarts after that point on the remaining input[2]. This restriction on unlimited backtracking serves to prevent an extracted slot from being associated with an unrelated slot later in the text.

WHISK rules also allow a form of disjunction. The user may define a semantic class, which is simply a set of terms that are considered to be an equivalence class. Class names are displayed in this paper as italicized words not enclosed by quotes. *Digit* and *Number* are special semantic classes that are hard-wired into WHISK.

In the Rental Ad domain the semantic class *Bdrm* has been defined as the disjunction:

Bdrm = (brs|br|bds|bdrm|bd|bedrooms|bedroom|bed)

Such ad hoc lists of functionally equivalent terms can be created by an naive user and need not be complete or perfectly correct. They simply help WHISK generalize rules beyond the combination of tokens found in a limited training set. Such semantic classes could be learned, but it takes a user less effort to create a list of terms than to hand tag a set of training examples that use all of the terms. In some of the experiments reported in this paper, semantic classes are presented to WHISK as part of WHISK's input.

The rule in Figure 8 shows this semantic class as well as *Nghbr*, a list of neighborhood names for the city from which the training data was taken. This rule extracts the first case frame shown in Figure 2, "Capitol Hill", "1" bedroom, and price "675".

ID:: 2
Pattern:: * (*Nghbr*) * (*Digit*) ' ' *Bdrm* * '$' (*Number*)
Output:: Rental {Neighborhood $1} {Bedrooms $2} {Price $3}

Figure 8. A rule using the semantic classes *Nghbr* and *Bdrm*

2.2. Extensions of the rules for grammatical text

The rule language presented so far is adequate for structured or semi-structured text. When applying WHISK to grammatical text, it makes sense to take advantage of the clausal structure of sentences and any other information that can be supplied by a syntactic analyzer.

The domain used to illustrate free text is the Management Succession domain from the Sixth Message Understanding Conference (MUC-6 1995). This is a collection of Wall Street Journal articles in which the relevant information is the relationship of persons moving into a top corporate position, persons moving out of that position, the position, and the company name. The output is represented as a case frame with four slots: PersonIn, PersonOut, Post, and Org. This is a simplification of the case frames used in MUC-6.

An example was given in Figure 3 in which "C. Vincent Protho, chairman and chief executive officer of this maker of semiconductors, was named to the additional post of president, succeeding John W. Smith, who resigned to pursue other interests." A syntactic analysis of this sentence is repeated here in Figure 9.

This is the type of input given to WHISK for this domain rather than the text itself. WHISK rules are applied to such *instances* rather than to the text directly. For some domains, such as the Rental Ad domain, there is minimal pre-processing to create instances. A web page of on-line rental ads was turned into a set of instances by segmenting the text on the HTML tag <hr> that serves as a delimiter between ads. No other modifications were done to the text to create instances. In other domains of structured or semi-structured text, the entire text may become a single instance. For free text, an instance is generally a clause or sentence as determined by a syntactic analyzer.

```
@S[
  {SUBJ   @PN[ C. Vincent Protho ]PN , @PS[chairman and chief
          executive officer ]PS of this maker of semiconductors, }
  {VB     @Passive was named @nam }
  {PP     to the additional post of @PS[ president ]PS , }
  {REL_V  succeeding @succeed @PN[ John W. Smith ]PN ,
          who resigned @resign to pursue @pursu other interests. }
]@S 8910130051-1
```

Figure 9. A Management Succession instance after syntactic analysis

Segmentation of the text into syntactic fields is indicated by curly brackets together with the field name, such as "{SUBJ ... }" for the subject of a clause. WHISK will accept whatever field names the syntactic analyzer provides, even ad hoc ones such as "{REL_V ... }" for a relative clause attached to a verb. Additional syntactic information may be included such as marking passive voice or supplying verb stems, which are annotated with the prefix "@".

For some domains, a prescribed set of terms is not sufficient for semantic classes. Special company name recognizers and person name recognizers were applied to Management Succession texts, as well as delimiting multi-word corporate positions. Explicit semantic tags are added to the text. Phrases with the class *Person* are marked with @PN[...]PN, *Corp* with @CN[...]CN and *Position* with @PS[...]PS.

It is useful in some cases for a rule to require that two terms of a pattern be found in the same syntactic field. WHISK rules distinguish between two wildcards: "*" means skip zero or more characters even if this spans fields; "*F" is restricted to skip only within the same syntactic field. As described earlier, the wildcard causes WHISK to skip until the next term of the pattern is matched.

The WHISK rule in Figure 10 illustrates several of these features. This is one of many possible rules that extract PersonIn, Post, and PersonOut from the instance in Figure 9.

```
ID:: 3
Pattern::  * ( Person ) * '@Passive' *F 'named' * {PP *F ( Position )
           * '@succeed ' ( Person )
Output:: Succession {PersonIn $1} {Post $2} {PersonOut $3}
```

Figure 10. A rule using syntactic and semantic tags

This pattern's first "*" means to skip over any input characters including field boundaries until the first person name is found. *Person* is shorthand for the pattern "@PN[*F]PN". The pattern skips past field boundaries again looking for the token "@Passive". The next wildcard is "*F" so the term "named" is required to be in the same field as the passive tag. The pattern matching continues, finding a prepositional phrase ("{PP") containing a position and later the verb root "@succeed" immediately followed by a person.

WHISK rules are free to use the syntactic tags or not, as needed. The above rule could have included the field tag "{VB" before the term "@Passive", but this would not have changed the discrimination power of the rule, since "@Passive" is only found in verb fields.

3. The WHISK algorithm

Having looked at WHISK's rule representation, we turn to the algorithm for inducing such rules automatically from training examples. WHISK is a supervised learning algorithm and requires a set of hand-tagged training instances. The tagging process is interleaved with the learning as described later. In each iteration WHISK presents the user with a batch of instances to tag, then WHISK induces a set of rules from the expanded training set.

3.1. Creating hand-tagged training instances

As mentioned in the previous section, what constitutes an instance depends on the domain and what pre-processing is done to the text before it is presented to WHISK. In some domains an entire text constitutes an instance with no alteration. For some domains of structured or semi-structured text, a text is broken into multiple instances based on HTML tags or other regular expressions. For free text domains, a sentence analyzer segments the text into instances where each instance is a clause, sentence or sentence fragment. Other pre-processing may be done to instances before passing them to WHISK, such as automatically adding semantic tags or other syntactic annotations.

WHISK begins with a reservoir of untagged instances and an empty training set of tagged instances. At each iteration of WHISK a set of untagged instances are selected from the reservoir and presented to the user to annotate. Details of how such instances are selected is discussed later.

The user adds a tag for each case frame to be extracted from the instance. If the case frame has multiple slots, the tag will likewise be multi-slot. Some of the "tagged" instances will actually have no tags if the user has determined that the instance contains no relevant information.

Instances that have been tagged by the user are referred to in this paper as *training instances* and collectively as the *training set*. This is to distinguish training instances from the set of untagged instances from which the training instances are drawn. The tags of training instances are used by WHISK to guide creation of rules and also to test the performance of proposed rules. If a rule is applied successfully to an instance, the instance is considered to be *covered* by the rule. If the extracted phrases exactly match a tag associated with the instance, it is considered a correct extraction, otherwise as an error.

Figure 1 showed an instance from the Rental Ad domain before it was tagged. This instance is shown in Figure 11 after a user has hand-tagged the case frames to be extracted. Each of the lines beginning with "@@TAGS" at the end of the instance represents a case frame with an ordered list of slots to extract, each enclosed in curly brackets. The instance itself is delimited by "@S[" and "]@S".

These tags are shown in a human readable format. A graphical user interface would display tags as highlighted text and record them internally as offsets within the text, for example character 0 through 11 for the Neighborhood slot and so forth.

3.2. Creating a rule from a seed instance

The top level of the WHISK algorithm is shown in Figure 12. At each iteration the user tags a batch of instances from the Reservoir of untagged instances. Some of these new training instances may be counterexamples to existing rules, in which case the rule is discarded so that a new rule may be grown. WHISK then selects an instance-tag pair for which the tag is not extracted from the instance by any rule in the RuleSet. This instance-tag pair becomes a "seed" to grow a new rule that covers the seed (extracts phrases matching the tag from the seed instance).

```
@S[
   Capitol Hill - 1 br twnhme. fplc D/W W/D. Undrgrnd pkg
   incl $675. 3 BR, upper flr of turn of ctry HOME. incl gar,
   grt N. Hill loc $995. (206) 999-9999 <br>
   <i> <font size=-2> (This ad last ran on 08/03/97.)
   </font> </i> <hr>
]@S 5
@@TAGS Rental {Neighborhood Capitol Hill} {Bedrooms 1} {Price 675}
@@TAGS Rental {Neighborhood Capitol Hill} {Bedrooms 3} {Price 995}
```

Figure 11. A training instance hand-tagged with two extractions

```
WHISK(Reservoir)
  RuleSet = NULL
  Training = NULL
  Repeat at user's request
    Select a batch of NewInst from Reservoir
    (User tags the NewInst)
    Add NewInst to Training
    Discard rules with errors on NewInst
    For each Inst in Training
      For each Tag of Inst
        If Tag is not covered by RuleSet
          Rule = GROW_RULE(Inst, Tag, Training)
  Prune RuleSet
```

Figure 12. Top level of the WHISK algorithm

WHISK induces rules top-down, first finding the most general rule that covers the seed, then extending the rule by adding terms one at a time. The metric used to select a new term is the Laplacian expected error of the rule, given by the following formula, where n is the number of extractions made on the training set and e is the number of errors among those extractions.

$$Laplacian = \frac{e+1}{n+1}$$

3.2.1. Anchoring the extraction slots This gives an estimate of the true error of a rule that is sensitive to the amount of support it has in the training set. For alternate rules with the same number of training errors, the Laplacian will be lowest for the rule with highest coverage. This version of the Laplacian formula gives an expected error rate of 0.5 for a rule that covers a single tag with no error[3].

Top-down rule induction typically begins with an "empty" rule that covers all instances, then adds terms to the rule, which reduces the number of instances covered monotonically. Unfortunately, things are not so simple with WHISK's information extraction rules.

Counter-intuitive as it may seem, the empty WHISK rule extracts nothing correctly, and adding terms to a rule may in some cases increase its coverage. This is an artifact of WHISK's restricted regular expression language. The empty rule for three slot extraction would be the following.

Empty rule: " * (*) * (*) * (*) * "

In WHISK's rule language the first wildcard of this empty rule means to skip until the following term is matched. The result is that the first wildcard matches the empty string and the next wildcard matches the entire input instance. This empty rule will make no correct extractions on any training instances. Before a rule can hope to cover any correct extractions, the boundaries of each extracted slot must be "anchored" with non-wildcard terms.

WHISK grows a rule from a seed, by starting with an empty rule and anchoring the extraction boundaries one slot at a time. To anchor an extraction, WHISK considers a rule with terms added just within the extraction boundary (Base_1) and a rule with terms added just outside the extraction (Base_2). It will often happen that these base rules are not constrained enough to make any correct extractions, in which case more terms are added until the rule at least covers the seed. The base rule is selected that covers the greatest number of positive instances among the hand-tagged training set.

As a concrete example, assume the seed is the instance shown in Figure 11 together with the first tag. The semantic class *Nghbr* matches the first and only term of slot 1, so this slot can be anchored from within by the pattern " * (*Nghbr*) ". The terms just outside this slot are the start of sentence and the space and hyphen that follow the neighborhood. This gives a second possible pattern " '@start' (*) ' -' ".

Each of base rule for slot 1 operates correctly on this instance. Base_1 skips until it matches a string of the class *Nghbr*, which it binds to the first slot. Base_2 extracts a phrase from the start of the instance until the first space followed by a hyphen. Base_1 applies correctly to more instances in the training set, so it is selected as the anchor for slot 1.

WHISK then considers two alternatives for extending the rule to cover slot 2. Base_1 for this slot uses the class *Digit* which matches the first and only term of slot 2. Base_2 uses the term just before this slot, '- ', and the term just after it, ' br'. Each of these operates correctly on the seed instance. Base_1 looks for the first digit after the first neighborhood. Base_2 looks for the first '- ' after the first neighborhood and extracts all characters up to the next ' br' as the number of Bedrooms. The process is then continued for slot 3.

The final anchored rule operates correctly on the seed instance, but may makes some extraction errors on other training instances. WHISK then continues adding terms to extend the rule until the rule operates reliably on the training set. Figure 14 shows the procedure growing a rule from seed.

Anchoring Slot 1:
  Base_1: * (*Nghbr*)
  Base_2: '@start' (*) ' -'

Anchoring Slot 2:
  Base_1: * (*Nghbr*) * (*Digit*)
  Base_2: * (*Nghbr*) * '- ' (*) ' br'

Anchoring Slot 3:
  Base_1: * (*Nghbr*) * (*Digit*) * (*Number*)
  Base_2: * (*Nghbr*) * (*Digit*) * '$' (*) '.'

Figure 13. Base rules anchoring each of slot of a three-slot rule

3.2.2. Adding terms to a proposed rule WHISK extends a rule by considering each term that could be added and testing the performance of each proposed extension on the hand-tagged training set. Since the new rule must apply to the seed instance, only terms from this instance need to be considered in growing the rule. If a term from the instance belongs to a user-defined semantic class or has been annotated by a semantic tagger, WHISK tries adding either the term itself or its semantic class to the rule.

Each word, number, punctuation, or HTML tag in the instance is considered a term. It is not self-evident whether some strings should be treated as a single term or as multiple terms. In a web page, should "<font size=+1>" be a single term? The current implementation of WHISK breaks it into separate tokens on the embedded blank and likewise treats the "=" and "+" as separate terms.

Line breaks are ignored in free text but contain useful information in some portions of semi-structured text. The current implementation of WHISK adds the token "@blankline" for blank lines in semi-structured text and also adds "@newline" if a line begins with a heading followed by a colon or begins with indentation.

Figure 15 shows the procedure to extend a rule. Each proposed extension of a rule is tested on the training set. The proposed rule with lowest Laplacian expected error is selected as the next version of the rule until the rule either makes no errors or until the Laplacian is below a threshold and none of the extensions reduce the Laplacian.

In a case where several proposed rules have the same Laplacian, WHISK uses heuristics that prefer the semantic class of a word, and that prefer syntactic tags over literals. The rationale behind these is to prefer the least restrictive rule that fits the data – a semantic class is more general than a member of that class, and syntactic field names tend to cover more instances than literals.

For domains with no semantic or syntactic tags, WHISK prefers terms near extraction boundaries. A stronger version of this heuristic is appropriate for some domains. WHISK can be given a window size of k tokens and only consider terms

```
GROW_RULE(Inst, Tag, Training)
    Rule = empty rule (terms replaced by wildcards)
    For i = 1 to number of slots in Tag
        ANCHOR(Rule, Inst, Tag, Training, i)
    Do until Rule makes no errors on Training or
            no improvement in Laplacian
        EXTEND_RULE(Rule, Inst, Tag, Training)

ANCHOR(Rule, Inst, Tag, Training, i)
    Base_1 = Rule + terms just within extraction i
    Test first i slots of Base_1 on Training
    While Base_1 does not cover Tag
        EXTEND_RULE(Base_1, Inst, Tag, Training)
    Base_2 = Rule + terms just outside extraction i
    Test first i slots of Base_2 on Training
    While Base_2 does not cover Tag
        EXTEND_RULE(Base_2, Inst, Tag, Training)
    Rule = Base_1
    If Base_2 covers more of Training than Base_1
        Rule = Base_2
```

Figure 14. The WHISK procedure for growing rules from seed

within k of an extraction slot. This can improve precision for some domains and speed up the algorithm, which takes time proportional to the number of features per instance.

3.3. *Hill climbing and horizon effects*

WHISK does a form of hill climbing and cannot guarantee that the rules it grows are optimal, where optimal is defined as having the lowest Laplacian expected error on the hand-tagged training instances. Terms are added and evaluated one at a time. It may happen that adding two terms would create a reliable rule with high coverage and few errors, but adding either term alone does not constrain the rule from making extraction errors.

If WHISK makes a “wrong” choice of terms to add, it may miss a reliable, high-coverage rule, but will continue adding terms until the rule operates reliably on the training set. Such a rule will be more restrictive than the optimal rule and will tend to have lower coverage on unseen instances.

```
EXTEND_RULE(Rule, Inst, Tag, Training)
    Best_Rule = NULL
    Best_L = 1.0
    If Laplacian of Rule within error tolerance
        Best_Rule = Rule
        Best_L = Laplacian of Rule
    For each Term in Inst
        Proposed = Rule + Term
        Test Proposed on Training
        If Laplacian of Proposed < Best_L
            Best_Rule = Proposed
            Best_L = Laplacian of Proposed
    Rule = Best_Rule
```

Figure 15. The WHISK procedure to extend a rule

3.4. Pre-pruning and post-pruning the rules

Continuing to extend a rule until errors on the training set have been reduced to zero may be too strict a stopping condition. A rule with terms added until it operates perfectly on the training set may overfit the training. In this case performance on previously unseen instances will be improved by pruning away the last term or terms. Stopping the induction while the rule still makes a small percentage of errors on the training is a form of pre-pruning. WHISK uses a threshold for the Laplacian error rate as a guide for when to stop expanding a rule.

For example, with the error tolerance threshold is set to 0.10, a rule that applies 20 times with 1 error (L=0.095) will be accepted unless an extension is found that covers 10 or more with 0 errors (L=0.091). If the best extension has coverage of only 5 with 0 errors (L=0.167) this is not considered a more reliable rule and WHISK keeps the rule with coverage 20 instead.

When the entire rule set has been generated, some of the rules will have low coverage on the training set. A post-pruning step that discards all rules with Laplacian expected error greater than a threshold, has the effect of removing the least reliable rules.

Other methods of post-pruning could be explored. A form of reduced error pruning using a separate validation set might do a better job of optimizing performance metrics. This would assess the global effect of pruning terms from a rule or discarding a rule. The Laplacian only measures each rule in isolation and ignores the effect of overlapping coverage. A drawback of reduced error pruning is that it requires a larger number of tagged training instances. WHISK's scheme for minimizing the number of tagged training instances is the topic of the next section.

4. Interactive preparation of training

In a supervised learning system, it is desirable to reduce the amount of hand-tagging as much as possible. Finding a reservoir of untagged instances is relatively cheap. The expense comes in the tagging effort.

The need to reduce the amount of instances to tag is particularly great for information extraction from free text. The phenomenon of interest is often sparsely represented in a corpus of representative texts. In the Management Succession corpus, less than 2% of the instances contain relevant information. For a particular combination of slots the data is even more sparse. In a collection of about 500 news stories comprising over 10,000 instances, fewer than 50 instances had all four slots PersonIn, PersonOut, Post, and Org. An extremely large number of training instances would be needed to learn a four-slot rule for this domain if instances are selected at random.

4.0.1. Selecting informative instances In each iteration of WHISK, a batch of instances is selected from the reservoir of untagged instances, presented to the user for tagging, and then added to the training set. Arbitrarily selected instances may be predominantly irrelevant or already covered by reliable rules. Neither of these cases will lead to improvement of the coverage or accuracy of the rule set. The amount of hand tagging required can be greatly reduced through *selective sampling*, a form of active learning (Cohn *et al.* 1994, Lewis and Gale 1994, Dagan and Engelson 1996).

When the rule set is empty, the best that can be done is to select instances at random from a reservoir of untagged instances. If the instance space is sparse, the initial pool of instances to tag may be sampled from instances that satisfy a set of key words. For the Management Succession domain, sampling from instances that match "(was named|appointed|resigned|retired)" produces instances about half of which contain relevant information to tag.

After an initial rule set has been created, active learning seeks to present instances to be tagged that are near the decision boundary. Systems that assign a confidence level to classifications can present untagged instances whose confidence level indicates uncertainty. Systems that pool the votes of a committee of classifiers can present instances about which there are conflicting votes.

WHISK rules do not assign confidence levels or use a voting scheme, but the instances can be divided into three classes in relationship to the rule set. WHISK randomly samples from the three classes shown in Figure 16 for instances to be tagged. The proportion of instances from each of these three categories can be set by the user, with the default being one third of the instances from each.

Those covered by existing rules are useful to increase precision of the rules. They either increase the support for reliable rules or offer counter-examples that lead to refinements of the rule set.

Seed instances that will increase the coverage of the rule set are harder to find when relevant information is sparse. A "near miss" of an existing rule will often produce useful training instances that increase recall (the percentage of relevant

1. Instances covered by an existing rule
2. Instances that are near misses of a rule
3. Instances not covered by any rule

Figure 16. Three classes of untagged instances

information found by the rules). A near miss is an instance not covered by any rule, but covered by a minimal generalization of a rule.

The third class is included for the sake of completeness. These are instances randomly sampled from those not covered by any rule. In order to find the covered and uncovered instances efficiently, WHISK marks each instance in the training reservoir when it is covered by a new rule being added to the rule set.

4.0.2. When to stop tagging How can the user gauge when to stop adding to the training data? An informal analogy to PAC analysis may be used (Valiant 1984). The PAC formalism calculates how many randomly sampled training examples must be examined so that the probability is less than δ that the error rate of a rule set is greater than ϵ. For a messy problem involving natural language, δ and ϵ cannot be expected to approach zero even with unlimited training.

An estimate of the precision of a rule set may be gained by presenting the user with a sample of N covered instances. WHISK adds tags automatically according to its rule set, then the user makes corrections. The percentage of correct tags added by WHISK is an estimate of precision, with a confidence interval depending on N. This would be a straightforward extension to WHISK, although one that has not been implemented.

To estimate recall, a large set of N randomly selected instances is needed. In the Management Succession domain, the user would be obliged to review several thousand instances to find a few dozen relevant instances. WHISK's recall is estimated by comparing the instances tagged automatically to the total possible tags after the user has reviewed that batch of instances.

5. Empirical results

This section presents results of experiments for three text genres: structured, semi-structured, and free text. Structured texts are formatted so uniformly that a few examples are enough to learn perfect text extraction rules. Structured text is often found on Web pages that have been created automatically, usually from an underlying relational database or as a response to a search query.

This class of texts will be represented by two domains: CNN weather forecast web pages, and the BigBook searchable telephone directory. The CNN Weather Forecast domain consists of web pages for various domestic and international cities with four-day weather forecasts. BigBook is a web site with a searchable directory that returns lists with name, address, city, state, and phone number.

The second, more challenging genre is semi-structured texts. These have fairly stereotyped information but are not rigidly formatted. Experiments are reported from three domains of semi-structured text: the Rental Ads, Seminar Announcements, and Software Jobs domains[4].

The Rental Ads domain was collected from the Seattle Times on-line classified ads. The target concept is the three-slot relationship Neighborhood, number of Bedrooms, and Price, which may occur multiple times in a single ad. An extraction may have a range of prices (e.g. "2 BR $595-$695"), a range of bedrooms (e.g. "Studios, 1 & 2 bed from $400") or a range of neighborhoods (e.g. "Fremont/Wallingford -"). The HTML source file for one day of rental ads was broken into instances on the HTML tag <hr>, giving 415 instances. This was used as a training reservoir for WHISK and a blind test set of 405 instances was taken from the same source six weeks later.

The Seminar Announcement domain is a collection of on-line university seminar announcements collected at CMU by Dayne Freitag. The target concept has four slots: StartTime, EndTime, Speaker, and Location. Ten-fold cross validation was done on a set of 100 randomly selected announcements. Each text is considered as a single instance. The texts were tagged for single-slot extraction, with a separate tag for each mention of the StartTime, a separate tag for each mention of the EndTime, and so forth.

The Software Jobs domain has on-line job postings for programmers and systems analysts collected by Mary Elaine Califf (University of Texas, Austin). Each of these is tagged with up to 17 slots such as City, State, Country, Language (e.g. VISUAL BASIC), Platform (e.g. Windows 95), Application (e.g. SQL Server), and Required_years of experience. Ten-fold cross validation was done on a set of 100 randomly selected postings. As with the Seminar Announcement domain, each text is one instance and these were tagged for single-slot extraction.

The Management Succession domain[5], representing free text, is a collection of Wall Street Journal articles, approximately half of which contain a "management succession" event. The target concept is a four-slot relation of corporation, position, person moving into that position, and person moving out of the position. Governmental positions and appointments to board of directors are not considered relevant and are often hard to distinguish from corporate succession events.

WHISK performed multi-slot extraction wherever related slot fillers are found in the same instance. These multi-slot case-frames are often a subset of the four slots, since all slots are only rarely specified in the same sentence.

The original text was pre-processed by a semantic tagger as well as a syntactic analyzer before the instances were passed to WHISK. An instance for this domain is a sentence or clause as identified by the syntactic analyzer. The training reservoir for the Management Succession domain has 599 news articles comprising 14,052 instances. The test set is the collection of 100 news articles used as the official MUC-6 conference test set. It has 2,839 instances with 169 tags containing 84 PersonIn slots, 100 PersonOut, 148 Post, and 92 Org.

Tagging of test set instances was based on the official MUC-6 answer key, which lists the desired case-frame slot fillers but does not specify where each filler is found in the text. Strings from the answer key were tagged as target slots for WHISK only in contexts where their role could be inferred from within the sentence. For example Mr. A may be tagged as a PersonOut in a sentence that says he resigned, but not tagged in the sentence, "Mr. A declined to comment."

5.1. Methodology and metrics

For each experiment the collection was divided into a reservoir of untagged training instances and a blind test set that was annotated with the correct extractions. WHISK was run on the reservoir of untagged instances and repeatedly selected batches of instances to be tagged. The tagging process was simulated in these experiments by having WHISK look up the tags from a file that had been previously hand-tagged. Rules were grown with an error tolerance of 0.10 then evaluated on the blind test set. Results are reported without post-pruning and with post-pruning that discards rules whose Laplacian expected error is above 0.10.

The ten-fold cross validation experiments did not use WHISK's mechanism for interleaving training and tagging. A set of hand-tagged instances was divided randomly into ten partitions. Each partition in turn was set aside as a blind test set and WHISK learned rules from the remainder of the instances.

Multiple rules from the rule set may apply to a given instance, in which case the output is merged. If the output has case frames A and B, they are merged if the slots in B are contained in the slots of A. Each merged case frame in WHISK's output is compared with the hand-coded extractions in the test instance.

To be considered correct, all the slots of an output case frame must match an extraction for the test instance. Each of these slots is counted as a true positive. If the output contains an extra slot, all slots of that output case frame are considered incorrect (false positives). The test set extraction may have more slots than the output, in which case slots not matched by any correct output case frame are counted as a false negatives.

The metrics used in information extraction are borrowed from information retrieval, *recall* and *precision*. These metrics focus on how well the system performs on identifying the relevant information. Recall is the percentage of relevant information that is correctly reported by the system. Precision is the percentage of the information reported as relevant by the system that is correct.

The following equations define recall, precision, and accuracy in terms of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Recall and precision are sensitive to how well the system performs on the true positives, and ignore true negatives altogether.

$$
\begin{aligned}
\text{Recall} &= \text{TP} \,/\, (\text{TP} + \text{FN}) \\
\text{Precision} &= \text{TP} \,/\, (\text{TP} + \text{FP}) \\
\text{Accuracy} &= (\text{TP} + \text{TN}) \,/\, (\text{TP} + \text{TN} + \text{FP} + \text{FN})
\end{aligned}
$$

5.1.1. Structured text (1): CNN weather domain When the text is rigidly structured, text extraction rules can be learned easily from only a few examples. Structured text on the World Wide Web is often created automatically by a formatting tool that delimits the variable information with exactly the same HTML tags or other labels. WHISK's task for such documents is reverse engineering to recover an underlying relational database entry.

The CNN weather forecast pages are typical of this genre of text. Figure 17 shows an entry that gives the weather for Thursday as "partly cloudy" with high of 29 C / 84 F and low of 13 C / 56 F. This is the HTML source text – the web page is presented to the user with an image of a partly cloudy sky beneath the word "Thursday". Each weather forecast page for international cities has four instances identical to this one except for the variable information. Domestic cities have a slightly different format with temperatures in Fahrenheit only.

```
<TD NOWRAP> <FONT SIZE=+1> Thursday </FONT> <BR>
<IMG SRC="/WEATHER/images/pcloudy.jpg"
ALT="partly cloudy" WIDTH=64 HEIGHT=64> <BR>
<FONT SIZE=-1> partly cloudy </FONT> <BR>
<FONT SIZE=-1> High: </FONT> <B> 29 C / 84 F </B> <BR>
<FONT SIZE=-1> Low: </FONT> <B> 13 C / 56 F </B> </TD>
```

Figure 17. An example of structured text from a CNN weather forecast page

WHISK can learn a pattern to extract this information from as few as two training instances as long as the variable information is not accidentally identical. The rule shown in Figure 18 extracts a word that has semantic class *Day* followed by an end font tag to fill the Day slot of a Forecast case frame. The string between the next "1>" and the following end font tag fills the Conditions slot[6]. The High and Low temperatures are found between the next two pairs of begin bold and end bold tags.

```
ID:: 4
Pattern::  * ( Day ) ' </FONT>' * '1> ' ( * ) ' </FONT>'
           * '<B> ' ( * ) ' </B>' * '<B> ' ( * ) ' </B>'
Output:: Forecast {Day $1} {Conditions $2} {High $3} {Low $4}
```

Figure 18. One of two rules sufficient for 100% recall and precision on CNN weather forecast pages

This rule for international cities, together with a similar rule for domestic cities, gives recall 100% at precision 100%. Perfect recall and precision can generally be obtained for structured text in which each set of slots is delimited with some unique sequence of HTML tags or labels. WHISK can also achieve high performance on semi-structured text, as shown later in this section, but a larger number of training examples are needed and perfect recall and precision are generally not attainable.

5.1.2. Structured text (2): BigBook domain The BigBook dataset was provided by Nick Kushmerick (University of Washington). It contains 235 web pages that BigBook returned for various queries. Each source text has a list of 20 entries, each entry having the following slots: Name, Addr, City, State, AreaCode, and Phone.

WHISK learned the rule in Figure 19 from a single web page with responses to the query "state=NY" and "category=Art Printing". The document had 9,976 tokens as segmented by WHISK, with 1,631 of them before the first target slot.

```
ID:: 5
Pattern::  * 'pkey' * 'SIZE' * '">' ( * ) '</FONT>' *
           '">' ( * ) '</FONT>' * '">' ( * ) ', ' ( 'NY' ) *
           '@LPAREN ' ( Number ) * ( Number * Number )
Output:: BigBook {Name $1} {Addr $2} {City $3} {State $4}
                 {AreaCode $5} {Phone $6}
```

Figure 19. A rule learned for the BigBook directory

The delimiters around the first slot, " '">' (*) '</FONT>' " are not in themselves sufficient to identify the Name of the first entry. The previous context " * 'pkey' * 'SIZE' * " is necessary to move past the rather long section that precedes the first entry.

This rule is equivalent to the rule learned by the Wrapper Induction system (Kushmerick *et al.* 1997), except that the rule in Figure 19 specifies that the state must be 'NY', since all its training had that in common. A rule learned from two web pages for different states would correct this. With the State slot anchored from the outside, the rule has 100% recall at 100% precision.

WHISK is not as efficient as Wrapper Induction for this problem, since WHISK takes time proportional to the number of tokens in the instance for each term it adds to the rule. Wrapper Induction focuses exclusively on tokens immediately adjacent to the extracted phrase to find a left and right delimiter.

5.1.3. Semi-structured text (1): Rental ads Results are presented for three domains of semi-structured text: Rental Ads, Software Jobs, and Seminar Announcements. Figure 20 shows recall and precision on the Rental Ad domain as the number of training instances grows. These are the average performance of ten runs. Each run begins with a random sampling of instances that contain the string "$", which starts WHISK out with instances that are predominately relevant. Each iteration adds 25 instances to the training, using WHISK's default of randomly se-

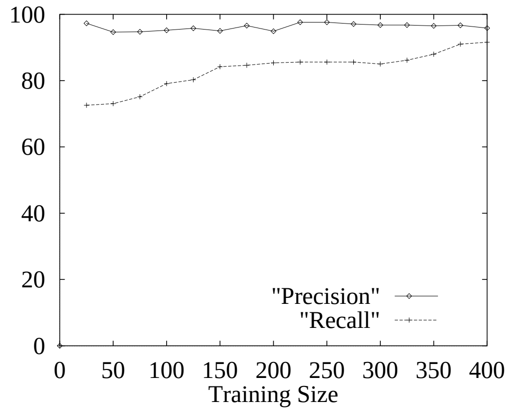

Figure 20. Learning curve for the Rental Ad domain

lecting one third each from instances covered by the current rule set, near misses of the rules, and instances not covered by any rule.

The recall and precision in Figure 20 are for rules grown with an error tolerance of 0.10 and post-pruned to remove rules with Laplacian expected error greater than 0.10. With post-pruning, precision is nearly flat, fluctuating between 94% and 98%. Recall begins at an average of 73% from 25 instances and climbs to 92% when 400 instances have been tagged.

Without post-pruning, recall is ten points higher than the pruned rule set at 25 instances, and reaches 94% at 400 instances. Without post-pruning the precision is lower, beginning around 85% and gradually climbing to 91%.

* (*Nghbr*) * (*Digit*) ' ' *Bdrm* * '$' (*Number*)

Figure 21. A pattern that covers 70% of the Rental domain recall

WHISK can gain such high recall from only 25 training instances because of a particularly strong regularity that is discovered early on. The pattern in Figure 21 covers nearly 70% of the recall with precision of 97%. Later rules boost the recall by covering instances that do not fit this pattern, such as those with a range of bedrooms or a range of prices.

5.1.4. Semi-structured text (2): Seminar announcements Table 2 shows recall and precision for the Seminar Announcement domain. These results are for ten-fold cross validation on a randomly selected set of 100 texts. Results are shown both without post-pruning and with a post-pruning threshold of 0.10. Terms were selected from a window beginning five terms before the extraction slot to five terms after the slot.

Table 2. Performance for Seminar Announcement, cross validation on 100 texts

Slot	Unpruned R	Unpruned P	Pruned R	Pruned P
StartTime	100.0	86.2	100.0	96.2
EndTime	87.2	85.0	87.2	89.5
Speaker	11.1	52.6	0.0	0.0
Location	55.4	83.6	36.1	93.8

The entire text is taken to be one instance in this domain. Since all slots from a given document belong to a single case frame, WHISK learned rules for each of the four slots separately: StartTime, EndTime, Speaker, and Location. This experiment used no semantic classes other than *Number* and *Digit*.

Recall was computed to take into account the effect of redundant information in the document. About half the references to StartTime and to EndTime are redundant references. For example an announcement has "Time: 10:30 PM - 11:30 PM" in the heading section and a second reference to "10:30pm-11:30pm" in the body of the text. WHISK often picks up one reference and misses additional references that are in less clear context. WHISK is given full credit for recall if it identifies a target concept at least once, even if it misses an additional reference. This is in keeping with the scoring methodology used for the SRV system (Freitag 1998) on this data set.

StartTime and EndTime show strong recall at high precision with post-pruning, 100% recall at 96% precision and 87% recall at 89% precision, respectively. WHISK had low performance for the Speaker slot and mainly learned rules based on speaker's names, which did not transfer well to the test data. Recall for the Location slot is 36% at precision 94% with post-pruning. Rules for Location were also mainly based on names of buildings on campus, but this has utility on unseen announcements.

These results are not as high as those reported for the SRV system (Freitag 1998) on the same dataset. SRV finds a speaker that is correct 62% of the time, a location that is correct 75% of the time, start time 99%, and end time 96%. The results for SRV are from a five-fold cross validation on 485 documents.

Much of the difference can be attributed to SRV's more expressive feature set. Where terms in WHISK rules can be tokens or semantic classes, SRV can make use of capitalization, word length, and length of extracted slot. WHISK cannot express a rule for Speaker that looks for a phrase of up to three capitalized words whose first token is "Dr.". SRV can express such a rule.

Figure 22 show examples of patterns learned in the Seminar Announcement domain. Rules for StartTime and EndTime took advantage of the classes *Digit* and *Number* and tokens such as ":", "-", "am", "pm", or "TIME" to produce rules with high generality. Location rules typically include the name of a building on campus.

When a WHISK rule has a wildcard within the extraction field, such as the pattern for StartTime " * 'Time: ' (*) ' -' ", there is a danger that the literal after the extraction will not be found until much later in the text. This would

Pattern:: * '@newline Time: ' (*Digit* *) ' @newline'
Output:: Seminar {StartTime $1}

Pattern:: * '- ' (*Number* * 'PM') ' @newline'
Output:: Seminar {EndTime $1}

Pattern:: * ': ' ('Alan' *) ','
Output:: Seminar {Speaker $1}

Pattern:: * ('Wean Hall ' *Number*)
Output:: Seminar {Location $1}

Figure 22. Some typical Seminar Announcement rules

cause WHISK to extract an extremely long string. The experiment reported here included an extraction length limit for each slot, so that no extractions were longer than the longest fill for that slot found in the training set.

Of the domains tested, the Seminar Announcement domain was the only one in which this length limit made a significant difference. Without this length limit, StartTime drops to 81% at 94% precision with post-pruning, EndTime to 82% recall at 78% precision, and Location and Speaker are the same with or without the limit.

5.1.5. Semi-structured text (3): Software jobs In the Software Jobs domain, as in the previous domain, one source text corresponds to a single case frame, so extracting isolated slots is sufficient. Each case frame has up to seventeen slots, not all of them filled for a given job posting: ID, Title, Salary, Company, Recruiter, State, City, Country, Language, Platform, Application, Area, Req_years_experience, Desired_years_experience, Req_degree, Desired_degree, and Post_date.

Figure 23 shows results of ten-fold cross validation on 100 randomly selected texts from the Software Jobs domain. Rules were learned at error tolerance of 0.10 and a window of five tokens before and after the extraction. No semantic classes were used other than *Digit* and *Number*.

As the post-pruning threshold varies from Laplacian 0.10 to 1.00 (which is effectively no post-pruning), recall begins with 39% at precision 92%. At post-pruning threshold 0.20 recall is 52% at precision 88%, at 0.35 recall is 55% at precision 85%, and with no post-pruning recall is 57% at precision 80%. Along this recall-precision trade-off are results nearly identical with those reported for RAPIER, which had recall 53% at precision 84% for this domain (Califf and Mooney 1997).

Except for a few slots such as ID, Post_date, and City that occur in highly predictable contexts, most of the rules WHISK learns for this domain are highly specific rules based on exact tokens. A separate rule learns that "C++" is a Language, that

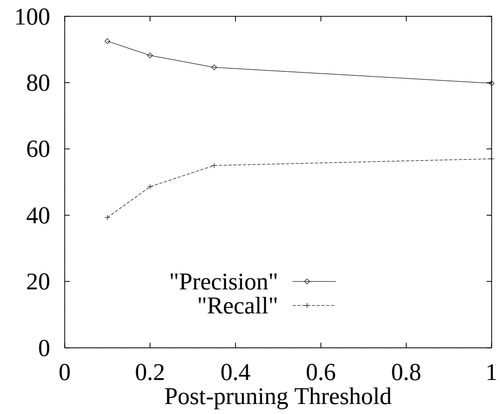

Figure 23. Performance in the Software Jobs domain as post-pruning threshold varies

"Lotus Notes" is an Application, and that in some, but not all contexts, "database" is an Area.

Many of these terms were hard to learn, since their meaning changes when used in a longer phrase. "Java" is not tagged as a Language when part of the phrase "Java Beans". Similarly "Windows" is a Platform unless part of the phrase "Windows 95" or "Windows NT", in which case the longer phrase should be extracted as the Platform. WHISK could only learn to identify such terms in restricted contexts, such as when the word Windows is immediately followed by a slash.

WHISK is unable to represent negated features such as a rule that extracts Java as a Language if *not* followed by Beans. An alternate approach would be for WHISK to automatically merge any output that is a substring of another extracted phrase. The internal metrics would then not count an extraction as an error if it is a substring of a correct extraction.

Could WHISK have profited from semantic classes for *Language*, *Platform*, *Application*, and so forth? An experiment was conducted in which WHISK was given lists of terms for these classes, but this failed to help performance other than a small boost at low training levels. Too many of the relevant terms in this domain undergo shifts of meaning depending on context for simple lists of words to be useful.

5.1.6. The impact of semantic classes A major reason that WHISK attains high performance from low training levels in the Rental Ad domain is that semantic classes are quite effective for this domain. The classes *Nghbr* and *Bdrm* can be assigned to isolated words regardless of context. When these classes are disabled, WHISK must learn separate rules with literals for each neighborhood name and each abbreviation for bedroom.

The results shown earlier in Figure 20 used the semantic classes *Nghbr* and *Bdrm*. Figure 24 shows a rule learned in the Rental Ad domain from 100 training instances without those semantic classes. This rule covers five training instances with no errors for a Laplacian of 0.167. This means that it would be discarded at a post-pruning threshold of 0.10.

ID:: 6
Pattern:: * ('Wallingford') * (*Digit*) ' br' * '$' (*Number*)
Output:: Rental {Neighborhood $1} {Bedrooms $2} {Price $3}

Figure 24. A rule not using the classes *Nghbr* and *Bdrm*

Recall grows much more slowly in the absence of the classes *Nghbr* and *Bdrm*. In the absence of semantic classes, WHISK must see multiple examples of each particular term in each context that makes it relevant. By 200 training instances recall is only at 27% at precision 93% with post pruning as opposed to 85% recall at 94% precision with the semantic classes. Without post-pruning, recall is 56% at precision 76% without the semantic classes, as opposed to 90% recall at 86% precision with semantic classes.

5.1.7. Free text: Management succession domain WHISK was tested on the Management Succession domain to assess how well it can handle the complexities of free text. Extraction from news stories presents a challenging linguistic complexity far beyond that of the semi-structured text domains presented here. Also, much of the information to be extracted is implied rather than plainly stated in the text.

WHISK's performance on the Management Succession domain is shown in Table 3 as the number of tagged training instances grows. These figures are the average recall and precision for ten runs. In each run the tagged training set began with 50 instances and was increased by 50's up to 200 instances, by batches of 100 up to 400, then by batches of 200. Data points shown in the table are for tagged training sets of 100, 200, 400, and 800 instances.

At each iteration WHISK selected a batch of instances from a reservoir of 14,052 untagged instances. Hand-tagging was simulated by looking up tags in a file of pre-tagged instances. Rules were grown with error tolerance of 0.10 and are shown with no post-pruning and with post-pruning at a threshold of 0.10.

Training on 400 or even 800 instances is a small fraction of the instances for this corpus. To get a sense of WHISK's ceiling performance in this domain, rules were also induced from a tagged training set of 6,915 instances, which comprised all instances that contained either a person name, company name, or corporate position. Recall reached 46% at precision of 69% for this exhaustive training set with post-pruning, recall of 61% at precision 48% without post-pruning.

At 100 training instances none of the runs had more than four rules with enough support to survive the post-pruning threshold of 0.10. Three iterations had no rules. A threshold of 0.10 eliminates rules based on fewer than nine instances.

By 400 training instances, an average of fifteen rules survived this pruning threshold out of an average of 368 rules before pruning. Retaining these low coverage rules

Table 3. Performance for the Management Succession domain as training levels increase

Training	Unpruned R	Unpruned P	Pruned R	Pruned P
100	51.5	24.1	9.6	45.6
200	49.9	31.5	13.9	62.1
400	53.5	36.0	19.3	70.5
800	56.3	42.9	31.0	70.6
6,900	61.0	48.5	46.4	68.9

increases recall from 19% to 54% but reduces precision from 70% to 36%. Recall for post-pruned results continues to grow slowly as the training set grows, and by 800 instances has reached two thirds of the recall of the exhaustive training set.

WHISK's active learning mechanism is effective in finding informative training instances in sparse domains such as this. Table 4 shows average performance of ten runs where instances are drawn at random from the reservoir of 14,052 untagged instances. As in the previous experiment, hand-tagging is simulated by consulting a file of pre-tagged instances.

At a training level of 400 instances, only one run out of ten has any rules with Laplacian error below 0.10. By 800 instances some of the runs still have zero recall with post-pruning, while most have recall similar to 100 instances with active learning but at higher precision. The difference between random training and active learning is less dramatic without post-pruning. Performance from 800 random instances without post-pruning is roughly that of 200 instances with active learning.

Table 4. Management Succession results without active learning

Training	Unpruned R	Unpruned P	Pruned R	Pruned P
100	33.7	21.3	0.0	0.0
200	34.6	24.5	0.0	0.0
400	42.0	31.3	0.6	63.9
800	47.1	35.6	5.0	68.0

The difficulty WHISK has with this domain at low training levels is primarily due to the wide variability in how the target concepts are expressed. A few high frequency patterns are responsible for much of the 46% recall reached in the exhaustive training run. Most references to the target concepts, however, use combinations of terms and syntactic patterns that are found only a few times in the entire corpus. It is extremely difficult for a machine learning system to distinguish these low frequency patterns from features accidentally associated with the concept.

How good is 46% recall on a domain of this difficulty? One comparison is with other systems that participated in the MUC-6 conference. Unfortunately, the only scores reported are for full system performance, which includes merging information across sentences, resolving pronouns and generic references, and formatting the output.

The highest performing MUC-6 systems, which used hand-crafted rules, had recall of 47% at precision 70%, recall of 50% at precision 59%, and recall of 47% at precision 62%. Performance was not reported for the system modules that extracted information at the sentence level. We can assume that performance was somewhat higher at the sentence level than full system performance.

The closest comparison possible is with CRYSTAL, which uses machine learning and operates at the sentence level. CRYSTAL was trained and tested on the same Management Succession texts as WHISK. Performance was reported for each combination of slots separately. A weighted average of all multi-slot extractions gives overall recall of 55% at precision 72% for a ten-fold cross-validation on all 599 texts.

While the MUC-6 systems are tackling a somewhat harder problem than WHISK, CRYSTAL has made the problem somewhat easier than WHISK. CRYSTAL identifies the syntactic field, such as subject or direct object, that contains the target phrase, but does not exactly delimit the target phrase. Later processing is required to identify the exact phrase within that field. CRYSTAL might identify the PersonOut, Post, and Org as all being found in the subject of a sentence without indicating which phrase plays which role. An example is a sentence whose subject field consists of "David A. Fleck, 41 years old, previously chief executive officer of Fleck Co."

Figure 25 shows how WHISK compares with the top three MUC-6 systems and with CRYSTAL on the Management Succession domain. All the systems cluster near the center of the recall-precision graph in a region from recall 47% to 55% and precision from 59% to 72%. At the exhaustive training level, WHISK is in the upper part of this cluster near the top MUC-6 system. Bear in mind that this is not a straightforward comparison, since the tasks are not identical. When the relative difficulty of the problem is taken into account, WHISK and CRYSTAL have roughly equivalent performance, and both lag behind the hand-crafted MUC systems.

5.2. Discussion

WHISK can learn rules for extraction from a wide range of text styles with performance that varies with the difficulty of the extraction task. What is it about WHISK's representation and learning algorithm that gives it perfect results in some domains and difficulty breaking recall of 50% in others?

Rules in the form of regular expressions are well suited to structured and semi-structured text, where there is a predictable order of fixed tokens surrounding the variable information. WHISK's rules match either literals or a semantic classes found in the text, skipping over portions of the text with its wildcard "*" until the following term in a rule is matched.

For this rule representation to work, there must be an identifiable pattern to the input. A phrase to be extracted may be self-delimiting, such as a semantic class

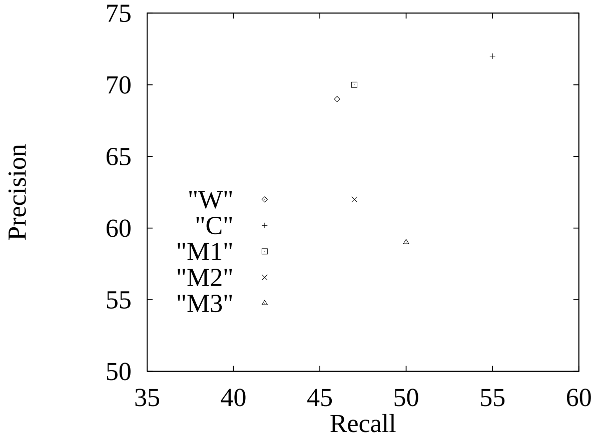

Figure 25. A performance comparison: W is WHISK trained on 6,900 instances, C is CRYSTAL, M1, M2, and M3 are the three top MUC-6 systems

(*Person*) or exact token ("C++"). In the Seminar Announcement domain some Location slot fills are identified by patterns such as " * ('Wean Hall ' *Number*) " which specifies a building name by exact token followed by a semantic class.

Phrases that are not self-delimiting must be delimited by unique and predictable patterns of tokens. In the CNN Weather domain, the Conditions slot is always immediately preceded by an HTML tag ending in '1>' and followed by ' </FONT>'. Together with other terms to further specify the context, this produces a rule that is 100% reliable. In the Seminar Announcement domain, it is similarly easy to learn rules for StartTime for input such as "Time: 4:00 PM ".

WHISK often has trouble in unstructured portions of text when the surrounding context is highly variable. A reference may have clear delimiters such as a StartTime with "... starting at 3:30". However, these patterns may be hard to distinguish from irrelevant look-alikes such as "Refreshments will be served starting at 3:15 pm." SRV's approach of combining multiple classifiers comes in handy here, since seminars *never* start at 3:15 at CMU, a fact learned by a classifier that simply remembers all the slot fills seen in the training.

Another difficulty that WHISK has with free text is the high variation in ways to express a target concept. This creates sparse training for individual patterns. Seminar announcements often include a second reference to a speaker in the body of the announcement. An example is "Dr. Kennedy will be discussing a project ...". This was the only case of "Dr. * will" in 100 texts.

In such cases WHISK needs additional knowledge provided in its input. Ideally a semantic tagging procedure would identify relevant phrases such as person names. This leaves WHISK with the task of determining the context in which a person plays a relevant slot-filling role. Even a semantic tagger that recognized capitalized words as proper names would help. The title of "Dr." followed by a proper name is nearly always a Speaker in the Seminar Announcement domain. Whether a person is relevant in the Management Succession domain is trickier to determine.

Even with person names and corporate names tagged, more than half the rules in the Management Succession domain at each training level have coverage of only one or two, contributing more errors than correct extractions. The rule shown in Figure 26 looks quite general, but applies only twice among 600 training instances. This rule covers instances such as "Mr. A succeeds Mr. B, chairman of XYZ Inc."

ID:: 7
Pattern:: * (*Person*) * *Succeeds* * (*Person*) * (*Corp*)
Output:: Succession {PersonIn $1} {PersonOut $2} {Org $3}

Figure 26. A low coverage rule generated from 600 training instances

WHISK's multi-slot extraction patterns must specify the relative order of the slots. For example a pattern that applies to "Atlas announced the resignation of its president Mr. Adams" has Org, Post, and PersonOut, while a pattern that applies to "Adams stepped down as president of Atlas" has PersonOut, Post, Org. These are sufficiently different contexts to warrant separate rules. WHISK is saved from considering an exponential number of orderings of k slots because WHISK only grows rules from actual occurrences of those slots found in the training set.

6. Related work

The main advance that WHISK makes is combining the strengths of systems that learn text extraction rules for structured text with those that handle semi-structured text. The following gives a brief overview of how WHISK compares to other IE learning systems. These systems are described later in this section as well as related work in machine learning.

The Wrapper Induction system (Kushmerick *et al.* 1997) works only on rigidly structured text. Wrapper Induction performs multi-slot extraction that bundles together related information into a single case frame[7]. This is important for domains where a text has several case frames and it is critical to associate the extracted slots together properly. An example of this is a web page with dozens of names, addresses, and phone numbers or item numbers, descriptions, and prices.

SRV (Freitag 1998) and RAPIER (Califf and Mooney 1997) extends information extraction to semi-structured text, but each of these systems extracts only isolated slots. Related information must be re-assembled into a case frame by later processing. This is adequate in domains where each text has only one case frame, such as seminar announcement postings.

WHISK shares SRV and RAPIER's ability to handle either structured or semi-structured text, but does not have their limitation of single-slot extraction. Like

Wrapper Induction, WHISK associates related information into multi-slot case frames.

For extraction from free text, performance of machine learning systems still trails behind that of systems with hand-coded rules. WHISK can handle free text as well as structured or semi-structured when it is provided with syntactically analyzed input, and has performance comparable to CRYSTAL (Soderland 1997) a system designed specifically for free text IE.

6.1. IE systems for structured text

The driving force behind research on information extraction from structured and semi-structured text is the World Wide Web. Heterogeneous database applications and software agent technology can be enabled by IE systems that transform a web page into the equivalent of database entries. The Wrapper Induction system (Kushmerick *et al.* 1997) was developed to learn a "wrapper" for web pages that allows software robots to treat the text as if it contained a set of relational tuples.

Wrapper Induction examines a set of web pages that have been labeled with extraction slots 1 to k and learns string of text that delimit each slot. The system looks for a common suffix for characters to the left of the slot and a common prefix of characters to the right of each slot. In addition, the system searches for strings that delimit text that forms the header or tail of a web page from the tabular information.

The class of web pages that Wrapper Induction can handle are those that conform to a single "HLRT" rule: a Head delimiter, a set of Left and Right delimiters for each slot, and a Tail delimiter. Pages that conform to this regularity are nearly always automatically formatted responses to a search index or a listing of an underlying database.

This process of finding delimiters is similar to WHISK's step of anchoring each extraction slot with terms immediately outside the left and right boundaries of an extracted phrase. One difference is that Wrapper Induction assumes that delimiters can be found that apply to *all* the training examples. The system fails to learn a wrapper if this is not the case. WHISK looks for delimiters that cover as many correct extractions as possible, but assumes that multiple rules may be needed to cover the entire training set.

WHISK also differs from Wrapper Induction by allowing rules to use terms inside the extraction boundary as well as just outside it. WHISK rules may also have terms that are not immediately contiguous to a target slot and allow semantic classes, while Wrapper Induction uses only literals.

If an HLRT rule exists that extracts all the data from a set of web pages, the Wrapper Induction system is guaranteed to find it. WHISK offers no such guarantee. The flexibility of WHISK's rule representation produces an enormous search space. WHISK uses a greedy search to add terms to its rules and may make choices that lead to overly restrictive rules that do not generalize well to new instances.

Another system that induces rules for structured text is that of Ashish and Knoblock (Ashish and Knoblock 1997). This system is designed for documents with a recursive structure: a regular pattern of sections divided into subsections. It learns a parser for such hierarchically structured text, using cues such as font size of headings and indentation. Human review of the output guides the system to refine its parsing rules. This system handles a different, but complementary, problem than WHISK.

6.2. Systems for semi-structured text

6.2.1. SRV The SRV system (Freitag 1998) adopts a multi-strategy approach and combines evidence from three classifiers[8]. The IE problem is transformed into a classification problem by limiting the system to single-slot extraction. All possible phrases from the text up to a maximum length are considered as instances. The system assigns to each of these phrases a metric indicating confidence that the phrase is a correct filler for the target slot.

For example, if a slot can be filled by a phrase with length from one to ten tokens and there are one thousand tokens in the text, SRV treats the text as the equivalent of ten thousand instances, nearly all of which are negative instances of a correct extraction.

The simplest of the three classifiers is a "rote" learner that simply compares the phrase to a list of all correct slot fillers found in the training. The second is a naive Bayes classifier that computes an estimated probability that the tokens in the phrase are found in a correct slot filler.

The third classifier is a relational rule learner that does a top down induction similar to FOIL (Quinlan 1990). The relational learner induces a set of constraints such as the length of the phrase and features of particular words in or near the phrase. These features can specify an exact token or derived features such as whether the token is numeric or capitalized. An example of a clause learned by the SRV relational rule learner is the following.

```
some(?A [prev-token prev-token] capitalized true)
```

This clause requires that there be some token A which is preceded by a capitalized token two tokens back. The rule may have several constraints on A and on other tokens in the phrase to be extracted. A rule for identifying speakers in the Seminar Announcement domain could specify a phrase of no more than four words, all of them capitalized with the first word "Dr." and a ":" immediately preceding "Dr.".

SRV's rule representation is expressive and able to incorporate orthographic features and other evidence such as part of speech or semantic classes when available. No prior syntactic analysis is required.

In domains where there is assumed to be one and only one correct slot filler per text (with possible redundant references), the system apparently ranks all candidate phrases by confidence level and outputs the most likely extraction for that slot. This works well in the Seminar Announcement domain. The system identifies a speaker for 98% of the announcements with 62% accuracy. Extraction for location has 75% accuracy, start time 99% and end time 96%. The SRV relational rule learner without the other two classifiers has accuracy of 61%, 76%, 99%, and 94% respectively.

SRV has also been applied, without syntactic analysis, to a collection of Reuters news stories on corporate acquisitions. No recall and precision figures are reported, but of the texts that had relevant information, the system found the correct acquired company 46% of the time, the correct purchaser 47% of the time, and purchase amount 70% of the time. For texts that had no relevant slot fillers, the system presumably has a tendency to report spurious extractions.

The rote learner, naive Bayes, and SRV relational learner are all designed to extract single slots. It is not clear that SRV can be adapted to multi-slot extraction and remain computationally tractable. To learn a rule with k slots would require training on *all* combinations of k phrases of suitable length in the document. Applying a rule to unseen text would also involve considering an enormous number of candidate phrases.

6.2.2. RAPIER The RAPIER system (Califf and Mooney 1997) also uses a form of logic programming and requires no prior syntactic analysis. The text is considered to have three fields centered around the target phrase: the target phrase itself, a pre-filler of tokens before the target phrase, and a post-filler of tokens after it.

RAPIER's rules specify an ordered list of items to be matched for each of these fields, with lexical constraints, semantic constraints, and/or part of speech constraints on each item. The items in RAPIER's pre-filler field are a list that matches the k tokens immediately before the target phrase and post-filler matches a list of tokens immediately after the target phrase. In addition, a maximum number of tokens for the field may be specified.

RAPIER operates bottom-up, beginning with the most specific rule that matches a target slot in the training. Pairs of rules are chosen at random and a beam search is conducted to find the best generalization of the two rules, taking a least general generalization, then adding constraints until the proposed rule operates correctly on the training. This is repeated until no progress is made in several successive iterations.

Like SRV, RAPIER performs well on domains where single-slot extraction is sufficient. RAPIER had recall of 53% at precision 84% on the Software Jobs domain using a part-of-speech tagger but no semantic classes.

Could RAPIER be extended to handle multi-slot extraction? Rather than divide the text into three fields, it could be divided into multiple fields: the target phrases and fields of tokens immediately before and immediately after each target phrase. How this would affect time complexity is not clear.

6.2.3. Discussion of structured and semi-structured text systems Many of the differences between SRV, RAPIER, and WHISK are in representation and supporting knowledge sources. WHISK allows constraints on exact tokens, semantic classes, and syntactic tags if any have been inserted by prior syntactic analysis. RAPIER's rules can include exact tokens, part-of-speech tags, and semantic classes.

SRV uses exact tokens and a variety of derived features such as length of word, capitalization, and semantic class. SRV's is a versatile representation, allowing it to learn personal names from orthographic features, where WHISK must rely on prior semantic tagging of person names and RAPIER relies on part-of-speech tags that distinguish common nouns from proper nouns.

WHISK's regular expression rules require it to specify a relative ordering of terms, with wildcards that allows zero or more intervening terms. Both SRV rules and RAPIER rules specify absolute rather than relative order of tokens. SRV does this with a prev-token and a next-token operator and RAPIER with an ordered list of items for each field.

In cases where the immediate context is sufficient to identify a target phrase, selecting terms that are immediately adjacent to the extraction slot is a useful heuristic. WHISK can add any token in the instance to a rule, which gives flexibility to the rules, but also leads to overfitting on tokens far removed from the extraction.

Extracting phrases that are of varying length without including too many or too few tokens is difficult for each of these systems. Both SRV and RAPIER address this problem with a feature that WHISK lacks, constraints on the length of an extracted phrase. WHISK's wildcard "*" will skip over tokens indefinitely until the next literal in the rule. WHISK would profit from the ability to constrain its wildcard character, perhaps "*1" meaning skip no more than one token and "*k" meaning skip no more than k tokens.

These difference of features available to the rules are a matter of implementation, rather than a fundamental difference between algorithms. WHISK and RAPIER could add features such as capitalization or number of characters in a token. SRV could search for informative tokens further removed from the target phrase. WHISK could add quantifiers to limit the scope of its wildcards.

The most important difference between WHISK and the two other systems, is WHISK's ability to learn multi-slot extraction. Extracting each slot in isolation is appropriate for some domains. This is not the case in other domains, such as texts that list dozens of items with prices and descriptions. Unless the item, price, and description can be extracted together, important relationships are lost.

For rigidly formatted text with tabular information that conforms to an HLRT rule, either the Wrapper Induction system or WHISK is suitable. If a web page is known to conform to HLRT, Wrapper Induction will find a rule more efficiently than WHISK.

For semi-structured text that is known to have at most one case frame, either SRV, RAPIER, or WHISK will serve. Performance between the three is roughly comparable, particularly if each system is extended to include features used by the others. For semi-structured text where multi-slot extraction is desirable, WHISK is the only system developed so far.

6.3. IE systems for free text

Research in applying machine learning to natural language processing has been primarily at the level of semantic disambiguation of individual words or gathering lexical statistics to guide parsing. Systems that learn text extraction rules for free text are rare. The systems most closely related to WHISK are AutoSlog and CRYSTAL. These are discussed as well as LIEP, HASTEN, and PALKA.

6.3.1. AutoSlog AutoSlog (Riloff 1993) was the first system to learn text extraction rules from training examples. AutoSlog handles only single-slot extraction and uses heuristics to create a rule from relevant examples that extracts the correct information from that example. Each rule includes a "trigger" word or words and a semantic constraint on the extracted field. If the rule succeeds, a case frame is output that identifies the syntactic field containing the target phrase.

In the MUC-4 domain of news stories about terrorism, an AutoSlog rule might be triggered by the verb "kidnapped" in the passive voice and look for the semantic class *HumanTarget* in the subject. Another rule might be triggered by the noun "attack" and look for a prepositional phrase with the preposition "on" and the semantic class *PhysicalTarget*.

AutoSlog rules operate at the granularity of syntactic fields and assume a prior step that has bracketed the sentence into fields such as Subject, Verb, Object, and PP (prepositional phrase). A rule does not specify the target phrase exactly, but simply identifies which field contains the information to extract. Later processing is needed to trim away extraneous words or phrases within that field.

AutoSlog also assumes that a semantic tagging step has been done to assign a semantic class to entire fields. A rule to identify kidnapping victims only extracts fields that have been previously tagged as *HumanTarget*.

AutoSlog has no automatic induction step and instead passes its output to a human to accept or reject each rule. Typically about 30% of the rules are retained. Despite its simple architecture, AutoSlog achieved 98% of the performance of the hand-crafted rules used in the University of Massachusetts MUC-4 system. This is with AutoSlog used in conjunction with the same syntactic analyzer, semantic tagger, and discourse processing routines, as the hand-crafted rules.

6.3.2. CRYSTAL A second system to learn text extraction rules is CRYSTAL (Soderland *et al.* 1995, Soderland 1997), which like AutoSlog, takes input that has been processed by a syntactic analyzer and a semantic tagger. CRYSTAL uses a bottom-up covering algorithm that begins with the most specific rule to cover a seed instance, then generalizes the rule by merging with similar rules.

CRYSTAL can learn multi-slot case frames and does not rely on heuristically chosen trigger words or fixed semantic constraints. Like AutoSlog, CRYSTAL relies on accurate syntactic processing and semantic tagging of individual terms in the input text. Also like AutoSlog, it identifies relevant information at the granularity of syntactic fields.

CRYSTAL has no explicit ordering constraints on syntactic fields or on terms within those fields., although the names of fields often imposes an implicit ordering. The Subject can be assumed to be before the Verb, which is before the Object, but a PP (prepositional phrase) can be anywhere.

CRYSTAL has also been applied to semi-structured text. but only if supplied with an appropriate syntactic analyzer that allows the text to be treated as if it were grammatical (Soderland 1997a).

6.3.3. LIEP, HASTEN, and PALKA Other systems that learn text extraction rules are LIEP, HASTEN, and PALKA. LIEP (Huffman 1996) uses heuristics in a manner similar to AutoSlog, but learns multi-slot rules. In contrast to systems we have previously seen that cannot handle multi-slot extraction, LIEP cannot handle single slot extraction. It finds context for a slot only in terms of its syntactic relationship to other slots.

LIEP's rules set a semantic constraint on target phrases and posit binary syntactic relationships to form paths linking multiple slots. For example slot A is the subject of a verb and slot B is the direct object of the same verb. If more than one syntactic path is possible to link the slots, LIEP selects the one that performs best on other training instances.

Terms included to form the syntactic links are expressed as exact words or as verb roots. Verbs that are incorporated in a rule may have further constraints such as active or passive voice. If other training instances match an existing LIEP rule except for an exact word or root, the rule is expanded to include a disjunctive list of terms.

LIEP rules are induced from positive training instances only, after a hand-coded set of key words have filtered out irrelevant sentences. The same key word filtering is done before applying the rules on new text. There seems to be no mechanism for guarding against rules that overgeneralize, but high precision is reported when the test set has only relevant sentences.

HASTEN (Krupka 1995) uses a k-nearest neighbor approach with a set of hand-picked instances as exemplars. Each exemplar represents a positive example of a multi-slot extraction. Each element in the exemplar is a token or phrase from the sentence along with constraints such as semantic class, verb root, verb voice, exact word. The exemplar also gives the relative order of these elements.

During training, a weight is learned for each exemplar based on its performance on training sentences. To apply the HASTEN classifier, a new sentence is compared to each of the exemplars and a goodness-of-fit metric is multiplied by the learned weight which discounts unreliable exemplars. HASTEN was used by one of the highest scoring systems in the MUC-6 evaluation.

PALKA (Kim and Moldovan 1993) uses an induction method similar to Mitchell's candidate elimination algorithm. PALKA is computationally intensive and was implemented on a parallel computer. Each new instance that is read may cause an evolving rule to either be generalized to include a new positive instance or specialized to avoid a negative instance.

PALKA's rules have a constraint on a verb root that serves much as the trigger word in AutoSlog. Syntactic fields that contain target slots have semantic constraints from a semantic hierarchy. Rules are generalized by moving up in this hierarchy and specialized by moving downward or by replacing a semantic class with some of its children. The system has no noise tolerance mechanism.

6.3.4. Discussion of free text systems One aspect of WHISK that distinguishes it from these other IE rule learners is WHISK's rule representation. WHISK rules can use either literals or semantic classes drawn from any portion of the instance, whether a target slot or not.

Most of the other systems, AutoSlog, LIEP, HASTEN, and PALKA, allow only semantic class constraints on extracted slots and allow only exact word or verb root constraints for other sentence elements. If a particular word within a target phrase is good evidence that the phrase is relevant, but the class of that word is not a reliable indicator, these systems will be unable to learn a rule. Neither WHISK nor CRYSTAL have this restriction.

WHISK is also free to use or ignore syntactic tags that have been provided by a syntactic analyzer. This gives it an advantage over CRYSTAL, AutoSlog, and PALKA, that are obliged to create separate rules if a relevant phrase can be found in the subject, direct object, or prepositional phrase. By ignoring the syntactic label of a field, WHISK can create rules of greater generality, as can LIEP and HASTEN.

What WHISK is not free to do is ignore the relative order of terms in its rules. This is a drawback when compared to the other systems that can leave the order of sentence elements free, although this is not an issue for major clausal elements, whose relative order is predictable.

One comparison point is the amount of human engineering required by each of these systems. AutoSlog requires manual verification of the rules it produces. LIEP uses a hand selected set of key words to filter out irrelevant input. HASTEN uses a hand selected set of relevant sentences as exemplars. PALKA begins its induction with a small set of hand coded rules that are then refined by training. WHISK and CRYSTAL are the most fully automated of these systems, with no hand-coded rules or key words needed to start the induction and no manual review needed of the resulting rule set.

Another comparison point is the granularity of extraction. WHISK rules specify exact delimiters on the target phrase. AutoSlog, CRYSTAL, and PALKA identify the syntactic field that contains the target phrase. Later processing is required to identify the phrase more exactly. LIEP and HASTEN operate on simple noun phrases and are able to identify the target phrase directly, as WHISK does.

The issue that was important in distinguishing systems for semi-structured text is less important with free text. All the systems except AutoSlog handle multi-slot extraction.

One point in common to all of these systems that handle free text is that none of them does the entire information extraction task. Each of them requires a syntactic analyzer and semantic tagger to prepare the input for rule learning. In a full

IE system the job is not finished after extraction rules have been applied. Later processing is needed to merge information across sentences, to resolve pronouns and generic references, and to perform some normalization of the output.

6.4. Related machine learning algorithms

WHISK belongs to the family of machine learning algorithms know as covering algorithms (Michalski 1983) and shares a general methodology with algorithms that learn classification rules by top-down induction (Quinlan 1990). WHISK begins with an empty rule, then selects terms to add according to a term selection metric. Terms are added to a rule one at a time until the errors are reduced to zero or a pre-pruning criterion has been satisfied. The process is repeated until a set of rules has been generated that cover all positive extractions from the training, at which time post-pruning is done to remove rules that may be overfitting the data.

WHISK does not "divide and conquer" or even "separate and conquer" – The entire training set is available for testing further rules as in RISE (Domingos 1994). This makes the best use of small training sets. Selection of additional instances to be hand-tagged is interleaved with the learning, to provide new training instances most likely to help increase precision or recall of the existing rule set. This is akin to the active learning paradigm (Cohn *et al.* 1994, Lewis and Gale 1994, Dagan and Engelson 1996). The bias of WHISK is to find the most general rules that fit the training data.

While each aspect of WHISK's general methodology is shared with other machine learning algorithms, there are characteristics of information extraction that make existing machine learning systems difficult to apply directly. Much of the machine learning research has centered on problems that can be reduced to *classification* problems.

Reducing an information extraction problem to a classification problem can turn each text into an extremely large number of instances. This is particularly true of multi-slot extraction. Each phrase in the text of appropriate length could be considered as a positive or negative instance of a slot filler. This results in $O(mn)$ instances if a slot fill may have up to m tokens and there are n tokens in the instance. For multi-slot extraction with k slots, there are $O(mn^k)$ combinations of phrases that could fill the k slots.

A more natural characterization of an IE problem is to consider the rules themselves as either correct or incorrect when applied to an instance, rather than to classify the instances. The number of instances remains proportional to the length of the text and is independent of the number of slots or the number of tokens per slot filler.

The implementation of many inductive learning systems makes an assumption that a term (or test) operates on an instance independently of other terms, and that the test partitions the instances into disjoint classes. This is a key assumption in top-down decision tree induction such as C4.5 (Quinlan 1993) and CART (Breiman *et al.* 1984) and in the RIPPER rule induction system (Cohen 1996).

For the IE problem that WHISK addresses, it is not possible to tabulate how often a term is associated with correct or incorrect extractions. Whether a term in WHISK's rules is associated with a correct extraction from a given instance depends on the other terms in the rule and their relative position in the rule. It is not clear whether this is an artifact of WHISK's rule representation or whether this is intrinsic to the information extraction problem.

Another anomaly that is closely related to WHISK's rule representation is the behavior of empty rules. Top-down rule induction (Cohen 1996) and top-down decision tree induction (Quinlan 1993) (Breiman *et al.* 1984) assume that an empty rule covers all possible instances and that adding a term or test reduces the coverage of a rule monotonically.

This would be the case if WHISK's rules were true regular expressions. An empty rule would apply an exponential number of ways to every instance, extracting all combinations of zero or more characters in each of the target concept slots. Such a representation would make rule application prohibitively expensive for an on-line application.

As described earlier, WHISK's pattern matching language interprets an empty rule as skipping zero characters and then extracting the entire instance to fill the first slot, leaving remaining slots empty. In nearly all applications, an empty rule will cover *no* correct extractions. It is often the case that any specialization that adds one term to the empty rule will also have zero coverage, giving no guidance for term selection.

This behavior of under-specified rules and WHISK's solution of anchoring the extraction slots seems to be unique to learning text extraction rules. WHISK anchors slot i by adding terms until the rule correctly extracts the first i slots from the seed instance. A hill climbing search is made for the most general rule with all slots anchored. WHISK proceeds then to expend the rule one term at a time to reduce errors on the training set.

7. Conclusions

Information extraction is an enabling technology for systems such as intelligent text retrieval or text routing, software agents whose actions are based on an understanding of text data, and data mining from text. IE systems are knowledge-intensive, however, and require a separate set of text extraction rules for each domain, which makes machine learning an attractive alternative to creating rules by hand. End users who would be daunted by building an expert system can easily annotate training instances.

7.1. The capabilities of IE systems

Information extraction is feasible with current technology primarily because IE systems do *not* attempt in-depth understanding. The goal is to transform the text into a structured format, equivalent to case frames or relational database entries. This in itself limits the problem to well-defined and fairly repetitious information.

A few examples are news stories about companies engaged in corporate acquisitions; lists of products with prices and brief descriptions; entries returned by a searchable web page; or on-line postings about rentals, seminars, or jobs.

IE systems rely on certain characteristics of the text style and information content as well. The relevant information must be expressed in short phrases, where local context is sufficient to identify the phrase as relevant.

The easiest slots to extract are numeric or frequently used labels together with numbers, such as time of day, room number, price. Company name and person name recognizers have become a reliable IE component, which makes proper names an easy field to extract. Long and highly varied phrases are quite difficult to extract, such as a reviewer's opinion of the film or a customer's description of a problem. These are only accessible when they are clearly labeled entries in a form.

As long as these criteria are met, IE systems can be developed for a wide range of text genres, and machine learning can be applied successfully to the knowledge acquisition for new domains. The simplest text genre is structured texts, typically created by text formatting software. The relevant information can be transformed into a table of database tuples. The Wrapper Induction system (Kushmerick *et al.* 1997) learns rules effectively for this type of structured text.

The next level of difficulty is semi-structured text, often telegraphic in style with much of the relevant information in a fairly small number of stereotyped contexts. On-line rental ads, seminar announcements, and job listings are examples of this genre. SRV (Freitag 1998), RAPIER (Califf and Mooney 1997), and WHISK have been developed specifically to process semi-structured text.

The most challenging text style is free text, such as news stories, where there is wide variation in how relevant information is expressed and where judgments about relevancy often require subtle inferences. Text extraction rules are one of several components in an IE system for free text. A previous syntactic analysis step and some form of semantic tagging are performed before the text extraction rules are applied.

Several IE systems have achieved useful performance on free text, although none comes close to human competence and machine learning fall short of hand-coding for text extraction rules. Despite this, there is a need for machine learning because of the effort and expertise required for manually constructing text extraction rules.

7.2. Contributions of WHISK

WHISK is the first system to learn text extraction rules for the full range of text styles from structured to semi-structured to free text. For structured text, WHISK can learn rules with perfect recall and precision, just as the Wrapper Induction system does.

Both WHISK and Wrapper Induction perform multi-slot extraction, transforming a text into a table of relational database tuples. Neither requires syntactic preprocessing for this genre of text.

Each of the systems that learns extraction rules for free text has a variety of strengths and limitations. All of them, including WHISK, must operate in conjunction with a syntactic analyzer and a semantic tagger. WHISK and CRYSTAL (Soderland *et al.* 1995, Soderland 1997) have a more expressive representation than the other systems and are more fully automated. The other systems require hand-crafted rules, hand-selected key word lists, or hand-selected examples to begin the rule induction or require human review of the proposed rules.

WHISK operates at a finer granularity than systems such as CRYSTAL and AutoSlog (Riloff 1993). Those systems identify the syntactic field, such as subject or direct object, that contains the target phrase, but do not identify the target phrase itself. WHISK learns the exact delimiters of the target phrase and requires no later processing to trim away extraneous words.

It is on semi-structured text that WHISK offers a definite advance. WHISK is the first system that learns multi-slot extraction for semi-structured text. Both SRV and RAPIER can only extract single case frame slots in isolation. Like SRV and RAPIER, WHISK requires no syntactic analysis for semi-structured text. WHISK can take advantage of syntactic or structural information, however, if available.

7.3. Future directions

The field of information extraction is fairly young and the application of machine learning to IE is an even more recent development. As new systems come along, they can combine ideas from the various systems described here. SRV's feature detectors are worth emulating, especially those that detect orthographic features such as capitalization and word length.

An adaptive system could restrict its search space on problems where the text is highly structured, and gain the efficiency of the Wrapper Induction system. If the system detects too much variability in the text, it could enlarge the search space to something resembling WHISK or RAPIER's representation. SRV's approach of pooling evidence from several learning systems is also a promising direction. Even if some of the learners are necessarily single-slot classifiers, these can help inform the confidence in an extraction by a multi-slot learner.

Free text IE systems are highly modularized, with machine learning assisting in many areas, from part-of-speech tagging, to probabilistic parsing, to learning extraction rules from example, to coreference resolution. The lower levels of processing have seen more success in applying machine learning than the higher levels. Some parts of a full IE system still require considerable knowledge engineering. The goal remains to create a fully trainable IE system that can be instantiated for a new domain by an end-user.

Acknowledgments

I would like to express my thanks to the helpful feedback from Ray Mooney, Claire Cardie, and the anonymous reviewers. This research was funded in part by Office of Naval Research grant 92-J-1946, by ARPA / Rome Labs grant F30602-95-1-0024, by a gift from Rockwell International Palo Alto Research, and by National Science Foundation grant IRI-9357772.

Notes

1. The database community uses the term semi-structured for what this paper calls structured text.
2. If the pattern has a non-wildcard term immediately following an extraction, this token must be matched as well for the extraction to be bound.
3. A standard Laplacian formula for two classes is $L = (e + 1)/(n + 2)$. This assumes an even distribution of classes and gives L = 0.5 when n=0, e=0. WHISK's formula gives L = 1.0 when n=0, e=0, which better reflects the skewed distribution of correct extractions from all possible rules.
4. http://www.cs.washington.edu/homes/soderlan/Rental_data.tar.gz, Seminar_data.tar.gz, and Jobs_data.tar.gz.
5. The MUC-6 corpus on which this domain is based is available through the Linguistic Data Consortium for a fee, http://www.ldc.upenn.edu/ldc.
6. The implementation of WHISK breaks the stream of input characters into tokens, treating "<FONT SIZE=-1>" as five tokens, "<FONT " "SIZE" "=" "-" "1>". It could just as well have been implemented to treat entire HTML tags as single tokens.
7. or database tuple, in Wrapper Induction's terminology
8. SRV is actually the name of one of the classifiers, the relational rule learner

References

Ashish, N. and Knoblock, C. (1997). Wrapper generation for semi-structured Internet sources. *SIGMOD Record 26(4)*, 8-15.

Breiman, L., Friedman, J., Olshen, R., Stone, C. (1984). *Classification and regression trees.* California: Wadsworth International Group.

Califf, M. E. and Mooney, R. (1997). Relational learning of pattern-match rules for information extraction. *Working Papers of ACL-97 Workshop on Natural Language Learning*, 9-15.

Cohen, W. (1996). Learning trees and rules with set-valued features. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 709-716.

Cohn, D., Atlas, L., Ladner, R. (1994). Improving generalization with active learning. *Machine Learning, 15(2)*, 201-221.

Dagan, I. and Engelson, S. (1996). Sample selection in natural language learning. In Wermter, S., Riloff, E., and Scheller, G. (Eds.), *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing.* Berlin: Springer.

Domingos, P. (1994). The RISE system: Conquering without separating. *Proceedings of the Sixth IEEE International Conference on Tools with Artificial Intelligence*, 704-707.

Fisher, D., Soderland, S., McCarthy, J., Feng, F., Lehnert, W. (1995). Description of the UMass system as used for MUC-6. *Proceedings of the Sixth Message Understanding Conference*, 221-236, San Fransisco, CA: Morgan Kaufmann.

Freitag, D. (1998). Multistrategy learning for information extraction. *Proceedings of the Fifteenth International Machine Learning Conference*, 161-169.

Huffman, S. (1996). Learning information extraction patterns from examples. In Wermter, S., Riloff, E., and Scheller, G. (Eds.), *Connectionist, Statistical, and Symbolic approaches to Learning for Natural Language Processing.* Berlin: Springer.

Kim, J. and Moldovan, D. (1993). Acquisition of semantic patterns for information extraction from corpora. *Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications*, 171-176. IEEE Computer Society Press.

Krupka, G. (1995). Description of the SRA system as used for MUC-6. *Proceedings of the Sixth Message Understanding Conference*, 221-236. San Fransisco, CA: Morgan Kaufmann.

Kushmerick, N., Weld, D., Doorenbos, R. (1997). Wrapper induction for information extraction. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 729-737.

Lewis, D. and Gale, W. (1994). A sequential algorithm for training text classifiers. *Proceedings of ACM-SIGIR Conference on Information Retrieval*, 3-12.

Michalski, R. S. (1983). A theory and methodology of inductive learning, In Michalski, Carbonell, and Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Palo Alto, CA: Tioga Publishing.

MUC-6. (1995). *Proceedings of the Sixth Message Understanding Conference*, San Fransisco, CA: Morgan Kaufmann.

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

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

Riloff, E. (1993). Automatically constructing a dictionary for information extraction tasks. *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 811-816.

Soderland, S., Fisher, D., Aseltine, J., Lehnert, W. (1995). CRYSTAL: Inducing a conceptual dictionary. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1314-1321.

Soderland, S. (1997). Learning text analysis rules for domain-specific natural language processing. Ph.D. thesis, technical report UM-CS-1996-087 University of Massachusetts, Amherst.

Soderland, S. (1997a). Learning to extract text-based information from the World Wide Web. *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining.*

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM, 27(11)*, 1134-1142.