Property-Based Feature Engineering and Selection

by

Noppadon Kamolvilassatian, B. Eng.

THESIS

Presented to the Faculty of the Graduate School
of the University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF ARTS

THE UNIVERSITY OF TEXAS AT AUSTIN
December, 2002

Property-Based Feature Engineering and Selection

APPROVED BY
SUPERVISING COMMITTEE:

supervisor _____

Raymond J. Mooney

_____

Joydeep Ghosh

To my mother.

Acknowledgments

First, I would like to thank my advisor, Prof. Raymond Mooney, whose advice and suggestions are indispensable for the completion of this work. His philosophy on doing scientific research provided me with invaluable lessons that will benefit many aspects of my life. I am also grateful to Prof. Joydeep Ghosh, who kindly read and gave comments that helped improve the thesis.

I am grateful to Un Yong Nam, who tagged additional documents used in the evaluation of this work, and Ruifang Ge who worked on the evaluation of RAPIER with new data. Discussions with Prof. Claire Cardie and Prof. Andrew McCallum helped clarify several issues during the formation of ideas for this thesis. I also appreciate the help of and enlightening discussions with Prem, Joseph, and Aniket, and members of the UTCS machine learning group.

The Fulbright program and Thai-US Educational Foundation provided full support over course of my study in the Masters program.

I have been lucky enough to have many good friends, especially Johnson H. Lee, Tim Mccullough, Prem Melville, Amol Nayate, and Puay Sasipongpairoege, who provided priceless friendships and encouragement over the period that I worked on this thesis and beyond.

My gratitude to my mother, Saowaluk Kamolvilassatian, can never be fully expressed in words. Her love and upbringing form an essential part of my life.

NOPPADON KAMOLVILASSATIAN
The University of Texas at Austin
August 2002

Abstract

Property-Based Feature Engineering and Selection

By

Noppadon Kamolvilassatian, M. A.

The University of Texas at Austin, 2002

SUPERVISOR: Raymond J. Mooney

Example representation is a fundamental problem in machine learning. In particular, the decision on what features are extracted and selected to be included in the learning process significantly affects learning performance.

This work proposes a novel framework for feature representation based on *feature properties* and applies it to the domain of textual information extraction. Our framework enables knowledge on feature engineering and selection to be explicitly learned and applied. The application of this knowledge can improve learning performance within the domain from which it is learned and in other domains with similar representational bias.

We conducted several experiments comparing the performance of feature engineering and selection methods based on our framework with other approaches in the Information Extraction task. Results suggested that our approach performs either competitively or better than the best heuristic-based feature selection approach used. Moreover, our general framework can potentially be combined with other feature selection approaches to yield even better results.

Table of Contents

List of Figures

1. Introduction

1.1 Representational Issues in Machine Learning

"How should we represent real-world problems inside computer programs?" is a fundamental question that has enormous implications for the performance of computer applications. This question is particularly important for computer programs that learn. It is known that using different representations result in significantly different performances in learning systems (e.g., (Mccallum and Nigam, 1998) (Asker and Maclin, 1997)).

From a theoretical perspective, each representation choice creates a different hypothesis space for learning. We know that performance of learning algorithms depend on the size of the hypothesis space and whether the target concept is contained in it (Mitchell, 1997). In particular, a smaller hypothesis space that still contain the target concept would result in better learning performance with the same amount of data (Mitchell, 1997). In contrast, if the hypothesis space does not contain a good approximation to the target concept, no matter how many examples are available, the learning performance will be low.

1.2 Inspiration: Cognitive Science

Human representational constraints most likely affect the way we solve problems. An influential paper in cognitive science shows that we have limited short-term memory (seven plus or minus two memory 'chunks') (Miller, G. A., 1956). We can apply this knowledge into the design of machine learning systems that deal with linguistic information, for example, by limiting the scope of contextual information used. The features that are too far from the point of focus might not be relevant and can be filtered out. This will reduce the hypothesis space and we can still be reasonably sure that it contains the target concept.

This leads us to believe that in many, if not most, learning domains, there are different kinds of features with varying effectiveness when used in learning. This thesis is an attempt to create a framework that embodies this concept. By exploring the notion of feature types, we

can gain new insights into the way we should extract features from real-world data and select them for learning.

Cardie (2000) incorporated human information processing limits found in cognitive science literature as prior in feature weighting for case-based learners. Empirical results on several linguistics tasks suggest that the cognitive-bias approach performs better than an information-gain-based feature selection method known to perform well on several natural language learning problems. The feature selection approach used there, however, did not employ a general framework that can be easily extended beyond linguistic tasks.

1.3 Needed: A New Approach to Feature Engineering and Selection

So far, the process of feature engineering for machine learning is conducted in a more or less ad hoc basis. The researcher or practitioner uses their intuition to determine the kinds of features to be extracted from raw data. In many cases, their intuition is right and good learning performance ensues. In many other cases, the features selected are not optimal for the learning problem at hand. The lack of a well-defined framework for determining the types of features that are more useful might have hindered performance of many learning systems. In the information extraction domain, for example, it is demonstrated in (Freitag, 1998) and (Califf and Mooney, 1999) that the performance of a naïve Bayesian learner is very low. But as we will see in chapter 6, the low performance was the result of feature engineering more so than the algorithm, since we have received much better performance using the same naïve Bayesian algorithm with more careful feature engineering.

Feature selection based on statistical heuristics (for example, in (Mladenic and Grobelnik, 1999)) and wrappers (Kohavi and John, 1998) are useful and do improve learning performance in many cases, but they do not usually help researchers and practitioners gain insights about the kinds of features that should be used for a problem.

If we have knowledge about the kinds of features useful for each problem, we can apply that knowledge to speed up the process of feature extraction. Moreover, we can transfer this

knowledge to other datasets in the same or similar domain that might share representational bias.

In this thesis, we propose a new framework for feature representation based on *properties* of features. We apply the framework to the task of information extraction and demonstrate how knowledge on feature engineering and extraction can be learned and applied. Our experiments suggest that this approach may yield better results than formula-based feature selection. In addition, we demonstrate that the transfer of this knowledge from one domain to another can improve learning performance.

The contributions of this work can be summarized as follows:

1. It proposes a new framework for feature representation based on the *properties* of features. The framework helps machine learning practitioners gain insights into the kinds of features that are or are not useful for each learning domain.
2. It presents a method for learning and transferring knowledge on feature engineering and selection.
3. It demonstrates that applying knowledge on feature engineering and selection based on the proposed framework improves learning performance in the information extraction domain over not using feature selection, and in some cases, over using the best feature selection heuristic we experimented with. Moreover, it shows that feature engineering knowledge can be beneficially transferred across domains.

1.4 Our Proposal: A Framework for Property-Based Feature Engineering

This section describes the overall framework of Property-Based Feature Engineering that we propose, which is also depicted in Figure 1.1.

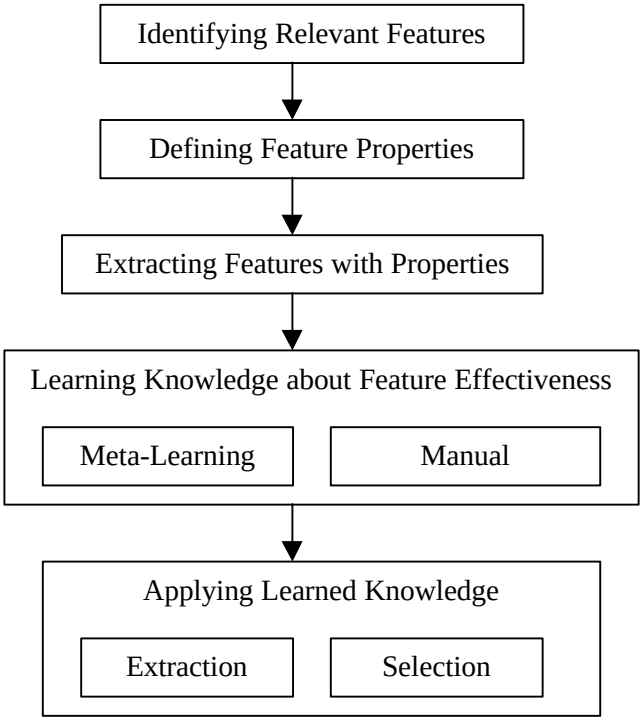

Figure 1.1 The Process of Property-Based Feature Engineering

The following is a simplified description of the process we use in experiments. More details of the process will be discussed in subsequent chapters.

1. From raw data, identify features that are potentially relevant to the task.
2. Study the features and define the parameters that characterize each feature type. We call these parameters *feature properties*.
3. When extracting features, collect and store feature properties with them.
4. Learn knowledge about feature effectiveness

   I. Meta-Learning Method

      A. Experiment with different feature selection heuristics. Evaluate the results. The best heuristic will be used to measure feature effectiveness in Step 4.I.B.

B. Generate the Features meta-dataset, in which each instance represents a feature. Each instance in the Features meta-dataset consists of properties of the feature and a measure of its effectiveness based on the best heuristic from Step 4.I.A.

C. Employ a rule learner to detect salient patterns of feature effectiveness based on feature properties.

II. Manual Method

A. Experiment with different sets of features by using the features with certain properties. Record results.

B. Repeat Step 4.II.A several times until the learning results are satisfying. (We can use hill-climbing or beam search strategies to revise the values of feature properties used to avoid combinatorial explosion.)

5. Apply learned knowledge about feature effectiveness.

I. On Feature Extraction. This knowledge can be used to selectively extract only features that are likely to be highly effective. We can also call this method *Feature Pre-Selection*.

II. On Feature Selection. This method deals with the case when all features have been extracted. We just use this knowledge (in the form of rules) to filter out features. This method is the same as the way standard feature selection works.

III. Across Domains. Many learning domains share learning bias. A domain with limited data would benefit if knowledge about feature engineering is transferred from another domain, where data is less scarce.

As we will see in the following chapters, the property-based feature engineering approach helps machine learning practitioners to form new insights about the kinds of features useful for each domain, enables transfer of knowledge on feature engineering across domains, and can improve learning performance over other feature selection methods.

1.5 Organization of the Thesis

The rest of the thesis is organized as follows: Chapter 2 presents background knowledge on the role of bias in machine learning, feature engineering and selection, transfer of learning, meta-learning, and information extraction. Chapter 3 discusses the use of naïve Bayesian algorithm for information extraction, which is used as the case study for our feature engineering framework. Chapter 4 explains and gives an example of the property-based feature engineering framework that we propose. Chapter 5 describes how knowledge on feature engineering and selection can be learned and transferred. Chapter 6 discusses experimental evaluation on different feature selection approaches for information extraction and analyzes the results. Potential extensions and applications of the work are presented in chapter 7. Finally, chapter 8 gives conclusions from the work.

2. Background and Related Works

Our work touches on many aspects of machine learning. This chapter reviews the related works that helped form the foundations for us to build on.

2.1 Role of Bias[1] in Machine Learning

An introduction to bias selection by desJardins and Gordon (1995) provides a search-based framework to the problem. The normal machine learning is performed at the base level, which is affected by a set of biases. We can learn to select the appropriate set of biases at this level. The learning for bias selection can still be characterized by a set of biases--this time at a higher-level, and continue ad infinitum. Two types of biases are defined: representation bias, which defines the hypothesis space, and procedural bias, which determines the traversal order within this hypothesis space. In general, representational bias is restrictive--no concepts can be found that lie beyond the representational language used, while procedural bias is preferential--it affects the speed of learning and the accuracy of concepts learned for a particular set of training examples, but does not limit the performance of learning algorithm in all possible cases.

In their "Inductive Policy", Provost and Buchanan (1995) extend the model of inductive bias to include rule and example spaces, the methods for searching the spaces, the relations between the examples and the methods for searching the rule space, and the relations between the rules and the methods for searching the example space. They analyzes the existing bias selection systems according to this model. They then propose a framework for bias selection that deals with all six bias categories in the model. Given an inductive policy, or a bias selection strategy, as an input, tradeoffs between accuracy and the cost of errors can be made within the framework.

---

[1] The term ‘bias’ used in this thesis refers to representational and procedural biases in machine learning. It is different from the concept of bias in statistics.

Brodley (1995) describes a rule-based method to select bias recursively from feedback of the learning process. The heuristic rules are developed by manual trial-and-error and encoded to the system's knowledge base. Three base representation biases are used--univariate decision tree, linear discriminant function, and k-NN classifier. The system either selects one of the three base representation or forms a hybrid of them. The results obtained show that classification accuracy of the representation bias selected is at least as good as the best of the primitive learning components, and sometimes significantly better.

For other related works, Cohen (1990) provides a framework for analysis of changes in the representation of training examples in concept learning. Bias selection in the inductive logic programming context is explored by Stahl (1995) and Ade et al (1995).

2.2 Feature Selection and Weighting

Feature selection is an important problem in machine learning. In many real-world learning domains with a large number of features, feature selection can help improve accuracy, speed up training, and reduce the complexity of the concepts learned (e.g., Wettschereck et al, 1997; Koller and Sahami, 1996).

Most feature selection methods in the literature fit into either the *filter* or the *wrapper* approach. The filter methods analyze the data and select seemingly relevant features independent of the learning algorithm used. In contrast, the wrapper approach takes feedback from learning on a holdout validation set and picks a feature subset that performs well on this set. There is substantial evidence that the wrapper approach generally yields better prediction accuracy than the filter methods since it also takes the inductive bias of the particular learning algorithm used into account (Aha and Bankert, 1995; Wettschereck et al, 1997; Kohavi and John, 1998). However, a cost is paid in terms of much higher computational time. The wrapper approach can be orders of magnitude slower than the filter methods.

We can think of feature selection as a specialization of feature weighting where feature weights are limited to zero or one. Feature weighting can sometimes perform better than

simple feature selection especially when features may not be equally important (Wettschereck et al, 1997). Kohavi et al, (1997) report interesting evidence that under the wrapper framework for nearest-neighbor algorithms, the number of possible weights allowed should not be too large or the classification error may increase due to overfitting and the presence of higher variance.

Feature selection and weighting is most frequently studied under the context of case-based learning since it is well-known that the presence of irrelevant features or inappropriate feature weights can have significant adverse impact on prediction accuracy for this class of learning algorithms.

Koller and Sahami (1996) proposes a filter method for feature selection based on information theory. The method uses cross-entropy to keep the conditional probability of the class given the features close to the original distribution. Since the algorithm does not take the learning algorithm's inductive bias into consideration, however, improvements in accuracy are sometimes modest.

John et al (1994) argue for a need to distinguish further among relevant features. In particular, they categorize features as *strongly relevant* or *weakly relevant*.. In short, strongly relevant features cannot be excluded in any learned hypotheses without suffering accuracy loss and *weakly relevant* features are those that sometimes can contribute to prediction performance but dispensable in other cases. In addition, the paper proposes a wrapper approach to feature selection which takes feedback from induction performance over hold-out validation set to either select a feature into the subset (forward selection) or eliminate one from it (backward elimination). Their empirical results using wrapper with C4.5 (Quinlan 1992) and ID3 do not show significant improvement in accuracy except in few cases (where the performance without feature selection is relatively low). The most tangible benefit is in getting simpler decision trees as a result of having fewer features.

Kohavi and John (1998) builds on the previous work of John et al (1994) by employing best-first search to explore the feature subset space more throughly and *compound operators* to speed up search. Empirical evaluation comparing this improved wrapper and the feature

filtering algorithm Relieved-F on 8 real-world datasets and 6 artificial datasets using C4.5 and Naive Bayesian algorithms indicates that the wrapper approach yields significantly better accuracy in most cases. Moreover, the wrapper algorithm usually selects a smaller feature subset, which results in more comprehensible learned concepts. The main drawback of the wrapper approach is that it is significantly more computational intensive than filtering.

Work by Andrew Ng (1998) using the wrapper model has led to an algorithm that performs better than the standard wrapper in the case that the number of relevant features are small compared to the number of all available features. In fact, the paper theoretically proves that if the search for the relevant feature subset can be performed optimally, the upper bound for sample complexity is only logarithmic in the number of features. In other words, under the assumption that there are relatively few relevant features, the new algorithm can tolerate having a number of irrelevant features exponential in the number of training examples. Empirical results shown in the paper, though promising, are limited to artificial datasets specifically generated to test the algorithm.

Aha and Bankert (1995) evaluates variants of forward and backward sequential feature selection methods for a cloud pattern classification task. The dataset used has relatively few examples and contain numerous features. The comparison using a nearest-neighbor algorithm as the classifier suggests that 1) feature selection improves accuracy on this task, 2) the wrapper approach performs better than using the Calinski-Harabasz separability index as feature filter, and 3) beam search for feature selection gives better accuracy than hill-climbing.

Wettschereck et al (1997) reviews different feature weighting methods in the context of standard lazy learning algorithms. The paper proposes a framework that distinguishes feature weighting methods based on the dimensions of bias (performance-based or preset), weight space (continuous or binary), representation (given or transformed), generality (global or local), and knowledge (poor or intensive). Their empirical evaluations suggest that performance-based methods (i.e., wrapper approach) generally require less pre-processing, handle feature interaction better, and has faster learning rate. Also, the experiments indicate

that continuous weighting methods tend to perform better than feature selection (i.e., binary weighting) for tasks where some features are useful but less important than others.

Cardie (2000) takes a distinctive approach to this problem by employing human information processing limits found in cognitive science literature as prior in feature weighting for case-based learners. A combinatioin of different cognitive biases are selected either greedily or exhaustively using wrapper-based approach. Empirical results on several linguistics tasks suggest that the cognitive bias approach performs better than an information gain-based feature selection method known to perform well on several natural language learning problems. Her result also suggests that exhaustive search for the best combination of cognitive biases does not give significantly better prediction accuracy than greedy search.

2.3 Learning to Learn and Transfer of Learning

An algorithm that possesses the ability of learning to learn can transfer knowledge or experience that it learns in some tasks to improve learning on other tasks (Thrun and Pratt, 1998). The algorithm may be able to improve its generalization accuracy or require fewer examples to reach a certain accuracy when the transfer of learning occurs. Several approaches to learning to learn have been investigated by researchers, including learning representations, distance functions, invariances, parameters, learning rate. The representations that are widely used include neural networks and memory-based. Some of these works are described in the following paragraphs.

In "Multitask Learning", Caruana (1998) shows that back-propagation neural networks can transfer information learned from one task to help others. They can also discover the relationships between tasks without explicit signals on task relatedness. The work shows that multitask learning also applies to K-nearest neighbor and kernel regression.

Thrun (1998) discusses that learning to learn can be viewed as learning bias. The work demonstrates learning to learn using memory-based and neural networks representations. Distance function is modified through learning in the memory-based approach and

representation is adapted through learning in both approaches. The algorithms were tested in an object recognition task, where learning to learn exploits the information from support sets (support sets comprise of images from other objects--not the one they try to recognize). Learning to learn mostly outperforms the normal approach, especially when there are few training examples.

On the theoretical side, Baxter (1998) analyzes and proves the theorems on learning many tasks from the same environment--or bias learning--using two models, empirical process discovery and hierarchical Bayes model.

Silver and Mercer (1998) introduces a measure of task relatedness which allows neural networks to select related tasks for the tranfer of knowledge. Another approach explored by Thrun and O'Sullivan (1998) is clustering of tasks into relatedness hierarchy. In the case that tasks are not appropriatedly related, selective transfer from the most related tasks cluster reduces the amount of training data needed substantially.

Transfer of learning is also discussed from the cognitive science perspective in Robins (1998). Results emerged from research in cognitive psychology include the importance of surface similarity and the ability of subjects to map from surface content to its structure.

2.4 Meta-Learning

Meta-learning has been used to automatically predicts the algorithm that best matches the learning problem at hand. For this purpose, learning tasks are usually characterized by a set of (meta-)attributes. Different algorithms are tested with the learning tasks to create training examples for meta-learning. A meta-learning algorithm can then be applied to the new task (defined in terms of meta-attributes) to predict the algorithm that will perform best on the task. (King et al, 1995)

A more sophisticated strategy investigated by Pfahringer et al (2000) uses results of simple learners to determine the location of the learning task in the possible space of learning problems--in other words, to 'landmark' the learning task. Efficient learners, or landmarkers,

from different learning paradigms--a naive bayes learner, the C5.0 tree learner, and a linear discriminant learner--are chosen to predict relative performance of more sophisticated learners--the discriminant tree learner LTREE (Gama, 1999), boosted C5.0 trees, the rule learner RIPPER (Cohen, 1995), and a nearest neighbor learner. The authors found that landmarking (on its own) outperforms meta-attributes-based approach. They also tried combining both approaches by taking landmarking result as an attribute and apply the same meta-learning procedure described in the above paragraph.

Another thread of work using meta-learning deals with learning from multiple subsets of data. For some real-world data mining applications, the whole dataset may not be able to fit into computer's memory at once. An alternative is to divide the data into subsets, learn from each subset separately, and combine the results from each individual classifier to yield the final prediction. (Chan and Stolfo, 1993) A simple-minded approach, however, may reduce the learner's accuracy. (Chan and Stolfo, 1995) takes predictions from individual learners for an example together with other information as a meta-learning instance, and learn the relationship between these predictions and the correct prediction. The individual predictions are then combined using a variety of strategies taken into account this learned relationship. Empirical results show that this produces better final accuracy than other simpler methods such as weighted majority voting or Bayesian combination.

2.5 Information Extraction

"Information extraction is the problem of filling out pre-defined structured summaries from text documents." (Freitag, 1998) In recent years, there are a number of machine learning algorithms developed for the task of information extraction.

Muslea (1999) surveys extraction patterns produced by machine learning systems and discusses their differences.

The CRYSTAL system described by Soderland (1995) automatically creates "concept nodes", or slot-filler patterns that represent groups of related information (concepts), from

training examples. First, each concept node definition is created from a single training instance. The system then generalizes by unifying several concept nodes together.

Soderland (1997) extends CRYSTAL to deal with textual information from the Web by introducing Webfoot, a preprocessing step that uses text-layout tags to group text into related fragments. The generic HTML segmenter implemented in Webfoot is based on HTML tags and can partition texts into four levels of granularity.

SRV developed by Freitag (1998) learns relational extraction rules top-down from examples. The representations of possible rules are more expressive than most other systems and can take into account field length, absolute and relative position, existensial predicates, link grammar, and Wordnet information. However, additional linguistics information does not seem to help much in many fields tested in the seminar announcement domain. SRV compares favorably with rote learning and naïve bayesian algorithms.

The RAPIER system developed by Califf and Mooney (1999) uses inductive logic programming (ILP)-inspired algorithm to induce extraction patterns bottom-up from examples. The algorithm has high learning rate and high precision. The system can use syntactic, semantic, and field length information, in addition to words. Plain words, however, seem to account for much of RAPIER's performance. RAPIER competes well with other similar systems (SRV and WHISK) and outperforms them in several fields from the seminar announcement domain.

WHISK (1999) induces a special type of regular expressions with two components: one for context description, another for exact delimiters of the phrase to be extracted. WHISK's rules extract multiple slots at once, while SRV and RAPIER extracts a single slot with each pattern.

Freitag (1998a) offers strategies to combine predictions from many information extraction methods (rote learning, naïve bayes, and SRV are used). The combination technique is based on regression. The multistrategy technique used improves the final result over using SRV in isolation, but the improvement is not large. The result from the combined extractor seem to

rely mostly upon SRV, the best of the three techniques tested. This could result from the fact that SRV is substantially better than other two techniques. It would be interesting to see if the result will improve much further from pure SRV if the multistrategy system include other advanced systems, such as RAPIER and WHISK, into its components.

Riloff and Jones (1999) uses a mutual bootstrapping technique to learn both semantic lexicon and extraction patterns simultaneously. A second level of bootstrapping is added to maintain only the most reliable lexicon patterns found.

McCallum et al (2000) employs maximum entropy Markov models, which is a probabilistic tool adapted from Hidden Markov models, to perform text segmentation. The model allows the use of overlapping features (such as word, capitalization, formatting, part-of-speech) to be represented in Markovian observation sequence. The model might be useful for information extraction tasks as well.

RoboTag described by Bennett et al (1997) learns to extract information using decision trees. For each field, RoboTag builds two decision trees; one to predict begin tags, another to predict end tags. The results of these classifiers are combined using a tag matching algorithm to produce complete fields.

Freitag and Kushmerick (2000) uses a boosting algorithm to induce extraction patterns. Each extraction pattern is simple and has low-coverage, but boosting creates new patterns to compensate the weaknesses of existing patterns. The approach is shown to work well in several domains and is competitive with many other leading algorithms (e.g. SRV, RAPIER, HMM).

3. Naïve Bayesian Learning for Information Extraction

In this chapter, we explains the use of naïve Bayesian learning for information extraction. The problem is used as the testbed for the feature engineering framework that we propose, which is described in the next chapter. The reason of choosing this task is that it is complex enough that the most straightforward feature representation might not yield good performance (Freitag, 1998b), while simple enough to be the first case study of a new framework. For an introduction to the domains and datasets we applied this learning system to, see Section 6.1 Domain and Task Descriptions.

3.1 Problem Definition

We formulate information extraction as the tasks of tagging beginning and ending positions of fields in a document. The task is reduced to the classification of whether to insert a tag at each potential insertion point in the document, and if that is so, what tag should be inserted.

As an example, the text:

> PHP/ASP Web developer seeking job in London

contains potential insertion points as indicated by the mark • as follows:

> •PHP•ASP•Web•developer•seeking•job•in•London•

(Note that slashes are considered part of an insertion point.) The problem is to extract useful features from the surrounding context and determine whether and what tag to be put at each point.

The problem does contain a good deal of ambiguity, since several tagging schemes might be equally acceptable. In a document in the dataset that we use, this is manually tagged as:

<title>PHP/ASP Web developer</title> seeking job in London

However, it is also reasonable to tag the line as:

<language>PHP</language>/<language>ASP</language> <title>Web developer </title> seeking job in London

Therefore, even among humans, perfect match is hard to achieve (except when all taggers follow the same strict rules.) This makes the problem more interesting and also harder for machine learning to deal with.

To measure the performance of machine learning systems in this task, however, we usually take the field boundaries tagged by humans as the gold standard, and compare the result from automatic tagging against them. This usually understates the performance of machine learning systems in the eyes of the third-party observer, since both manual and automatic tagging can be equally acceptable in practice as discussed above. In this work, we follow the convention used in research community and measure our system against manually tagged datasets. The standard measurements for information extraction are precision and recall defined as follows:

precision = # of correct fillers extracted / # of fillers extracted
recall = # of correct fillers extracted / # of all manually tagged fillers

The term 'filler' denotes the text between open and end tags. To be considered correct, exact matches with manual tagging at both begin and end boundaries are needed. Precision measures the proportion of predictions by the learner that turn out to match manual tags in the test set. Recall measures the proportion of manual tags that can be detected by the learner. In addition, another measure was defined in MUC conferences (DARPA, 1992) to summarize

the performance of information extraction systems with a single number. It is defined as follows:

$$F = (ß^2 + 1) PR / (ß^2P + R)$$

where P is precision, R is recall, and ß is used to weight the measure to prefer better precision or better recall. When we want to weight precision and recall equally, we set ß to 1 and call it the F-1 measure. F-measure is usually a better tool to summarize precision and recall than a simple average . With a simple average, a system with 1% precision and 99% recall would be considered equally desirable to another system with 50% precision and 50% recall, which is generally considered to be a much more useful achievement. On the F-1 measure, however, the first system will however get 1.98%, while the second gets 50%, which better reflect their usefulness.

3.2 Algorithms

When we define the information extraction problem as the decision to insert tags in the document, we can basically divide it into two subproblems:

a) How to classify whether and what tag to be inserted at each point.
b) How to determine most likely field boundaries using information from the classifiers.

Note that solving sub-problem b) would be easy if the classification result from sub-problem is perfect. Since we cannot solve sub-problem a) perfectly, the issue of determining most likely field boundaries is not trivial. We will now describe the algorithm we use to solve each problem in turn.

To classify whether and what tags should be inserted at each point, we employ an algorithm based on the Bayes equation (Mitchell, 1997):

$$P(Tag \mid InsertPo\,\text{int}) = \frac{P(Tag)\prod_i P(Feat_i \mid Tag)}{P(InsertPo\,\text{int})}$$

which allows us to compute the probability that a tag is present at an insertion point given the features that appear in its context. The equation contains an important assumption that all

features are independent from one another, which is obviously not true in the text domain, as some words tend to occur together and we can better predict the existence of a word based on knowledge of whether the other word is present. Despite the problem with this 'naive' assumption, the simple Bayesian algorithm has been shown to work well in many real-world domains (Domingos, P. and Pazzani, M., 1996).

In our domain, it is possible that several tags are present at an insertion point, therefore we decided to employ several binary Bayesian classifiers, one for each tag type, for tag predictions. Each classifier computes the statistics:

$$\frac{P(Tag \mid InsertPo\,\text{int})}{P(\sim Tag \mid InsertPo\,\text{int})}$$

which is the ratio of probability that the tag is there given the insertion point (or more precisely, the contextual information surrounding the insertion point). We call this specific odds ratio the *confidence ratio*. Each insertion point is characterized by several features extracted from its context. All features used in our system are binary. The features we extract from the context include words, word distance, token length, and capitalization. The features will be described in detail in Chapter 4.

From the two equations above, we have:

$$\frac{P(Tag \mid InsertPo\,\text{int})}{P(\sim Tag \mid InsertPo\,\text{int})} = \frac{P(Tag)\prod_i P(Feat_i \mid Tag)}{P(\sim Tag)\prod_i P(Feat_i \mid \sim Tag)}$$

We compute the confidence ratio above using the features that are present at each insertion point only; features that are absent are ignored. This is a very close approximation to the full model which uses the probability of all possible features. This makes the computation much more efficient. It works because in our case both $P(\sim Feat \mid Tag)$ and $P(\sim Feat \mid \sim Tag)$ are very close to 1.0. The ratio of these two numbers are therefore very close to 1.0.

$P(Tag)$ is computed as follows:

$$P(Tag) = \frac{\#Tag}{\#InsertPo\,\text{int}}$$

where #Tag is the number of the given tags found in the training set and #InsertPoint is the number of insertion points in the training set.

Each P(Feat$_i$|Tag) is computed using m-estimate (Mitchell, 1997) to avoid the value zero which will dominate the equation. Therefore, the conditional probability for feature given tag is defined by:

$$P(Feat_i \mid Tag) = \frac{\#(Feat_i \wedge Tag) + priorEst * equiSampleSize}{\#Tag + equiSampleSize}$$

where $P(Feat_i \wedge Tag)$ is the number of the feature that appears together with the tag in the training set, *priorEst* is the prior estimate used and *equiSampleSize* is equivalent sample size. The same method is used to compute $P(Feat \mid \sim Tag)$. We use the equivalent sample size of 1.0 and prior estimates of 0.00002 for computing $P(Feat \mid Tag)$ and 0.00008 for $P(Feat \mid \sim Tag)$. These estimates are based on the statistics we observed from a dataset. However, varying these priors by an order of magnitude does not make a difference in the classification results in our experiments (since the value of equivalent sample size is also low).

After the confidence ratio $\frac{P(Tag \mid InsertPoint)}{P(\sim Tag \mid InsertPoint)}$ for every tag is computed, only the tags that have this ratio equal to or higher than the *prediction threshold* would be selected. This prediction threshold is a parameter in the system and can be easily adjusted in command-line parameters. The natural threshold is 1.0, since it predicts that the tag is more likely to appear than not. Only the top three tags that have confidence ratios higher than the prediction threshold are ultimately selected. These top picks will be used by the tag matching algorithm described below. For many insertion points, no tag will be predicted since the ratios for all tags are lower than the threshold. Another advantage for having prediction threshold is that we can conveniently trade-off between precision and recall by raising or lowering the prediction threshold. In particular, lowering the threshold will usually result in higher recall and lower precision. Raising the threshold usually has an opposite result.

After we have predicted tags using the classifiers, we filter out non-matching tags (i.e. with only open tag or end tag) by looking for a matching tag within a window. The size of this window depends on the length of fillers found in the dataset (In our experiments, this is set to 3).

We may still have several filler candidates for each insertion point, we now try to rank the field candidates based on a scoring method which takes into account the frequency that the filler text appear in the training set and the length of filler. The scoring formula is:

filler score = frequency score + length score

frequency score = 1000 * (#fillers in training set + #virtual examples) /
tags in training set

length score = (max length difference – length difference)

The number of virtual examples used is 0.5. The maximum length difference allowed is 5.0. The filler selected must have the top score among all candidates present and also the score must be 0.5 or higher. Basically, these formulas employ statistics of filler frequency and length in the training set to choose between top filler candidates from classifiers. This scoring method is inspired by the method used in RoboTag (Bennett et al, 1997). Note that we are making a simplifying assumption that there is only one starting tag for each insertion point, which is not always true in the domains we explore. However, this assumption is mostly applicable and does not adversely affect our main purpose, which is comparing feature selection methods, we therefore decided to use it.

3.3 Discussions

When we consider the equation for computing confidence ratio based on naïve Bayesian formula, we see that the statistics for every single feature may influence the final decision in the level unproportionate to its real relevance. In the case where many features are present or there is limited training data, it is likely that there will be some flunctuations in the statistics P ($Feat_i$|Tag) which may easily cause incorrect classifications. In such cases, it is very important that we select only relevant features to be included in the classification decision.

This contrasts with the systems based on relational rule learning (e.g. (Freitag, 1998) (Califf and Mooney, 1999)), for example, which handle features separately, or systems based on decision trees (e.g. (Bennett et al, 1997)) which has feature selection built into the learning algorithm.

A prior information extraction system based on naïve Bayesian learning described in (Freitag, 1998) and (Califf and Mooney, 1999) performs badly in moderately complex tasks. We believe that this is mostly due to the lack of appropriate feature engineering and selection. The experiments that demonstrate this point are discussed in Chapter 6.

Another point that has been raised against naïve Bayesian algorithm for information extraction is that the independence assumption is obviously wrong. This is true, but it does not necessarily hurt performance. It is shown that for a wide range of applications, naïve Bayesian classifier works well despite the violation of this assumption (Domingos, P. and Pazzani, M., 1996).

A major advantage of naïve Bayesian learning is its classification speed. Our system is fast and scales linearly with the size of document and dataset because of its use of naïve Bayesian learning.

4. Property-Based Feature Engineering

en·gi·neer·ing *n.* 1 a. The application of scientific and mathematical principles to practical ends such as the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems.

The American Heritage® Dictionary of the English Language, Fourth Edition

We define feature engineering as the process of designing features to be extracted from raw data, and the construction[2], selection, and weighting of features for effective use in machine learning. It is recognized in (Asker and Maclin, 1997), for example, that in real-world domains, feature engineering often turns out to be more important than the algorithm used for learning.

In this chapter, we outline a new approach to feature design which explicitly defines *feature properties*. A feature property characterizes an aspect of the feature. We then show how this framework can be applied to the domain of textual information extraction. This framework directly addresses two important aspects of feature engineering, namely, feature extraction and selection. It can also be easily extended to handle feature weighting.

4.1 The Framework: Features and Feature Properties

The proposal here is to recognize explicitly that features we extract and use in learning contain some distinctive aspects. We define feature properties to characterize aspects of each feature. A feature property may apply to some features and not to some others.

---

[2] Feature construction involves the creation of new features from existing ones. For example, if raw data contain heights and weights of hospital patients, it might be desirable to construct a new feature, the ratio of height per weight, which can readily capture some disease patterns better than either height or weight.

The hypothesis we have in mind is that

If patterns in a domain can be captured by some kinds of features but not others, learning performance would improve when we employ only the kinds of feature useful for the domain.

The word ‘employ’ above implies four different meanings: 1) feature extraction (from raw data), 2) feature construction, 3) feature selection, and 4) feature weighting. In this thesis, we only address feature extraction and feature selection.

This hypothesis is shown to be true for the domain of textual information extraction in Chapter 6.

4.2 A Case Study: Textual Information Extraction

As with the study of genomics, we use a Textual Information Extraction task as the *drosophila* domain before conducting further studies with more complex organisms like those of mammals and humans.

We start out by identifying different feature types that are potentially useful for the task. This might include the frequency of a given token within a certain window, the distance of a given token from the potential insertion point, the capitalization of nearby tokens, for example. Some less obvious but potentially relevant features are also included, such as the length of the previous line (if zero, it may signify that the current line is a new section heading), the distance of the insertion point from the line start (a certain tag might often be placed one token after the line start, for example).

We characterize every feature based on two principal properties: Class and ObjectType. The Class property defines the ‘feature class’ which indicates the major aspect of features that also influence other properties. Class indicates whether a feature describes length, distance, frequency, or capitalization of context. ObjectType indicates the kinds of objects that the feature describe. It can be a character, a token, a line, or a special type (which means that it is not any of the former). Other properties may be present or not depending on the Class of the

feature. To distinguish other properties from Class and ObjectType, we call them ‘parameters’. The set of Class, ObjectType, and parameters together uniquely define a feature.

Every feature used in the system is binary (i.e. the value of the feature can only be true or false). This design decision allows us to treat all features uniformly in the information extraction task. Uniform treatment of features should minimize the possibility that experimental results are affects by factors other than feature engineering and selection, and therefore lead to a better scientific study.

Below is the definition of feature Class, ObjectType, other properties, with examples.

1. Frequency

Definition: Whether *a specific character/token* appears in the symmetric window of *a given size* surrounding the insertion point *a given number of times*.

Parameters: Content, Window Size, Number of Times

Example:

```
[Class=frequency, ObjectType=token, Content=Developer, Window Size=6, Number of Times=1]
```

The token ‘developer’ appears once in the window of size 6 surrounding the insertion point.

Note: The system uses the variable max window size is defined to limit the scope of feature extraction and reduce computation. It can be adjusted in the system’s parameter.

2. Length

Definition: Whether a token/line *at a given distance* from the insertion point has *a certain length***.**

Parameters: Distance, Length (with natural units, i.e. characters for a token, and tokens for a line)

Example:

```
[Class=length, ObjectType=line, Distance=-1, Length=4]
```

The previous line is four tokens long.

Note: Negative values for Distance mean distance to the left of the insertion point; positive ones indicate distance to the right.

3. Distance

Definition: Whether *a specific character/token* is at *a certain distance* from the insertion point

Parameters: Content, Distance (in the unit of chars for a char object, unit of tokens for a token object)

Example:

```
[Class=distance, ObjectType=token, Content=Subject, Distance=-5]
```

The token 'Subject' is five tokens to the left of the insertion point.

Notes:

- Unit of distance varies according to the object. Distance to a char is the number of characters. Distance of a token is the number of tokens.
- The variable max distance is defined to limit the scope of feature extraction and reduce computation. It can be adjusted in the system's parameter.

4. Distance from Line Start

Definition: Whether the insertion point has *a certain distance to the right of the line start*

Parameters: Unit (characters or words), Distance from Line Start

Example:

```
[Class=distance from line start, ObjectType=special, Unit=characters, Distance from Line Start=4]
```

The insertion point is four characters to the right of the line start.

Notes:

- We define the new feature property 'distance from line start' instead of using the feature property 'distance' in feature Class 3, since they have different meanings. The property 'distance' denotes the distance of a given object to the insertion point, while 'distance from line start' indicates the distance of the insertion point itself to the line start.
- The ObjectType value for this feature Class is always equal to Special.

5. Capitalization

Definition: Whether the token at *a given distance* from the insertion point has *a certain capitalization form*.

Parameters: Distance, Capitalization Form (possible values: first upper, all upper, all lower, mixed, others)

Example:

```
[Class=capitalization, ObjectType=token, Distance=2, Capitalization Form=all lower]
```

The second token to the right of the insertion point is composed of all lowercase characters.

Notes:

- 'others' applies whenever there are any non-letter symbols in the token.
- 'first upper' applies when the first character of the token is a capital, while all following characters are lowercase.
- 'all upper' and 'all lower' are self-explanatory.
- 'mixed' applies when the token does not meet any of the above cases (i.e. contain only letters but with mixed lowercase and uppercase ones).

We can see that other feature classes, including those that use syntactic and semantic knowledge of the language could be defined as well. They can be important for the domains that depend heavily on linguistic knowledge. Our experiments in Chapter 6 use only the feature classes defined above.

4.3 Discussions

Our effort in the previous section is to include features of several kinds and granularity from raw text data. We still use some bias based on knowledge of the English language in designing these features, as clearly shown in the feature class 'Capitalization', for example. The bias is also present at the tokenization process itself. For example, some other languages (such as Thai) may not use space as the word boundary, and tokenization based on space would be misleading.

It is possible in theory to ignore much of these biases and construct features using only very primitive knowledge (such as knowledge that English words are composed of characters and the way characters are written), just like what an alien from another planet (computers can be considered alien beings) might do to learn a human language from text without teachers or other learning materials. In practice, however, computational time and sample size needed in learning would be overwhelming. Therefore, some bias at the basic level would still be necessary.

Even when some bias is used, it is still possible to construct many different kinds of features. It is important that we do not overlook all potentially useful possibilities. Real-world domains are complex and in many cases it is counterproductive to determine what kinds of features would be useful beforehand. The procedures we propose in the next chapter should help us determine this more effectively based on real-world data.

5. Learning and Transferring Feature Engineering Knowledge

In this chapter, we describe how feature engineering and selection knowledge can be learned based on the property-based feature engineering framework. The learned knowledge is in the form of decision list, the decision list can then be applied to select features from the test data in the same dataset, or another dataset that shares representational bias.

5.1 Learning Decision Lists

A decision list is an ordered list of conjunctive rules. Our goal is to predict the patterns of feature properties that associate with effective features. Decision list is selected because it is easy for humans to comprehend, therefore helping humans to gain insights into the kinds of features that are useful.

To learn the feature engineering decision list, we first need to generate datasets about features and feature effectiveness. All potentially relevant features are extracted from training documents in the dataset and a measure of feature effectiveness is applied to each of them. Each feature is represented as one instance in the Features dataset.

The Features dataset can be considered a meta dataset, since it does not present the real learning task, but represents a task *that indirectly affects* the real learning task. The feature effectiveness measure identifies the features that are more relevant in classification. We chose Odds Ratio for this purpose, as it is one of the best feature selection heuristics when naïve Bayesian classifier is applied and the domain has highly unbalanced class distribution (Mladenic and Grobelnik, 1999). The class distribution in our domain is highly skewed toward negative for binary classifiers, since most tags do not appear in most insertion points. Also, our own experiments comparing the performance of several feature selection heuristics in the domains we explore confirmed that Odds Ratio is the best heuristic for our purpose.

A few instances from the feature and feature effectiveness dataset (from now on will be called the Features dataset for brevity) is shown below.

```
</area>,3,2,230,?,?,3,?,?,?,?,-2.4257
</area>,1,2,858,10,1,?,?,?,?,?,-1.1621
</area>,1,2,4428,6,1,?,?,?,?,?,1.2585
```

Each line in the Feature dataset represents the properties of one feature. Notice that there are several missing values for each instance. This is because only some feature properties are applicable for each feature Class. The properties of each instance (which represents one feature) are delimited by commas and are list in the following order: Tag String, Feature Class, Object Type, Content ID (each number uniquely represents a content string), Window Size, Freq, Distance, Distance from Line Start, Length, Capitalization, Unit Type, and Odds Ratio.

We generate the Feature dataset in the format that can readily be used by the learning system WEKA (Witten and Frank, 2000). The learning algorithm we choose is PART (Frank and Witten, 1998), which is a recent decision list learner implemented in WEKA. Other types of rule or decision tree learners can potentially serve the same purpose.

We cannot feed the dataset above to PART directly. There are two issues here. First, PART can handle only nominal attributes and therefore all numeric attributes need to be discretized. Second, the number of instances in the Feature dataset is very large and cannot be handled with PART as implemented in WEKA with the computational resource that we have.

To solve the first issue, we exploy the discretization algorithm implemented in WEKA that optimizes the number and value range of bins with a restriction that the maximum number of bins is four (this number is determined experimentally). The optimization process uses a cross-validation procedure that maximizes the estimated likelihood of the data. The discretization of Odds Ratio results in four bins or nominal values as follows:
`(-inf--2.044225], (-2.044225-2.65175], (2.65175-7.347725], (7.347725-inf)`

The following are some examples from the Features dataset after discretization.

```
</area>,3,2,230,?,?,(-0.5-4.75],?,?,?,?,(-inf--2.044225]
</area>,1,2,858,(6-inf),(-inf-3],?,?,?,?,?,(-2.044225-2.65175]
</area>,1,2,4428,(4-6],(-inf-3],?,?,?,?,?,(-2.044225-2.65175]
```

The number of instances in the Feature dataset is very large, because it is equal to the number features in the training set multiplied by the number of tags. For the Job600 dataset (see Chapter 6), for example, the number of instances in this dataset is over 8 million. For PART to be able to handle it, we perform stratified random sampling[3] to select 2000 instances from the dataset—500 instances from each bin of Odds Ratio values. The stratification is required so that the resulting decision list is not weighted heavily toward any single bin. The size of the data is mainly constrained by the memory consumption using the implementation of PART in WEKA (We used about 500 MB of memory).

5.2 Applying the Learned Decision List

The decision list learned by PART from the Features-Resumes dataset is shown here:

```
Rule 1: ObjectType = Token && Class = Distance  ➔ HIGH_EFFECTIVENESS
Rule 2: Default ➔ LOW_EFFECTIVENESS
```

Each rule starts with a set of conditions in the antecedent and ends with the consequence which predicts whether the feature that meets the conditions would have high or low effectiveness.

The decision list is to be interpreted in order. The first rule is tried out first, if the feature in consideration does not match the conditions set by it, the second rule is tried, and so on. The first rule means that the features based on tokens and distance from insertion point are highly effective. The second rule above is a default rule since it has no condition, and says that whenever the conditions of the first rule is not met, the feature has low effectiveness. The

[3] Stratified random sampling is a kind of random sampling which we select an equal number of instances from each category of data.

decision list learn from the Jobs dataset is comprised of 24 rules and is included in the Appendix.

Since we will use only feature selection and not feature weighting, only two possible values in the consequence are needed. In our experiments, only the highest bin out of four is assigned to be highly effective. If feature weighting is performed, a MEDIUM_EFFECTIVENESS consequence can be added.

With the decision list ready, we can now apply it to perform feature selection. Each feature and tag in consideration are passed to the matching function. The function looks up the feature properties and match them against the condition in rule antecedent. If the tag condition is present in the rule, it will also be considered. Only the features predicted to be highly effective are selected for classification.

5.3 Transferring Knowledge

The decision list learned can be used to select features in the same domain, or alternatively, to the test set in a different domain that share common representational bias. The rules learned are not specific to a particular dataset. When the decision list is used in a different dataset or domain, this process can be considered a form of 'learning to learn' where learning performance improves with the number of datasets that are used (S. Thrun and L. Pratt**,** 1998). Our method is the transfer of representational bias through the reduction of the hypothesis space that needs to be considered by the learning algorithm.

Knowledge transfer has been shown to be useful in several domains. Our experiments show that it is useful in textual information extraction as well. In paricular, we show that using the decision list learned in one domain to select features in another improve performance. The approach is especially important when data is plentiful in the original domain, but scarce in the destination domain. As a practical matter, this is often the case when data is cheap to acquire in the original domain and expensive in the destination domain (S. Thrun and L. Pratt**,** 1998).

6. Experimental Evaluation

6.1 Domain and Task Descriptions

We test our system in two information extraction domains. The first domain is Jobs, which consists of a set of 600 computer-related job postings from the newsgroup `austin.jobs`. Half the documents in this dataset are those used in the experimental evaluation of RAPIER and tagged by Mary E. Califf (Califf, 1998); the other half was tagged by Un Yong Nam. The task is to identify and tag key pieces of information in each job posting, such as the job title, the salary, the company, the languages and platforms involved, and the required years of experience and degrees. There are in total 17 different kinds of tags in this domain.

Another domain is Resumes, of which we use a dataset consisting of 300 computer-related resumes from various newsgroups. This dataset was tagged by Un Yong Nam. There are 13 different kinds of tags in this domain, including those that cover the name, the desired position, the areas of expertise, the number of years of experience, the salary requirements, and the educational degrees of the applicant.

These two domains share several commonalties, especially those involving technical terms and job titles in computer fields. A major difference is that the company's name and location tags that are present in the Jobs domain, but not in Resumes. The similarities between these two domains make them prime targets for transferring representational bias, which we will show to be effective in our experiments.

6.2 Data Pre-Processing

We extract tokens from the document with a very simple tokenization technique. A decision is made here not to return any white-space characters (space, tab, carriage return, form feed) as tokens, while punctuation, such as colons, parentheses, ampersands, used as token separators are also returned as single character tokens.

Another pre-processing step is to identify potential tag insertion points. Spaces and most punctuation, with the noticeable exception of underscores, are part of a potential insertion point. Underscores are excluded because it usually connects two separate words together to form a proper noun (e.g. in john_doe@yahoo.com).

Although these pre-processing steps seem trivial, the decision made at this point in fact significantly affects extraction performance. For example, when a dot (.) is considered part of an insertion point, recall will generally increase while precision will decrease from when it is not part of an insertion point. (Our experiments below do not consider a dot as part of an insertion point.)

Notice that we do not perform any sophisticated tokenization that employs knowledge of common words and abbreviations in English (while RAPIER does). The extraction performance of our system will likely increase if better pre-processing is used. The point of our experiments is to show the differences between feature-selection methods, however, and leaving the pre-processing step simple does not hurt our comparisons.

6.3 Feature Selection Approaches Compared

There are three major feature selection approaches used in our experiments: heuristics-based, rule-based, and manual. Simple heuristics based on document and class statistics have been used for feature selection for a long time. The rule-based and manual approaches both rely on the property-based feature extraction framework we outlined in Chapter 4 and are our novel contributions.

Note that we perform feature selection for each instance (or insertion point) and, with some approaches, the features selected also depend on the category (or tag) of concern. An implication is that feature selection is computed at the testing phase.

Heuristics-Based Approach

Statistics based on the probability that a feature or a class appears, or a feature appears given a class, a class appears given a feature, have been used for feature selection. These include

information gain, mutual information, feature frequency, and odds ratio. We experimented with most of the feature selection heuristics presented in (Mladenic and Grobelnik, 1999), which studied them for the case of unbalanced class distribution and naive Bayer learner. Our experiments show that odds ratio is the best among these heuristics for our information extraction setup. The results are congruent to those reported in (Mladenic and Grobelnik, 1999), where Odds Ratio is one of the best performing heuristics. In subsequent experiments, we use Odds Ratio as the representative for the heuristics-based feature selection approach.

Manual Feature Extraction Approach

This approach performs *pre*-selection of feature by limiting the extraction of features in the first place. The kinds of features that are generally not very useful are simply not extracted. The major advantage for this approach is its low training and testing time and low memory consumption. Several experiments will be needed, however, to find out what *combinations* of feature types are most useful. When there are many kinds of feature properties, exhaustive search is not possible because of the combinatorial complexity, but manual hill-climbing or beam search usually finds good solutions. This higher-level search process for good combinations of feature types can potentially be automated as well, although we did not implement it.

Rule-Based Approach

This approach uses rules that are learned from examples in the same domain or a similar one to select features, as described in Chapter 5. It uses data of feature effectiveness (based on, for example, Odds Ratio) to learn higher-level rules. This effectively automates the search through the space of useful features. A limitation for this approach is that it cannot detect the results from feature interactions, while manual selection can, since it receives feedback from experiments that use all features together.

6.4 Relevant Parameters

Given a set of the pre-processing steps, the feature types extracted, the feature selection method, and the learning algorithm used, there are still two parameters that affect extraction performance. They are explained here:

a) Feature types extracted and window size for extraction. The types of features we choose to extract (from all the features defined in Chapter 4) significantly affect performance. Since we generally do not know a priori which kinds of features are most useful, we use all feature types by default. The obvious exceptions are when manual selection of feature types is used. Another important factor is the maximum size of window the system considers when extracting features. By default, we use the window size of 10. Therefore, the parameter Maximum Window Size in Section 4.2 is 10 and the parameter Distance has a range of -5 to +5.

b) The number of selected features. For each binary classification whether a tag should be put into a *given* insertion point, we select only the features that are deemed most relevant by our feature selection heuristic (when it is used). Since feature selection heuristics return their feature evaluation results in varying, incompatible numerical scores, we select a fixed number of top features evaluated. The number of selected features may affect the classification performance. The default value of this parameter is 7, which is experimentally determined.

c) Prediction threshold. As described in Chapter 3, the naive Bayesian classifiers we use determine whether a tag should be predicted for a given insertion point based on the probabilities that the tag is present versus that it is not. The natural decision threshold for this ratio is 1.0 (when the probability that the tag is present is equal to the probability that it is not), which is the default value we use in most experiments. However, we find that shifting this threshold in either high or low directions sometimes increase extraction performance. We usually discuss about the threshold using the *logarithm* of its value, since its range is very wide.

6.5 Experimental Methodology

For all experiments, we use ten-fold cross-validation, in which one-tenth of each dataset is reserved as an independent test set. The size of the training set is varied in some experiments to produce learning curves. Except when noted otherwise, the X-axis on learning curves indicates the number of training documents. By default, however, the training set consists of

540 documents in the Jobs domain and 270 documents in the Resumes domain. The test set always consists of 60 documents in the Jobs domain and 30 documents in the Resumes domain. The same training and test set splits are always used for every experiment; therefore direct comparisons can be done across experiments. We use two-tail paired t-test to measure statistical significance, where it is conducted.

Precision, recall, and F-1 measures, as described in Chapter 3, are reported in percentage. Training and testing time are reported in seconds. The terms training time and testing time denote the average time used for each split in ten-fold cross-validation. We also report the total cross-validation time, which denotes the combined training and testing time for all ten folds.

All experiments are run on computers with the following specification: 1.0 GHz Pentium III CPU, 256 KB Cache, and 512 MB of RAM. The system is developed in Java and compiled using Sun Microsystems' JDK 1.3.1.

6.6 Comparing Feature Selection Heuristics

In this section, we show the results of experiments comparing different feature selection heuristics used in (Mladenic and Grobelnik, 1999). The heuristics used here are odds ratio, weighted odds ratio, cross entropy, feature frequency, information gain, log probability, and mutual information. We use the maximum number of training examples in each domain for these experiments (540 for Jobs and 270 for Resumes). The number of top features selected for classification is set at 7 by default.

Since the extraction performance using these heuristics is particularly sensitive to the *prediction threshold* (described in section 6.4), we conducted experiments for each heuristic and no heuristic with different thresholds. *Logarithms* of the threshold we use vary from -16, -8, -4, -2, 0, 2, 4, 8, to 16. We report the result for each heuristic here using the prediction threshold that maximizes the F-1 measure of extraction performance. In Figure 6.1, the number that follows each heuristic name is the logarithm of the prediction threshold reported here.

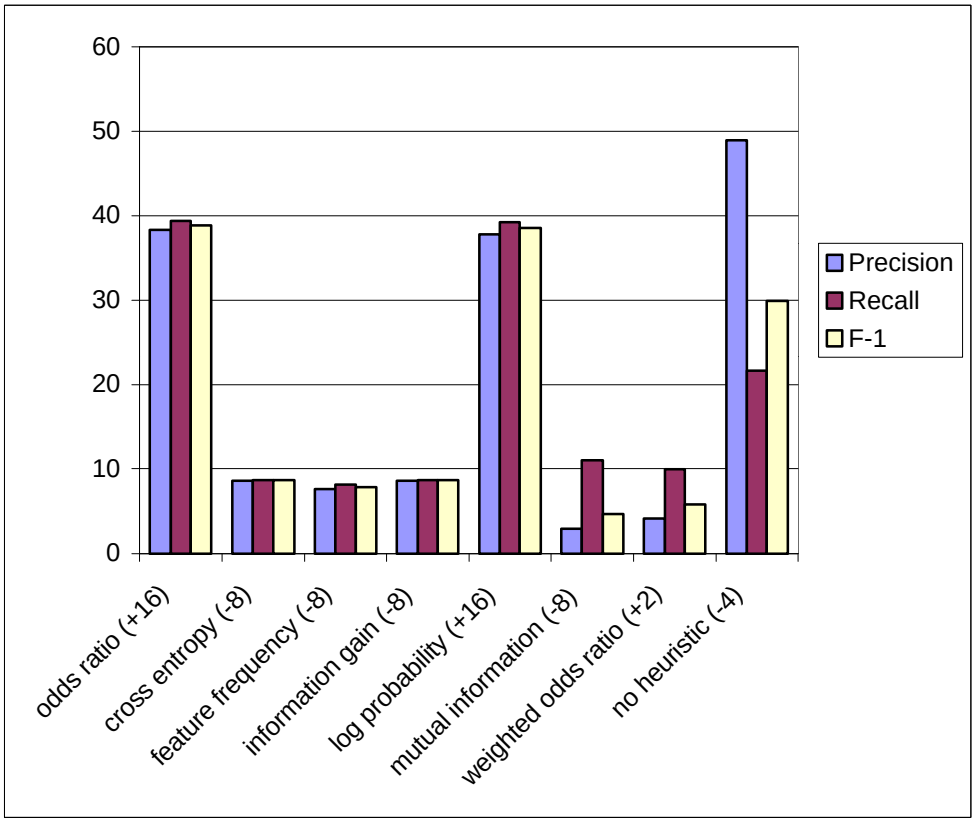

Figure 6.1 a) Comparing extraction accuracy when using different heuristics in the Jobs domain.

Note that "No Heuristic" indicates that no heuristic is employed, but we still use only seven features for each binary classification, in order to make the results comparable with using heuristics.

We can see that in both domains, many feature selection heuristics actually perform worse than not using feature selection heuristic at all. In both domains, the best performing heuristics are Odds Ratio and Log Probability Ratio, respectively. Both heuristics favors the features that appears in positive instances (the cases which the tag is present) over those that appear in negative ones (which the tag is not present). Both these heuristics and Weighted Odds Ratio are among the best performers in (Mladenic and Grobelnik, 1999). A surprising difference between our results and theirs is that Weighted Odds Ratio performs

unsatisfactorily here. We believe that the factor (the weight) P(F) that is multiplied to simple Odds Ratio is the culprit here. P(F) is not a good indicator for feature effectiveness, as we can see that the Feature Frequency heuristic performs badly. It also varies greatly in our system and greatly sway the final value away from the well-performing Odds Ratio.

For further discussions about the use of above feature selection heuristics for unbalanced class distribution with naïve Bayes, the reader is referred to (Mladenic and Grobelnik, 1999).

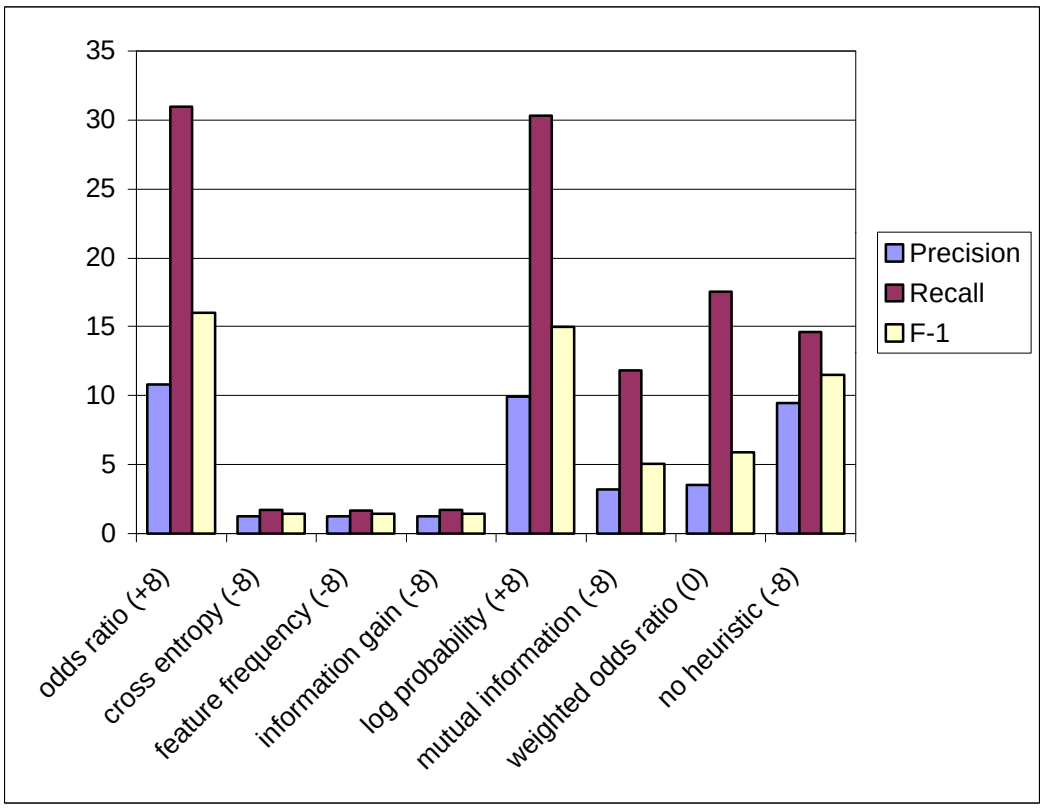

Figure 6.1 b) Comparing extraction accuracy when using different heuristics in the Resumes domain.

The testing time for all feature selection heuristics are comparable (the training time are about equal by nature since the heuristics are computed at the testing phrase). The testing time for these heuristics, except Feature Frequency, range from 200 to 340 seconds per split in the Jobs domain and 220 to 360 seconds in the Resumes domain. Feature Frequency is simpler to compute since it is collected in a data structure when we extract features in the first place. Thus, when it is used, the testing time is lower, ranging from 80 to 90 seconds in the Jobs domains and 95 to 140 seconds in the Resumes domain.

Figure 6.2 shows the effects of different prediction thresholds on extraction performance when we use Odds Ratio for feature selection. A clear trend is that when the threshold is lower, the precision decreases. At the same time, lower threshold tends to increase recall, but lowering the threshold beyond a certain point does not seem to help increase recall further.

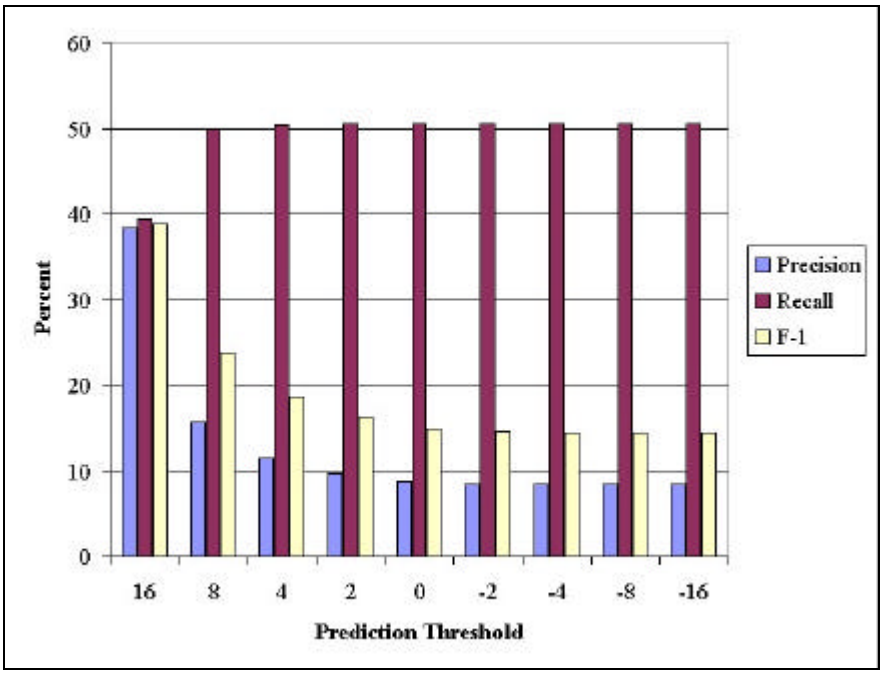

Figure 6.2 a) Extraction accuracy when using Odds Ratio with different prediction thresholds in the Jobs domain.

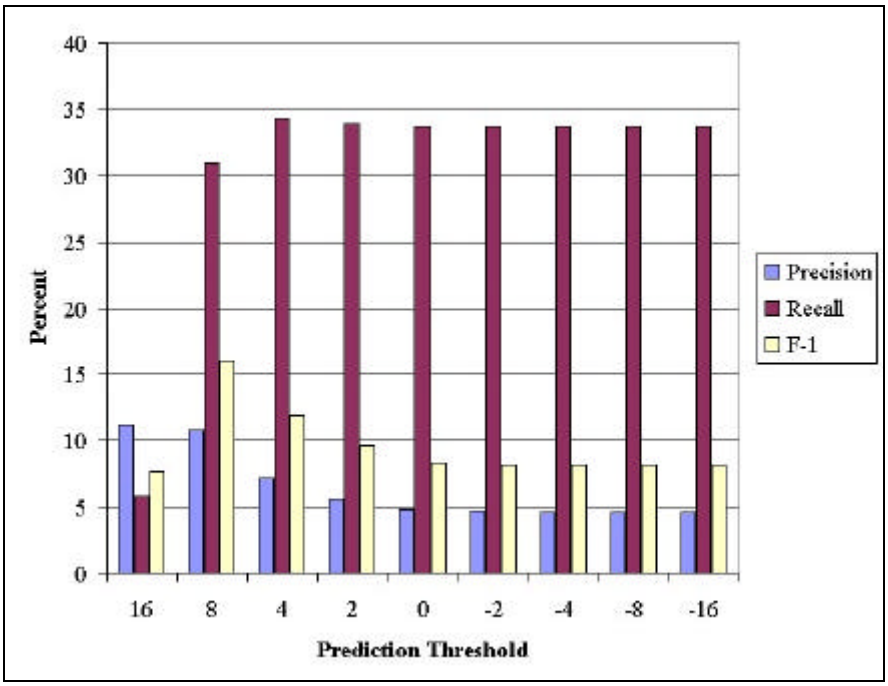

Figure 6.2 b) Extraction accuracy when using Odds Ratio with different prediction thresholds in the Resumes domain.

6.7 Rule-Based Feature Selection

To learn rules about feature effectiveness, we create a dataset comprising of feature properties and a measure of its effectiveness, as described in Chapter 5. The effectiveness measure we use is Odds Ratio, which depends on the feature and the tag involved. We call this the Features meta-dataset since it contains meta-information about the problem domain. For brevity, sometimes we simply call it the Features dataset.

The Features datasets produced from the Jobs and the Resumes datasets are very large. Their sizes are equal to the number of features in the dataset multiplied by the number of tags. For example, in the Jobs dataset we use, there are 30 tags and over 400,000 features; the resulting Features dataset thus has over 12 million instances. We performed discretization and stratified sampling as described in Chapter 5 and use a subset of 2,000 instances from the Jobs-Features dataset and 1,600 instances from Resumes-Features dataset to learn decision lists of feature engineering knowledge.

The PART decision list learner creates 24 rules from the Jobs-Features dataset and only 2 rules from the Resumes-Features dataset. Despite the differences in numbers of rules, both decision lists share the key rule, which is:

```
ObjectType = Token && Class = Distance  ➔ HIGH_EFFECTIVENESS
```

which states that features that are based on tokens and in the class Distance usually are highly effective. As we will see, it turns out that manual trial-and-error searches for the kinds of features that are effective result in the same knowledge.

It should be mentioned that most of the rules in the decision list learned from the Jobs-Features dataset deals with features about highly specific content. There are however two more rules in the list that are quite general; they are:

```
ObjectType = Special ➔ LOW_EFFECTIVENESS
ObjectType = Line ➔ LOW_EFFECTIVENESS
```

Note that ObjectType can be Special only when the feature class is Distance from Line Start (see Chapter 4) and that the ObjectType of Line is only used in the feature class Length. These two rules also match the results from manual experiments with different feature types in Section 6.9. However, they do not affect the performance of rule-based feature selection, since the default rule (when none of the other rules in the decision list applies) also has `LOW_EFFECTIVENESS` as its consequence.

The decision list learned from Resumes-Features dataset is simple and can be stated in full here:

```
ObjectType = Token && Class = Distance  ➔ HIGH_EFFECTIVENESS
Default ➔ LOW_EFFECTIVENESS
```

The decision list states that when the feature is based on the distance to a given token, it is highly effective, otherwise (by default) it has low effectiveness. The reader may also check the full decision list learned from the Jobs-Features dataset in the Appendix.

6.8 Comparing Different Feature Selection Approaches

This section discusses the differences in extraction performance between major feature selection approaches. The following approaches are compared:

a) No Feature Selection

b) Rule-Based Selection Features are selected by using the decision lists learned from the datasets in two domains: Jobs and Resumes. Transfer of learning occurs when the decision list learned from one domain is used to select features in another domain. We label these *jobrule* and *resumerule* in the charts below. With this approach, we do not limit the number of selected features.

c) Manual Selection In this approach, only features of certain types are extracted and used in classification. Two major parameters are feature types and the size of window considered. We experimented with many combinations of parameters. One of the best is when feature Class is Distance and Object Type is Token (see Section 4.2 for definitions) and window size is 8. The following result for *manualboth* uses this parameter combination. ('Both' refers to the fact that we manually set both feature types and window size.)

d) Heuristic-Based Selection We choose Odds Ratio as the representative of heuristic-based Feature Selection, since it is the best performing heuristic as shown in the Section 6.5. The results reported as *oddsratio* below uses prediction threshold of 10 to the power of 16.

We also tested another way of using Odds Ratio. Instead of selecting the top features evaluated regardless of its Odds Ratio value, we now set a minimum value of Odds Ratio that all features selected must surpass. The *number of selected features* parameter is not used in this case, since all features with Odds Ratio higher than this minimum value are selected. This minimum value is set to the lowest value of the high bin when we discretize the Odds Ratio value in the Features datasets as described in Chapter 5. We call this method *filtered odds ratio*.

Except as noted, the number of selected features is set at 7, and prediction threshold at 1.0 for all the following experiments.

Figure 6.3 to 6.5 respectively show the F-1, Precision, and Recall results for different approaches outlined above. Note that the values of *jobrule* and *resumerule* are identical in every case (therefore the two lines overlap in the graphs). This is because most of the rules learned from the Jobs dataset predict low feature effectiveness for features with specific content. These rules do not overlap with the only significant rule in the list, which predicts high effectiveness for features with token as object type in the feature class Distance. This important rule is the same one the system learned from the Resumes dataset. For our purpose, therefore, both decision lists have the same effect.

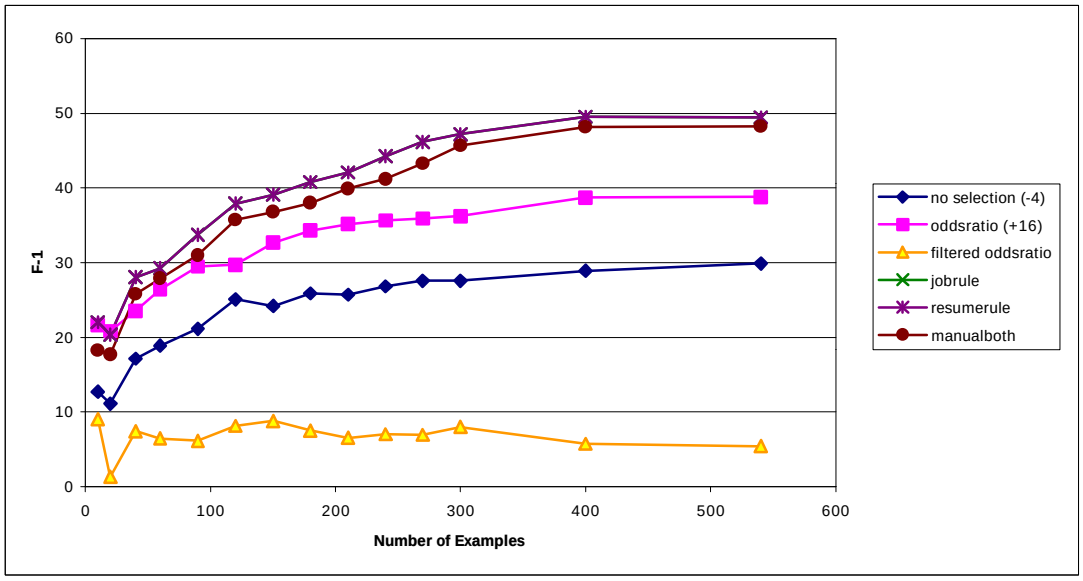

Figure 6.3 a) Resulting F-1 values when applying different feature selection approaches to the Jobs domain.

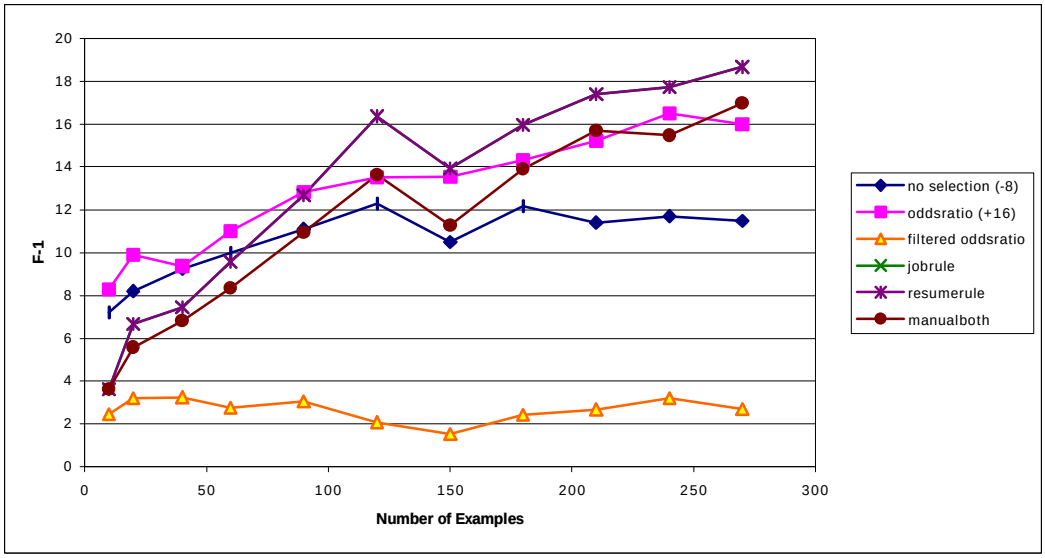

Figure 6.3 b) Resulting F-1 values when applying different feature selection approaches to the Resumes domain.

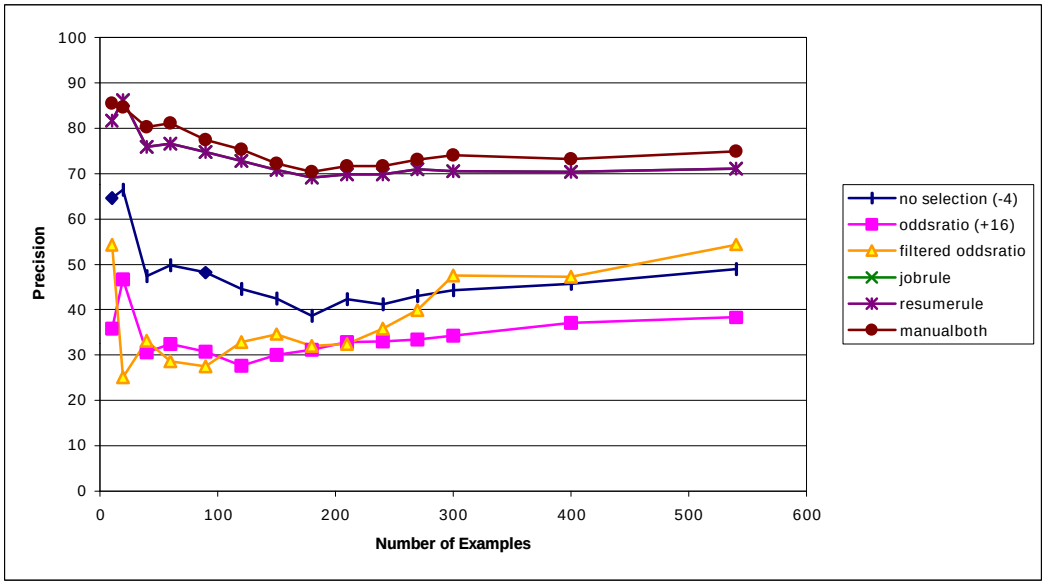

Figure 6.4 a) Resulting precision values when applying different feature selection approaches to the Jobs domain.

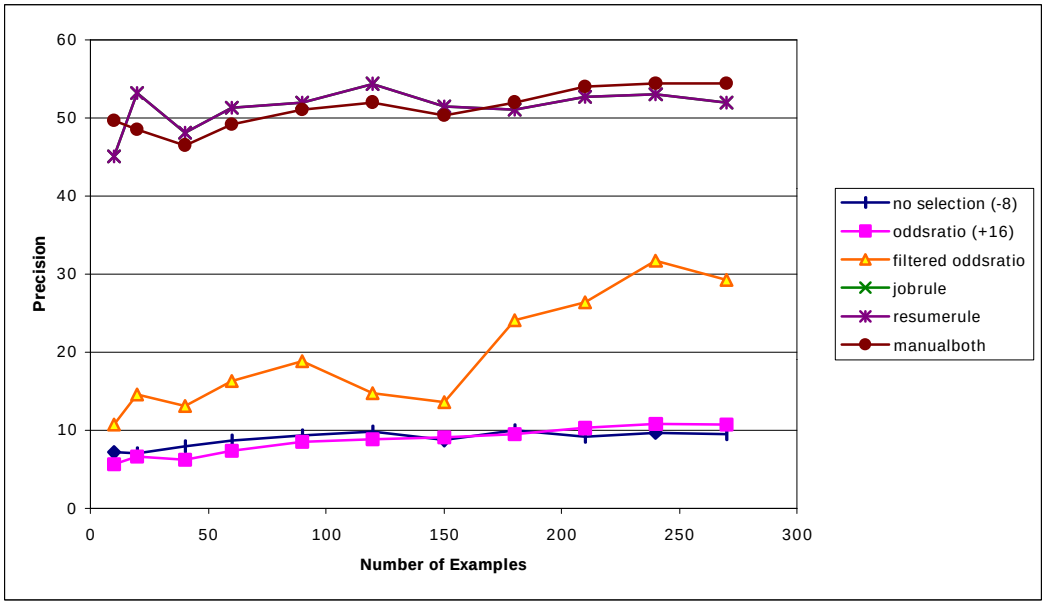

Figure 6.4 b) Resulting precision values when applying different feature selection approaches to the Resumes domain.

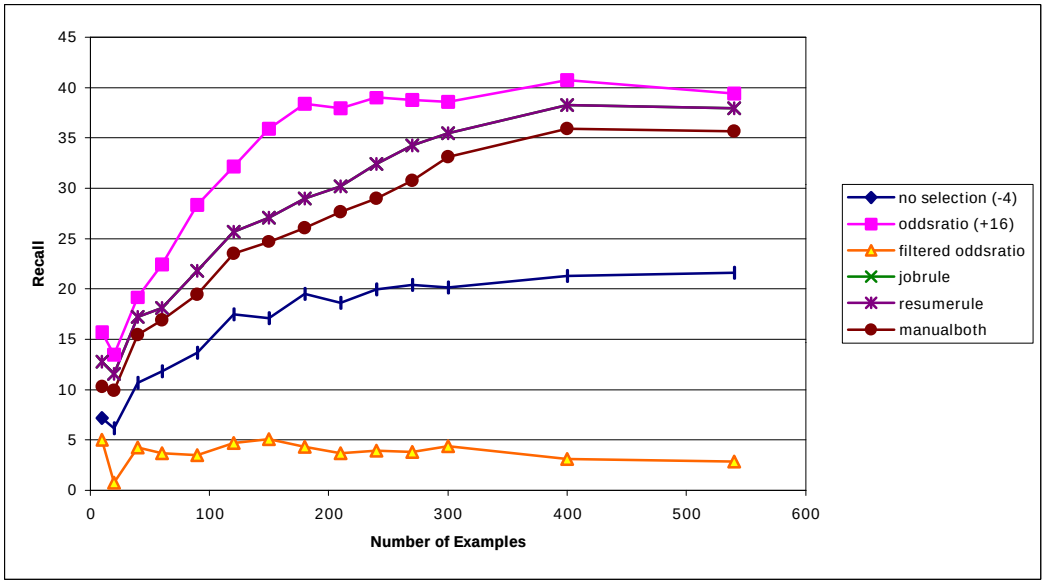

Figure 6.5 a) Resulting recall values when applying different feature selection approaches to the Jobs domain.

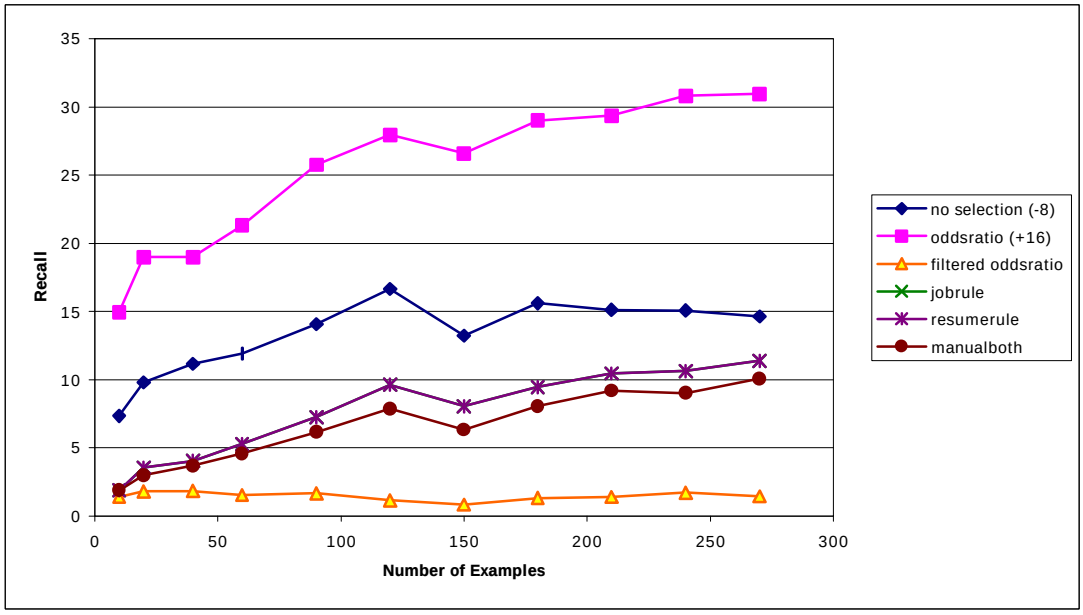

Figure 6.5 b) Resulting recall values when applying different feature selection approaches to the Resumes domain.

Basically, rule-based feature selection and *manualboth* perform similarly in all cases. This is because the important rule requires the selection of the types of features also used by the manual selection. This shows that the meta-learning process does capture a pattern for feature effectiveness that is discovered by manual trial-and-errors. Performance differences between these two approaches stem from the different uses of the parameter *number of selected features*. (Rule-based selection does not use it; manualboth does).

The results summarized by F-1 show that all feature selection approaches, except filtered odds ratio, outperforms using no feature selection. When we inspect the detailed data, only a few features have Odds Ratio value higher than the minimum required by *filtered oddsratio*. For many insertion points, no features are selected at all. This helps explain its low performance. This suggests that although we can discover useful patterns for feature effectiveness through discretization in meta-learning, the exact value used in the discretization should not be used to filter out features at the classification time.

The extent that Odds Ratio, rule-based, and manual feature selection outperforms using no feature selection is larger in the Resumes domain than in the Jobs domain. We believe that more complex patterns and a larger number of irrelevant features in the Resumes domain require more judicious selection of features used in classification.

The F-1 measure of Rule-Based and Manual approaches is higher than that of *oddsratio* in the Jobs domains and competitive with it in the Resumes domain. The precision of *oddsratio* is low in both domains, even compared to using no feature selection. This is also due to the *prediction threshold* that we set to maximize F-1. As a result, recall for Odds Ratio is slightly higher than those of Rule-Based and Manual approaches in the Jobs domain and much higher than the recall of those two approaches in the Resumes domain.

Note that we did not select a prediction threshold to maximize the F-1 values of every approach, except *oddsratio* and *no selection*, the results above may understate the performance of the feature selection approaches we propose.

We therefore perform additional experiments with *manualboth* and rule-based approaches by varying prediction thresholds as well (using the same threshold points as in Section 6.6). The best F-1 value for rule-based selection in the Job domain is 49.78% (precision = 64.73% and recall = 40.46%) when the log of prediction threshold is set to –2.0. For rule-based selection in the Resumes domain, the best F-1 is achieved when the log of prediction threshold is set to –8.0. The best F-1 in this case is 26.14% (precision = 25.09% and recall = 27.30%) which is quite a bit higher than the ones shown in Figure 6.3 b) and is clearly better than *oddsratio*. The best F-1 for *manualboth* in the Jobs domain is 49.51% (precision = 66.19% and recall = 39.56%), achieved when the log of prediction threshold = -2.0. For manualboth in the Resumes domains, the best F-1 is 26.19% (precision = 36.56% recall = 20.43%), again clearly better than the result from *oddsratio*. This F-1 is achieved when the log of prediction threshold is equal to –4.0.

The training time is basically the same for all approaches, except with manual feature extraction. This is because all feature selection is performed at the testing time. Manual extraction reduces the time it takes to extract features from documents in the first place.

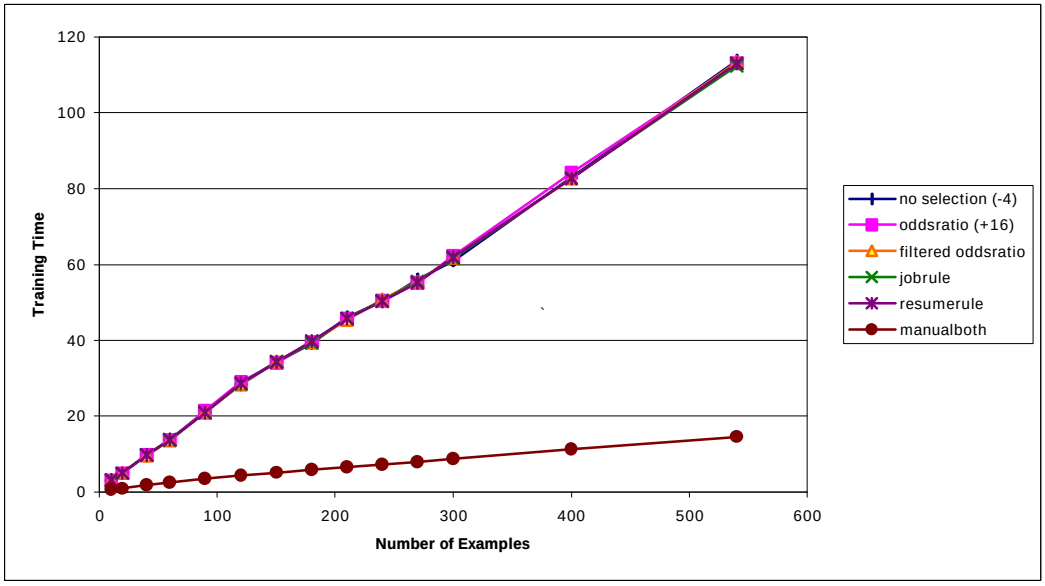

Figure 6.6 a) Training time used when applying different feature selection approaches to the Jobs domain.

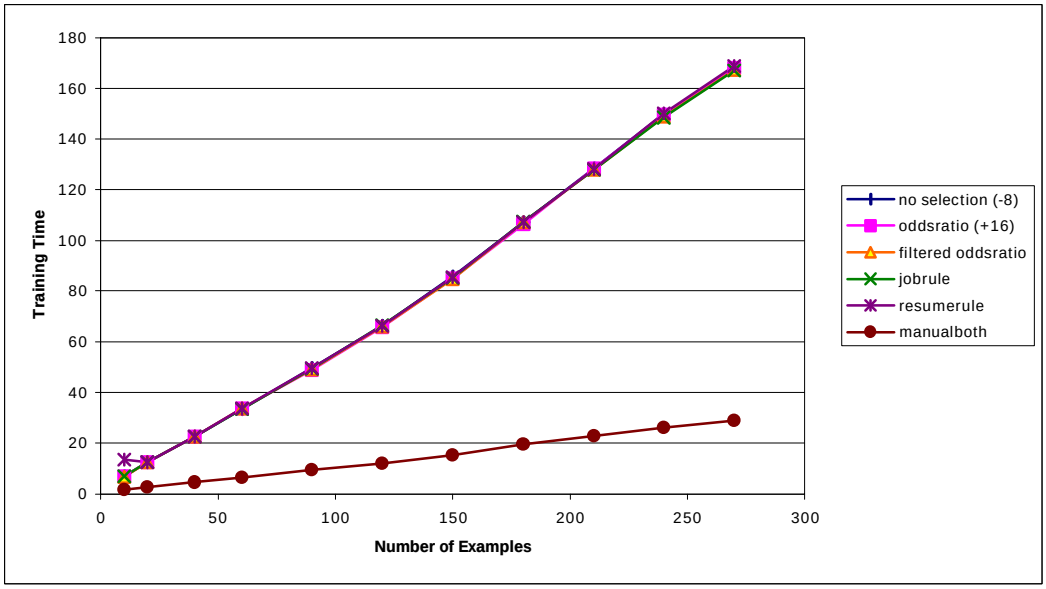

Figure 6.6 b) Training time used when applying different feature selection approaches to the Resumes domain.

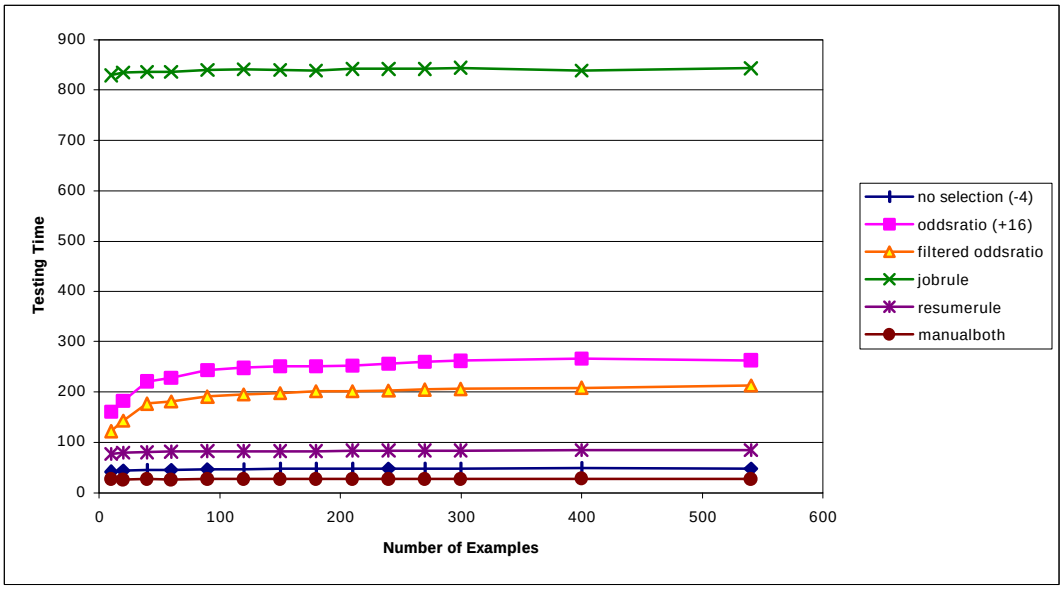

Figure 6.7 a) Testing time used when applying different feature selection approaches to the Jobs domain.

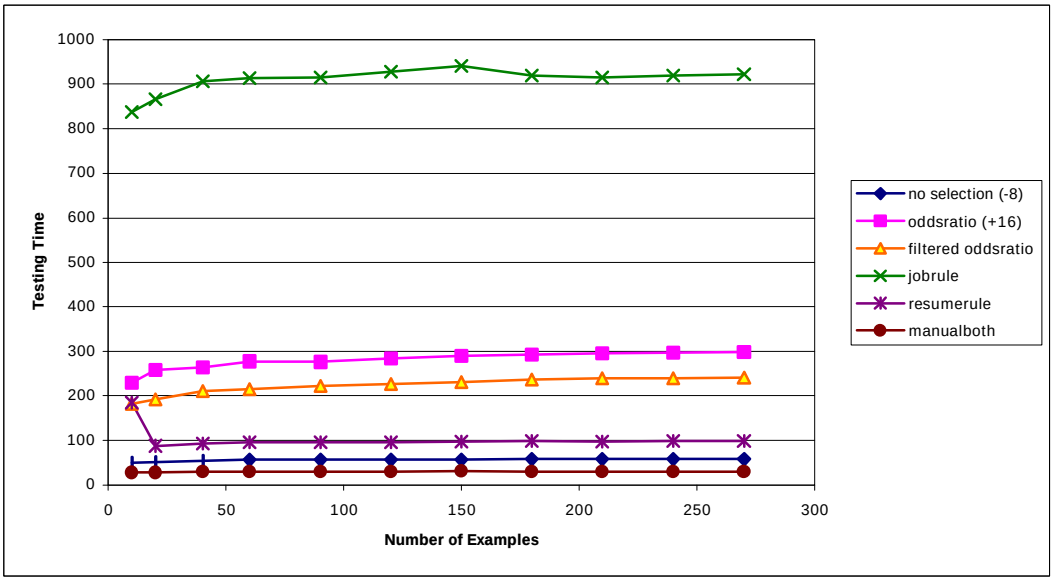

Figure 6.7 b) Testing time used when applying different feature selection approaches to the Resumes domain.

During testing, the manual approach again has the advantage, since it deals with a smaller number of features. All methods, except *jobrule*, spend from less than one second to five seconds per test document (recall that there 60 test documents used in each split). *jobrule* takes about 15 seconds to process each test document. The large difference in testing time between *resumerule* and *jobrule* is caused by the different numbers of rules in the decision lists (2 versus 24, respectively).

6.9 Comparing the Use of Different Feature Types and Window Size

In this section, we describe the performance of different feature types used in information extraction. We vary two main variables: one is the window size considered in extracting features; another is the class and object types of features extracted.

In the first set of experiments, we varied the window size from 2 to 32, with the exact window sizes used as follows: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 24, 28, 32. Note that in all these experiments, all feature types are extracted and no further feature selection is used.

The trend from the results are clear. Both precision and recall continually decreases as the window gets larger than a certain size. For the Jobs domain, the F-1 value peaks at the window of size 4. In the Resumes domain, using the window sizes of 4 and 6 achieve the highest level of F-1. Using the smallest window size of 2 achieves the highest precision, but its recall is lower than using the window of size 4, resulting in lower F-1 for the window size of 2.

The training and testing time also increase linearly as the window expands, except with the expansion from 28 to 32 where the increase seems to be super-linear. We believe that this is the result from the need to swap data from the main memory to hard drive, which is much slower, as the memory consumption grows beyond the machine's main memory's capacity.

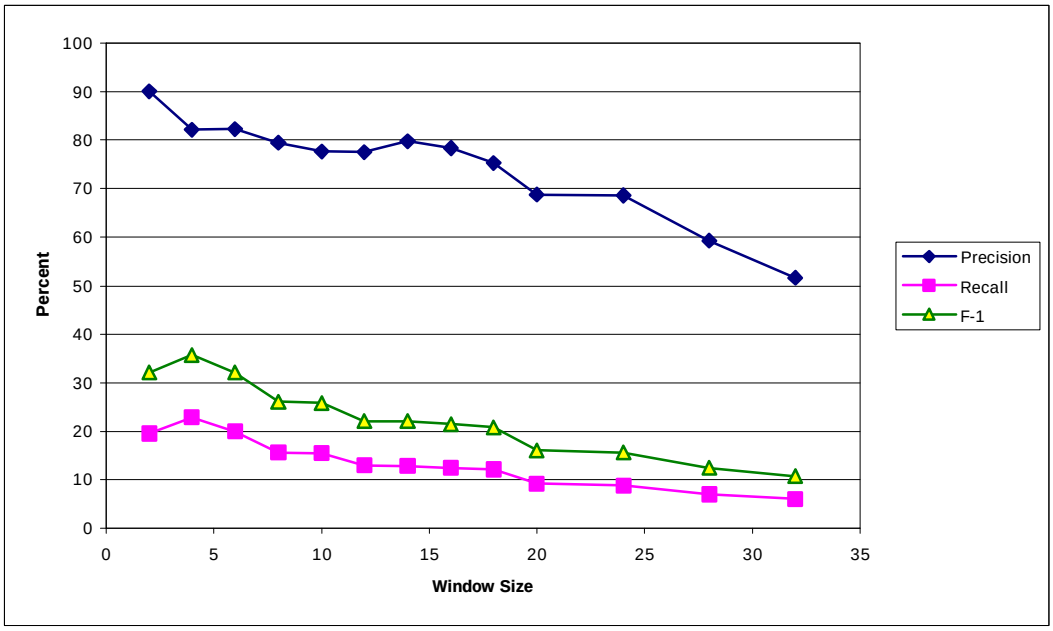

Figure 6.8 a) Comparing extraction accuracy when using different window sizes for feature extraction in the Jobs domain.

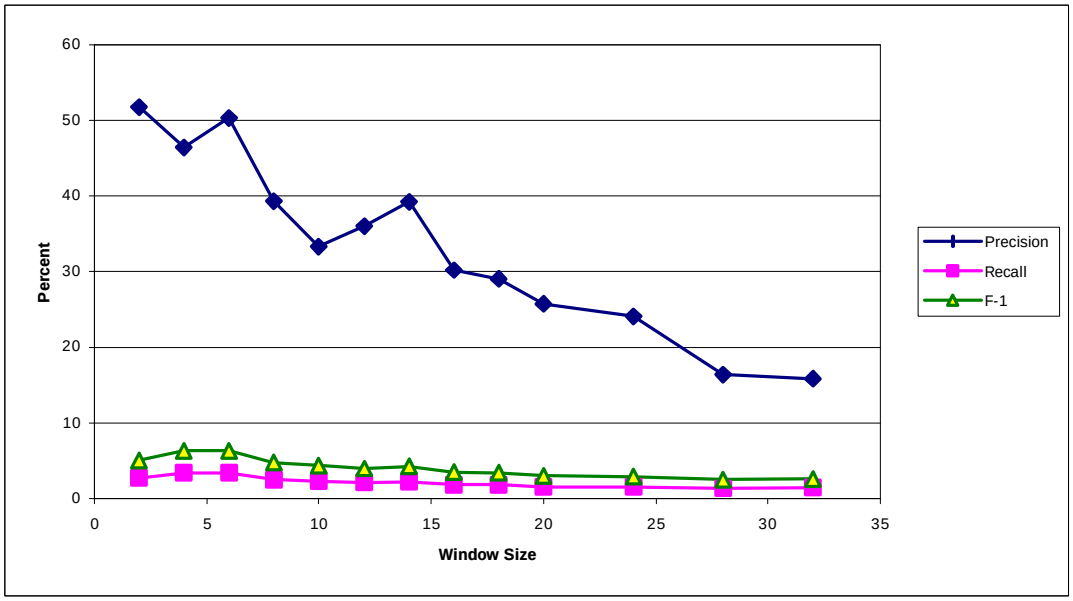

Figure 6.8 b) Comparing extraction accuracy when using different window sizes for feature extraction in the Resumes domain.

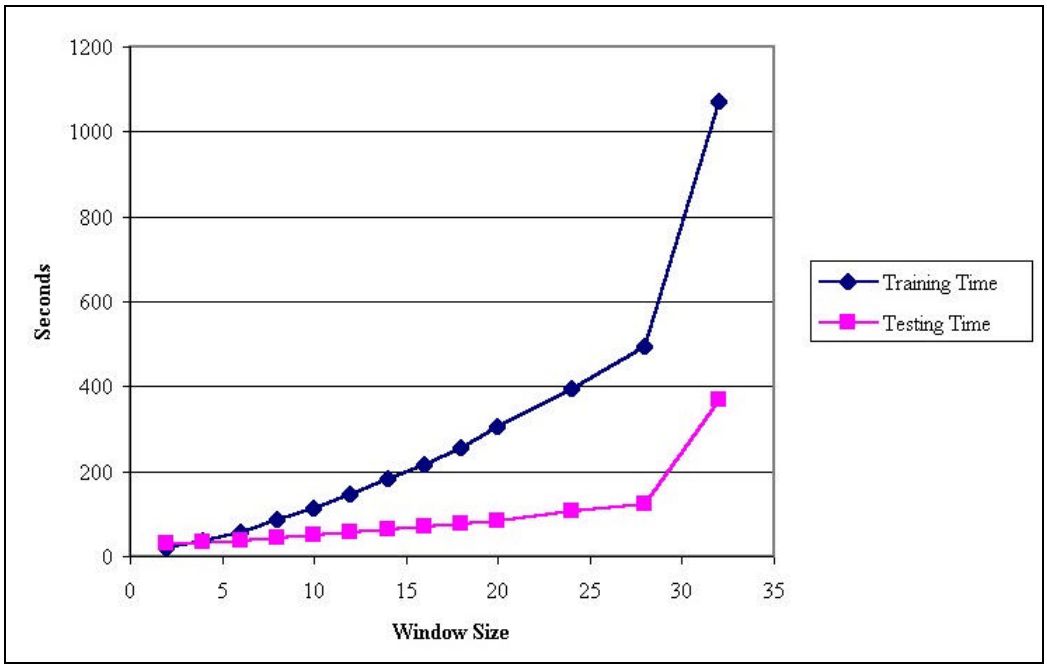

Figure 6.9 a) Comparing training and testing time when using different window sizes for feature extraction in the Jobs domain.

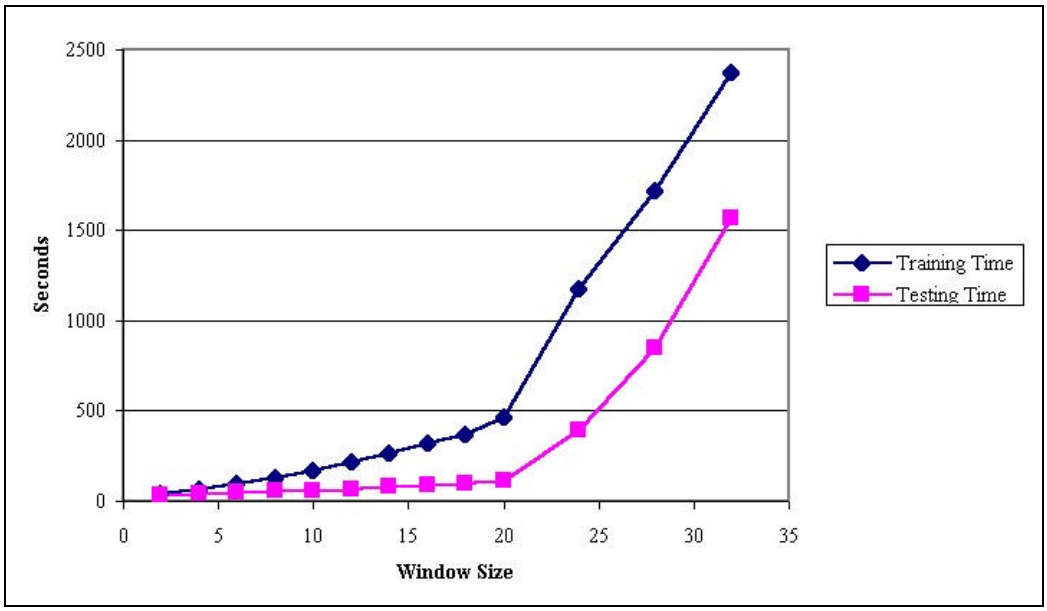

Figure 6.9 b) Comparing training and testing time when using different window sizes for feature extraction in the Resumes domain.

In the second set of experiments, we compare different feature selection approach under the very large window size of 32. The purpose is to discern how much improvement we will achieve from using feature selection when a large number of irrelevant features are present.

In Figure 6.10, we compare using no feature selection, feature selection with Odds Ratio (with prediction threshold = 0), and using manual selection for feature types (only features in the class Distance that are based on tokens are extracted). Manual selection outperforms no feature selection and Odds Ratio by about a factor of three in F-1 measure for the Jobs domain. Odds Ratio and Manual selection however achieve similar F-1 performance in the Resumes domain and the F-1 measure for both roughly doubles the case when no feature selection is used. These large improvements are similar to those observed when the smaller window size of 10 is used (as reported in Figure 6.3).

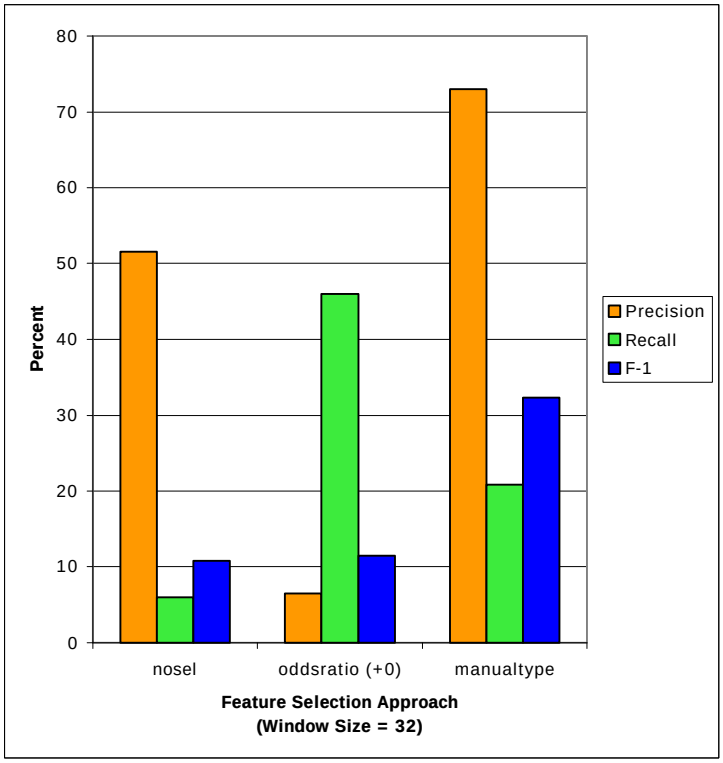

Figure 6.10 a) Comparing extraction accuracy for different feature selection approaches when using the window size of 32 in feature extraction for the Jobs domain.

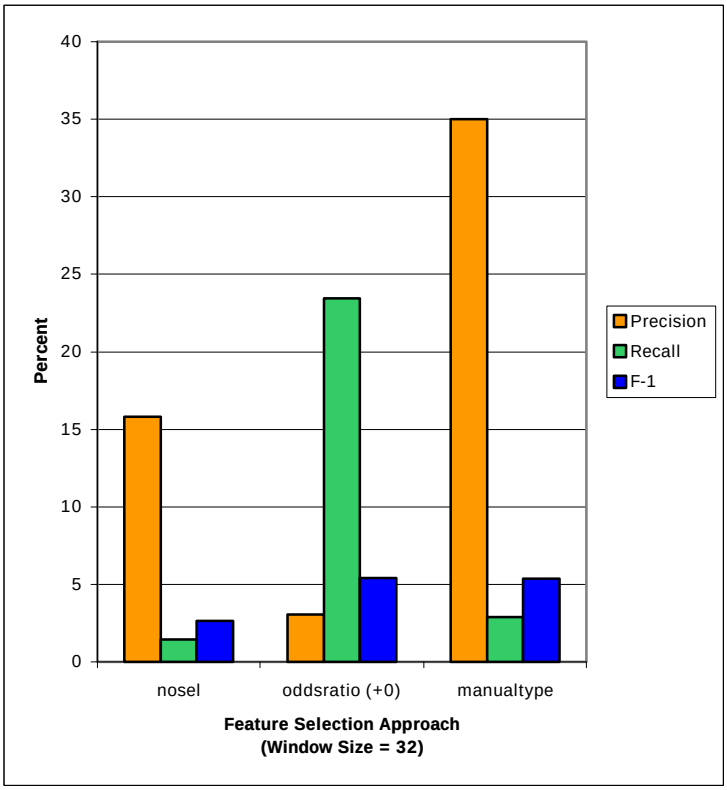

Figure 6.10 b) Comparing extraction performance for different feature selection approaches when using the window size of 32 in feature extraction for the Resumes domain.

The third set of experiments involves the selective extraction of different feature types from the raw data. In facilitate further discussion, we introduce some abbreviations to represent the types of features as follows:

Abbreviations for Object Types:

tok = Token, char = Character

Abbreviations for Feature Classes:

freq = Frequency, dist = Distance, length = Length, cap = Capitalization, diststart = Distance from Line Start

We concatenate these abbreviations together to indicate the mixture of feature types used. For example, *tokfreq* means the features in the class Frequency that are based on tokens. The same convention is used in the cases when we use more than one types. For example, *tokfreqdist* means that we use the features based on tokens in both the Frequency and the Distance classes. Because of the feature mixtures that we tested on, this simple convention suffices without causing confusion.

The first observation is that *diststart* results in zero precision and recall in both domains. This implies that the distance from the line start alone generally carries too little information for the classification algorithm to work correctly.

Notice that *tokcap* achieved near perfect precision in the Jobs domain, but only 10 percent precision in the Resumes domain, we investigated this by inspecting the tags it predicts in the Jobs domain. We found that *tokcap* predicts only <id> and </id> tags. Apparently, these two tags can be predicted very precisely by the capitalization category of its nearby tokens. This suggests that using different feature types tailored to each tag the system is dealing with can be useful.

The top performer in the Jobs domain is *tokdist*, following by *tokfreqdist*. In the Resumes domain, the reverse is true, *tokfreq* outperforms *tokfreqdist*. The performance of *tokfreqdist* seems to be in between those of *tokfreq* and *tokdist* in both domains.

The fact that *tokfreq* outperforms *tokdist* in the Resumes domain does not appear in the learned decision list about feature effectiveness from the domain. This might be because the decision list is learned from the values of individual feature effectiveness and does not take feature interaction into account. However, the difference in F-1 between *tokdist* and *tokfreq* in the Resumes domain is also not large (12.2 versus 15.4) and *tokdist* does have twice as high precision as tokfreq (48.3 versus 23.0), although the lower recall (7.0 versus 11.7) weights its F-1 measure down.

Generally, using features based on tokens alone achieve better results than using both token-based and character-based features. The difference in performance is small in the Jobs domain, but quite large the Resumes domain. This suggests that the character-level features cannot handle more complex patterns in the Resumes dataset very well. Again, these results occur in the context that no further feature selection (e.g. based on Odds Ratio) is employed.

Although the training and testing time vary somewhat depending on the feature types used, they are not significant in practical applications since it takes less than 2 minutes in every case we experimented with to train using 540 documents in the Jobs dataset and 270 documents in the Resumes dataset, and takes less than 1 second to process one test document in both domains. Also, the system scales linearly with the size and the number of documents, as already shown in Figure 6.6.

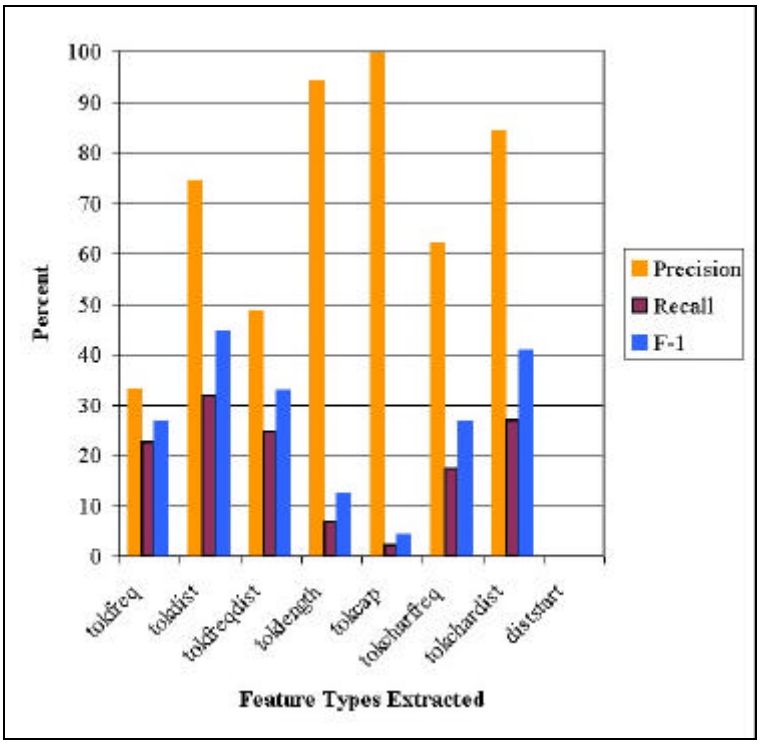

Figure 6.11 a) Comparing extraction performance when using different feature types in the Jobs domain.

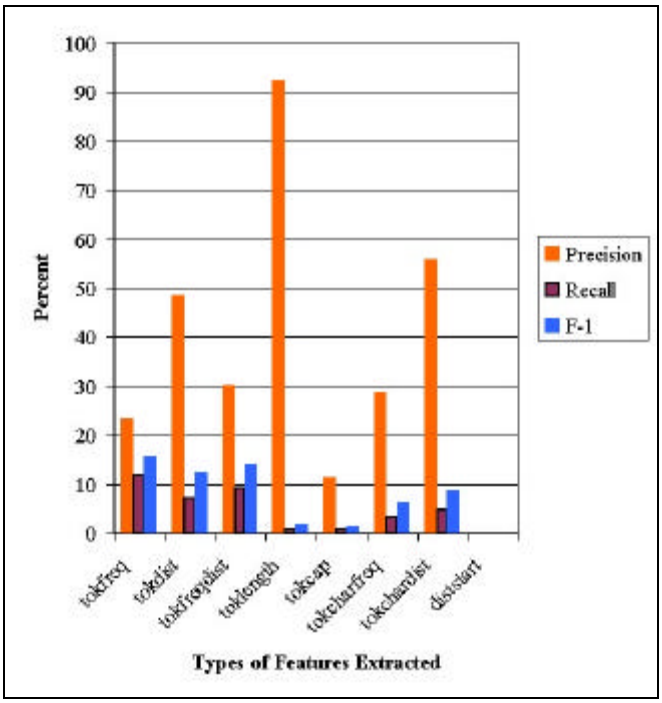

Figure 6.11 b) Comparing extraction performance when using different feature types in the Resumes domain.

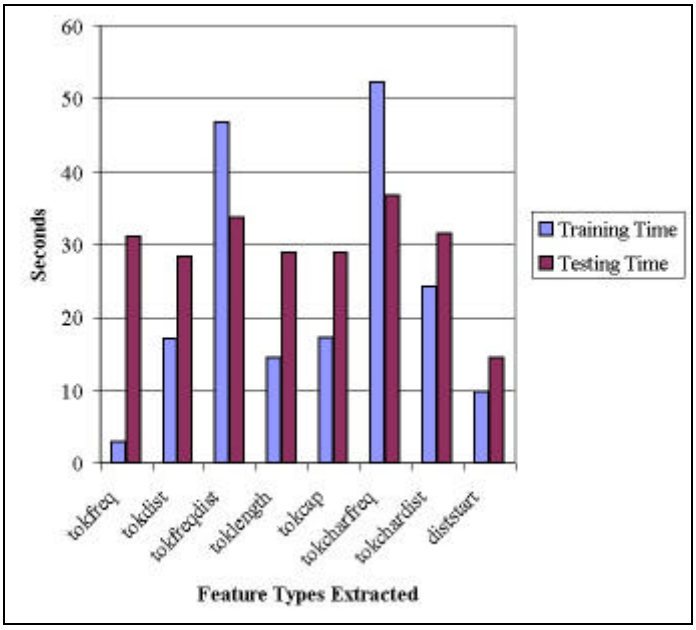

Figure 6.12 a) Comparing training and testing time when using different feature types in the Jobs domain.

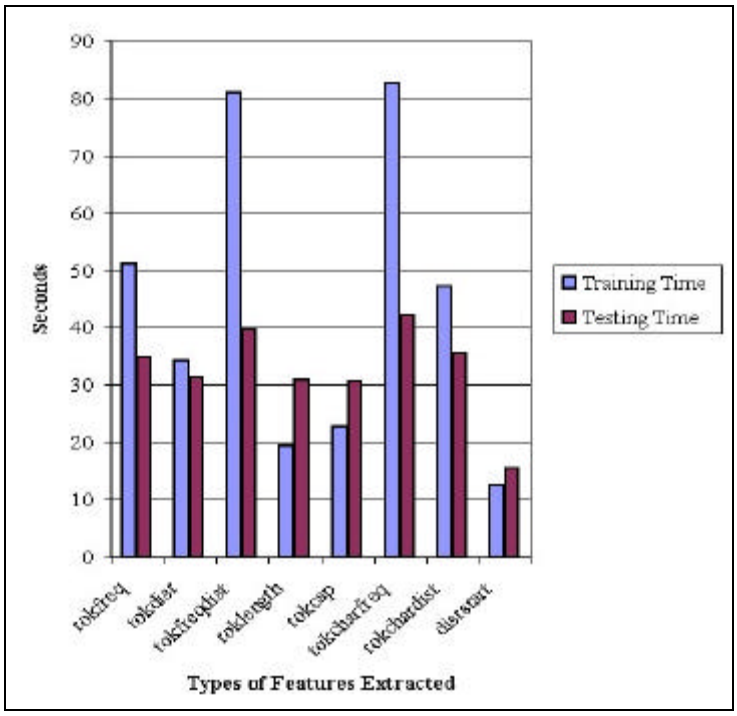

Figure 6.12 b) Comparing training and testing time when using different feature types in the Resumes domain.

6.10 Comparing Our System with RAPIER

In this section, we compare the results from RAPIER (Califf, 1998; Califf and Mooney, 1999) with our system when using different feature selection approaches. The results of these approaches have been reported in previous sections, but we select one of the best variants for each approach and summarize them here. For *Odds Ratio*, we report the result that uses 100 selected features for each classification and prediction threshold equals to 1.0 here. The representative of *Manual* selection here uses token-based features in the class Distance within the window of size 8 and uses 9 features for each classification.

The results from RAPIER are run on 540 examples for the Jobs domain. RAPIER cannot finish its run on the Resumes after over two days. Ruifang Ge generously helped with the experiments with RAPIER—the results of which we present here. RAPIER's F-1 performance on the Jobs domain is 90.2 percent, with 53.7 percent precision and 67.3 percent

recall. The recall and consequently F-1 reported for RAPIER here is lower than those in (Califf, 1998) because we strictly match both the locations and the texts of RAPIER's filler prediction with the manual tags here, while in (Califf, 1998) only filler texts regardless of their locations are used to make the matches. Note that the results from our system always require strict location as well as textual matches.

Comparing with the representative of the Manual approach, which is the best among our results here, RAPIER's precision is 17.1 percent higher, its recall is 13.4 percent higher, and consequently its F-1 is 15.4 percent higher. This indicates that RAPIER's relational representation still has an advantage over our fine-tuned propositional representation. But it might also reflects RAPIER's more sophisticated data pre-processing step (which recognizes common abbreviations and consequently correctly break text into sentences).

Another relational learning system WHISK (Soderland, 1998), whose results are reported in (Califf, 1998) based on a single trial with 200 training documents, achieved 76% precision and 40% recall. These results from WHISK are very close to the results from our Manual approach here. This suggests that the performance of relational learners in information extraction tasks could very well overlap with that of propositional ones, although it should be noted that WHISK used only 200 documents for training while our system used 540 documents to achieve the quoted results.

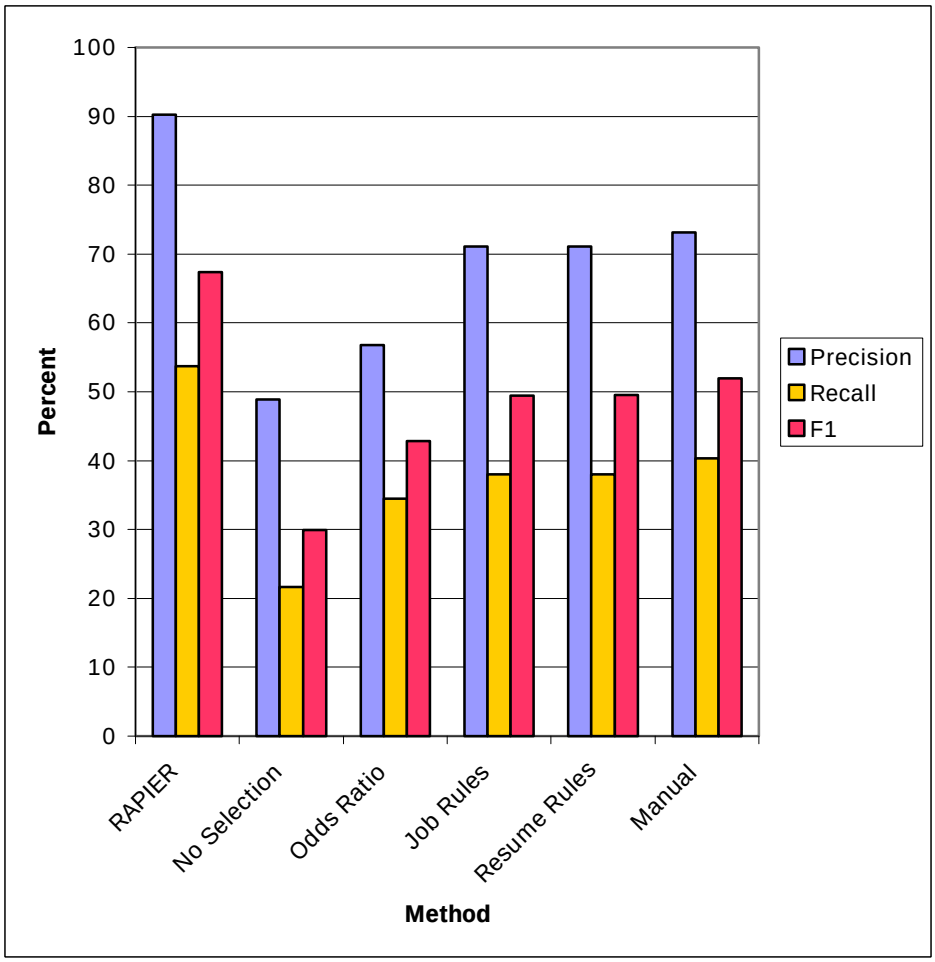

Figure 6.17 Comparing extraction accuracy between RAPIER and variants of our system in the Jobs domain

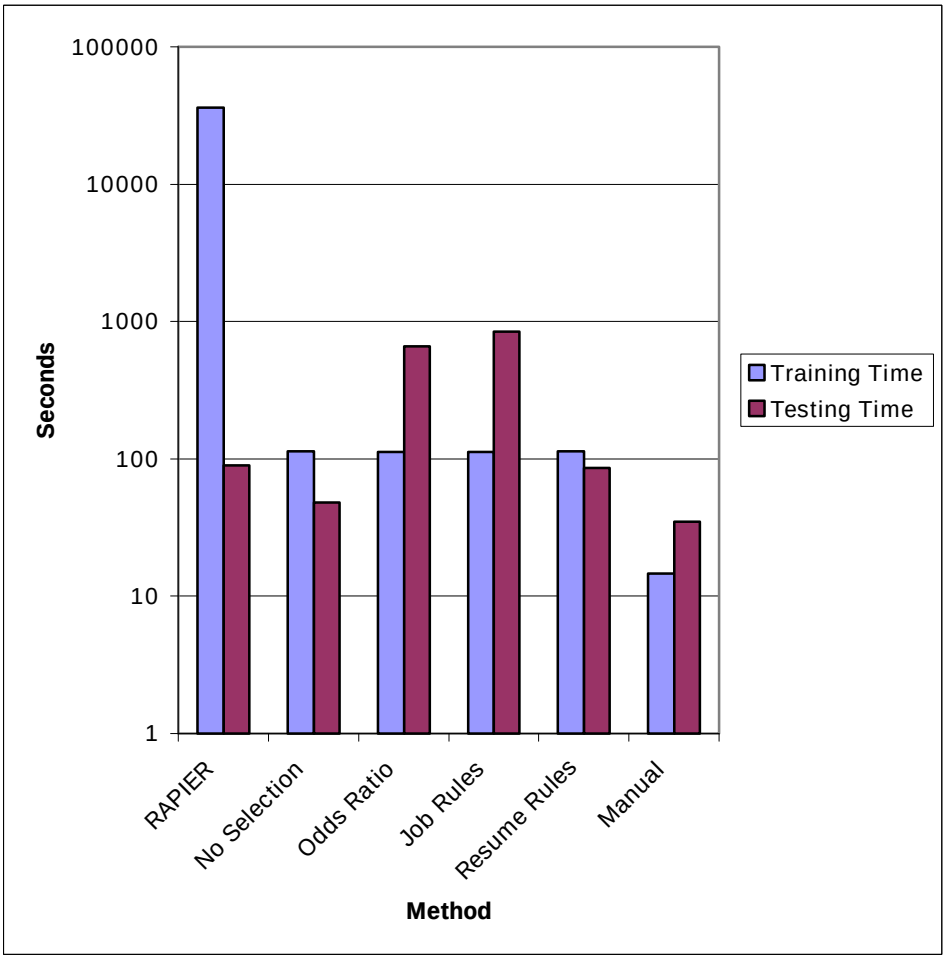

Figure 6.18 Comparing training and testing time between RAPIER and variants of our system in the Jobs domain

RAPIER's training time using 540 documents as reported here is approximate only and is the lowest training time on any split we trained on. The test time for RAPIER is quoted from Section 4.9 of (Califf, 1998), where it was run on a different, supposedly slower, machine.

In terms of training time, however, our system is much faster than RAPIER. Note the logarithmic scale of the Y-axis in Figure 6.18. We can see that RAPIER takes over two orders of magnitude more time to train than all variants of our system. Moreover, our system scales up linearly with the size and number of training documents, while RAPIER apparently

does not (As reported in Section 4.9 of (Califf, 1998), RAPIER's training time increased between 52 and 111 folds when the number of training documents increased from 10 to 270 or 27 folds).

As for testing time, RAPIER is competitive with our system. The variants Odds Ratio and Job Rules of the our system are significantly, but less than one order of magnitude, slower than RAPIER and other variants.

6.11 Discussion

In the information extraction domains we explored, our experiments with manual extraction of features and in learning feature selection rules from the Features dataset suggest the same thing: "It is important to use the right feature types". Selecting for feature types provide better performance improvements than using heuristics such as Odds Ratio, and limiting the window size used in feature extraction.

Although this result may sound obvious to some, it is not a foregone conclusion before these experiments shown it to be so. More important, even when we know that feature types are important, we often do not know a priori what combination of feature types would work best. A clear example in this case is that the Naïve Bayesian information extraction system developed by Dayne Freitag (1998a; 1998b; Califf, 1998) performs much worse than the version of our system that uses feature selection based on feature types. The features used in his naïve Bayes system are based on the frequency of words within a window. Our experiments have shown that this feature type does not work as well as the features based on token distance for this domain. In this case, using inappropriate representation hurts performance significantly. This case study clearly demonstrates the importance of careful feature extraction and selection.

Our own experiments summarized in Section 6.11 also clearly show that the F-1 measure for the cases which we select for feature types approximately doubles compared to no feature selection. Section 6.9 also demonstrates the two domains we explored have slightly different preference for feature representation. Features based on token distance works much better

than those based on token frequency in the Jobs domain; while token-distance features are slightly less effective than token-frequency features in the Resumes domain.

Learning decision lists for deciding the types of useful features was able to detect prominent patterns about feature effectiveness. Despite the fact that the decision list are learned from the Features meta-datasets produced using Odds Ratio as a measure for feature effectiveness, the decision lists turn out to be more effective for feature selection than Odds Ratio itself. This is a surprising result that requires explanation. We believe that this happens because the statistics used to compute Odds Ratio for each individual feature and the given tag are too sparse in most cases (this is particularly true in text learning and pattern recognition applications because there are a large number of features), resulting in unreliable Odds Ratio for individual instances. Machine learning from the Features meta-dataset has the advantage of using a global view based on many features and tags, therefore allowing us to compensate for the sparse-data problem. Further work to explain this phenomenon in a more concrete, theoretical term, might aid future research into the problem of feature selection.

Despite its overall effectiveness, a disappointment we have with the decision lists learned from the Features dataset is that it could not detect some other, less prominent, patterns in feature effectiveness that manual experiments with different feature extraction parameters can identify. For example, the decision lists learned do not include rules that indicate the features more distant from the insertion point are less effective than the features close to it. Another inherent weakness of the rule-based approach is that the learning process cannot in general capture the effects of feature interaction, which may affect performance when a particular combination of features are used together.

The property-based feature engineering framework allows the machine learning practitioner to complement her intuition with careful experiments performed manually or using machine learning from the Feature meta-dataset to gain insight into the types of features most useful with a learning algorithm in the given problem domain. Our experiments show that more effective feature engineering can significantly improve learning results. The improvement tends to be greater in a more complex domain (e.g. Resumes versus Jobs in Section 6.8).

7. Future Work

In this chapter, we identify important future research topics that can make property-based feature engineering more effective and useful.

7.1 Extensions of Property-Based Feature Engineering

7.1.1 Addressing Feature-Category Interaction

In our experiments, the system did not isolate the performance measure for each category (or tag); it measured only global performance based on all categories. It is likely that each category or tag requires a different representation bias, as we have seen from Section 6.9 and Figure 6.11 that the tags <id> and </id> in the Jobs domain can be predicted with near 100 percent precision from the feature of type *Capitalization Form* alone.

Tailoring representation to each tag type might improve extraction accuracy and will also increase the speed in training and testing processes.

When using the rule-learning method, we can generate a meta-dataset for each category and learn rules that are specific to that category. Although the meta-datasets we used in the reported experiments do contain tag type as a property of each instance, the limited number of examples we fed to PART due to memory constraints might have prevented it from learning rules for specific categories. Learning from several meta-datasets, one for each category, will also increase the number of examples the rule learner can use without requiring special parallel implementation or more computational resources.

With the manual approach, we can simply isolate the performance measures for each category and select the best-performing representation for each of them. If we would like to vary the parameters in manual trials to achieve best performance for individual categories, automatic trials-and-errors will be very helpful as the number and range of experiments we need to perform increase with the number of tags. This leads us to the next promising research topic in the following section.

7.1.2 Automated Search for Effective Feature Combinations

Our experiments show that manual experiments can potentially detect subtler patterns about feature effectiveness better than meta learning, since its global perspective can capture feature interaction. However, straightforward manual experiments can be time-consuming and slow. Standard AI techniques such as beam search and genetic algorithms (Russell and Norvig, 1998) can be used to automate the process of trials and revisions of feature combinations and system parameters. This high-level search can revise system settings, perform experiments, record results, keep the best sets of parameters, and continue until a stopping condition has been reached (such as when the extraction accuracy does not improve further after a number of trials).

7.1.3 Feature Weighting

With feature selection, a feature is either filtered out or selected in. When a feature is used, it has an equal weight to every other feature. In many cases, a more general approach of feature weighting might be more desirable, since it allows the system to put different weights on different features. Extending our framework to handle feature weighting is very straightforward. We can add a MEDIUM_EFFECTIVENESS category for the consequence of the rules and assign another bin of discretized Odds Ratio values to this label, for example. More thoughts are required for using feature weighting with different learning algorithms. Feature weighting is natural for some algorithms and less so for others.

7.2 Combining Feature Selection Approaches

We used either property-based or heuristic-based feature selection in each experiment, in order to study and compare their performance. In fact, the two approaches are not mutually exclusive. We can employ both of them together. For example, the learned rules can be used to selectively extract features and then Odds Ratio can be used to filter out some of those extracted. The final performance may be better than using either approach alone.

Another kind of experiments worth performing is using wrappers (Kohavi and John, 1998) to measure feature effectiveness in the meta-dataset. If wrappers are better at measuring feature

performance than Odds Ratio does, we might be able to learn more effective feature engineering rules.

7.3 Applications in Pattern Recognition

We believe that the property-based feature engineering framework will be most effective in the domains that deal with complex patterns. The framework incorporates human intuition about feature types and properties into the definition of features, allowing for richer feature representations and more insightful experiments to be performed. Fields dealing with a multitude of different kinds of features, such as bioinfomatics, image processing, natural language processing, computer vision, and robotics will likely benefit the most from this new approach.

8. Conclusions

Feature engineering and selection make major differences in learning performance in information extraction domain. Much of the performance difference observed between systems is not caused by the learning algorithm, but the representation, as our experiments have shown. We suspect that this conclusion also applies to many other domains, especially those characterized by complex patterns.

The framework of Property-based Feature Engineering allows machine learning practitioners to gain insights into the types of features that are or are not useful for each problem domain. This insight is often not obvious a priori to a practitioner. Careful experiments using the framework developed here allow for effective features engineering, which might increase learning performance significantly.

Learning rules for feature selection automates the process of learning insights on feature engineering. It also allows for the transfer of knowledge about feature engineering from one domain to another similar domain. This is particularly useful in the case when the data in the destination domain are scarce while there are enough data in the original domain.

Appendix

The decision list learned from the Jobs-Features dataset. See discussions about the process of learning this list in Chapter 5 and its characteristics in Section 6.7.

```
RULE 1:  ObjectType = Character && Content = t  ==> LOW_EFFECTIVENESS
RULE 2:  ObjectType = Character && Content = @  ==> LOW_EFFECTIVENESS
RULE 3:  ObjectType = Character && Content = o  ==> LOW_EFFECTIVENESS
RULE 4:  ObjectType = Character && Content = d  ==> LOW_EFFECTIVENESS
RULE 5:  ObjectType = Character && Content = Newsgroups  ==> LOW_EFFECTIVENESS
RULE 6:  ObjectType = Character && Content = m  ==> LOW_EFFECTIVENESS
RULE 7:  ObjectType = Special  ==> LOW_EFFECTIVENESS
RULE 8:  ObjectType = Character && Content =
          hub1.bbnplanet.com!news.bbnplanet.com!nntp1.crl.com!tfs.com!
          news.wlk.com!news.onramp.net!news  ==> LOW_EFFECTIVENESS
RULE 9:  ObjectType = Character && Content = a  ==> LOW_EFFECTIVENESS
RULE 10: ObjectType = Character && Content = .  ==> LOW_EFFECTIVENESS
RULE 11: ObjectType = Character && Content =
         relay.us.dell.com!jump.net!uunet!in5.uu.net!newsm.ibm.net!
          newsm.ibm.net!ibm.net!infeed1.internetmci.com!newsfeed.internetmci.com!131.
          103.1.102!news2.chicago.cic.net!iagnet.net!su  ==> LOW_EFFECTIVENESS
RULE 12: ObjectType = Line  ==> LOW_EFFECTIVENESS
RULE 13: ObjectType = Character && Content = news  ==> LOW_EFFECTIVENESS
RULE 14: ObjectType = Character && Content = onramp.net  ==> LOW_EFFECTIVENESS
RULE 15: ObjectType = Character && Content = g  ==> LOW_EFFECTIVENESS
RULE 16: ObjectType = Character && Content = w  ==> LOW_EFFECTIVENESS
RULE 17: ObjectType = Character && Content = 52  ==> LOW_EFFECTIVENESS
RULE 18: ObjectType = Character && Content = VISUAL  ==> LOW_EFFECTIVENESS
RULE 19: ObjectType = Character && Content = from  ==> LOW_EFFECTIVENESS
RULE 20: ObjectType = Character && Content = -  ==> LOW_EFFECTIVENESS
RULE 21: ObjectType = Character && Content = j  ==> LOW_EFFECTIVENESS
RULE 22: ObjectType = Character && Content = h && Distance = -99999.99--5.75 && Class
          = Distance  ==> LOW_EFFECTIVENESS
```

RULE 23: ObjectType = Token && Class = Distance ==> HIGH_EFFECTIVENESS

RULE 24: Default ==> LOW_EFFECTIVENESS

References

L. Asker and R. Maclin (1997) **Feature engineering and classifier selection: A Case Study in Venusian Volcano Detection**. Proceedings of the 14th International Conference on Machine Learning, pp. 3-11, Morgan Kaufmann.

H. Ade, L. de Raedt, and M. Bruynooghe (1995) **Declarative Bias for Specific-to-General ILP Systems.** Machine Learning, 20, 63-94.

D. W. Aha and R. L. Bankert (1995) **A comparative evaluation of sequential feature selection algorithms.** Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics, pp. 1-7.

J. Baxter (1998) **Theoretical Models of Learning to Learn.** In Learning to Learn, S. Thrun, and L. Pratt (eds.), pp. 71-94, Kluwer Academic Publishers.

C. E. Brodley (1995) **Recursive Automatic Bias Selection for Classifier Construction.** Machine Learning, 20, 63-94.

S. W. Bennett, C. Aone, and C. Lovell (1997) **Learning to Tag Multilingual Texts Through Observation.** Proceedings of the Second Conference on Empirical Methods in Natural Language Processing, pp. 109-116.

M. E. Califf. (1998) **Relational Learning Techniques for Natural Language Information Extraction.** Ph.D. Thesis, Department of Computer Sciences, University of Texas at Austin.

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

C. Cardie (2000) **A Cognitive Bias Approach to Feature Selection and Weighting for Case-Based Learners.** Machine Learning, 41, 85-116.

R. Caruana (1998) **Multitask Learning.** In Learning to Learn, S. Thrun, and L. Pratt (eds.), pp. 95-133, Kluwer Academic Publishers.

P. K. Chan and S. J. Stolfo. (1993) **Experiments on multistrategy learning by meta-learning.** In 2nd International Conference on Information and Knowledge Management.

P. K. Chan and S. J. Stolfo (1995) **A Comparative Evaluation of Voting and Meta-learning on Partitioned Data.** Proceedings of the Twelfth International Conference on Machine Learning, pp. 90-98, Morgan Kaufmann.

W. W. Cohen. (1990) **An analysis of representation shift in concept learning.** Proceedings of the Seventh International Conference on Machine Learning, Austin, Morgan Kaufmann.

W. W. Cohen (1995) **Fast effective rule induction.** Proceedings of the Twelfth International Machine Learning Conference, pp. 115-123.

DARPA (Ed.) (1992) Proceedings of the Fourther DARPA Message Understanding Evaluation and Conference, San Mateo, CA. Morgan Kaufman.

P. Domingos and M. Pazzani (1996), **Beyond independence: conditions for the optimality of the simple Bayesian classifier.** Proceedings of the Thirteenth International Conference on Machine Learning, Morgan Kaufmann.

E. Frank and I. H. Witten (1998) **Generating Accurate Rule Sets Without Global Optimization.** Proceedings of the Fifteenth International Conference on Machine Learning, pp. 144-151, Morgan Kaufmann.

D. Freitag (1998a) **Multistrategy learning for information extraction.** In Proceedings of the 15th International Conference on Machine Learning, pp. 161-169, Morgan Kaufmann.

D. Freitag (1998b) **Toward general-purpose learning for information extraction.** In Proceedings of COLING/ACL-98.

J. Game (1999) **Discriminant trees.** Proceedings of the Sixteenth International Machine Learning Conference, pp. 134-142.

D. F. Gordon and M. desJardins (1995) **Evaluation and Selection of Biases in Machine Learning.** Machine Learning, 20, pp. 5-22.

R.D. King, C. Feng, and A. Shutherland. (1995) **STATLOG: comparison of classification algorithms on large real-world problems.** Applied Artificial Intelligence, 9(3). May/June 1995, 259-287.

R. Kohavi and G. H. John (1998) **The Wrapper Approach.** In Feature Extraction, Construction and Selection: A Data Mining Perspective, Huan Liu and Hiroshi Motoda (eds), pp. 33-50.

R. Kohavi, P. Langley, & Y. Yun (1997) **The utility of feature weighting in nearest-neighbor algorithms.** ECML-97.

D. Koller and M. Sahami (1996) **Toward Optimal Feature Selection.** Proceedings of the Thirteenth International Conference on Machine Learning, pp. 284-292, Morgan Kaufmann.

G. John, R. Kohavi, and K. Pfeger (1994) **Irrelevant features and subset selection problem.** Proceedngs of the Eleventh International Conference on Machine Learning. Morgan Kaufmann.

A. McCallum, D. Freitag, and F. Pereira. (2000) **Maximum entropy Markov models for information extraction and segmentation.** Proceedings of the Seventeenth International Conference on Machine Learning, 2000.

A. McCallum and K. Nigam. (1998) **A comparison of event models for Naive Bayes text classification.** AAAI-98 Workshop on Learning for Text Categorization, 1998.

G. A. Miller (1956) **The magical number seven, plus or minus two: Some limits on our capacity for processing information.** Psychological Review, 63, 81-97.

D. Mladenic, and M. Grobelnik (1999) **Feature selection for unbalanced class distribution and Naive Bayes.** Proceedings of the sixteenth International Conference on Machine Learning, pp. 258-267, Morgan Kaufmann.

I. Muslea. (1999) **Extraction Patterns for Information Extraction Tasks: A Survey.** Workshop on Machine Learning for Information Extraction.

A. Y. Ng (1998) **On Feature Selection: Learning with Exponentially many Irrelevant Features as Training Examples.** Proceedings of the Fifteenth International Conference on Machine Learning.

B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. (2000) **Meta-learning by landmarking various learning algorithms**. Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann.

F. J. Provost and B. G. Buchanan (1995) **Inductive policy: The pragmatics of bias selection.** Machine Learning, 20, 35-61.

E. Riloff and R. Jones (1999) **Learning dictionaries for information extraction using multi-level bootstrapping.** Proceedings of the Sixteenth National Conference on Artificial Intelligence, pp. 474-479.

A. Robins (1998) **Transfer in Cognition.** In Learning to Learn, S. Thrun, and L. Pratt (eds.), pp. 45-67, Kluwer Academic Publishers.

D. L. Silver, and R. E. Mercer (1998) **The Parallel Transfer of Task Knowledge Using Dynamic Learning Rates Based on a Measure of Relatedness.** In Learning to Learn, S. Thrun, and L. Pratt (eds.), pp. 213-233, Kluwer Academic Publishers.

I. Stahl (1995) **The Appropriateness of Predicate Invention as Bias Shift Operation in ILP.** Machine Learning, 20, 95-117.

S. Soderland. (1997) **Learning to extract text-based information from the world wide web.** In Proceedings of Third International Conference on Knowledge Discovery and Data Mining (KDD-97).

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

S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. (1995) **Crystal: Inducing a conceptual dictionary.** Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 1314-1319.

S. Thrun (1998) **Lifelong Learning Algorithms.** In Learning to Learn, S. Thrun, and L. Pratt (eds.), pp. 181-209, Kluwer Academic Publishers.

S. Thrun and J. O'Sullivan (1998) **Clustering Learning Tasks and the Selective Cross-Task Transfer of Knowledge.** In Learning to Learn, S. Thrun, and L. Pratt (eds.), pp. 235-257, Kluwer Academic Publishers.

S. Thrun and L. Pratt (1998) **Learning to learn: introduction and overview.** In Learning to Learn, S. Thrun, and L. Pratt (eds.), pp. 3-17, Kluwer Academic Publishers.

D. Wettschereck, D. W. Aha, and T. Mohri (1997) **A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms.** Artificial Intelligence Review, 11, 273-314. Also available as NCARAI Technical Report AIC-96-006

I. H. Witten and E. Frank (2000) **Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.** Morgan Kaufmann.

Vita

Noppadon Kamolvilassatian was born in Bangkok, Thailand, in 1978. He grew up in Hatyai, a southern city of Thailand. He attended Sangthong-wittaya school for his elementary education and later studied in Hatyai-wittayalai school for his secondary education. He worked for his Bachelors of Computer Engineering degree at Prince of Songkla University. All three institutions mentioned are in Hatyai, his hometown.

In addition to school English, he studied the language with native English speakers at an extracurricular English center. Learning English there helped him break out of the pattern that most university graduates in Thailand cannot communicate well in English. That language learning experience has proved to be of tremendous value in his past, present, and future endeavors.

After graduation with a Bachelor of Engineering degree in 1998, he taught at the department he graduated from for two and a half years. During the time, he lectured on a number of subjects to several hundred students, supervised the senior projects of 10 students, and helped draft a new Bachelor's curriculum. He also published a paper on computer-aided learning based on his senior project and a book on electronic commerce.

In 2000, he came to the US to study for a Masters degree at the University of Texas at Austin. The Fulbright scholarship generously supported him during this Masters study.

The most valuable knowledge he learned during this period are not specific details of the subjects he studied, but knowledge of the US culture and people, research methodology, the philosophy of science, and the value of friendships. Moreover, he discovered where his personal interests and inclinations lie and understand better where he can contribute the most to the world.