Information Extraction for On-line Job Advertisements

Kwok-Chung Au and Kwok-Wai Cheung
Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Hong Kong
{william, henryau}@comp.hkbu.edu.hk

Abstract

The Web has widely been used as a low-cost and yet effective way to disseminate various kind of information, where job advertisements (ads) posting is one of the typical examples. Unlike many typical Web documents, on-line job ads contain grammatical, telegraphic as well as ungrammatical text. Their writing styles and layout structures are much less formal which make the traditional NLP-based information extraction techiques simply fail. The pattern matching approach is commonly used instead for this kind of documents. In this paper, an information extraction system using the pattern matching approach is described. The system represents the extraction pattern using lexical and typesetting information and the extraction rules are derived automatically via an induction process. Via experiments, we show that the proposed system can automatically choose the correct rule representation via the induction. In particular, the system can successfully extract company names from on-line job ads with an extraction accuracy of 92.13%.

1 Introduction

The Web has widely been used as a low-cost and yet effective way to disseminate various kind of information. However, the rapid growth of the Web leads to the well-known *information overload* problem and thus searching relevant information embedded in heterogeneous on-line repositories becomes a non-trival process. To alleviate the problem, some sophisticated tools (information extraction systems) for extracting semantic information from Web pages are inevitably required.

Information extraction (IE), originated from the natural language processing community, has a history of more than a decade and many related systems have been built [8] with the objective to extract target information from a large collection of documents. Related IE systems proposed in the literature include Rapier [2], WHISK [10] and SRV [4]. In terms of *representation*, these systems use a set of extraction rules to model the local patterns of the target information for identification. For extracting information from grammatical free text (e.g., articles and news), part-of-speech (POS) tagging (using a syntactic analyzer) and semantic tagging (using some dictionaries) are commonly adopted to first preprocess the document. Then, extraction rules can be specified using syntactic and semantic constraints for the local patterns.

For text with HTML layout tags, the use of HTML tags are found to be effective. Related extraction systems are commonly called *wrappers*. WIEN [6] is a wrapper which consists of various grammar classes of HTML tags, each corresponds to the extraction of information tagged in a particular grammatical structure. SoftMealy [5] summarizes the grammatical structures using a finite-state transducers. STALKER [9] allows extraction rules to be integrated in a hierarchical manner where the extraction rules can be embedded into one another.

For on-line advertisements like job ads, their contents seldom contain complete grammatical sentences. Instead, short and concise phrases with some HTML tags revealing the documents' layout (semi-structured text) are found. Most of the aforementioned techniques which use POS tagging will simply fail. On the other hand, performing the extraction solely based on the HTML tags is also insufficient. For example, it is common for a list of job requirements to be included within a paragraph marked with <p>. It is by no means the HTML tags can help extracting the individual information items out. The system we described in this paper is developed mainly for extracting information from these kind of advertisement documents.

The pattern matching approach is used in our system to be described in this paper where we represent the extraction patterns using only the words and their typesetting attributes. Rapier [2] is one of the extraction systems closest to our proposed system and has also been applied to job ads. The main difference between our system and Rapier is that Rapier uses POS information while we use typesetting attributes instead. It turns out that quite a number of specially designed pattern classes (later on called token classes in Section 2.1) are required and they are unique to the application. Due to the different representations, the underlying rule induction algorithm is also different.

We have evaluated our system using a dataset containing job ads ($\sim$500) collected from a local on-line classified post and obtained an accuracy of 92.13%. The remaining of the paper is organized as follow. In Section 2, we describe our rule representation and then the rule induction algorithm in Section 3. Section 4 is devoted to the experiment setup and evaluation. Section 5 concludes the paper.

2 Representation

Our proposed system adopts the pattern matching approach for information extraction. So, one of the important issue is the choice of rule representation, which greatly affect the power of the system. We model patterns of target information using a composition of:

Tokens which are derived from a set of predefined *token classes*, each characterizing a particular type of document items, and

Word Lists which are formed by collections of possible words.

2.1 Token Classes

Our system uses eleven different token classes inspired from Hsu *et al.*'s work [5]. Six of the classes are related to different types of typesetting and thus called *typesetting tokens*. [1] The remaining five are related to other item types and are here called *special tokens*.

- *Typesetting Tokens:*

  Uppercase Strings: [Upper,#num], e.g., "HONG KONG".

  Lowercase Strings: [Lower,#num], e.g., "computer science".

  Capitalized Strings: [Cap,#num], e.g., "Baptist University".

  Mixed Strings: [Mixed,#num], e.g., "MicroStation".

  sCapitalized Strings: [sCap, #num], e.g., "i-Cable, 7-Eleven".

  sUpper Strings: [sCap, #num], e.g., "i-CABLE".

- *Special Tokens:*

  Lexicon: e.g., [<Company>]. (to match strings predefined in the associated lexicon file)

  Punctuation Symbols: [Punct,#num], e.g., ",", ":", ...

  Numeric Strings: [Number,#num], e.g., "2000".

  Newline Character: [NEWLINE]. Match only the newline character

  HTML Tags: e.g., ["<HTML>"]. (default case insensitive)

where #num denotes the occurrence parameter of a token and is an integer greater than zeros. It corresponds to the upper bound of consecutive occurrences of the associated token instance.

2.2 Word Lists

Using solely the token classes for rule representation will result in systems which are over-generalized to extract all the formatted entities inside a document and thus a high false alarm rate. To reduce the false alarm rate, we use word lists as another cue orthogonal to token classes to restrict the over-generalized token-based extraction rules. To differentiate from the Lexicon token class, these word lists contain some more subtle vocabularies specific to particular application domains (e.g., company names, job natures) while the Lexicon token class refers to some more obvious vocabularies which can be easily created in an *a priori* manner like sets of synomyns.

2.3 Rule Modeling

Based on the token classes and their associated word lists, *extraction rules* can be defined to characterize the target information to be extracted. In our system, an extraction rule consists of five components, namely

1. the prefix sub-pattern right ahead of the target pattern, **b**,

2. the beginning sub-pattern of the target pattern, **B**,

3. the ending sub-pattern of the target pattern, **E**,

4. the suffix sub-pattern right after the target pattern, **e**, and finally

5. the maximum number of strings allowed in between **B**and **E**, **g**.

See Figure 1 for an illustrated example.

[1]Note that there are some token classes which are specifically designed for modeling company names.

3 Rule Induction

Based on the rule representation, one can manually create a set of extraction rules using domain knowledge. Thus, domain experts with prior experience in designing the extraction rules will be required, and yet trial-and-error iterations are inevitable in most of the cases. Inductive learning techniques have long been known to be useful for automating the rule generation process. For our system, the set of extraction rules is induced using a bottom-up learning process based on manually tagged training data, as described in Section 3.1.

3.1 Rule Learning

In order to induce the extraction rules, a set of on-line documents with the target information being tagged is required. Based on the tagged documents, individual target items together with their neighboring items can readily be obtained and form the training dataset for the subsequent learning. The learning process we adopted involves three stages, namely, *tokenization*, *initial rule set generation* and *rule generalization*.

3.1.1 Tokenization

This is the first stage where words within each training sample are tokenized according to the token definition. Some tokens are exclusive to each other while some are not. So, there are cases where an item can be represented by more than one token classes. For example, the word "Company" can be tokenized as a Lexicon token or a Capitalized token. We resolve these conflicts using a pre-defined precedence order. The logic behind our precedence order is to have the more restricted token classes to be used first. So, the Lexicon token has the highest priority, followed by the different format tokens, and lastly the String token. During our development, we found that many abbreviations are used within company names. This results in the fact that many capitalized or upper-case strings contain "dot" signs in them, e.g., "H.K.". Therefore, our implementation of those tokens deliberately allow the existence of some pranctuations (in particular ".").

For example, the job ad segment

"<u>Fortis Clearing (OPTIONS) Hong Kong Ltd</u> (a member of ..."

can be tokenized as:

```
"nil ** (Cap)(Cap) (Upper) (Cap)(Cap)
(<Company>) ** (Prun) "
```

where the two "**" indicate the boundaries between the target pattern and its neighborhood. Here we assume that only one item is to be extracted from the neighborhood (single-slot rules).

3.1.2 Initial Rule Set Generation

With all the items being tokenized in each sample, an intermediate format rule representation ($\hat{\mathbf{b}}$, $\hat{\mathbf{B}}$, $\hat{\mathbf{E}}$, $\hat{\mathbf{e}}$, $\hat{\mathbf{g}}$) is constructed by going through the following sequence of steps:

- "$\hat{\mathbf{b}}$": Scan from the left boundary towards left and stop just before the token type changes. Count the corresponding number of token instances encountered to compute the occurrence parameter of that token.
- "$\hat{\mathbf{B}}$": Scan from the left boundary towards right and repeat the same process.
- "$\hat{\mathbf{E}}$": Scan from the right boundary towards left and stop just before the token type changes or when a previously scanned item is encountered. Again, compute the occurrence parameter if needed.
- "$\hat{\mathbf{e}}$": Scan from the right boundary towards right and again stop just before the token type changes. Also, compute the occurrence parameter by counting.
- "$\hat{\mathbf{g}}$": Count the items between $\hat{\mathbf{B}}$ and $\hat{\mathbf{E}}$ which have not been scanned.

Thus, the rule generated for the above job ad segment is

" $\hat{\mathbf{b}}$=nil, $\hat{\mathbf{B}}$=(Cap,2), $\hat{\mathbf{E}}$=(<Company>), $\hat{\mathbf{e}}$=(Punct,1), $\hat{\mathbf{g}}$=3."

See Figure 1 for a pictorial illustration. Note that we use "()" instead of "[]" to indicate that it is an intermediate rule representation (with exact-length tokens) where the occurrence parameters of the tokens are interpreted as the exact counts, instead of the upper bound as defined in Section 2.1.

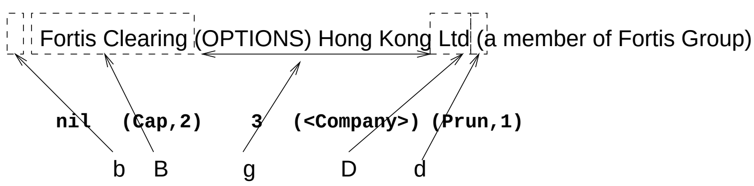

Figure 1. An example of intermediate format rule representation.

Lastly, the words associated with each token are added to complete the initial rule set generation, as shown in Figure 2.

$\hat{b}$	$\hat{B}$	$\hat{E}$	$\hat{e}$	$\hat{g}$
Format: nil List: nil	Format: (Cap,2) List: Fortis, Clearing	Format: (<Company>) List: nil	Format: (Punct,1) List: (	3

Figure 2. The Complete Rule Representation

3.1.3 Rule Generalization

After the inital rule set generation step, each training sample generates exactly one rule with obviously limited generalization capability. A bottom-up covering process (similar to the one used in [1]) is then applied to generalize the rules. The basic idea is to first group rules based on the token types to form a number of disjoint rule subsets. For each subset, we generalize the rules in it in a pair-wise manner with the goal of forming a rule set covering the largest number of examples without extraction errors. For example, the rule shown in Figure 2 belongs to the rule subset "`nil#Cap#<Company>#Punct`". During the generalization process, a pair of rules from it will be randomly picked. and generalized to a set of less specific rules. The maximum occurrence parameter of the rule pair is used in the newly generated rules. For example, if "Cap,1" is the beginning token of the first rule and "Cap,3" is that of the second, the final beginning token will be "Cap,3". We treat the maximum number of strings allowed parameter (**g**) in a similar manner. For the associated word lists, we combine the word lists of the two rules to form a new rule, and drop the word lists to form another.

These newly generated rules will not be adopted at once. They need to be tested with the training set to see if they can match all training data correctly. To avoid the rules to be too specific to the training data (overfitting), a certain degree of matching errors (later on called the *tolerated error rate*) is allowed. So, each new rule is first matched with the training data and the numbers of correct and failed matches are recorded. Here, we consider correct company name extractions as well as empty extractions from ads without company names to be correct matches. All the other cases are considered as failed. If the percentage of failed matches exceeds the tolerated error rate, the new rule will be discarded.

New rules that can survive will then undergo a *subsuming* process. Subsuming is a process that replaces specific rules by more generalized rules, i.e. can cover more data. For example, if there are two rules with identical format, the rule without a word list will be more general the one with a word list. In other words, the former rule subsumes the latter one and thus the latter can be removed. If both rules have a word list with them, the latter rule's word list should contain all the words in the former rule's word list to perform subsuming. After subsuming of the old rules in the group, the new rules will finally be added into the group. This generalization process iterates until there is no more change in the number of rules in the group for a certain number of iterations. Figure 3 summarizes the overall learning process.

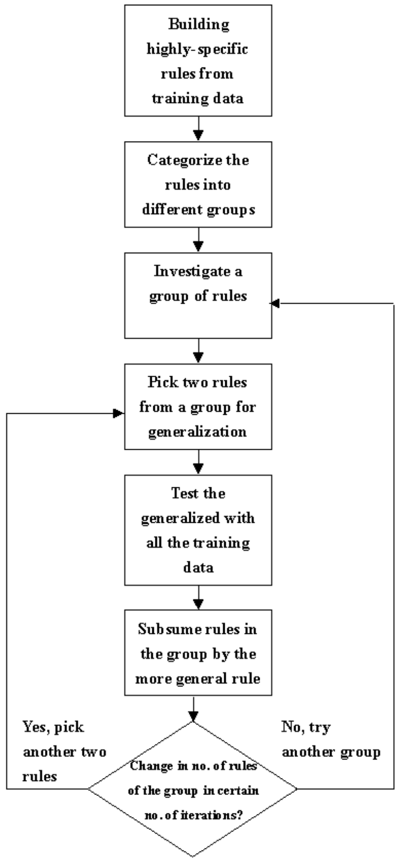

Figure 3. The whole process of rule generalization.

We believe that such a generalization process can eliminate rules with tokens carrying a big trunk of words as far as possible and retain those with tokens carrying a short list of important keywords.

3.2 Pattern Extraction

After rule learning, the resultant set of rules forms the representation of the target pattern. By applying the rules to an unknown input document of the same application domain, the system is expected to be able to extract all the relevant patterns out and only the relevant patterns. Any spurious extraction results should be considered as failure cases. It is also expected that the order of the rule application should be independent to the extraction results, i.e. the extraction results will not be affected by the order of applying the rules.

4 Experiments

A prototype of the proposed system has been implemented in Perl 5. To evaluate the system performance, we have created a dataset by retrieving job ads from a local online classified post and preprocessed them so that irrelevant information like company logos, graphical ad banners, etc. are removed. A total of 561 job ads which have been preclassified into 22 job categories, are solicited to form our dataset. Among the 561 job ads, only 241 of them contain company names.

4.1 Preprocessing

All the job ads in the dataset are in HTML format. As the core information of the job ad is found to be presented in a consistent manner (embedded inside a particular table), the preprocessing step is as simple as writing some simple pattern matching rules based on HTML tags. In general, this very first extraction step may not necessarily be that simple [3].

4.2 Extraction of Company Names

The proposed system has been applied to extract company names from the preprocessed job ads as described in Section 3.2. The labeled data are divided into the training set and the test set and cross-validation has been used to reduce the sampling bias. The rules are learned based on the training set and applied to the test set for evaluation. In order to demonstrate the effect of different rule representations and the advantage of using the induction process for rule creation, we set up four different experiments. In Experiment 1, we use the rule representation that contains only token classes but not word lists. Note that rule induction is not needed in this experiment. In Experiment 2, we allow the use of word lists in the prefix and suffix sub-patterns only. In Experiment 3, we add word lists in the target pattern representation instead. In both Experiment 2 and 3, the rule generalization process as described in Section 3.1.3 is used. Lastly, in Experiment 4, we allow the use of word lists in all parts of the rules and rely on the system to keep or drop the word lists or the token classes in both the prefix/suffix sub-patterns as well as the target patterns. Performance comparison based on the extraction accuracy is shown in Table 1, 2, 3 and 4 (also revealed in Figure 4). According to Figure 4, it is first observed that as the size of training set increases, the extraction accuracy increases monotonically in all the experiments. For the comparison of different rule representations, we see that introducing word lists in the prefix and suffix sub-patterns results in significant performance degradation (Experiment 2). This signs that including word lists only in the representation of the prefix and suffix sub-patterns fails to enhance the model accuracy.[2] However, significant performance improvement is achieved by adding word lists to the target pattern representation instead (Experiment 3). Lastly, the best performance is achieved by allowing word lists in all parts of the rule representation (Experiment 4). This evidences the system's capability of finding the optimal rule representation automatically. Other than comparing different representations, we have also evaluated the use of the tolerated error rate in enhancing the system's generalization performance. As described in Section 3.1.3, setting the tolerated error rate to $r\%$ means that an induced rule is admitted into the rule set as far as it makes less than $r\%$ extraction errors on the training set. In particular, when $r = 0$, all the induced rules are not allowed to make any error at all on the training set, and in general cannot result in optimal generalization performance. Figure 5 shows the performance comparison based on different values of r. We note that the use of the tolerated error rate always helps when the number of training data is significantly small (see the performance when the number of training data is 35). Also, we found that consistant improvement can be obtained by allowing a small degree of errors (tolerated error rate = 0.2).

# Training	35	49	81	161	193	207
Accuracy(%)	71.03	72.17	77.24	82.79	82.95	83.72

Table 1. Extraction results of experiment 1 where only typesetting information is used for the rule representation (tolerated error rate=0).

# Training	35	49	81	161	193	207
Accuracy(%)	68.76	70.38	74.20	79.76	79.90	80.48

Table 2. Extraction results of experiment 2 where typesetting information and word lists (limited to prefix and suffix) are used for the rule representation. (tolerated error rate=0)

# Training	35	49	81	161	193	207
Accuracy(%)	67.41	72.09	75.90	87.83	89.76	91.23

Table 3. Extraction results of experiment 3 where typesetting information and word lists (limited to company name) are used for the rule representation (tolerated error rate=0)

[2] At least, our existing induction process fails to learn a representation which can perform better than the one using only typesetting information in the representation.

# Training	35	49	81	161	193	207
Accuracy(%)	68.46	72.45	76.80	88.73	91.02	92.13

Table 4. Extraction results of experiment 4 where both typesetting information and word lists are used for the rule representation. (tolerated error rate=0)

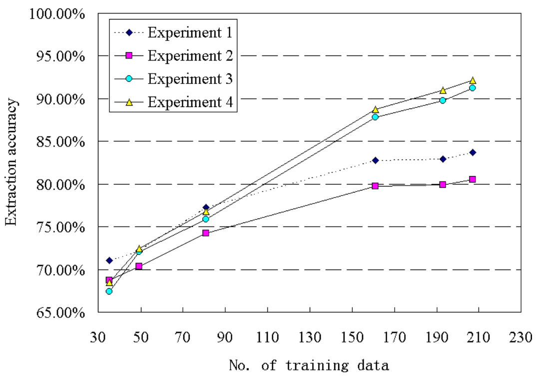

Figure 4. Comparing the performance of three different rule representations given the tolerated error rate = 0.

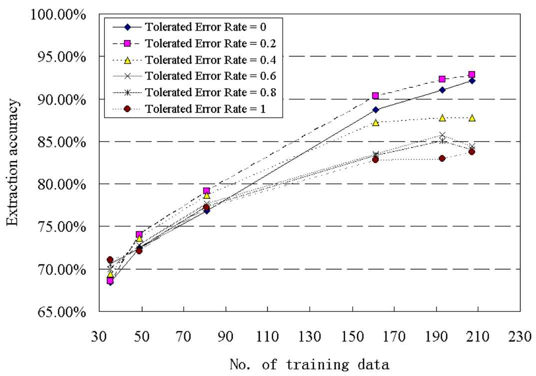

Figure 5. Comparing the performance for different values of tolerated error rates. Both word lists and token classes are used here for the rule representation.

5 Conclusion

In this paper, we have described an information extraction system we developed and show that our approach works well in company name identification in job ads (with an accuracy of 92.13%). We believe that extending it to extract job titles should be straight forward as job titles are normally highlighted in jod ads. However, when dealing with target items related to job natures and job requirements, typesetting matching will not be useful. Also, rule-based approaches are falling short in evaluating the strength of the evidence that guides extraction decisions [7]. More robust extraction technique are required. Currently, we are investigating the use of hidden Markov models to extract the fields other then company names. Also, coordinating the individual field wrappers to further improve the extraction accruacy is another worth-pursuing direction.

References

[1] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.

[2] M.E. Califf. *Relational Learning Techniques for Natural Language Information Extraction*. PhD thesis, Department of Computer Science, University of Texas, Austin, TX., 1998.

[3] D.W. Embley, D.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.K. Ng, and R.D. Smith. Conceptual-model-based data extraction from multiple-record web pages. In *Data and Knowledge Engineering*, November 1999.

[4] D. Freitag. Information extraction from HTML: Application of a general machine learning approach. In *Proceedings of the Fifteenth National Conference on Articial Intelligence*, pages 517–523, 1998.

[5] C.N. Hsu and M.T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.

[6] N. Kushmerick, D.S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of International Joint Conference on Artificial Intelligent*, 1997.

[7] Nicholas Kushmerick. Finite-state approaches to web information extraction, 2002.

[8] I. Muslea. Extraction patterns for information extraction tasks: A survey. In *Proceedings of AAAI-99 Workshop on Machine Learning for Information Extraction*, Orlando, Florida, July 1999.

[9] I. Muslea, S. Minton, and C.A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 2000.

[10] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, pages 1–44, 1999.