Information Extraction tasks: a survey

Gonçalo Simões, Helena Galhardas, Luísa Coheur

Instituto Superior Técnico, INESC-ID, DMIR, L2F

Abstract. An Information Extraction activity is a complex process that can be decomposed into several tasks. This decomposition brings the following advantages: *(i)* for each task it becomes possible to choose the best technique independently from the other tasks; *(ii)* an Information Extraction program can be developed as a set of independent modules (one for each task), making it easy to perform local debugging; *(iii)* it becomes easy to customize the Information Extraction activity through reordering, selection or even composition the tasks.
This paper presents a commonly used decomposition of the Information Extraction activities and gives detail about the most used machine learning and rule-based techniques for each task.

Keywords: Information Extraction, Natural Language Processing, Machine Learning, Declarative Languages

1 Introduction

With the increasing volume of publicly available information, companies need to develop processes for mining information that may be vital for their business. Unfortunately, much of this information is presented in the form of unstructured or semi-structured texts. Software tools are not able to analyze such texts and humans would take so much time to perform this task that the information would become obsolete by the time it was available.

Information Extraction emerged as a solution to deal with this problem. According to (Cowie & Lehnert, 1996), "*Information Extraction* starts with a collection of texts, then transforms them into information that is more readily digested and analyzed. It isolates relevant text fragments, extracts relevant information from the fragments, and then pieces together the targeted information in a coherent framework".

An *Information Extraction* activity can be very complex. Thus, it is common to decompose it into several tasks. This decomposition offers some advantages. First, it is possible to choose, for each task, the techniques and algorithms that better fit the objective of a particular application. Second, it is easy to locally debug an *Information Extraction* program since the module responsible for each task is completely independent from the others. Finally, an *Information Extraction* can be customized activity according to an application's needs, by reordering, selecting and composing some of the tasks.

This paper presents a possible decomposition of the *Information Extraction* activity in tasks. This decomposition is based on the work of (McCallum, 2005).

The considered tasks are: Segmentation, Classification, Association, Normalization and Correference Resolution. For each of these tasks we explain what is its purpose and which techniques can be used.

2 Segmentation

The *Segmentation* task divides the text into atomic elements, called segments or tokens. Next, we will present some problems that can be found in *Segmentation* for Western languages (Santos, 2002):

- *Whitespaces*: space-separated words may not match the different segments. For example, the string "New York" should be considered a segment.
- *Hyphens*: compound words in which there is a separation with an hyphen (e.g., "daughter-in-law") should be part of the same segment. However, there are situations in which we could separate a segment with an hyphen in two words (e.g. non-lawyer).
- *Apostrophes*: in languages like English or French, it is very common to find situations in which apostrophes are used in contractions. These situations can be difficult to handle. For instance, the expression "s" in "John's" can be the verb "to be" or the possessive form.
- *Full stop*: the full stop usually marks the end of a sentence. However, it can also be used to separate letters in an abbreviation (e.g. "United Nations" can occur as "U.N.").

Usually, these problems are solved using rules that show how these cases should be handled.

The major problems related to this task can be found in oriental languages. For example, the Chinese does not have whitespaces between words (Li Haizhou & Zhiwei, 1998). For this reason, solving the problems described above is not enough in this language. In these cases, it is typically necessary to use external resources. Lexicons and grammars can also be used in order to accomplish the task of segmentation using syntactic or lexical analysis. Another approach for segmentation in Chinese uses techniques based on statistics. An example is the system described in (Li Haizhou & Zhiwei, 1998), which uses *N-grams* and the *Viterbi algorithm* (Forney, 1973) applied to segmentation.

The techniques based on *N-grams* resort to word counts to define a conditional probabilistic model. In the model, the probability of a given word depends on the last N-1 words. As an example, let us consider a vocabulary with only two words, "A" and "B", and a training corpus "A B A A B B A". We want to train a bigram model so we need two types of counts: unigrams and bigrams.

Table 1.1 presents the frequency of each word in the training corpus. Table 1.2 presents, for each word in the lines, the frequency with which it appears in the text after the word in the columns. Note that, in these tables, we consider two symbols that do not appear in the text: $< s >$ which represents the beginning of the corpus and $< /s >$ which represents the ending of the corpus.

Table 1. Unigram (1) and Bigram (2) counts for the training corpus *A B A A B B A*

Table 1.1

$< s >$	A	B	$< /s >$
1	4	3	1

Table 1.2

	$< s >$	A	B
A	1	1	2
B	0	2	1
$< /s >$	0	1	0

As an example of using this model we calculate the probability that the sentence "ABA" occurs. With a bigram model we can compute the occurrence probability of this sentence by Equation (1)

$$P(\text{“}ABA\text{”}) = P(A| < s >)P(B|A)P(A|B)P(< /s > |A) \tag{1}$$

where P(i|j) is computed by the Equation (2)

$$P(i|j) = \frac{Table1.2(i,j)}{Table1.1(j)} \tag{2}$$

Thus, we have P(A| $< s >) = \frac{1}{1}$, P(B|A) $= \frac{2}{4}$, P(A|B) $= \frac{2}{3}$, P($< /s > |A) = \frac{1}{1}$, which means that $P(\text{“}ABA\text{”}) = \frac{1}{1}\times\frac{2}{4}\times\frac{2}{3}\times\frac{1}{1} = \frac{1}{3}$

The *Viterbi Algorithm* is a dynamic programming algorithm used to estimate the most likely sequence of states in a state machine, given a set of observable events. These events constitutes the input sequence of the state machine. The algorithm assumes some facts that may not necessarily be true. First, it assumes that the observed events are in sequence with the states of the states machine. Then it assumes that the probability of the sequence depends only on the event observed in a finite number of previous iterations.

The most likely sequence of states for the observable events, in each iteration, is through the computation of the Viterbi score, which may be computed in different ways depending on the technique used. A detailed example of the *Viterbi Algorithm* will be introduced in Section 3 as a technique for classification.

3 Classification

The *Classification* task determines the type of each segment obtained in the segmentation task. In other words, it determines the field of the output data structure where the input segment fits. The result of this task is the classification of a set of segments as entities, which are elements of a given class potentially relevant for the extraction domain.

The rule-based techniques used in the classification task are usually based on linguistic resources, such as lexicons and grammars (Farmakiotou et al., 2000). The input segments are compared with the elements of the lexicon until there is a match. Additionally, some techniques of morphological analysis can be used as support. The grammar supports the recognition of terms that are not in the lexicon and solves possible ambiguities when an entity has different categories in the lexicon (e.g., when there are homonymy relationships).

One of the most popular approaches to undertake classification is machine learning. Machine learning techniques used in this task are usually supervised, which means that an annotated corpus is needed. Five of the most common supervised learning techniques are the Hidden Markov Models (HMM), Maximum Entropy Markov Models (MEMM) (McCallum, Freitag, & Pereira, 2000), Conditional Random Fields (CRF) (Lafferty, McCallum, & Pereira, 2001), Support Vector Machines (Isozaki & Kazawa, 2002) and Decision Trees (Sekine, Grishman, & Shinnou, 1998).

HMM have been widely used in text classification tasks such as Part-of-Speech tagging that consists of classifying words according to the morphological class. These models are based on: *(i)* a set of hidden states that usually have a physical meaning (e.g., in the classification of tokens we can have each state associated with a type of entity); *(ii)* a set of observable symbols that correspond to observable events (e.g. tokens that occur at each time), *(iii)* a function that returns the probability distribution of the state transition, *(iv)* a function that returns the distribution of probabilities of observing a given symbol in a given state and *(v)* a vector corresponding to the distribution of probabilities in the initial state. A probabilistic model is built during the training phase. The classification is then obtained through the use of the generated model for finding the most likely classification for the test dada. The most likely classification can be computed using the Viterbi Algorithm.

Let us consider an example (Allen, 1994) of a prediction of Part-of-speech tagging. We assume the existence a trained HMM model for this task. This model considers for each word the probability of being a name, a verb, an article or a preposition:

$\mathrm{P}(\mathit{the}\vert\mathrm{ARTICLE}) = 0.540$	$\mathrm{P}(\mathit{like}\vert\mathrm{PREPOSITION}) = 0.068$	$\mathrm{P}(a\vert\mathrm{NAME}) = 0.001$
$\mathrm{P}(\mathit{flies}\vert\mathrm{NAME}) = 0.025$	$\mathrm{P}(\mathit{like}\vert\mathrm{NAME}) = 0.012$	$\mathrm{P}(\mathit{flower}\vert\mathrm{NAME}) = 0.063$
$\mathrm{P}(\mathit{flies}\vert\mathrm{VERB}) = 0.076$	$\mathrm{P}(a\vert\mathrm{ARTICLE}) = 0.360$	$\mathrm{P}(\mathit{flower}\vert\mathrm{VERB}) = 0.050$
$\mathrm{P}(\mathit{like}\vert\mathrm{VERB}) = 0.100$		

As we can see we do not present the probabilities for all the classifications for each words. If one probability is not presented we will consider to be zero. The model also considers the bigram probability of a given classification that depends on the word classification of the word before. This probability is presented as a Markov chain in Figure 1.

We will use this model and the *Viterbi algorithm* to predict the most probable Part-of-Speech classification for the sentence "Flies like a flower". We will assume that any bigram probability that is not shown in Figure 1 has a value of 0.0001.

The *Viterbi algorithm* is divided into three steps. The first one is the *Initialization Step* in which we compute a score for the first word (w_1) in the sentence. This score is given by Equation (3):

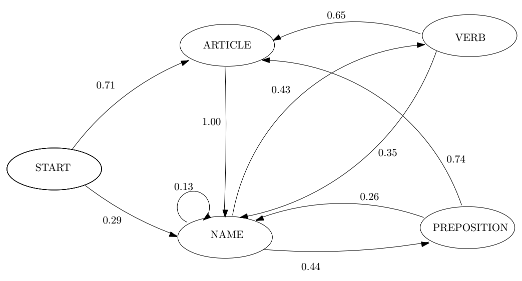

Figure 1. A Markov Model capturing the bigram probabilities of this model

$$v_1(j) = P(w_1|L_j) * P(L_j|START) \tag{3}$$

where L_j is each of the possible classifications: NAME, VERB, ARTICLE or PREPOSITION. For the example, the first word is "Flies". The scores for this word are:

$$v_1(VERB) = 0.076 * 0.0001 = 7.6 * 10^{-6}$$
$$v_1(NAME) = 0.025 * 0.29 = 7.25 * 10^{-3}$$
$$v_1(PREPOSITION) = 0 * 0.0001 = 0$$
$$v_1(ARTICLE) = 0 * 0.71 = 0$$

In the second step of the algorithm (*Iteration Step*), we will scan all the other words and compute the best score for each classification using the Equation (4):

$$v_t(j) = max_{i=1}^{N} v_{t-1}(i) P(Tag_j|Tag_i) P(Word_t|Tag_j) \tag{4}$$

Figure 2 shows the Iteration Step computation for the word "like". The solid arrows show the path which corresponds to the maximum of $v_{t-1}(i)P(Tag_j|Tag_i)$ in Equation (4).

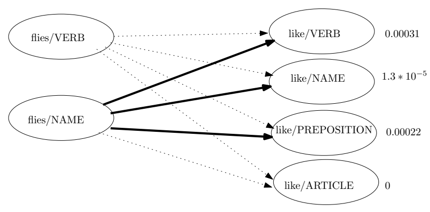

Figure 2. Score for the word "like" in the beginning of the *Iteration Step*

This step is repeated for all the words in the sentence. The result after the step is performed for the last word is shown in Figure 3.

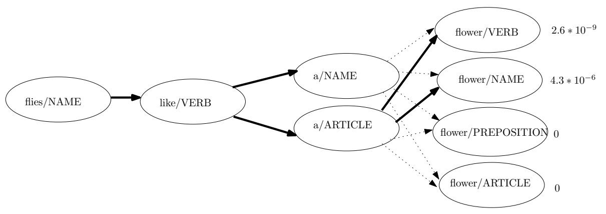

Figure 3. Score for the word "flower" in the *Iteration Step*

In the final step of the Viterbi algorithm (Backward step) the path with the highest score is scanned from the last one to the first one. In this scan, the algorithm gives a final classification to each word. In the example, we can see that the path with highest score is the one in which "flies" is a name, "like" is a verb, "a" is an article and "flower" is a name.

The HMM models for the classification task are based only on two kinds of probabilities: $P(Tag|Tag)$ and $P(Word|Tag)$. This means that the introduction of other knowledge in the classification process (e.g., information about capitalization ou letter size) is not easy. In HMM, if we want to model this knowledge we need to code it only with the two kinds of probabilities presented before. In order to solve these problems, some approaches began to emerge that seek to modify the HMM. Examples of these approaches are MEMM and CRF.

While the HMM use a generative model, which produces a probability density model over all variables in a system, the MEMM use a discriminative model, which makes no direct attempt to model the underlying distribution of the variables. Instead of having separate models for $P(Tag|Tag)$ and $P(Word|Tag)$, the MEMM train a single probabilistic model to estimate $P(Tag_i|Word_i, Tag_{i-1})$. This probabilistic model uses a Maximum Entropy classifier that estimates the probability for each local tag given an observed word, a tag for the prior word and a set of features corresponding to the modeling of additional information we want to introduce in the process.

Like in the HMM, the MEMM use the Viterbi algorithm to infere the tags associated to each word. The only difference between the two algorithms is in the way the probabilities are computed. The MEMM uses a modified version of the Viterbi score used for HMM (Equation 4) that is presented in Equation 5:

$$v_t(j) = max_{i=1}^{N} v_{t-1}(i) P(Tag_j|Tag_i, Word_t) \tag{5}$$

MEMM are a good solution if we want to introduce additional information in the statistical model but they suffer from a limitation that makes them weaker than HMM when no additional information is used. Consider the following example of a statistical model shown in Figure 4.

If we observe the local probabilities for each observation we can see that for state 1, the transition to state 2 is usually the one with the highest probability. We can see that the same happens with state 2. But if we use the Viterbi

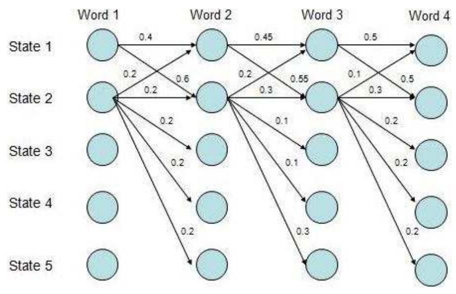

Figure 4. Example of a MEMM model with the Label Bias Problem

algorithm to compute the most probable path we find out that it is the path 1→1→1→1.

This happens because there are five transitions out of state 2 and only two transitions out of state 1. Since the probabilities of the transition from a given state in a MEMM are normalized, the model usually prefers states with lower number of transitions. This gives an unfair advantage to these states. This is called the *label bias problem*.

CRF were developed as a solution that offers all the advantages of MEMM but overcomes the label bias problem. The difference between CRF and MEMM is that the MEMM model uses an conditional probabilities exponencial model for each state while CRF uses a single exponencial model for the joint probability of the entire sequence of labels given the observation sequence. With this model it is possible to normalize the probabilities at a global level.

The Viterbi value used in the Viterbi algorithm for CRF is different from the ones used in HMM and MEMM. It is given by:

$$v_t(j) = max_{i=1}^{N} v_{t-1}(i) e^{\sum_k (\lambda_k f_k(Tag_i, Tag_j, Sentence, t))} \tag{6}$$

In Equation 6 f_k is a feature function. For each additional information about words of the text we want to use in the process there will be a different feature function. λ_k is a parameter estimated in the training phase that represents the importance of each feature function in the process.

The great disadvantage of the CRF models is that the training is very expensive, making them difficult to use if we want to keep training the model as long as new data appears.

Support Vector Machines assume that it is possible to map the segments of the text in a vector space according to linguistic (e.g., lexical information) or graphical (e.g., position or style in the text) properties of the segments and the words in its neighboring. Text segments are mapped into a vector space. Then, the ideia is to separate positive elements (elements that belong to the class) from negative elements (elements that do not belong to the class) by an hyperplane. In the training phase, the objective is to find the hyperplanes that achieve the better separation between positive and negative examples. In the testing phase, the classification is performed by looking at which side of the hyperplane the input is.

Let us consider the example of a classification of lines in a text as titles. We will use two graphical properties to classify the lines: the size of the letters and the line numbers. For the training phase some positive and negative examples are given. The examples are shown in a bidimensional space in Figure 5. The positive examples are the dots and the negative examples are the stars. Figure 5 also shows the best choice for the hyperplane.

In the testing phase, each line is mapped in the bidimensional space as done with the training examples and the classifier uses the hyperplane to classify it. Figure 5 shows two new lines (a square and a cross) that will be classified with this model. The square is positioned below the hyperplane, so it is considered that it is not a title. The cross is positioned above the hyperplane, so it will be classified as a title.

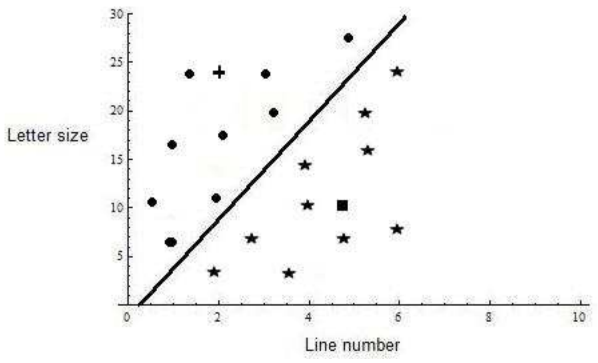

Figure 5. Testing phase in a SVM classifier

Decision Trees are also used for text classification. An example is described in (Sekine et al., 1998) for classifying Japanese texts. During training, a probabilistic decision tree is built based on the words' morphologic classification, type of characters and information of the dictionary in the neighborhood of the word (word right before, the word itself and the word right after). In the testing phase, the properties of the neighborhood of each word are analyzed and compared with the decision tree in order to associate a probability of belonging to a given class. Given that the probabilities of all the segments are computed, the task consists in discovering the most consistent sequence of probabilities. For that the Viterbi Algorithm can be used.

Let us consider a task in which we want to classify words as locations. We will use the decision tree of Figure 6 for this task.

From this decision tree we can get the probabilities of a given word to be or not to be a Location. Consider the sentence "I am going to Washington". To compute the probabilities associated to "Washington" we start from the root of

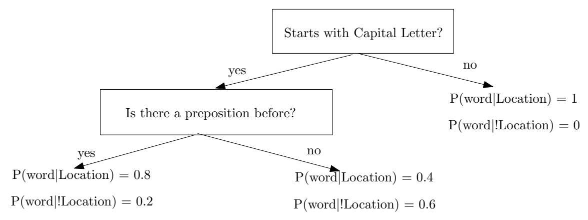

Figure 6. Simple Decision Tree to compute the probability of a word being a Location

the tree. Since “Washington” starts with capital letter, the algorithm follows the left path and gets to another node. In this node we need to observe the word before “Washington” and see if it is a preposition. Since “to” is a preposition we follow the left path again. We conclude that “Washington” has 80% chance of being a Location.

4 Association

The *association* task seeks to find how the different entities found in the classification task are related. The systems that perform extraction of relationships are less common than the ones that perform the classification task (McCallum, 2005). This happens due to the difficulty in achieving good results in this task.

Many techniques in the association task are based on rules. The simplest approach uses a set of patterns to extract a limited set of relationships. For instance, we can extract an affiliation relationship between a person and a company with the rule (7), in which, upon detection of the standard <Person> works for <Company>, the values of <Person> and <Company> are inserted in the relation worksFor(Person,Company).

$$< Person > worksfor < Company > \rightarrow worksFor(< Person >, < Company >) \tag{7}$$

Such an approach solves only very simple cases, where a big variety of relationships is not expected, since we need to develop a different rule to extract a different relationship.

A more generic rule-based approach for association is based on *syntactic analysis*. Often, the relationships that we want to extract are grammatical relationships (Grishman, 1997). For example, a verb may indicate a relationship between two entities. A complete syntactic analysis, where all the text is analyzed with a single syntactic tree, has the inconvenient of being very expensive and causing many errors (Grishman, 1997). Partial syntactic analysis, in which the text is divided into parts where each part has its syntactic tree, returns better results. With this kind of analysis some linguistic patterns, as conjunctions or modifiers, are ignored.

The association task can also use machine learning techniques. One of the first machine learning approaches was based on probabilistic context-free grammars (Miller et al., 1998). These grammars differ from regular context-free grammars, because they have a probability value associated to each rule. When the syntactic analysis is undertaken, it is possible to find many syntactic trees. By using probabilistic rules, the probability of each tree is computed and the most probable tree is chosen.

Since HMM (refered in Section 3) are usually more suitable for modeling local problems, its use for the association task is not the most appropriate (Zelenko, Aone, & Richardella, 2003). However, other approaches like MEMM and CRF (Section 3) may be adapted for the association task. To do so, all we need to do is to add information about the neighborhood of a word in the form of features. Instead of tagging each word with a given class, these models would tag with relationships with other words in the text.

5 Normalization and Correference Resolution

Normalization and Correference resolution are the less generic tasks of the *Information Extraction* process (Applet & Israel, 1999), since they use heuristics and rules that are specific to the data domain.

The *normalization* task is required because some information types do not conform to a standard format. For instance, there can be several representations of hours like "3pm", "3h", "15:00". This format heterogeneity may pose difficulties in the comparison between entities.

The normalization task transforms information to a standard format defined by the user. In the example of the hours mentioned above, the user can define that all the hours shall be converted to a standard format, for example, "15h00". This task is typically achieved through the use of conversion rules that produce a standard format previously chosen.

Correference arises whenever the same real world entity is referred in different ways in a text fragment. This problem may arise due to the use of: *(i)* different names describing the same entity (e.g., the entity "Bill Gates" can be found in the text as "William Gates"), *(ii)* classification expressions (e.g., a few years ago, "Bill Gates" was referred as "the world's richest man"), *(iii)* pronouns (e.g., in the sequence of sentences "Bill Gates is the world's richest man. He was a founder of Microsoft", the pronoun "He" refers to "Bill Gates").

Rule-based approaches for correference usually take into account semantic information about entities. With this information the detection of correferent entities is done through filtering. This means that only entities whose semantics information coincides can be correferent. The filtering can be done manually or using independent resources (e.g. Wordnet[1]) in order to find semantic information associated to each word. At the end of the filtering phase, it is necessary to determine which entities have the highest probability of being correferent.

[1] http://wordnet.princeton.edu/

A machine leaning approach for correference resolution is described in (Cardie & Wagstaff, 1999). This approach is based on clustering algorithms for grouping similar entities. Initially, the entities of the document are analyzed from the ending to the beginning of the document and the distance between each one of them is computed. The distance is computed using an incompatibility function and a set of weighting constants through the Equation (8).

$$dist(E_i, E_j) = \sum_{f \epsilon F} w_f incompatibility_f(E_i, E_j) \tag{8}$$

The incompatibility function uses attributes like name, position, number, semantic class and gender to determine the difference between two entities. If the distance between two entities, E_i and E_j, is less than the cluster radius which is pre-defined, then the entities belong to the same cluster. Consider the following example: “Bill Gates was one of Microsoft’s founders. He is not Microsoft’s CEO anymore”. With this approach, the distance between “Bill Gates” and “He” is expected to be small. Thus, the result must have at least one cluster: $C_1 = \{BillGates, He\}$. If two entities are contained in the same cluster of the final result, they are considered correferent.

Another machine learning approach for correference resolution is based on decision trees. Analogously to the clustering approach, the construction of the tree (training phase) makes use of a set of attributes such as name, position, number. After the training, the text is processed from left to right and each entity is compared with all the previous entities. For each pair, the tree is used to check if its elements are correferent.

Consider as an example, the decision tree of Figure 7.

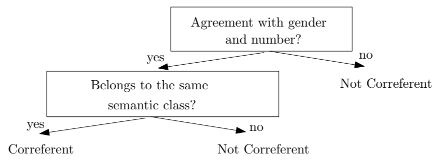
Agreement with gender and number?
yes
no
Belongs to the same semantic class?
Not Correferent
yes
no
Correferent
Not Correferent

Figure 7. Simplified example of a decision tree for correference resolution

Consider the sentence used in the last example. Let us use the decision tree to check if the pair of elements (Bill Gates, He) are correferent. As the two elements are singular and male, in the first level of the tree, we choose the option “yes”. Since “Bill Gates” and “He” belong to the class “Person” (could have been concluded in the classification task), we choose again the option “yes” and we get to a leaf node, concluding that the elements of the pair are correferent.

6 Conclusions

In this paper, we presented the decomposition of an *Information Extraction* activity into tasks and referred the techniques that are commonly used for each task. The goal of this paper is to supply material that can be used for the conceptualization of an *Information Extraction* program.

In future work, we will present a set of declarative operators with the semantics of the tasks described in this paper. These operators will offer the possibility to choose between different techniques for each task. It will also be possible to reorder and compose them in order to adapt the *Information Extraction* activity to the application requirements and input data.

References

Allen, J. (1994). *Natural language understanding (2nd edition).* Addison Wesley.

Applet, D., & Israel, D. (1999). Introduction to information extraction technology. In *Procedings of the* 16th *International Joint Conference on Artificial Intelligence.* Stockholm, Sweden.

Cardie, C., & Wagstaff, K. (1999). Noun phrase coreference as clustering. In *Proceedings of the Joint Sigdat Conference on empirical methods in natural language processing and very large corpora* (pp. 82–89). New Brunswick, NJ, USA.

Cowie, J., & Lehnert, W. (1996). Information extraction. In *Special natural language processing issue of the communications of the ACM* (Vol. 39, pp. 80–91). New York, NY, USA.

Farmakiotou, D., Karkaletsis, V., Koutsias, J., Sigletos, G., Spyropoulos, C. D., & Stamatopoulos, P. (2000). Rule-based named entity recognition for greek financial texts. In *Proceedings of the Workshop on Computational Lexicography and Multimedia Dictionaries (COMLEX 2000)* (pp. 75–78). Pyrgos, Greece.

Forney, G. D. (1973). The Viterbi algorithm. In *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE)* (Vol. 61, pp. 268–278).

Grishman, R. (1997). Information extraction: techniques and challenges. In *Information Extraction International Summer School SCIE-97* (pp. 10–27). Frascati, Italy.

Isozaki, H., & Kazawa, H. (2002). Efficient support vector classifiers for named entity recognition. In *Proceedings of the* 19th *International Conference on Computational Linguistics (COLING02)* (pp. 390–396). Taipei, Taiwan.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Procedings of the* 18th *International Conference on Machine Learning (ICML 2001)* (pp. 282–289). Williamstown, MA, USA.

Li Haizhou, B. S., & Zhiwei, L. (1998). Chinese sentence tokenization using viterbi decoder. In *Proceedings of the International Symposium on Chinese Spoken Language Processing (ISCSLP 1998).* Singapore.

McCallum, A. (2005). Information extraction: Distilling structured data from unstructured text. In *ACM Queue* (Vol. 3, pp. 48–57). New York, NY, USA.

McCallum, A., Freitag, D., & Pereira, F. (2000). Maximum entropy markov models for information extraction and segmentation. In *Procedings of the* 17^{th} *International Conference on Machine Learning (ICML 2000)* (pp. 591–598). Stanford, CA, USA.

Miller, S., Crystal, M., Fox, H., Ramshaw, L., Schwartz, R., Stone, R., et al. (1998). Algorithms that learn to extract information–BBN: Description of the SIFT system as used for MUC-7. In *Proceedings of the* 7^{th} *Message Understanding Conference (MUC-7)* (pp. 75–89). San Francisco, CA, USA.

Santos, M. (2002). *Extraindo regras de associação a partir de textos.* Mestrado em Informática Aplicada, Pontifícia Universidade Católica do Paraná, Curitiba, Brasil.

Sekine, S., Grishman, R., & Shinnou, H. (1998). A decision tree method for finding and classifying names in japanese texts. In *Proceedings of the* 6^{th} *Workshop on Very Large Corpora (WVLC-98)* (pp. 171–178). Montreal, Canada.

Zelenko, D., Aone, C., & Richardella, A. (2003, February). Kernel methods for relation extraction. In *Journal of machine learning research* (Vol. 3, pp. 1083–1106).