1 Introduction

Nowadays, the development of the Internet has created massive amounts of documents available on-line. This not only gives us the opportunity to access various sources of knowledge, but also raises an issue of how to efficiently exploit this knowledge. In this project, we present a system to assist users in extracting and clarifying terms from free-form documents. We developed Glossary Compiler, a system that automatically analyzes web documents in order to extract terms and their definitions. The purpose of an automatic glossary compiler is to aid in the construction of a list of definitions across a large collection of documents. Contrary to the basic search functionality, where the goal is to find single or multiple occurrences of a term or a set of terms, glossary is used to pinpoint definitions. Definition is a concise description of what an entity is. Therefore, glossary compiler should perform some basic semantic analysis to distinguish simple occurrences of a term from its actual definitions. There are several challenges to consider. First challenge is due to multiple ways to phrase a definition. Second challenge occurs when a single term has multiple definitions and it is necessary to cluster them according to the category each definition belongs to.

Ideally, everyone can benefit from an on-line glossary compiler, whether it's an ordinary user, who wants to define an unknown term or a scholar, working with a large collection of papers, which contain the definitions of terms. The major advantage of an on-line glossary compiler is twofold. On one hand, an on-line definition extraction tool can leverage the diversity of content on the Web and have immediate access to a considerably larger set of definitions. Each definition can view the defined term from a different or sometimes unexpected perspective. On the other hand, manual compilation of conventional paper glossaries takes significant amount of time and a user interested in finding the definition of a new term may simply not find it in paper glossaries. Since new information appears considerably faster on the Web than in paper sources, automatically compiled glossaries provide access to definitions at the same time they appear on the Web. However, while automatic identification and extraction of terms from text document have been widely studied in the linguistic literature, the automatic definition extraction problem is much less studied.

As opposed to previous similar projects, e.g. [3], which essentially were based on bag-of-words treatment of the text and hypertext markup heuristics, in this project we proposed a novel approach to definition extraction by looking at it as a search for subtrees in large set of trees. Our approach has strong foundation in theoretical linguistics.

2 Related Work

Glossary Compiler is an end-to-end system. Given a user query, it retrieves relevant documents, extracts definition-bearing passages from them, and presents the clustered results to the user. It is similar to definition-mining systems in that we extract definitions from a huge amount of text (e.g.[1,2,3,4,5,6,7]), but we also apply clustering techniques to provide context for the definitions extracted. Our definition extraction approach is similar to [1] and [3], where a set of definition patterns was manually constructed and used to identify definition-bearing passages. Search engines such as vivisimo (http://vivisimo.com/) are similar to our project, except that our retrieval objective is on pinpointing definitions from relevant web pages, while search engines focus on retrieving relevant web pages.

3 Solution Framework

Our system can be divided into 4 major components: query processing, definition extraction, definition clustering, and user interface. When a user first submits a term to be defined, we retrieve a set of relevant documents and process them into the format appropriate for definition extraction. Once definitions are extracted, we cluster similar definitions together and display the clusters to the users. The following sections describe the components of the system.
**Query Processing**

The major task of query processing is to automatically acquire a set of documents relevant to a query term. However, to facilitate the processing in later stages it needs to perform sub-tasks including format conversion, reorganization, and garbage cleaning, preferably in an unsupervised manner. Figure 1 shows the diagram of the query processing component.

**Figure 1. The Query Processing Component.**

In the first step, the given query term is submitted to the Yahoo API which returns a list of URLs for the query. Then the document, corresponding to each URL is fetched and converted to plain text format. The system works with two major formats of documents on the Web: HTML and PDF. The relative fraction of other text document formats (.doc or .pot) is relatively small and we left the support of those formats as an open space for later extension.

After the conversion to the text format, the document becomes more like a garbage bag containing a large amount of separate characters, short phrases, extremely long strings without actual words, and non-character symbols. The really meaningful sentences are scattered around. Finding those sentences is a non-trivial job. Currently, we use the sentence segmentation tool developed by Prof. Dan Roth’s group which performs reasonably well. The tool reads plain text, detects sentence boundaries and rewrites the document one sentence per line.

In the final step, the document is further cleaned to remove special tokens (e.g., &nbsp; and 0x12a6) and sentences which are merely a combination of words but do not constitute any meaning. Usually, those sentences are pretty long so we may easily locate them by checking the length. Query processing takes quite amount of time to execute, mostly due to the downloading of documents. In order to decrease the processing time we use multithreading. Each thread takes one URL and processes the document associated with it.

**Definition Extraction**

The major difficulty in identification of definitions or any other semantic patterns in free text is the variability of natural language. Every natural language sentence can be phrased in multiple ways, even if the information conveyed by the sentence is exactly the same in all variants. Similarly, there is no single way to define a term. For example, the term “data mining” can be defined in the following syntactically different sentences:

1. Data Mining can be defined as "The nontrivial extraction of implicit, previously unknown, and potentially useful information from data" and "The science of extracting useful information from large data sets or databases".
2. Data Mining, if you haven't heard of it before, is the automated extraction of hidden predictive information from databases.
3. Data Mining is, in some ways, an extension of statistics, with a few artificial intelligence and machine learning twists thrown in.

As can be seen from the above sentences, due to unbounded number of different ways to express definitions in natural language, straightforward approaches, such as regular expressions, words or phrases matching, do not work in the case of definitions extraction and, thus, deeper analysis is required. Therefore, we
need to consider the structure of sentences not only on the lexical level (the level of single words) but also on syntactic and semantic levels.

From the linguistic theory of dependency grammars we know that the structure of a sentence can be represented by a set of dependency relationships that form a tree. A dependency relationship is an asymmetric binary relationship between a word called the “head”, and another word called the “modifier”. Each word in the sentence may have several modifiers, but can modify at most one word. For example, the figure below shows the dependency tree for the sentence “Data Mining, also known as knowledge discovery in databases, is the process of automatically searching large volumes of data for patterns”. Each edge is labeled by the type of dependency relation. For example, the label “s” means that the modifier in this relation is the subject of a sentence. The direction of edges in the dependency tree is from the head to the modifier. The root of the tree is a word that is not modified by any others, which is usually the main verb in the sentence.

![Figure 2. Example dependency tree, produced by Minipar.](image)

Observe from the above example of dependency tree that the lower level nodes in the same branch of the tree refine the meaning of the higher level nodes. In order to covert the free text to dependency tree representation we used Minipar [9], a broad coverage dependency parser. Given a sentence, Minipar returns its dependency tree, in which the edges represent the types of relations and nodes are the words in the sentence with part-of-speech tags assigned to them. The evaluations of the parser have shown that its accuracy is about 89%.

Therefore, using dependencies and part-of-speech information we can create more general and accurate definition patterns. By analyzing a large number of definitions in real free text, we identified the following patterns that reveal the presence of a definition in a sentence:

1. A sentence contains a predicate, which is a noun and defines a subject. For example, in a sentence “data mining is identification of hidden patterns in large volumes of data”, “data mining” is a subject and “identification of hidden patterns in large volumes of data” is a predicate, the main word of which is a noun, so it’s a predicate noun phrase.
2. A sentence contains a phrase “known as”, which modifies the subject of a sentence. These sentences correspond to the definitions similar to “data mining, also known as …”.
3. A sentence contains a phrase “defined as”, which modifies the subject of the sentence. These sentences correspond to the definitions similar to “data mining, also defined as …”.
4. The subject of a sentence is modified by an appositive, which is a noun phrase that follows another noun phrase and renames or describes it. For example, “data mining, a relatively new branch of computer science, emerged in …”.

The above patterns can be represented as parameterized dependency trees and the problem of definition extraction now becomes the problem of identification of subtrees in a large collection of trees, representing the parsed free text. For example, pattern 4 from above can be represented as the following parameterized dependency tree:
The parameterized trees for all patterns are encoded in XML format in the configuration file of the definition extraction module:

```xml
<pattern id="1">
  <node id="0" parent="" word="is" role="i" index="0" />
  <node id="1" parent="0" word="" role="s" index="1" />
  <node id="2" parent="0" word="" role="pred" index="0" />
  <node id="3" parent="2" word="" role="det" index="0" />
</pattern>
```

In the sentence, whose parse tree contains a subtree corresponding to the definition pattern, the node having non-zero attribute “index” is indexed as a term being defined and the sentence is stored as a definition.

The dependency-tree based approach to definition extraction has the following advantages:

- Dependency tree based definition patterns are sufficiently general to overcome the problem of natural language variability;
- It allows to identify definitions of arbitrary complexity;
- In addition to single terms, it allows to identify definitions of phrases such as “comparative genome analysis”;
- The set of definition patterns is extensible as new patterns can be easily encoded in XML and added to configuration file.

### Definition Clustering

The output from Definition Extraction is a set of unique terms with respect to the query and definitions associated with these terms. The goal of definition clustering is to provide a context for these definitions in an online environment. We cluster definitions such that similar definitions are grouped together and different definitions are not. We treat this as a problem of document clustering, where each definition is a document.

There are several issues to consider when choosing a clustering algorithm. A key component in clustering is the features of data to cluster. Since we are clustering definitions and definitions can be noisy, we must look for salient terms in the definitions. Thus, stop words are removed for clustering. We do not apply stemming because we want to retain word discrimination. Another key issue to consider is the similarity metric. The vector-space model with normalized cosine [8] is a reasonable metric for similarity. We treat each definition as a document vector.

A similarity-based clustering algorithm can be hierarchical or search-based (e.g., K-Means). K-Means is more suitable for our project. For hierarchical clustering, we need to determine a stopping criterion apriori. After much experimentation, we noticed that the criterion varies, even given the same query. Since we are clustering in an online environment, we cannot train one criterion for every possible query.

K-Means is an iterative process that begins with random cluster centroids and adjusts them until no more centroid adjustments are needed. The main problem with K-Means is the initial conditions. The clustering performance is sensitive to the number of clusters k and the initial centroids. To get around the problem of initial centroids, we select top k most dissimilar definitions as initial centroids, using the vector-space model as the similarity metric.

It is difficult to determine k, since different concepts may have a different k. We make k adaptive in the following sense. Given the way the definitions are extracted, we can set k to be the number of unique terms
extracted. This $k$ is the upper bound on the number of clusters actually needed. We run K-Means using this $k$. If the clusters are too small (resulting many clusters with only one definition), we decrease $k$ and run K-Means on the new $k$.

This approach relies heavily on the assumption that each unique term provides distinct contextual background for the definitions. However, this assumption is very narrow. As a consequence, our clustering algorithm is not robust enough. We can use Latent Semantic Indexing to identify salient phrases or concepts for each unique term, collect all unique concepts identified, and construct $k$ from these concepts.

4 Results

In this part we present the user interface and some interesting results, obtained by running the system online. We start with the user interface.

The interface of the system is basically an IE toolbar plug-in shown in Figure 2. A user types the term or phrase, the definition of which she wants to find, in the toolbar textbox and the results will be shown in the content page. The left pane shows the terms and their corresponding definitions. Users can view the Web page, from which the definition was extracted, in the right pane by clicking on the definition.

![System interface](image)

Figure 4. System interface.

Since the system runs on-line, we don’t have access to the actual corpus, in which we can manually find the terms and definitions for evaluation. Ideally, the system should be run in the client-server framework as a service on the search engine side. We can only run it in the single computer mode and, thus, because we need to download and preprocess each page on the fly, the running time is slowed down. Next, we will show some interesting observations from the results we obtained.

The first interesting result we observed is that the system can handle both generic and specific queries. For example, a user can input the query “what is” and the system can extract various definitions, as shown in Figure 1. By posing both generic and specific queries to the system, we observed that the accuracy of the system is independent of the type of query. It only depends on the grammar complexity of the content sentences. For example, in the returned result “Pilot’s first data mining product, the Pilot Discovery Server, was released in August 1996 as part of the Pilot Decision Support Suite line of products”, the system identified “data mining product” as the term being defined, even though it is actually a definition.

The second result is that the clustering algorithm accurately classifies definitions into different clusters, each cluster containing definitions in the same category. For example, with query “data mining”, we can find 6 different definitions, which are clustered into several different categories. For example, the definitions of “data mining” that are related to massive data in the market are grouped into the same cluster:

- “Data mining is a powerful new technology with greater potential to help DB preemptively define the information market of tomorrow DB companies already know how to collect and refine massive quantities of data to deliver relevant and actionable business information”
• “Data Mining is an analytic process designed to explore data usually large amounts of data - typically business or market related in search of consistent patterns and/or systematic relationships between variables, and then to validate the findings by applying the detected patterns to new subsets of data.”

The definitions of “data mining” and “bagging” are grouped into the same cluster because they both are related to the machine learning, and the word “bagging” is used in the definition of the core of predictive data mining:
• “Data mining is, in some ways, an extension of statistics, with a few artificial intelligence and machine learning twists thrown in”;
• “Bagging: These techniques - which are often considered the core of predictive data mining - include: Bagging Voting, Averaging, Boosting, Stacking Stacked Generalizations, and Meta – Learning”

From the results shown above, we can see some of the novel observations about the accuracy and efficiency of the system, as well as its usefulness. In the future, we can try to have a full evaluation of the system, but this task requires separate research into possible evaluation criteria and methodology.

5 Conclusion

In this project, we have implemented Glossary Compiler, an on-line system to extract and cluster definitions from web documents. We developed a new approach to definitions extraction, which is based on syntactic analysis of sentences and has strong foundations in theoretical linguistics. Since having access to definitions is a critical step towards accessing the knowledge in a particular domain, we believe that our system could become a very useful tool and enhance the functionality of existing search engines. Our preliminary experimental results verify the merit and usefulness of the system including the friendly user interface, the ability to work with documents of various formats, and the accuracy of definition extraction and clustering. For future work, we consider to evaluate the system under larger testing set and more objective analysis, to extend the set of document formats, and to improve the robustness of the clustering component. The work in the team has been distributed in the following way:
• Alexander Kotov: overall idea of the project, definitions extraction component
• Hoa Nguyen: system integration, user interface
• Zhenyu Yang: query processing
• Hengzhi Zhong: definition clustering

6 References


