PicAChoo: A Tool for Customizable Feature Extraction
Utilizing Characteristics of Textual Data

Jaeseok Myung  Jung-Yeon Yang  Sang-goo Lee
School of Computer Science and Engineering, Seoul National University
Seoul 151-742, Republic of Korea
{jsmyung, jyyang, sglee}@europa.snu.ac.kr

ABSTRACT
Although documents have hundreds of thousands of unique words, only a small number of words are significantly useful for intelligent services. For this reason, feature extraction has become an important issue to be addressed in various fields, such as information retrieval, text mining, pattern recognition, etc. Numerous supporting tools for feature extraction are available, but most of them deal with text as a simple literal. Unfortunately, text is not just a literal, but a semantically significant unit including linguistic characteristics. So, we need customized extraction methods that consider the characteristics of source documents. PicAChoo stands for ‘Pick And Choose’, and it provides an environment which enables feature extraction methods using the structure of sentences and the part-of-speech information of words. Moreover, we suggest dynamic composition of different extraction methods without hard-coding.

Categories and Subject Descriptors
H.2.8 [Database Management]: Database Applications – data mining; H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing – linguistic processing; I.5.4 [Pattern Recognition]: Applications – text processing

General Terms
Design, Experimentation, Management

Keywords
Customizable feature extraction, Feature storing model, Complex feature

1. INTRODUCTION
As the number of documents on the World Wide Web is increasing dramatically everyday, many researchers are trying to analyze those documents to offer an intelligent service. However, acquiring knowledge from huge size of unstructured documents is not that easy. Even knowing which part of the document is important is difficult, because the document consists of so many individual terms.

In text analysis, feature extraction is one of the most significant tasks, as it tells which terms are more meaningful than the others. As a result, researchers have invented a variety of feature extraction methods and many tools supporting those methods have been developed by the researchers. Nevertheless, most of the tools still consider words just as a meaningless sequence of characters. In other words, many extraction methods only consider whether the term appears in the document or not. So, we need some hard-codings or manual efforts when we want to exploit the characteristics of textual data, such as a structure of a sentence and a part-of-speech information of individual words. Also, implementing our own methods from scratch can be stressful; and therefore, we need a feature extraction tool considering the characteristics of languages.

In terms of performance, the results of feature extraction methods can be measured in different ways: by the purpose of the applications or by the domain of the applications. The reason is that each application has different requirements and source documents, and that’s why we use some heuristics in our extraction methods to improve the performance. So we need a feature extraction tools considering refined dynamic operations for enabling customized feature extraction.

In this paper, we suggest a tool for customizable feature extraction named PicAChoo. As indicated in its name, it provides many functions in order to extract appropriate features. It also tries to give users an environment that takes advantage of characteristics of textual data, and it allows dynamic composition of feature sets to design a customized extraction method.

Rest of this paper is structured as follows. In section 2, we introduce some existing text analysis tools, and we explain our objectives more clearly in section 3. And in section 4, we suggest and describe some key concepts and details for our objectives. Section 5 discusses about the usability of PicAChoo with some case studies, and finally, we present a conclusion and suggested future works in section 6.

2. RELATED WORK
In terms of research area, feature selection and extraction are theoretically based on dimensionality reduction. Many statistical methods have been developed so far in that discipline. Especially, there are several tools available for free on the Internet for the text analysis. In this section, we introduce some popular open-source solutions for text feature extraction.

RapidMiner[1](formerly YALE) is an open-source data mining solution that is widely used by researchers. The modular operator concept of RapidMiner allows the design of complex nested operator chains for a number of learning problems. It’s
implemented in Java language, and about 400 operators are available with convenient graphical user interface.

GATE[2] stands for a General Architecture for Text Engineering. It includes a lot of algorithms for NLP(Natural Language Processing), and tries to divide overall processes into database schema, user interface, and algorithms. The separation of system architecture has brought reusability to the individual algorithms. It is also implemented in Java language with graphical user interface. Weka[3] is a Java software package including a collection of machine learning algorithms for data mining tasks. Weka contains tools for data preprocessing, classification, regression, clustering, visualization, and feature selection. It can be used in different ways such as application programming interface, command line, and graphical user interface.

Although there have been many commercial tools that provide a variety of functionality, their main functions are concentrated on fully automatic unsupervised learning methods rather than semi-automatic supervised methods. In other words, there is a limitation for designing a new type of methods with existing tools. In addition, there’s lack of consideration for complex feature[4] which consists of several terms so far.

3. DESIGN GOALS
In this section, we discuss three kinds of issues that we considered at the design time of PicAChoo.

3.1 Supporting Dynamic Composition of Feature Extraction Methods
Basically, feature extraction is a source-dependent and purpose-dependent research area. So, we have to consider the characteristics of source data, and we also have to think about the application domain of extracted features. In other words, we need appropriate methods depending on our purposes and data characteristics. So far, there have been several feature extraction methods for text analysis[5,6,7]. And some tools are already developed to support those methods.

Suppose that you want to build a subset of features by using a fully automatic extraction method, such as information gain; it might be good idea, because most of the existing tools can make you have the feature set. However, what if you want some changes of the information gain method? What if you want to make your brand-new method? What if you want to compose those methods? In these situations, you need to make your own tool from scratch.

We concentrated on providing an environment in which you can apply several feature extraction methods without hard-coding. To achieve this objective, we prepared four types of primitive methods to make a feature set, and we suggested two types of composite methods to compose different feature sets.

3.2 Supporting Separation of Preprocessing and Feature Extraction Methods
In text analysis, we have taken advantage of many NLP techniques to obtain linguistic features from raw texts. For instance, we have used POS(part-of-speech) tagger, stemmer, parser, and so on. In many cases, we have a preprocessing phase to apply those techniques because they give us more information that can be used in extraction methods.

While doing preprocessing, we need to consider the separation of NLP techniques and feature extraction methods. Sometimes we need to replace the NLP software because of a new version of the software, or when we need to change our target language from English to Korean. In those cases, if we don’t think about the separation, we have to redo the entire analysis processes for the new types of source documents if we cannot think about the separation. The separation will help us to analyze source documents in a flexible way.

In addition, the separation is also helpful for dynamic composition of feature extraction methods. The extraction method doesn’t have to worry about saving and loading features, and the preprocessing tool also doesn’t have to know about the extraction method. That makes flexible architecture so that we can mash our methods up.

We suggest a feature storing model which supports separation of preprocessing and feature extraction methods. We believe it is useful because we can replace preprocessing tools according to our purpose and target language. Moreover, the storing model enables dynamic composition of feature extraction methods.

3.3 Supporting Complex Features for Text Analysis
As we already mentioned, we need to consider what we call a complex feature, which is the characteristics of textual data as a linguistic resource. Generally, a word cannot be meaningful enough without any other information. Intuitively, the complex feature contains not only conventional features but also the contexts of the features. The context could be any type of information that describes the situations of the feature. So, there can be a lot of complex features which include the contexts of the feature. There were some researches which tried to utilize a complex feature[8, 9]. However, in terms of supporting tools, we have only a few options to take advantage of a complex feature to our intelligent services.

We want to provide you with a feature set, which consists of a pair of words extracted by co-occurrence, when you want to deal with complex features. The relationship between words could be defined by using POS tags so that you can put your intension into the definition. For example, you can gather a complex feature such as a pair of (adjective, noun) words which are adjacent to each other, and then you can exploit the feature for your purpose.

4. KEY CONCEPTS IN PICACHOO
In this section, we will discuss some of the key concepts of PicAChoo, which enable us to achieve above objectives. First of all, we explain the processing step to discuss how we approached to the goals.

4.1 Processing Steps
In an architectural point of view, PicAChoo has a workspace which includes several data sources. Each data source contains a huge number of documents to be analyzed, and each document is distinguished from one another by unique identifiers. A document is divided into terms through preprocessing. A term contains additional information such as POS tags and position of the term. All of the information is stored into a database to be exploited by the extraction methods. In addition, we extract some candidate features by using POS tags during the preprocessing step. In the
preprocessing step, we use a POS tagger[10] and stemmer[11]. We designed a database schema to store such information. We discuss about the storing model in section 4.2.

The next step is primitive extraction. A method for primitive extraction receives a set of candidate features as an argument, and it returns a set of scored features. Basically, there are three kinds of primitive extraction methods: namely, frequency based, co-occurrence based, and pattern based. In addition, there is a plug-in method which supports external implementations.

Primitive extraction could be followed by composite extraction. In composite extraction step, we receive one or more feature sets that are already scored and, we can apply logical and arithmetical operators to return a unified feature set. The details of primitive and composite extractions are described in section 4.3.

Finally, depending on the purpose, we can extract a context word that describes the feature. Each feature can have several context words, and every pair of features and context words constructs a complex feature respectively. Figure 1 shows the overall processing steps of PicAChoo.

Figure 1. Processing Steps of PicAChoo

4.2 The Feature Storing Model

Raw texts could be analyzed by various NLP tools, and the analysis is usually performed during preprocessing step. We perform POS tagging and stemming at that period. At the same time, we identify candidate features by using POS tags and length options defined by the user. For instance, suppose the user select options that POS tags of candidate feature is NN(noun), and the maximum length of candidate feature is 2. We can obtain all sequences of forms ‘NN’ and ‘NN NN’. All candidate features are stored into the feature storing model.

The feature storing model consists of four entities: Source, Feature, Usage, and Words. Source documents and candidate features are stored into Source and Feature entities respectively, and Words entity contains individual term data, and Usage entity indicates positions of candidate features. Figure 2 represents the feature storing model.

The feature storing model enables many kinds of analysis queries. It is a simple, very general and flexible model to design various types of feature extraction methods with. Some example methods will be presented in section 4.3.

Figure 2. The Feature Storing Model

Since the storing model is fixed, we can make any type of extraction methods by using the model. Moreover, the separation between preprocessing and extraction methods is achieved because of this model. Even if we want to change NLP tools or application domain, we do not have to change the existing extraction methods because the storing model will not be changed.

4.3 The Composition of Feature Extraction Methods

In most cases, the measurement of feature extraction is not clear because it depends on the purpose of the applications. So, it is more useful to design a customized method through modification and composition, unless you want a fully automatic unsupervised learning for your specific purpose. We suggest primitive methods and composite methods to develop more complex methods. Overall methods and options are represented in table 1.

Table 1. Feature Extraction Methods in PicAChoo

<table>
<thead>
<tr>
<th>Category</th>
<th>Method</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitive</td>
<td>Frequency-based</td>
<td>Frequently used(TF)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Widely used(DF)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Threshold</td>
</tr>
<tr>
<td></td>
<td>Co-occurrence</td>
<td>Fixed-size window (left, right, both)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sentence</td>
</tr>
<tr>
<td>Pattern-based</td>
<td>Sequential patterns (pos</td>
<td>literal)</td>
</tr>
<tr>
<td>Plug-in</td>
<td>Logical</td>
<td>And, Or</td>
</tr>
<tr>
<td>Composite</td>
<td>Arithmetical</td>
<td>+, -, *, /, ^, %</td>
</tr>
</tbody>
</table>

4.3.1 Primitive Methods

Primitive methods have two responsibilities. The first one is a selection of candidate features to build a subset of features. The second role is to decide the importance of the selected features. In other words, a specific algorithm is not important if it can make a feature set, including features and corresponding scores. We already developed some primitive built-in functions as you can see in Table 1.

The first type of primitive methods is frequency based methods. It concerns how many times the term appears(term frequency), and how many documents contain the term(document frequency). The
threshold value is provided to dynamically select features according to their appearance scores.

The second type of primitive methods is co-occurrence based methods. It concerns whether there is an appropriate term within a given range or not. The range can be a sentence or a fixed size window around a candidate feature. The occurrence condition can be described by using POS tags or literal. Every option is dynamic so that we can make results at runtime.

The third one is pattern based methods. Some features obeying the pattern rules are selected to build a subset of features. Users can define a sequential pattern to build a feature set. The pattern rule consists of a candidate feature, POS tags and literals. For example, ‘<DT> <feature> of’ means the POS tag of the former word is ’DT(a determiner)’, and the literal of latter word is ‘of’. Figure 3 shows formal definition of pattern rules.

![Figure 3. A Definition of Sequential Pattern](image)

Finally, the last type of primitive methods is plug-in methods. As we know, the plug-in architecture is one of the most popular design principles for software development. It enables us to make a new type of extraction methods by following some simple guidelines. Basically, the input and output of plug-in methods are a set of scored features. However, we also provide a method having a datasource object as an argument that contains a connection to the feature storing model. It can be used for generating and executing a new SQL query.

### 4.3.2 Composite Methods

Applying primitive methods makes a set of features including candidate features and corresponding scores. Every primitive method has a same output format, so we can compose two or more feature sets by using the id and score values of candidate features. Besides, the result of composition also has a same output format which is made up of features and scores, so we can apply composite methods repeatedly.

There are two types of composite methods: namely logical composite methods, and arithmetical composite methods. Logical composite methods use the id value, which distinguishes features from each other. It means even if a candidate feature is contained in several feature sets, the id value of the candidate feature is same. So we can use some logical operators such as ‘and’ or ‘or’. We will filter some candidate features by using ‘and’ operator, and merge feature sets by using ‘or’ operator.

Arithmetical composite methods use the score value. The value can be a real number, and we may be familiar with calculating some real numbers. In this type of methods, we can use six basic operators (+, -, *, /, ^, %) to merge scores. However, it is not enough to make all possible expressions, such as sigma, log, and so on. You may need to implement some plug-in method if you want some kinds of complex operators. But, those operators will be supported soon.

### Table 2. Converting from "<DT> <feature> 'of'" to SQL Query

```
select fid, f_stem, count(*) from (
    // a sub-part for the <DT> tag
    select fid, f_stem, sid, sno from (
        select fu.fid, fu.stem as f_stem, fu.len, fu.sid, fu.sno, fu.wno, wd.stem as w_stem, wd.tag from (
            select f.fid, f.stem, f.len, u.sid, u.sno, u.wno
            from [FEATURES] f, [USAGES] u
            where f.fid=u.fid
        ) fu, [WORDS] wd
        where (fu.sid=wd.sid and fu.sno=wd.sno and ((fu.wno-1) = wd.wno)) // position of the <DT> tag
    )
    intersect
    (select fid, f_stem, sid, sno from (
        select fu.fid, fu.stem as f_stem, fu.len, fu.sid, fu.sno, fu.wno, wd.stem as w_stem, wd.tag from (
            select f.fid, f.stem, f.len, u.sid, u.sno, u.wno
            from [FEATURES] f, [USAGES] u
            where f.fid=u.fid
        ) fu, [WORDS] wd
        where (fu.sid=wd.sid and fu.sno=wd.sno and ((fu.wno+(fu.len)+0) = wd.wno)) // position of the <DT> tag
    )
) group by fid, f_stem, sid, sno
```

- 703-
4.4 Context Extraction Methods

As discussed previously, enough sense cannot be made from one word; it needs contexts that describe the word. That is the reason why we need to deal with complex features. Unfortunately, defining a complex feature is quite difficult because there are a huge number of relationships between words. So, we introduce a separated step concerning complex features. In PicAChoo, primitive and composite extraction methods are used to extract simple terms, and we consider the complex feature in the context extraction step.

In context extraction step, we consider a relationship between candidate features and other words. The relationship is co-occurrence. It is similar to co-occurrence based extraction, which is one of primitive method types, but the only difference is a form of extracted feature. It means the complex feature contains a pair of feature and context words. In a case of primitive methods, we can only gather feature words without contexts.

When we are on the context extraction step, we need to think about the semantics of context extraction. For example, if we select frequently co-occurred words as context words, the relationship between features and context words can be named to ‘frequently co-occurred’. And, if we select an adjective word in front of candidate feature, the relationship can be named to ‘describe’. In these cases, the meaning of relationships must be different, and it should be treated in different manners. There can be numerous relationships according to your research purpose, so we need to identify the semantics of context extraction, and use it in a right way.

5. CASE STUDIES

Text analysis and feature extraction have various fields to be used, but we want to introduce some scenarios in a practical point of view.

5.1 Applying TF-IDF without Hardcoding

TF-IDF is a fundamental and representative term weighting method and it usually become a very first method when we need to extract features from raw texts. There are two typical methods to apply TF-IDF to our researches. The first one is to find a tool, including TF-IDF, and the second one is to implement it. However, the first way has a customization problem, and the second one has an implementation problem. It is why we need a customizable feature extraction tool.

First of all, preprocessing is required to make a datasource and to analyze source documents. The feature storing model is used for storing analyzed terms. And then, primitive extraction and composite extraction is progressed. Figure 4 shows datasources and a feature set created by frequency based extraction methods. We have two feature sets by term frequency(tf) and document frequency(df). As you can see in the figure, we can change options dynamically, and we can see usages of a feature, and we can remove a feature from a feature set. Moreover, we can export a feature set to Excel or XML file.

The next step is composite extraction which mixes tf and df feature sets. In this process, we can apply ‘and’ operator so that we can find features appeared in both feature sets, and we can calculate (tf * (1/df)) in a runtime. Figure 5 represents the result of composite extraction. Like a previous step, every option can be applied dynamically, and we can manipulate our feature sets.


5.2 Possibility of Complex Features

If we treat a feature as a semantically significant unit, we can find wide fields to use it. The context, which is one of essential elements of complex feature, could be a hint to make an intelligent service. So, it is natural that many text mining techniques are concentrating on the relationships between feature and its contexts. The complex feature represents a profound meaning of the relationship.

Suppose a complex feature (noun, adjective) as a (feature, context). We also can name the complex feature as (subject, predicate). That is one of the relationships that words can have. In our intelligent services, we can use the semantics of complex feature to do more accurate summarization and classification.

For example, we analyzed real product reviews to realize user's response for each attributes of product. We made complex features of which relationships represent (subject, predicate). For instance, we made a complex feature such as (size, big) by using a co-occurrence based context extraction method. In this case, source documents are related with products, and we can categorize some product documents according to features and contexts. It means we can analyze documents at a finer level. It also could be used for summarization and sentiment classification. Figure 6 shows selecting POS tags to extract complex features.

![Figure 6. Context Selection](image)

6. CONCLUSION

The techniques of feature extraction let us know which features are more important than the others. So, we can use those techniques to build a subset of relevant features which would be exploited for classifying, clustering, summarizing, and so on. Text analysis, especially, is a major area of feature extraction, and it needs more sophisticated operations because it is not just a simple literal.

We presented a customizable feature extraction tool named ‘PicACHoo’. Many linguistic features and extraction methods are supported by the tool. To support those objectives we are taking advantage of various key concepts. The feature storing model enables separation between preprocessing and extraction methods. The dynamic composition of feature sets can be used for developing a new type of feature extraction method. Finally, we introduce context extraction which considers the characteristics of textual data. Moreover, plug-in architecture is applied so that we support customized methods.

However, there are still some remained problems. First of all, we have only a few plug-in methods, as we are at the starting point. Second, the number of context extraction methods supported by the tool needs to be increased. We are supporting only co-occurrence based method; however, there are numerous relationships in real documents.

7. REFERENCES


[12] [http://ids.snu.ac.kr/wiki/PicACHoo_%28Pick_And_Choose%29](http://ids.snu.ac.kr/wiki/PicACHoo_%28Pick_And_Choose%29)