PicAChoo: A Text Analysis Tool for Customizable Feature Selection with Dynamic Composition of Primitive Methods

Jaeseok Myung
School of Computer Science and Engineering, Seoul National University
Email: jsmyung@europa.snu.ac.kr

Jung-Yeon Yang and Sang-goo Lee
School of Computer Science and Engineering, Seoul National University
Email: {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 text analysis. Thus, feature selection has become an important issue to be addressed in various text analysis studies. A number of techniques and algorithms for feature selection are available, but unfortunately, it is hard to say that a certain algorithm overcomes the others, because feature selection results mostly depend on the source documents. We should pick and choose the appropriate algorithm and the best subset of feature words whenever we need to analyze source documents. In this paper, we present a framework named 'PicAChoo', which stands for 'Pick And Choose' that enables customizable feature selection environments by composing several primitive feature selection methods without hard-coding. As indicated in the name, this framework provides many strategies for extracting appropriate features and allows dynamic compositions among several feature selection methods. In addition, it tries to give users an environment that utilizes linguistic characteristics of textual data, namely part-of-speech, sentence structures, and so on. Finally, we illustrate that selected feature words can be used for various intelligent services.**

***Index Terms*—text analysis, feature selection, dynamic composition, feature storing model, complex feature**

I. INTRODUCTION

As the number of documents on the World Wide Web is increasing dramatically every day, many researchers are trying to analyze those documents in order to offer an intelligent service. However, acquiring useful knowledge from the huge number of unstructured documents is not that easy. Even knowing which part of the document is important is difficult, because a document consists of so many individual words. Thus, in many cases of text analysis we usually define or extract a few important feature words to be used for our specific applications. For example, some applications utilize feature words for text classification[1][2], and others summarize users' opinions according to product features[3][4]. There have been numerous approaches for selecting useful feature words, and feature selection is the first and most significant task that should be considered.

Manual feature selection is the first option we can consider for identifying important feature words. In this approach, we are able to recognize only a few limited words but do so very accurately by actually reading real documents. However, it is a really time-consuming task, and we might miss some important features due to human error. Furthermore, sometimes we need to do feature selection once and again according to source documents, because the words contained in each document must be different from each other. Consequently, manual feature selection promises reliable feature words, but it is difficult to take care of all documents by hand.

Automatic feature selection[5] aims to reduce manual efforts effectively, and many researchers have intensively studied several aspects of the technique. Especially, a number of statistical approaches based on reasonable heuristics have been introduced so far. Some representative examples are document frequency, information gain, mutual information, and so on. These methods automatically estimate the importance of each feature word so that we can see which features are more important than others. The number of automatic feature selection methods is huge, and those methods actually work nicely. However, it seems that there are still some sophisticated tasks that require valuable manual work by researchers. As different features can be important in different source documents, we may need to choose different feature words for different documents. When going through this process, deciding an appropriate selection algorithm can be a serious problem. In addition, we need to check selected features in order to confirm their quality. In other words, we should 'Pick And Choose' the proper algorithm and a set of feature words by examining numerous candidates that would be able to show different benefits.

Implementing many different selection methods by hard-coding is very difficult and time-consuming. To remedy this problem, we looked for a supporting tool that helps us apply several selection methods in runtime. There have been a number of tools available for general

doi:10.4304/jsw.5.2.179-186

text analysis, but feature selection needs more sophisticated assistance. For example, tf-idf is one of the most popular methods for feature selection. Almost all existing tools support the method naively. However, we need to think of the method as actually a composition of term frequency and inverted document frequency. Traditional tools deal with the tf-idf method as an atomic unit, and they have focused on complete analyzing tasks and high-level mining tasks such as clustering, classification, etc. We believe if we define some primitive methods and provide an environment allowing dynamic composition of primitive methods, we will be able to pick and choose appropriate features in a more sophisticated way.

In this paper, we propose a tool that reduces manual efforts for feature selection, and we focus on a specific feature selection stage enabling customizable feature selection. To do that, we defined and implemented some primitive and composite methods. The rest of this paper is organized as follows. In section 2, we introduce the background and related work. Section 3 stresses our objectives in terms of designing a tool, and section 4 gives a detailed explanation of the implementation. In section 5, we examine cases in which the system was applied, and we draw a conclusion in section 6.

II. Related Work

Theoretically, the origin of feature selection is dimensionality reduction, which is a preprocessing step of data mining. The objective of dimensionality reduction is reducing the number of target source documents as well as keeping the level of achievement of the data mining task. In other words, the result derived from reduced source documents should be similar to the result from original documents. In fact, the preprocessing task can be categorized into feature selection and extraction. The difference between the two is that feature selection identifies important feature words from entire source documents, but feature extraction represents a document as a vector of feature words. In other words, feature selection prepares features to be used for learning a model, and feature extraction actually uses the model in order to classify documents. In this paper, we deal with feature selection, and we believe it helps people to acquire better understanding of their data by telling them which features are important and how they are related to each other.

Generally, feature selection for textual data goes through the following stages:

Preprocessing: First of all, we should tokenize a document into individual words. At the same time, we may be able to attach additional information to the words. Many NLP (natural language processing) techniques can be used for gathering this information. In addition, during this stage, we generate candidate feature words that would be evaluated later. Most researchers have determined nouns or noun phrases as candidate features[6][7].

Scoring: By using statistics of the candidate features, we are able to determine the importance of each candidate feature. A number of measures have already been examined so far[8], but we still have a chance to apply our brand-new heuristics and statistical analysis. For example, we have considered a document as a bag-of-words model that doesn't think of the sequence of words, but there can be another document model from computational linguistics[9].

Filtering: Candidate features get corresponding scores after the scoring stage. Then, we can ignore some ineligible candidates below the threshold. We are allowed to finalize feature words manually.

Since feature selection is a complex task that has to deal with a huge number of data, some supporting tools have already been developed. Especially, there are several tools available for free on the Internet for text analysis. RapidMiner[10] (formerly YALE) is an open-source data mining solution that is widely used by researchers. The modular operator concept of the solution allows the design of complex nested operator chains for a number of learning problems. It was implemented in Java language, and about 400 operators are available with a convenient graphical user interface. GATE[11] stands for a General Architecture for Text Engineering. It includes many 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 individual algorithms. GATE was also implemented in Java language with a graphical user interface. Weka[12] 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.

Unfortunately, many earlier tools tried to support an entire data mining processes, which means there is a limitation to design a new type of method with existing tools. In other words, to the best of our knowledge, the earlier tools only support to apply existing feature selection methods rather than to create a new method. In addition, most previous tools consider textual data as a meaningless literal; namely, it remains to be seen how to apply the characteristics of textual data. For this reason, PicAChoo concentrates on a specific feature selection area, and enables dynamic composition of several selection methods. Moreover, PicAChoo applies both several NLP tools and statistical analysis tools in order to deal with textual data characteristics.

III. Design Goals

As we have mentioned, our system has two major design goals; the first is supporting dynamic composition among primitive methods, and the second is utilizing textual data characteristics. Basically, feature selection is a source-dependent and purpose-dependent task. To obtain the appropriate feature, we need to select proper algorithms according to our purpose and application domain. Consider a situation in which you want to select features by using an existing selection method, such as

TABLE I. PROCESSING STEPS OF PICACHOO SYSTEM

Processing Step	Applied Technologies	Description
Preprocessing	NLP Techniques Stemming, Tagging, Parsing	It takes a huge number of documents as input, and generates candidate features indicating word sequences of nouns
Scoring	Term Weighting Methods with Statistics	With given candidate features, it determines significance for each candidate feature. A (candidate feature, score) pair is called as a scored feature. A scored feature set consists of scored features
Customizing	Composition with Logical and Arithmetical Operators	It takes two or more scored feature sets as input, and compounds features with logical and arithmetical operators. Finally, it yields a composite scored feature set
Filtering	Threshold & Manual Filtering	It reduces scored features by applying threshold values dynamically, and users may finalize selected features with manual filtering

information gain; it is a very simple task, because most of the existing tools can make you have the features. However, what if you want to make some changes to the information gain method? What if you want to create a brand-new method? What if you want to compose those methods? In this situation, you have to make your own tool from scratch. We concentrated on providing an environment in which you can apply several feature selection methods without hard-coding. To achieve this objective, we prepared four types of primitive methods that make a feature set, and we suggested two types of composite methods to compose several feature sets.

One of the most important things we have to consider is that text is not just literal but also a semantically significant unit including linguistic characteristics. In text analysis, we have taken advantage of many NLP techniques to obtain linguistic characteristics from raw texts. For instance, we have used part-of-speech tagger, stemmer, parser, and so on. Nevertheless, in the field of feature selection, most conventional tools still consider words just as a meaningless sequence of characters, and only consider whether the specific word appears in the document or not. In contrast, we attempt to utilize some linguistic characteristics for feature selection. We applied not only traditional NLP techniques but also some statistics such as co-occurrence relationships between feature words and sequential patterns in a sentence. In addition, we also considered context words that describe the context of the selected feature words. Generally, a word cannot be meaningful enough without other information. For example, in product review sentences, ‘size’ can be a good feature word, but we cannot understand what customers tried to say. ‘Big,’ ‘small,’ ‘good,’ ‘bad,’ or some other words will be able to describe the feature word, and make the feature word useful.

It should be noted that we need a special storing model to enable the above design goals. For example, in order to utilize these textual data characteristics, we should exploit a more complex document model than the conventional bag-of-words model[13]. In the bag-of-words model, a text is represented as an unordered collection of words. On the other hand, our model considers additional metadata for individual words, such as part-of-speech, the position in the document, the stemming form, and so on. Furthermore, we need to support many types of statistical feature selection methods. In other words, all of our primitive and composite methods have to be based on a certain feature storing model. For this reason, the storing model must contain additional information to be used for feature selection methods. The statistical information can be redundant, but it reduces the need for processing entire documents every time. Finally, we designed a feature storing model, and it will be discussed later.

IV. PICK AND CHOOSE IN ACTION

In this section, we describe the processing steps of the system. In particular, the implementation of each stage will be illustrated with detailed examples.

A. Processing Steps

The overall processing steps of our system are simply expressed in Table I. There are four processing steps: namely, preprocessing, scoring, customizing, and filtering. Three steps are the same as in the conventional feature selection tasks, but the customizing step is what we have introduced in this paper. In section 2, we have already seen what preprocessing, scoring, and filtering do, and briefly, the customizing step merges several feature sets by using pre-defined composite methods.

As each stage has different input and output formats, many technologies are applied to the formats. For example, the preprocessing step takes a huge number of documents to be analyzed, and makes some candidate features as a result. Since we generate candidate features as sequences of nouns that appear in documents, we need to tokenize documents and have to attach part-of-speech information to each word. In the case of the scoring stage, we use several methods to estimate the importance of each candidate feature. We define a primitive method as a method that can calculate the corresponding score for each candidate feature. A number of traditional selection methods can be adopted as primitive methods by following this definition. Especially, statistics based on the occurrence of each feature can be used for the scoring stage. In terms of the customizing stage, we receive scored features and adjust their scores. Different primitive methods make different feature scores, and we can compose those scores by using logical and

arithmetical operators. To enable the composition, the customizing stage uses the same formats as the input and output. The composition of the scores can be seen as a composition of primitive methods. In the tf-idf example we mentioned, idf can be seen as (1/df). Therefore, we can create a complex feature selection method by going through the customizing stage. Finally, the filtering stage applies the threshold value dynamically in order to hide ineligible features, and we can manually finalize selected features to be used for intelligent user applications.

B. Document Preprocessing

In the preprocessing phase, the system receives a number of documents and extracts some useful data from those documents. Raw text could be analyzed by various NLP techniques, and the analysis is usually performed by the preprocessor. The preprocessor tokenizes documents by using some useful tools[14][15], and simultaneously, it enriches each word with additional information, such as part-of-speech, a stemmed form of the word, document ID, including the word, a sentence number, position of the word in the sentence, and so on. At the same time, the preprocessor also extracts and stores candidate features. We identify candidate features by using part-of-speech information and length options defined by the user. For instance, suppose the user selects an option that the maximum length of candidate feature is 2. We can obtain all sequences of the forms 'noun' and 'noun noun.' In addition, we store statistical information about candidate features as well as tokenized documents. The positions of occurrence for each candidate are preserved for the purpose of primitive methods.

As a result, to store and to utilize all above information, we have designed a feature storing model. It contains tokenized documents, candidate features, and occurrence data. In particular, the physical schema of the feature storing model is introduced in the appendix. Although we briefly discuss the feature storing model, the model plays the most important role in the system. The feature storing model enables many kinds of analysis queries in runtime. This is a simple, but very general and flexible model for designing various types of feature selection methods with. Moreover, because the storing model is fixed, we can make any type of selection method based on the model. Finally, a separation between the preprocessing and selection methods is achieved because of the fixed model. Even if we want to change NLP tools, we do not have to change the existing selection methods because the storing model will not be changed.

C. Scoring Candidate Features

The scoring step follows the preprocessing step. A primitive method receives candidate features as an argument, and returns scored features. Basically, we suggested three kinds of primitive methods: namely, frequency based, co-occurrence based, and pattern based. In addition, there is a plug-in method that supports external implementation. The overall methods and options are represented in Table II.

Primitive methods have two responsibilities. The first one is a selection of candidate features to build a subset

TABLE II. FEATURE SELECTION METHODS IN PICACHOO

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

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 II. The first type of primitive methods is a frequency-based method. It concerns how many times the word appears (term frequency) and how many documents contain the word (document frequency). The threshold value is given for selecting features according to their appearance scores. The second type of primitive method is a co-occurrence-based method. It concerns whether there is an appropriate word 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 part-of-speech tags or string literals. Every option is dynamic so that we can create results at runtime. The third one is a pattern-based method. Some features obeying the pattern rules are selected to build a subset of features. Users can define a sequential pattern in order to build a feature set. The pattern rule consists of a candidate feature, part-of-speech tags, and string literals. For example, '<DT> <feature> of' means the part-of-speech tag of the former word is 'DT(a determiner),' and the literal of the latter word is 'of.' Fig. 1 shows the pattern rules definitions. The last type of primitive method is the plug-in method. 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 selection method by following some simple guidelines. Basically, the input and output of the plug-in methods are a set of scored features. However, we also provide a method having a connection to the feature storing model.

It can be used for generating and executing a new SQL query. A considerable point of primitive methods is that PicAChoo dynamically generates an SQL query according to the user options. It is a good point we can

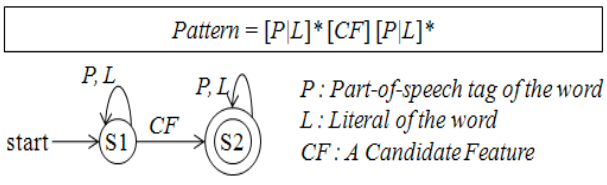

Figure 1. A Graphical Representation for Sequential Patterns

TABLE III. CONVERTING FROM "<DT> <FEATURE> 'OF'" TO SQL QUERY

```
select feature_id, f_stem, count(*) from (
    (   // a sub-part for the <DT> tag
        select feature_id, f_stem, document_id, sentence_no from (
            select fo.feature_id, fo.stemmed_form as f_stem, fo.len, fo.document_id, fo.sentence_no, fo.word_no, td.stemmed_word as w_stem,
td.part_of_speech from (
                select f.feature_id, f.stemmed_form, f.len, o.document_id, o.sentence_no, o.word_no
                from [CF] f, [OC] o
                where f.feature_id=o.feature_id
            ) fo, [TD] td       // position of the <DT> tag
            where (fo.document_id=td.document_id and fo.sentence_no=td.sentence_no) and ((fo.word_no-1) = td.word_no)
        ) where pard_of_speech=q'#DT#' group by feature_id, f_stem, document_id, sentence_no
    )
    intersect
    (
        select feature_id, f_stem, document_id, sentence_no from (
            select fo.feature_id, fo.stemmed_form as f_stem, fo.len, fo.document_id, fo.sentence_no, fo.word_no, td.stemmed_word as w_stem,
td.part_of_speech from (
                select f.feature_id, f.stemmed_form, f.len, o.document_id, o.sentence_no, o.word_no
                from [CF] f, [OC] o
                where f.feature_id=o.feature_id
            ) fo, [TD] td       // position of the <DT> tag
            where (fo.document_id=td.document_id and fo.sentence_no=td.sentence_no) and ((fo.word_no+(fo.len)+0) = td.word_no)
        ) where w_stem=q'#of#' group by feature_id, f_stem, document_id, sentence_no
    )
) group by fid, f_stem order by count(*) desc
```

get by fixing the feature storing model. Table III illustrates an example of the translation from a pattern-based method to the corresponding SQL query based on the feature storing model, which is introduced in the appendix. [CF] indicates a candidate feature table of the feature storing model, and [OC] refers an occurrence table, and so on. The translation of the other types can be achieved in similar ways.

D. Customizing Scored Feature by Dynamic Composition

Applying primitive methods makes a set of scored features including candidate features and corresponding scores. Every primitive method has the same output format, and the composition result also has the same output format, which is made up of features and scores, so we can apply composite methods repeatedly.

As we can see in Table II, 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. This means that even if a candidate feature is contained in several feature sets, the ID value of the candidate feature is the same. We use the value in order to apply some logical operators such as ‘and’ or ‘or.’ For example, we find some intersect features by using the ‘and’ operator and merge different feature sets with the ‘or’ operator. After that, arithmetical composite methods are applied. The second method uses real numbers, and we can use six basic operators (+, -, *, /, ^, %) to merge scores.

In procedural point of view, logical and arithmetical operators cannot be separated. A logical operator determines whether a candidate feature will be contained into a new feature set, and an arithmetical operator determines a corresponding score for the candidate feature. Hence, a composite method can be evaluated by both logical and arithmetical operators. The logical evaluation is followed by the arithmetical evaluation.

Unfortunately, it is not enough to make all possible expressions. For example, many feature selection methods use sigma or logarithms that are not supported by the system. Hence, you may need to implement a plug-in method if you want those kinds of complex operators. But we plan to support those operators soon.

E. Filtering Scored Features

As the last step for feature selection, we need to finalize important feature words. The threshold can be used for this stage, because the input of this stage is scored features. PicAChoo allows adjustment of the threshold value during the runtime environment so that users pick and choose appropriate feature words. Finally, users are able to check selected features and can manually filter features.

F. Enriching Selected Features

An extra stage still remains that we have not introduced yet. As we have mentioned before, enough sense cannot be made from one word, because text is not just a literal but also a semantically significant unit. Consequently, the objective of this stage is enriching each selected feature by attaching additional information to the feature. Recently, some researchers have tried to adopt the notion of the complex feature[16] that utilizes the context of the feature word. However, unfortunately, defining a complex feature is quite difficult because there are a huge number of relationships between words. In PicAChoo, we consider a co-occurrence relationship between selected features and other words that describe

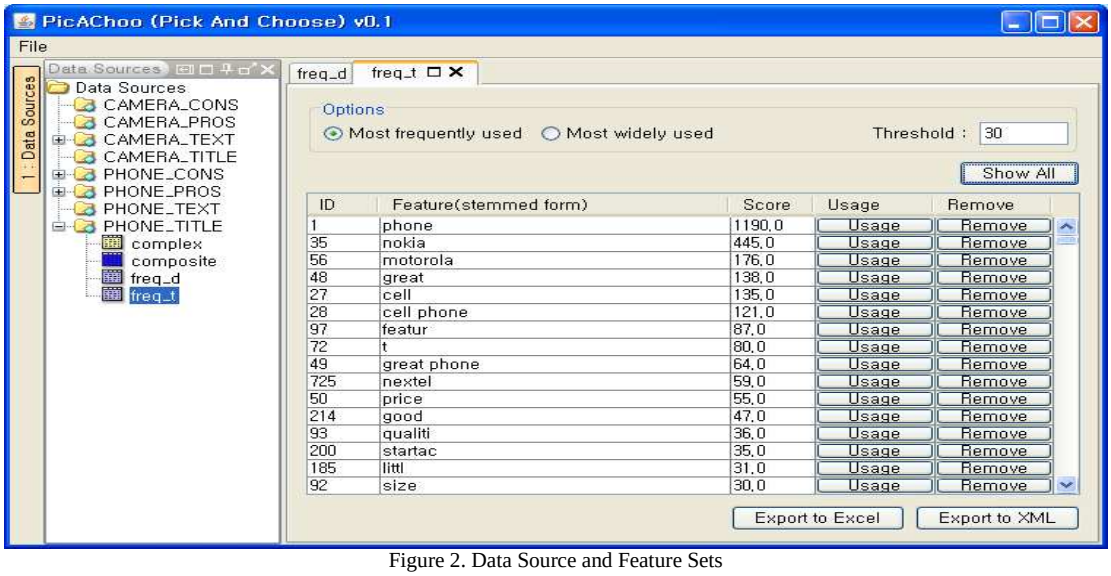

Figure 2. Data Source and Feature Sets

the feature word. For example, suppose a feature word, 'apple,' the feature can be more useful if it contains another word such as 'red' that is near the feature. It is similar to the co-occurrence-based selection method, which is a type of primitive method, but the difference is a form of the selected feature. It means the complex feature contains a pair of feature and context words. In the case of the primitive method, we can gather only feature words without contexts.

If we treat a feature as a semantic unit, we can find a wide field in which to use the feature. On the other hand, it is considerable that relationships between words can be expressed in a variety of ways. For example, if we select frequently co-occurring words as context words, the relationship between features and context words can be named as 'frequently co-occurring.' And, if we select an adjective word in front of the selected feature, the relationship can be named 'describe.' In these cases, the meaning of the relationships must be different, and should be treated in a different manner. There can be numerous types of relationships according to the research purpose, so we need to identify the semantics of the context word, and use it in the right way.

IV. Case Studies

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

A. Applying TF-IDF without Hard-coding

The tf-idf method is a fundamental and representative term weighting method, and it usually becomes the very first method when we need to extract features from raw text. There are two typical methods to apply tf-idf to our research. The first one is to find a tool, including tf-idf, and the second one is to implement it. However, the first method has a customization problem, and the second one has an implementation problem. This is why we need a customizable feature selection tool.

In our system, users are supposed to register a datasource that has documents to be analyzed. After that, preprocessing is required to analyze source documents and to generate candidate features. The feature storing model represented in the appendix is used for storing tokenized documents and candidate features. And then, the next stage is scoring. In Fig. 2, we can see registered datasources and a feature set created by frequency-based selection methods. We have two feature sets by term frequency (freq_t) and document frequency (freq_d). As you can see in the figure, we can change options dynamically, and we can see usages of a specific feature word. In addition, we are allowed to manually remove a feature word from a feature set. Finally, we can export a feature set to an Excel or XML file. The following step is customization that mixes freq_t and freq_d. In this stage, we can apply the 'and' operator so that we can find features that appear in both feature sets, and we are able to calculate (freq_t * (1/freq_d)) in a runtime. Fig. 3 represents the result of the composite method. Like the previous step, every option can be applied dynamically, and we can manipulate our feature sets.

B. Summarizing Product Reviews with Selected Features

Selected features can be used for many applications. An interesting example is text summarization. Especially, as online shopping is

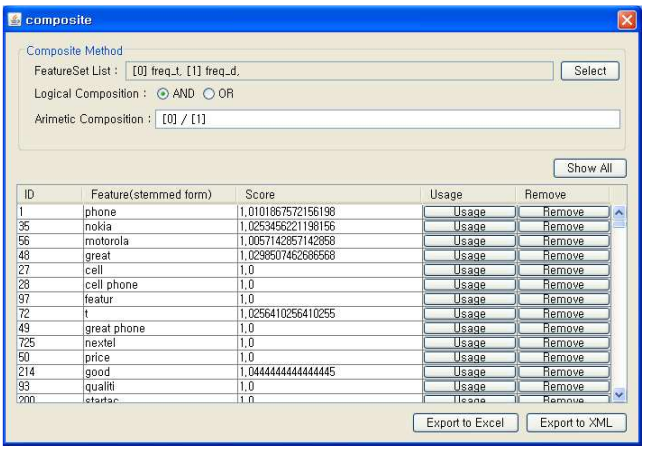

Figure 3. Customizing Stage with Composite Methods

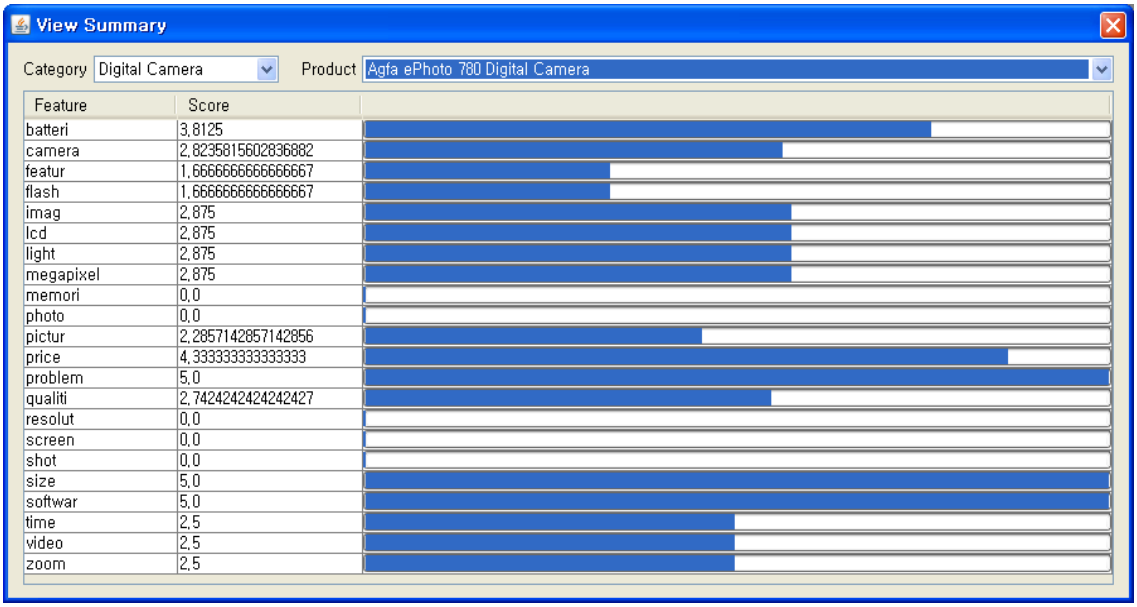

Figure 4. An Example Using Selected Features

becoming commonplace, more and more product information and product reviews are posted on the Internet. Because customers cannot see and feel the products directly, product reviews are becoming an essential source of qualitative information. As a result, the volume of reviews is increasing drastically, and review summarization is becoming very important for the Web 2.0 environment. We have conducted research about review summarization[17], and PicAChoo was responsible for extracting useful feature words and corresponding opinion words. Review summarization is not within the scope of this paper, but the important point is identifying appropriate feature words, and PicAChoo can be used for a number of applications.

V. Conclusion

Feature selection techniques let people know which features are more important than others. Therefore, we generally use several selection methods in order to build a subset of relevant features that would be exploited for classifying, clustering, summarizing, and so on. Text analysis, especially, is a major area of feature selection, and it needs more sophisticated operations because text has a number of linguistic characteristics.

We presented a text analysis tool named 'PicAChoo' for customizable feature selection with dynamic composition of primitive methods. Many linguistic features and selection methods are supported by our system dynamically. We defined primitive methods for scoring each candidate, and to enable customization of primitive methods, we also provided logical and arithmetical composite methods. Every selection method is translated into an SQL query based on the feature storing model, and the threshold value can be used to filter inappropriate feature words. Moreover, a set of selected feature words can be enriched by taking a context word that describes the context of the feature word. As a result, we would be able to use selected features to offer an intelligent service. Unfortunately, there are still some problems that we have not taken care of yet. The number of implemented plug-in methods is very small yet, as we are at the starting point. We believe that if we provide some important mathematical functions, such as sigma, log, and so on, we would be able to implement any kind of mathematical composite expression during the runtime environment. Additionally, we are planning to optimize the physical schema of the feature storing model. It is one of the most important plans, because feature selection generally deals with a huge number of documents.

Appendix A The Physical Schema for the Feature Storing Model

A fixed storing model enables dynamic SQL query generation. The physical schema is designed as Fig. 5. The feature storing model consists of four entities: Document, Tokenized Document, Occurrence, and Candidate Feature. As indicated in the names, Document and Candidate Feature store source documents and generate candidate features. Tokenized Document contains additional information about individual words, and Occurrence indicates the position of each candidate feature. Actually, the physical schema does not follow the conceptual ER model, and it has redundant data like Occurrence entity. However, sometimes, the redundancy helps the statistical analysis. Probably, we would be able to have a chance for optimizing the storing model.

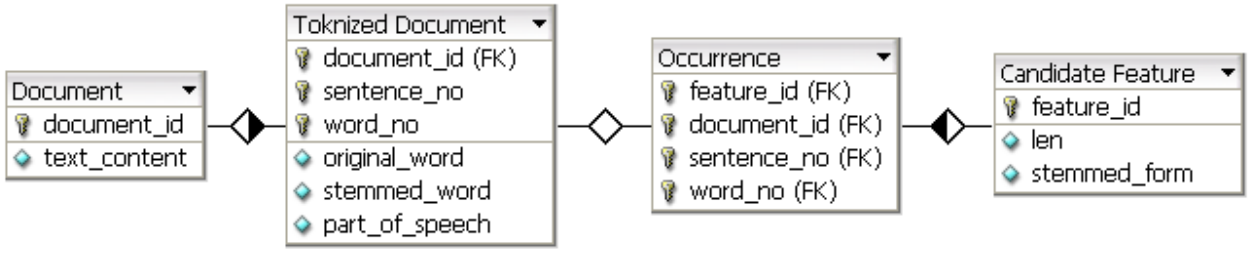

Figure 5. A Physical Schema for the Feature Storing Model

ACKNOWLEDGMENT

This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency). (grant number NIPA-2009-C1090-0902-0031)

REFERENCES

[1] David D. Lewis, Feature selection and feature extraction for text categorization, Proceedings of the workshop on Speech and Natural Language, February 23-26, 1992 Harriman, New York.
[2] Wenqian Shang et al., 2007, A novel feature selection algorithm for text categorization, Expert Systems with Applications: An International Journal, 1-5
[3] Ding X., Liu B. and Yu P., 2008, A holistic lexicon-based approach to opinion mining, ACM WSDM conference
[4] Christopher Scaffidi et al., Red Opal: product-feature scoring from reviews, ACM Conference on Electronic Commerce 2007, pp. 182-191
[5] Rada F. Mihalcea, Word sense disambiguation with pattern learning and automatic feature selection, Natural Language Engineering, v.8 n.4, pp. 343-358, December 2002
[6] Archak, N., Ghose, A., Ipeirotis, P. G., Show me the money! Deriving the pricing power of product features by mining consumer reviews, In Proceedings of the Twelveth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2007, pp.56-65
[7] Liu, B., Hu, M., Cheng, J., Opinion observer: Analyzing and comparing opinions on the Web, In Proceedings of the 14th International World Wide Web Conference (WWW 2005), pp. 342-351
[8] Yiming Yang, Jan O. Pedersen, 1997, A Comparative Study on Feature Selection in Text Categorization, Proceedings of the 14th International Conference on Machine Learning, pp. 412-420
[9] Morris, J., G. Hirst, 1991, Lexical cohesion computed by thesaural relations as an indicator of the structure of text. Computational Linguistics, 17(1), pp. 21-48
[10] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler, 2006, YALE: rapid prototyping for complex data mining tasks, In proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 935-940
[11] Hamish Cunningham, Yorick Wilks, Robert J. Gaizauskas, 1996, GATE: a General Architecture for Text Engineering, In Proceedings of the 16th conference on Computational linguistics, vol 2, pp. 1057-1060
[12] Ian H. Witten et al, 1999, Weka: Practical Machine Learning Tools and Techniques with Java Implementations, Proceedings of ICONIP/ANZIIS/ANNES'99 Workshop on Emerging Knowledge Engineering and Connectionist-Based Information Systems, 192-196
[13] C. Boulis and M. Ostendorf, Text classification by augmenting the bag-of-words representation with redundancy-compensated bigram, Workshop on Feature Selection in Data Mining, in conjunction with SIAM conference on Data Mining, 2005
[14] The Stanford Natural Language Processing Group, Stanford Log-linear Part-Of-Speech Tagger
[15] Martin Porter, The Porter Stemming Algorithm
[16] William W. Cohen, Yoram Singer, 1999, Context-sensitive learning methods for text categorization, ACM Transactions on Information Systems, vol. 17, issue 2, pp. 141-173.
[17] Jung-Yeon Yang, Jaeseok Myung, Sang-goo Lee, A Holistic Approach to Product Review Summarization, 2009, Proceedings of International Workshop on Software Technologies for Future Dependable Distributed Systems

Jaeseok Myung received his Bachelor degree in Computer Science from SungKyunKwan University at Suwon, Korea, in 2007. Currently, he is working on his joint doctoral degree in Computer Science from Seoul National University. His research interests include opinion mining, and e-Business technology.

Jung-Yeon Yang received his Bachelor degree in Computer Science and Engineering from ChungNam University at Daejeon, Korea, in 2002. Currently, he is a candidate for the degree of Ph.D. of Computer Science and Engineering in Seoul National University. His research interests include opinion mining, semantic technology, and e-Business technology.

Sang-goo Lee received the B.S. degree in Computer Science from Seoul National University at Seoul, Korea, in 1985. He also received the M.S. and Ph.D. in Computer Science from Northwestern University at Evanston, Illinois, USA, in 1987 and 1990, respectively. He is currently a professor in School of Computer Science and Engineering at Seoul National University, Seoul, Korea. He has been the director of Center for e-Business Technology at Seoul National University, since 2001. His current research interests include e-Business technology, database systems, product information management, and Ontology.