A Hybrid Architecture for Text Classification

Michael S. Register and Narasimhan Kannan
Digital Equipment Corporation
1110 Chapel Hills Drive, CXN2/39
Colorado Springs, CO 80920

Abstract

Text classification applications are emerging as an important class of text processing applications. SKIS is a prototype system that allows for the construction and use of text classification applications. SKIS uses a hybrid combination of knowledge-based techniques, statistical techniques, morphological processing, and relevance feedback learning techniques to perform text classification. SKIS has been used to construct a prototype text classification application for the routing of customer service requests within the Digital Customer Support Centers (CSCs). In this paper, we describe the SKIS run-time architecture, describe the development and knowledge maintenance environment, describe how SKIS is used, elaborate on the benefits of combining knowledge-based and statistical techniques for text classification, and compare SKIS with other text classification systems.

1 Introduction

Due to the increasing volume of information needing to be processed, many businesses today are looking to automate the handling and routing of text throughout their organizations. Text classification (categorization) applications are emerging as an important class of text processing applications. Text classification is defined as follows: given a natural language input text and a set of predefined categories, determine which categories are most similar to the input text. Text classification applications can solve a variety of problems in the routing and indexing of text. SKIS is a prototype system that allows for the construction and execution of text classification applications. SKIS provides a development environment for the building of the knowledge needed to perform text classification, a run time engine for performing text classification and an application programmer's interface for embedding text classification applications inside of other software systems. SKIS uses a combination of knowledge-based techniques, statistical techniques used in information retrieval systems [4, 5], morphological processing, and relevance feedback learning techniques [5] to perform text classification.

SKIS has been used to construct a prototype text classification application for Digital's Customer Service organization. The application classifies incoming customer service requests submitted electronically to Digital's Customer Support Centers (CSCs). Once classified, the service requests can be routed to the technical support specialist with the required expertise to address the customer problem.

In the remainder of this paper, we discuss the technology used by SKIS in greater detail. First we present the SKIS run-time architecture, then we describe the development and knowledge maintenance environment. We also describe in more detail our prototype text classification application. Finally we elaborate on the benefits of the SKIS architecture and compare it with other text classification systems.

2 SKIS Run Time Architecture

The architecture of SKIS is presented in Figure 1. The boxes on the left side of the diagram are the major components of the SKIS run time system; the boxes on the right side are the major components of the domain specific knowledge base that SKIS uses to perform text classification.
Figure One: SKIS Architecture
2.1 Morphological processor

The job of the morphological processor of SKIS is to extract all the relevant information that is explicitly contained in the input text. To accomplish this task, this component uses a lexicon of keywords, phrases, and regular expressions. This lexicon contains all the information that is considered relevant for extraction purposes.

SKIS allows the inclusion of single word nouns and verbs and multiple word noun and verb phrases into its lexicon. The morphological processor of SKIS will recognize the root form and it will also recognize morphological variants of the root. SKIS also allows synonyms of a keyword to be entered into the lexicon; these are useful when defining keyword classes or when writing disambiguation rules.

The SKIS lexicon can also include single token (single word) regular expressions. If a regular expression is in the lexicon, SKIS will identify any token in the input text that matches against the regular expression. Being able to define regular expressions in the lexicon gives the lexicon maintainer more flexibility than being restricted to defining literal words and phrases. For example, the maintainer can define the expression "SYS\$n*" to match all the VMS' system service routines instead of having to enter the name of every system service routine directly into the lexicon.

A limitation of the SKIS morphological processor concerns spell checking. Primarily due to time and resource constraints we have not yet added a spell checking capability to the morphological component. However, we have designed SKIS in such a way that adding a spell checking component would not be a difficult task.

The output of the morphological processor is a list of lists: each sublist corresponds to a single sentence in the input text and contains all the recognized keywords, phrases and regular expressions in that sentence. This list is passed onto the intelligent inferencer component for further analysis and possible augmentation.

2.2 Intelligent inferencer

The intelligent inferencer takes the information extracted directly from the problem statement by the morphological processor and attempts to add to that information by deducing further facts that are implied by the keywords and phrases identified. This component uses a hierarchically structured knowledge base of keyword classes. Each class contains a group of keywords, phrases, and/or regular expressions (already defined in the lexicon) that share something in common. The classes are structured into a hierarchy such that classes themselves can be members of other classes.

What is useful about these classes is that we can attach facts to them to deduce implied information if a member of a class is found in the input text. If a keyword class member is identified then all the facts attached to that class are inferred and added to the deduced facts list. In addition, all the facts attached to the parent classes are inferred as well.

In addition to inferring new facts with keyword classes, we can also substitute more general descriptions of an identified keyword in an attempt to match other key phrases. We call this process "keyword substitution" and it is an attempt to match key phrases in the lexicon that couldn't be matched explicitly. For example, let's say we want to match the phrase "Analyze Disk" every time we see "Analyze X", where X is a specific disk device. We would also like to be able to do this without having to enter a single verb phrase for every specific disk device (e.g., "Analyze RD54", "Analyze RA81", etc.) into the lexicon. Using keyword substitution, the group of like devices can be grouped into a class and a word attached to the class to substitute for matching phrases in the lexicon. Going back to our example, we can define a class of disk devices and associate the keyword "disk" as a substitute. This way we can recognize "Analyze " as "Analyze Disk" without having to put "Analyze RD54" into the lexicon.

The output of the intelligent inferencer is a list of all the extracted keywords, phrases and regular expressions and a list of all the deduced facts that the intelligent inferencer was able to infer. Associated with each extracted keyword is a number designating the frequency of the keyword in the input text.

\footnote{VMS is a trademark of Digital Equipment Corporation.}
\footnote{RD54 and RA81 are trademarks of Digital Equipment Corporation.}
2.3 Similarity measuring component

This module is responsible for returning a numeric similarity score for each category in the knowledge base. Each score indicates how similar a given category is to the input text. This component is implemented using relevance feedback technology for information retrieval applications [4,5]. We have modified the relevance feedback techniques somewhat to make them applicable to text classification.

The similarity measuring component of SKIS uses a knowledge base of keyword->category profiles to determine similarity scores for all of the categories defined. Each category has its own profile containing the keywords, phrases and regular expressions that are relevant to that category. Once a text input is parsed by the first two components of SKIS, we have a list of all the keywords present in the input text, as well as the number of times they occur in the input text (called term frequency). The category profile can be represented as a \( n \)-dimensional vector of the form \( C = (c_1, c_2, ..., c_n) \), where \( n \) equals the total number of possible keywords in the lexicon and the individual \( c_i \) elements represent the corresponding profile weight of keyword \( i \) in the category profile. The input text can also be represented as a \( n \)-dimensional vector of the form \( T = (t_1, t_2, ..., t_n) \), where \( n \) is as above and \( t_i \) represents the corresponding weight of keyword \( i \) in the input text. Similarity between a category and an input text can then be measured as the inner product between these corresponding vectors, i.e.,

\[
Sim(C, T) = \sum_{i=1}^{n} c_i \ast t_i
\]

The main question that remains is how to come up with initial weights for category profiles and how to come up with keyword weights for input texts. The formulae SKIS uses take both term frequency and collection frequency as input. In text classification terms, collection frequency is the number of category profiles a specific keyword occurs in. The profile weight calculation formula is: \( PW = \log \left( \frac{CAT}{CF} \right) \), where \( CAT \) equals the total number of defined categories and \( CF \) equals the collection frequency of the given keyword (this formula uses only collection frequency). Notice that as \( CF \) increases, the profile weight decreases. This makes sense because if a keyword provides evidence for a large number of categories then its profile weight should be lower than a keyword that provides evidence for a small number of categories.

The keyword weight calculation formula is:

\[
KW = \frac{TF \ast \log \left( \frac{CAT * CF_i}{TF} \right)}{CKW}
\]

where \( CAT \) and \( CF \) are as above, \( TF \) equals the term frequency of the keyword in the input text, and \( CKW \) is the combined keyword weight and is calculated as follows:

\[
CKW = \sqrt{\sum_{i=1}^{n} \left( tf_i \ast \log \left( \frac{CAT * cf_i}{TF} \right) \right)^2}
\]

where \( n \) is the total number of keywords found in the input text. \( tf_i \) and \( cf_i \) are the term and collection frequencies for one of the found keywords, and \( CAT \) is as previously defined.

Once similarity scores have been calculated for all categories, a dynamic threshold is applied to the list of categories; this threshold is a given tunable offset from the similarity score of the most similar category. In other words, if \( N \) is the highest similarity score for the input text and \( M \) is the pre-defined threshold offset, then \( N-M \) is the threshold value. All categories whose similarity scores are below the threshold value are discarded and those above the threshold value are passed to the next SKIS component, along with the extracted keywords and deduced facts.

2.4 Category disambiguation

The category disambiguation component uses a rule base to select certain categories over other categories based on the extracted and deduced information from the input text. The idea here is to use rules to decide the appropriate category when more than one category is a potential candidate for being the most similar. The left hand sides of the rules consist of CATEGORY and KEYWORD slot-value pairs and deduced facts. The right hand sides of
the rule merely assert SKIS preference for one category over another category (or set of categories).

When this module is invoked, all the rules that can apply to the given input text are fired and all the category preferences are recorded by SKIS. SKIS then presents to the user as the final output of the run time system an ordered list of the most similar categories that do not have any other category with preference over them. Once a category has been identified as having at least one other category with preference over it, it is eliminated from the most similar categories list. In addition to the list of most similar categories, SKIS also returns to the user as output the list of extracted and deduced information from the input text.

The category disambiguation module is detachable from the SKIS run time architecture. If a particular text classification application has no heuristics for category selection, then we can remove the category disambiguation component and rely solely on similarity scores to determine the most similar category. Detaching the rule base will most likely result in a decrease in the accuracy of the classification; but for some applications no such rule base exists. By making the rule base detachable, we have widened the range of potential applications that can be developed using SKIS.

2.5 Relevance feedback learning component

Once text classification is done, this component can be used to adjust the keyword->category profile weights to achieve better accuracy. The component uses a modified relevance feedback technique [5]. The way this component works is as follows:

- Collect all the text classifications over a given period. The classifications should include the input text, the chosen most similar category, and the keyword weights for the extracted keywords.

- For each category profile do the following:
  - Collect all the text classifications where the particular category was identified as the most similar.
  - Determine which input texts were correctly classified and which were not.
  - For all the correctly classified input texts, add their keyword weights to the corresponding keyword profile weights in the category profile.
  - For all the incorrectly classified input texts, subtract their keyword weights from the corresponding keyword profile weights in the category profile. Also, determine the correct category and add the keyword weights to that category's profile.

3 Development Environment

SKIS provides a graphical user interface (Motif based) for the development of text classification applications and the acquisition and maintenance of text classification knowledge. By providing a graphical user interface we have made the development and maintenance task easy enough for a non-programmer to perform. Instead of entering SKIS knowledge in terms of the syntax and semantics of rule based and lexicon based languages, the knowledge base developer is guided by a graphical system that knows about language syntax and can hide many of the details from the developer.

The development environment also includes a graphical user interface debugging environment for testing a knowledge base to verify its accuracy. The developer can type in any text and run the SKIS run time engine to classify it using the developer's knowledge base. Once classification is finished, the development environment will display all the identified keywords, deduced facts, most similar categories, and category preferences. The developer can then save the classification results to a file, or can get explanations of how the results were either inferred or calculated.

4 A SKIS Application

Recently, Digital began a new service that allows customers to send service requests electronically to Digital's Customer Support Centers (CSCs). One of the problems with the service is that currently a customer has to determine which one of over 200 electronic mail addresses within Digital to send their service request. The mail addresses correspond to the technical skills needed to solve customer problems. Customers sometimes have dif-
difficulty determining the proper address and as a result, the service requests end up going to the wrong technical specialist at the CSC.

To address this problem, we have developed a prototype application using SKIS that automatically determines which mail address is the correct one given the customer service request. Our prototype application includes a lexicon of over 2000 keywords, 90 regular expressions, 134 categories (mail addresses), 24 keyword classes and a knowledge base of 182 category disambiguation rules.

The prototype application covers a range of topics concerning all of the software products supported out of the Colorado Springs CSC. The categories in the prototype sometimes map directly to software products, but in other cases they represent particular components of a software product (e.g., the I/O subsystem of an operating system).

SKIS provides an Application Programmer's Interface (API) for embedding a text classification application within another application. This facility allows us to easily embed our text classification prototype within the current service request submission system. The API provides the programmer with three simple functions. An initialization function is called at the beginning of the programmer's application to initialize the internal data structures of the text classification application (it is called only once). To parse text, a parsing function is provided that takes text as input and returns a data structure containing the extracted keywords, the deduced facts, the most similar categories, and the category preferences. This function can be called as many times as desired. Finally, the API provides a cleanup routine that deallocates the memory originally allocated to the internal data structures. This function is called after the application has completed all of the text classifying work it has to perform.

Initial tests have returned an accuracy figure of 80 percent for our prototype without using the relevance feedback learning component (there currently exists a prototype of the relevance feedback learning component, but it has not yet been fully integrated with the rest of the SKIS architecture). We believe that this figure can be improved upon in two ways: use of the relevance feedback learning module and further tuning of the category disambiguation rule base. Future research will determine how much accuracy is gained by using the learning component and how best to use the learning component (e.g., what is the most appropriate size for a learning set).

5 Benefits Of Architecture/Related Work

Perhaps the text classification system that mostly closely resembles SKIS in function and architecture is TCS (Text Categorization System) [2]. TCS is a predominantly rule-based system. It parses the input text using morphology techniques, attempts to recognize concepts in the text, and then uses a rule base to map from identified concepts to categories. Applications developed using TCS have shown to have very high accuracy (above 90 percent) at the cost of a large amount of knowledge engineering to tune the rule base to achieve the high accuracy [3].

SKIS shares with TCS a morphological component as well as a rule base component, but SKIS is also different from TCS in several ways. First, TCS does not include a similarity measuring component. TCS also does not include any learning component. TCS requires the building of a rule base, which SKIS does not (rule bases are optional in SKIS). We believe that integrating statistical and learning techniques into the SKIS architecture makes SKIS less dependent on the rule base component and thus applications developed using SKIS require much less knowledge engineering. This assertion is supported when we compare the amount of knowledge engineering time spent to develop the Digital CSC service request classification prototype (approximately 3 months) versus the knowledge engineering required to develop applications with TCS (approximately 3 years -- [3]). Currently SKIS applications do not match the accuracy performance of TCS developed applications. We will need to perform more tests to determine if the SKIS learning component can improve accuracy to near TCS performance without extensive knowledge engineering of the rule base.

Another interesting text classification system is the CBR-PRISM system [1]. CBR-PRISM is a text classification application developed for routing bank Telexes. The system uses a case-based retrieval mechanism to retrieve previously routed telexes to determine where to route a new telex. About the only part of CBR-PRISM that is common with SKIS is the morphological front end that recognizes keyword patterns in the input text. Once the keywords are identified, CBR-PRISM immediately tries to match them to a previously routed telex and come up with the correct classification. CBR-PRISM does this through an inductive learning algorithm; it builds up a hierarchical library of past cases to search on. CBR-
PRISM does not include a relevance feedback similarity measuring component nor does it include a rule-based component. CBR-PRISM seems to have also achieved one of the goals of SKIS -- high accuracy (around 90 percent) with a low knowledge engineering cost (because of its reliance on past cases as opposed to a rule base).

6 Summary

This paper has presented another approach to the text classification problem. We concur with others that there are a large body of applications that can be developed by text classification systems. Our approach attempts to minimize the amount of knowledge engineering required to build such applications without sacrificing much in terms of accuracy of classification. By including statistical and relevance feedback learning techniques into our architecture we believe we have achieved this goal.

References


