Trouble Ticket Solution Provider

Using Natural Language Processing to Perform Information Extraction

Christopher Stoll & Patrick Lemmon
Department of Computer Science
The University of Akron
Akron, Ohio, USA

I. Problem Statement

Most large corporations provide various information technologies to their workers. To support the provided hardware and software they often have a help desk or service desk which is made up of people and software. Service desk employees are often entry level and have little understanding of the business and its specific technical problems, so software is employed to route tickets to the appropriate staff and record the knowledge which is learned along the way. For improved efficiency it is important to be able to access the information that has been amassed in the service desk software, but simple text queries often fall short. The purpose of this experiment is to use natural language processing techniques to extract information from a series of service desk tickets, then structure the information in such a way that it can be retrieved with more precision than is provided by simple text queries.

II. Approach

A. Information Extraction

The process begins with information extraction. The approach selected for this task is that of the cascaded finite state transducer. A batch sequential architecture is used to deterministically transform raw service desk data into clean, useful information. Each step in the sequence is described below.

First, service desk tickets which are handled by strictly defined processes or potentially contain sensitive information are removed from the raw data.[1] For example, if a user requests to have their password reset a rigid and well-known process is followed, so there is no need to provide a solution to this type of problem. Also, if the information extraction process works well, then it could be

[1] In script ie_preproc_a.py

possible for people to mine confidential information, thus every attempt is made to remove records which may contain sensitive information.

Next, boilerplate text is removed from the raw data.[2] The people using the service desk software often write courteous notes to the users when their problems are resolved, this is good for informing the user, but it does little to help dedifferentiate individual problems. Also, email correspondences are recorded in the service desk, and these often contain the senders contact information along with the standard email header. This is also of little use in classifying the problems; if it was useful to have the user's contact information, then it would be cleaner to pull it from a distinct field. This step also removes username information.

After the boilerplate text is removed from the raw data, regular expressions are used to remove a broader range of useless data.[3] Dates and times found in the raw data are not useful for classifying and solving problems, so they are removed. If temporal information was desired it could be extracted from the database in a uniform format. During this step special characters are also removed and white spaces are compressed.

With preprocessing complete, the next step is natural language processing. Natural language processing is in itself a multi-stage process.[4] First, the text data from the corpus is broken into sentences, then the sentences are broken into words. Each word is tagged for its part of speech. Next, each word is stemmed using the Lancaster stemmer provided with the Natural Language Took Kit (NLTK). In the same block where stemming occurs the program discards sentence fragments; those sentences which have either no verbs or no nouns. Named entity recognition is next run against the text to identify proper nouns. Then, phrases are created using a few simple grammar rules. The combination of phrases extracted from each service desk problem is used to represent a summary of the problem.

At this point the original problem statement of *"—melenrdr - 04/11/2012 - 06:53 1 3 - - - - - - - - - - - - - -*

From: Sevel, Evan NI/IBC-SIM

[2] In script ie_preproc_b.py

[3] In script ie_preproc_c.py

[4] In script ie_proc.py

Sent: Wednesday, April 11, 2012 12:13 AM To: IT-Support Schaeffler-Group North America Subject: 500 Internal Server Error When trying to access the Self Service to access my benefits and payment, I receive the error '500 Internal Server Error.' Evan P. Sevel International Key Account Manager Schaeffler Group USA Inc. Mobile: 704-519-8474 E-mail: sevelean@schaeffler.com Catalog: http://medias.schaeffler.com/medias/en!hp/" would be converted into the following phrases: *"server error when", "tri", "access the self servic", "access, benefit and payment", "receiv", "the error intern server error"*. These results are typical, and, on average, provide a good summary of the actual problem.

B. Machine Learning

After the natural language processing utilities generate the summarized data, a method needs to be used to query the data.[5] It is assumed that the above described process would be ran on any new queries, so the query value would also be formatted as above. A database is created with all of the training data, and the index of this database are the distinct phrases generated by the natural language processing. A relation between each distinct phrase and each occurrence of the phrase in a problem is established. The phrase's importance in the sentence is calculated along with the phrase's importance in the corpus.

When a query is performed a set is created of all the service desk problems with phrases that overlap the phrases from the query. A dynamic programming algorithm is then used to compare the the query to each of the problems in the set. (Note: the present dynamic programming algorithm is extremely simplistic; it does not use the probability of each phrase when generating a match score, it only uses ones and zeros.) If the number of matches between the two strings is greater than fifty percent of the longest string, then it is considered to be a match. This is an arbitrary number, but it seems to yield resonably precise results. More work could be done here to more scientifically balance precision and recall.

III. Design and Implementation

Since this application was experimental and it could potentially take a lot of time to

[5] See script ml_basic_b.py

process an entire corpus, it was written as a set of independent Python scripts which can be run independently from the command line. Python was chosen because that is the native language of the Natural Language Toolkit (NLTK). No interactive query interface was designed as of yet, only the train and test portions were designed in order to analyze the performance of the application.

IV. Results

The results of this system depend greatly upon the inputs. The data for this project was provided by a large international firm and showed considerable variation between data sets. The initial test data for the system consisted of 12,729 records from a few plants in the United States. After preprocessing 5,171 records remained. Below are the testing results.

	Training records	Test records	Test solved	Test sovled
Group 1	4134	1037	107	10%
Group 2	4110	1061	100	9%
Group 3	4137	1034		%
Group 4	4151	1020		%
Group 5	4105	1066		%

The second data set for the system consisted of 10,666 records from any plant in the company's global operations. Only 3,610 records made it through the preprocessing step, yet many of those left were still not user generated records; they were made by automated or semi-automated processes. It seems that the higher solution rate is most likely due to the high percentage of machine generated tickets.

	Training records	Test records	Test solved	Test sovled
Group 1	2902	708	345	49%
Group 2	2932	678	324	48%
Group 3	2870	740	331	45%
Group 4	2878	732	366	50%
Group 5	2855	755	342	45%

V. Conclusions and Future Work

The approach taken here yields very poor results with the given corpus sizes. The number of correct solutions may improve with larger corpus sizes, but further work would need to be done to determine this. Improving the phrase matching algorithm may also help, but the biggest improvement would likely come from using a different approach.

The data sets have also been processed using Weka to check for correlations between the service desk ticket problem and the solution category. Initial tests with the second data set seem to suggest that categorization rates of better than 85% are possible. However, categorization is just one step

towards providing a solution. More work needs to be done in systematically verifing the categorization rates and then providing solutions in that even more limited domain.

VI. Remarks

The contributions of each of the team members can be seen by looking at the project source repository on Github. The master commit history can be found here: https://github.com/stollcri/UA-3460-560-P2/commits/master