Content Rating for Online Social Networks

Bryan Nehl

Kansas State University
bnehl@k-state.edu

Abstract

The author will demonstrate that text data from an Online Social Network can be tokenized, filtered, expanded with synonyms and rated with an approximated Vector State Model cosine Θ function. The resulting value is used by an agent to rate and possibly filter subscribed content. Additionally, the agent can monitor a public content source and identify and display content that may be potentially interesting to the consumer.

Introduction

Today there are numerous sources for real time information resulting in information overload. Colloquially, we have so much information streaming in that it is referred to as "drinking from the fire hose." One approach to dealing with this overload is to not receive the information at all. That approach is quickly dismissed as not feasible since there would be information the consumer would want to see that would be lost. With the *SkimmerAgent* project the researcher takes the approach of rating *Twitter* content from friends and "skimming" the public time line for interesting content.

The organization of this paper is as follows: this introduction, discussion of the concepts used in this project, technical details of how the project was implemented, results, possible future research and conclusions.

Background

One of the many source of information today are online social networks (OSN) such as *Twitter*, FaceBook and LinkedIn. *Twitter* lets users post short SMS sized messages telling people what they are doing. *Twitter* users primarily view messages (tweets) from people they follow (friend). However, there is also a public timeline which displays tweets from users system wide. If the user has not marked their tweets as protected any user can see their tweets in the public timeline. SkimmerAgent will interact with the user's *Twitter* account, specifically, rating and displaying tweets from their friends as well as skimming the public timeline for interesting content.

According to Manning, Raghavan, & Schütze (2009), tokenization is the process of breaking a given document up into tokens. *SkimmerAgent* needs to use tokens to analyze the content of the tweets.

SkimmerAgent also makes use of the concepts of content identification and filtering. *SkimmerAgent* is able to identify user names, hash tags and URLs in the tweets. Filtering is implemented to remove URLs so that they are not considered in the content rating.

Case folding is the process of eliminating case from being taken into account when doing term matching. For instance, after case folding "CaSe" and "case" are equivalent. *SkimmerAgent* implements case folding as a filter.

Stop words are words that are considered to be common or frequently used and of little or no value for classification purposes. For example, words such as: and, if, then are stop words. *SkimmerAgent* uses the stop words list that MySQL has published.

The vector space model (VSM), Lee, Chuang, and Seamons (1997), is a way of looking at documents and document sets. The VSM consists of a vector of tuples that consist of the term, term frequency and document frequency. Document similarity can be obtained by calculated the angle between the query VSM and the source VSM. However, computing the angle is an expensive calculation so the cosine Θ approach or other approximation is often substituted. *SkimmerAgent* uses an alternate approach proposed by Lee.

From discussion with Dr. Hsu and the Turney (2005) paper the researcher implemented the idea of synonym expansion. That is, the query terms are expanded with additional terms that are synonyms of the query terms.

Precision is the fraction of returned documents that are relevant. Recall is the computed by dividing the number of relevant items retrieved out of the number of all relevant documents. There is an inverse relationship between recall and precision. That is, if you have a 100% recall (like this application) expect that you will have a low precision.

PEAS

This section will review the Performance, Environment, Actuators and Sensors associated with the *SkimmerAgent*.

Performance

The agent performance is measured by the unit tests created in development. These are covered in detail in the Evaluation Methods section.

Environment

The agent will be working in an environment that includes interaction with *Twitter*, the user and other agents. The *Twitter* world is partially observable. The agent may perform queries of the *Twitter* environment. However, it does not have all information about what is happening in the *Twitter* world.

The agent may be considered strategic since it is in an environment where it can work to alter its own environment. However, there are other agents that are also interacting with the *Twitter* world which will partially determine the agents view.

The agent is working in an episodic manner. It checks the state of the world, processes and then acts.

The *Twitter* world does change while the agent is deliberating. For instance, other agents can be posting information.

The agent is limited in the actions that it can perform therefore it is considered a discrete environment. However, the actions can take nearly unrestricted parameters.

The environment is multi-agent as the *SkimmerAgent* interacts with the user and indirectly with other agents via the *Twitter* world.

Actuators

SkimmerAgent has the ability to send messages to the *Twitter* world. Currently, *SkimmerAgent* only sends those messages that the user requests it send.

SkimmerAgent also displays processed friend and public timeline data to the users display.

If query term expansion is active, the agent displays the list of expanded query terms to the user.

Sensors

The user may interact with the agent by setting options via a menu interface, specifying a query sentence or inputting a message to tweet.

The agent also retrieves and processes the user's friend and public timelines.

Methodology

The researcher chose to implement this project with Microsoft Visual Studio 2008 in C# with the .Net Framework 3.5 as it is the environment with which they are the most productive. The application was created with an iterative approach utilizing test driven design (TDD). Code decoupling was enhanced by the use of an interface based thought process. By tending towards a fluent interface it is thought that code should be easier to comprehend.

The architecture of the application has been modularized to allow for testing and reusability. For example, there is a Filtering project and a corresponding Filter Tests project. The filtering DLL may now be used in other projects. The GUI of the application uses a custom component for tweeting and implements a Model-View-Controller (MVC) pattern in the overall primary form.

Connecting to Twitter

To minimize the development time the *TweetSharp Twitter* API was utilized. *TweetSharp* is a C# API wrapper around the standard REST style *Twitter* API. The *TweetSharp* API uses a fluent interface and includes classes to ease deserialization of the various *Twitter* feeds.

OAuth is a library that was needed in order for the application to be able to work as a proxy for the user and to have *SkimmerAgent* be identified when the user tweeted. The application was registered as an *OAuth* application and in code the application launches a web browser for the user to authorize the application. The user has to copy the PIN number that is returned from *OAuth* and plug it into Skimmer. With this PIN *SkimmerAgent* can work on behalf of the user. The current version of *SkimmerAgent* requires PIN authentication for every run.

In this first iteration the agent simply connects to the *Twitter* service.

Posting a Tweet

Partially in order to verify connection with the *Twitter* world and to provide more utility to the user, the next step was to be able to send a message or "tweet" to *Twitter*.

A user control was created which would display the number of allowed characters remaining allowed. The control has an event to which other controls may subscribe. When the event is triggered the control container requests that the message be sent to *Twitter*.

Retrieving Tweets

In order to do content rating, we must first have content! The next iteration focused on retrieving the friends timeline and the public timeline.

The *TweetSharp* API was used to retrieve the timelines and put them into an object structure which could be processed.

Information Processing

To implement tokenization a Tokenizer class was created which splits the incoming tweet up into terms based on a set of delimiters. A pre-filter had to be implemented to remove URLs. Otherwise, the Tokenizer would split up the URL as if it were sentences that had been run together.

A filter interface is specified as taking a list of tokens, performing the filtering function and returning the filtered terms. *SkimmerAgent* uses filters to identify and remove stop words and user Ids. The stop word list is the list used by MySQL. Another filter is implemented which removes the leading # from hash tags. Case folding is also implemented as a filter. The implementation of IFilter, IExpand and a fluent interface paradigm enabled the chaining of methods. For instance, the content to rate could be created as the result of chaining like: StopWordsFilter(ToLowerFilter(UserIdsFilter(StripHashes Filter(Tokenize(PreFilter(source)))))).

Synonym Expansion

The WordNet library was used in the term expansion class. The Expand method would create a new list of tokens to be returned. Initially, the list would include the original query tokens. Then the WordNet library was used to obtain synonyms for each token. These synonym tokens are then added to the expanded terms list which is returned.

Transformation from Tokens to VSM

Before the content can be rated it must first be transform the tokens into a Vector State Model (VSM). This was one of the most challenging aspects of the project.

The query document and source documents must be converted into a VSM representation. The VSM vector is made up of VSM terms which map to tokens in the originating document. Each token in the originating document is *AddTerm()* to the VSM object. The *AddTerm()* method looks for an existing VSM term. If a match is found, the term frequency is incremented. If the term is not found, a new term is added to the VSM vector and the initial frequency is set to 1.

When it comes to computing document similarity the researcher added a short circuit to the calculation. If either VSM contains no terms, the resulting similarity is 0. This raises the question, what if we compare two empty documents?

Content Rating

The tweet ranking method implemented an approximation that was described in Lee (1997). The similarity between the query document and the search document is computed by using the summation of the term frequency divided by the square root of the number of terms in the document space. The document set frequency is ignored since we are only looking at individual tweets and not the overall content from a single given user.

SkimmerAgent pulls the users friends timeline and rates all of the incoming tweets. The public timeline is queried once a minute, which is the delay between updates from *Twitter*. There is the potential to connect to an alpha service which provides a sample of the full *Twitter* stream. The display criteria is set to >= 0 so that all tweets are displayed. The application is set to display any content that is ranked >= 0.01. This is a very low threshold. However, it seems to work very well for our purposes.

The author has made available a quick video how-to and executable code on the *SkimmerAgent* blog.

Evaluation Methods

As part of TDD, unit testing was employed throughout the development process. In this way, the researcher was able to verify expected functionality.

Pre-filtering for URLs was tested with source strings in numerous formats. The unit tests include empty strings, basic URLs, fully qualified URLs and multiple URL occurrences.

Tokenization was tested in a similar fashion to URLs. Numerous tests were constructed using single and multiple delimiters. The tests then checked that the expected tokens were ascertained.

Username identification was tested by running a test set of terms through the filter and verifying that the UserIds were removed from the list of terms. Stop words filtering was done in the same manner.

Lowercasing or case folding was provided tokens of mixed case, all upper case and all lower case as test inputs.

Proper functioning of the synonym expansion was verified by testing words with known synonyms.

The content rating calculation was tested by creating query and source documents that would satisfy conditions such as no similarity, full similarity and then checking the evaluation against expected results.

One of the benefits of unit testing is that if a potential defect is spotted in production a test can be written to reproduce the problem. Then the developer can revise the code to handle the new test case while not breaking any of the existing functionality. The other test cases must still be passing after modifying the code for the new test case.

Results

Initially a simple content rating had been implemented which was purely term frequency occurrence based and normalized for a result in the 0 to 100 range. For our purposes this seemed to be an acceptable calculation. After implementing the cosine theta approximation Lee (1997) the resulting rating is no longer normalized for size. Using the content rating *SkimmerAgent* has successfully identified “interesting” content from friends as well as displayed content picked up from the public timeline.

WordNet synonym expansion yielded some unexpected results. First, some of the synonyms given were multi-word phrases which *SkimmerAgent* is not currently equipped to identify. The multi-word phrases were returned strung together with underscores where the spaces would be. This would mean additional tokenization of the phrase and then a more advanced search through the source document. The second surprise was the expansion of an airport code, SPF, to Sun_Protection_Factor!

In the initial implementation of *SkimmerAgent* the weight of original terms is equivalent to that of the expanded terms. Some words get expanded to very common terms such as "word." This can lead to many false positives being displayed in the skimmed public timeline.

Precision and Recall

For *SkimmerAgent*, recall is always 1. That is, it always has a 100% recall because the agent retrieves all tweets for analysis. Of course, this results in a lower precision.

To evaluate the content rating algorithm, the researcher sampled the CNN twitter feed. A console application was created to rate the content of the tweets. An initial run with a single query term set to the word "president" yielded no hits—a 0 for precision. Reviewing the source data, the researcher chose the term "boss." This yielded a precision of .1. There was no difference in using an expanded goal terms set.

Next, the researcher used a carefully crafted data set. The data set was built in a manner that with synonym expansion there would be false positive ratings. The query term used the word "president." The data set included the single word "president" along with the word used in sentences. There were also sentences constructed that referred to chairman of the board, chairwoman, and chair as a piece of furniture.

Running test data through the core processing routines without synonym expansion yielded a precision of .3. Running test data through the core processing routines with the synonym expansion resulted in an increase of precision to 0.6. However, there were 3 false positives. Or an error rate of 50%!

Possible Future Research

The username and URL filters could easily be used as the basis for creating content classifiers which could be used to feed other methods. For instance, capturing the URLs could provide a capability to aggregate URLs throughout the day, pre-fetch the web pages and provide the user with thumbnails for review.

Once a tweet with interesting content has been identified on the public time line, a full document set rating could be done against the tweets of the posting person. That is, the Lee (1997) computation with document frequency term could be implemented. If a strong rating is returned, *SkimmerAgent* could propose that the user follow this person—or automatically follow them!

There is also the potential to use the same technique against the document set (tweets) of existing friends with the potential of suggesting people to prune from the list of followed people.

The existing agent could be expanded to handle phrases as terms and not just single words. It would also be nice to be able to specify which words to expand.

During research, the only application that was found to be using supervised learning was at the website www.philtro.com. However, at the time of this writing the website is not available. The website displayed your tweet information and the user would mark it as having content that they would like to see come through in the filtered feed.

A nice enhancement would be to add a learning facility. The agent would start with using user provided terms as a *base*. The user could then train with additional positive matches (generalization) as well as identify false positives for term removal (specialization). The existing architecture of filtering and expanding could be used to implement the specialization and generalization respectively. The associated filter or expansion method work with the list of terms provided to it by the agent.

I've also thought about the idea of a setting which would request that the agent occasionally let random content through the public timeline filter. Or the agent could monitor trending topics and provide a tweet from that context. The idea behind these ideas is to potentially introduce the user to some content outside of their normal interests.

There is also the potential to aggregate and skim data from other OSNs. An agent that aggregated data from numerous networks and by default only displayed sentences with ratings above a user configurable threshold could save that user significant time.

Conclusions

During the course of developing *SkimmerAgent* the researcher learned about different AI areas. Development of the agent gave birth to numerous additional possible research projects in the area of Online Social Networks. The agents content rating of friends tweets has been useful for "at a glance" review of tweets. Having a skimmed public timeline has led to seeing more interesting content and prompted further research on a number of occasions. The researcher feels that *SkimmerAgent* could serve as a good starting point for these new questions.

References

Manning, C., Raghavan, P., & Schütze, H. (2009). *An Introduction to Information Retrieval: Online edition*. Cambridge, England: Cambridge University Press.

Retrieved from http://nlp.stanford.edu/IR-book/information-retrieval-book.html

Lee, D. L., Chuang, H., and Seamons, K. (1997). Document Ranking and the Vector-Space Model. *IEEE Softw.* 14, 2 (Mar. 1997), 67-75. DOI= http://dx.doi.org/10.1109/52.582976

Turney, Peter D. (2005) *Measuring Semantic Similarity by Latent Relational Analysis.*

Salton & Buckley (1988). *Term-Weighting Approaches In Automatic Text Retrieval.* Information Processing and Management Vol. 24, No. 5, pp. 513-523

Gulutzan, P. (n.d.) *MySQL stop words.* Retrieved November 8, 2009, from http://dev.mysql.com/tech-resources/articles/full-text-revealed.html#stopwords