A Crowd-Sourced Associative Rule Mining Natural Language Engine*

Justin Mrkva[1]

***Abstract*— This project studies the potential for creating a natural language engine based on association rule mining. The goal is not to create a complete artificial intelligence, but rather to predict an upcoming stream based on entered data. In this form, starting to type a sentence will begin looking up the next series of words based on likely candidates from the rule miner.**

I. INTRODUCTION

Natural language engines are a tricky thing. English speech is complex, and creating meaning from an algorithm has long been a primary goal of artificial intelligence (AI) systems. In order to appear intelligent, an AI must be able to create language.

This kind of processing is difficult, computationally expensive, and non-obvious. Going about creating a system that appears to be able to think for itself is difficult to do when the underlying hardware and software follows utterly different paradigms than human thought. Applying human thought patterns to a machine is a broad field of study.

In this paper, a method is presented that generates content from nothing but crowd-sourced data. Specifically, Twitter data is collected and used to produce output based on a seed input. While this is not a complete artificial intelligence model, it can provide any number of uses. One possible use is a learning grammar checker. Another is an intelligent autocomplete engine for mobile devices or assistive input systems. The possibities grow with the application of different input data.

In the remainder of this paper, we will discuss the process of collecting and analyzing crowdsourced Twitter data using various software tools, as well as the results of a test implementation of such an algorithm.

II. PRELIMINARIES

A. Collected Data

The data collected are comprised of a set of Twitter posts, or 'tweets,' collected over some period of time. These tweets include different languages and dialects, different sentence structures, and different intents. They also contain junk data like URLs and emoticons, plus other artifacts like misspellings.

The raw twitter was initially stored in databases and has the following structure:

TABLE I
DATA FIELDS

Column	Type
twitter_id	INTEGER
lang	TEXT
timestamp	TEXT
latitude	REAL
longitude	REAL
user_id	TEXT
user_name	TEXT
reply_of	INTEGER
message	TEXT

Of note are the 'lang' and 'reply_of' columns, which indicate language and what tweet the second was a reply to. In the initial filter, 'lang' was used to find tweets designated as English in order to focus the data set. Although not studied in this paper, reply chaining using 'reply_of' could be used to provide a conversation pattern to base replies on.

In the initial stage, various modifiers will be applied to the data in order to reformat it into a cleaner format, eliminating artifacts and leaving as little language cruft as possible. For example, the tweet "RT @SomeRandomPerson: Check out this image! http://some.random.url/3k32sdh2.png" would be rewritten as "check out this image". Stripping retweet tags, URLs, and punctuation allows the algorithm to focus on the language progression and not on the extra artifacts.

Following this preprocessing, association rule mining will be applied to the data to determine the rate of positive implication for various words and phrases. To achieve this, the proportional value of implications from one set of words to the next will be used to build a sparse correlation matrix. For each word or phrase, sets of implied words or phrases will be noted. A weighting algorithm will determine the most likely sequence based on these rules. Based on the rules, alternatives and rankings can also be determined.

III. OVERVIEW OF PROPOSED SYSTEM

A. Data Collector

The data collector component is responsible for collecting streaming Twitter data and storing it in a database. A local database file is used as concurrent access is not necessary. This simplifies the implementation of the data collector.

B. Stage 1 Preprocessor

The Stage 1 preprocessor (denoted as P_1) is responsible for extracting a subset of the dataset and cleaning it up. The actions performed in P_1, which includes validating the language of the tweets and stripping extra data, do not have to be performed offline. In a robust implementation, P_1 could be integrated into the streaming collector in order to process the data in realtime. This would allow such a system to update its state relatively easily. P_1 also performs clustering based on

*This work was not supported by any organization

[1] mail@justinmrkva.com

the language variable; everything except English is thrown away.

C. Stage 2 Preprocessor

The Stage 2 preprocessor (denoted as P_2) transforms the data into a sparse correlation matrix using association rule mining. Due to the broad scope of this kind of AI, the ruleset will be very large. In order to mitigate this, the proposed implementation of P_2 for this project will compute the highest correlation rule for each item and throw out all other rules.

This is done in order to save space; however, in an online system, this would not be a wise decision. Additional work could be done to allow online updating. Keeping all of the rules would not be efficient due to the vast amount of space necessary; however, the Misra-Gries algorithm could be used to keep a subset of rules for each source item and update it as new data arrived. Such a system, if properly designed, may be capable of dealing with the incoming volume of data, given an efficient implementation. It is likely that a truly online system would require a distributed sharded database, and a system of this magnitude is beyond the scope of this project.

D. Stage 3 Preprocessor

The Stage 3 Preprocessor (denoted as P_3) is responsible for maintaining the database that will allow fast association rule lookup. It is this database that allows the system to work in apparent realtime for end users. In this project, P_3 simply takes each rule produced by P_2 and saves it to the database, along with a likelihood constant. In a more robust system as mentioned above, this stage would implement the Misra-Gries algorithm during the save to the database. The database would need to be substantially larger. However, because this version is designed to be run offline, the Misra-Gries algorithm is not necessary, and the overhead can be avoided.

E. Data Presentation

The final component is the data presentation component. This component is a website in which a series of words can be entered; the algorithm uses the association rules to suggest likely completions to the word sequence.

IV. EXPERIMENTAL EVALUATION

A. Experimental Setup

The experiments were performed on a Mac mini server running Mac OS 10.9.2 Mavericks. A dedicated gigabit network gave the server a fast connection in order to collect the Twitter data. The server hardware is a 2.6 GHz quad-core Intel Core i7 with 16GB of RAM, and internally has two 1TB hard drives available. An additional 1.5TB RAID served as a high-speed data collection drive, although the extra speed proved to be unnecessary.

The data collection component was written in Python and saved to SQLite databases stored on the RAID and backed up periodically. The Python language was chosen for the following reasons.

- Python is very efficient to code and can be run as a daemon in order to maintain continuous data collection.
- Python has a very good 3rd party Twitter library that supports streaming.
- Python's SQLite support is extremely good and I have substantial experience with it.

This script was run as a daemon for several days. Some difficulties were encountered during data collection. In particular, the rate of disk activity rose dramatically during the first few days, which was alleviated by saving to the database in blocks of 10,000 tweets. Although this is not a permanent solution, splitting data storage into atomic chunks is a reasonable strategy for dealing with an extremely fast sequence of very small data items. In the case of solid-state drives, which have a limited number of write cycles, this strategy is advised to help extend life. A few other minor setbacks included an apparent disk write error which was due to an administrative mistake which set a disk quota on the administrator account. While it did not affect the storage RAID or the backup drive, the SQLite caches count towards the disk quota, which caused write errors. This error was mitigated temporarily but the root problem was not discovered until later.

The data stream collected accounts for 1% of the entire worldwide Twitter firehose and yielded roughly 5 million tweets per day. A total of 25GB of data was amassed during the data collection phase.

The first test dataset, after running P_1, contains a subset of 9,911,337 English tweets collected over a period of several days, consisting of approximately 101,236,475 words. The file, including metadata, measures 806MB (226MB compressed with *bzip2*). The output data is substantially smaller than the input due to filtering and because it is plain text and not a binary database.

A subsampling of tweets was taken from the dataset after running P_1 to test the replacement algorithms. The following table shows the regex substitutions made in P_1 in order to clean the data:

- `s|'https?://\[^ ]+'|''|`
- `s|'[\n\r\t]+'|' '|`
- `s|'([Rr][Tt] )?\@[^ ]+'|''|`
- `s|'[^A-Za-z0-9 ]'|''|`

This regex sequence strips URLs, replaces newlines and tabs with spaces, strips retweets and Twitter handles, and strips special characters to retain only alphanumeric characters and spaces. An additional function reduces the entire string to lowercase before saving it to a file.

To implement P_2, a dedicated Hadoop environment was configured inside a virtual machine running on VMware Fusion 6 Professional. The virtual machine was configured with 4 virtual cores and 4GB of memory, and runs CentOS 6.5 and Hadoop 2.4.0. Pig 0.12.1, Hive 0.13.0, and Mahout 0.7 and 0.9 were also installed, but were not used.

The Hadoop program was written in Java in order to take advantage of Hadoop's *bzip2* compression capabilities. By

reducing the file size by over 70%, the load on the virtual machine's disk and the file size on HDFS was minimized. The virtual machine was networked using *sshfs* so that all output files, once copied from HDFS, resided on the host machine's RAID.

For the 9.9 million tweet dataset, a complete Hadoop run of P_2 for single-word pairs (the most simple implementation) took 37 minutes and produced 179MB of data composed of 12,079,050 rules. For a three/two predictive pair, a run took 160 minutes and produced 650MB of data composed of 31,205,541 rules.

An interesting aside: during the map-reduce process, Hadoop appears capable of beginning the reduce stage before the map is complete. Although this would seem to be problematic, especially if two of the keys overlap between stages, there are several ways this might be optimized in a way that allows for an early start to the reduce stage. In the single-word pair case, the reduce stage started when the map stage was at 92% and reached 17% quickly, then stayed at 17% until the map stage was finished. Observation of processor utilization indicates that as many as two of the map stages finished early; the reason for this is unknown. After the map phase, the reduce phase finished within a couple minutes.

The final stage is P_3 in which the map-reduce output is converted into a database format. For storage, PostgreSQL on a second CentOS virtual machine was used to store the data. This virtual machine was configured with 2 processors and 4GB of RAM in order to provide high caching performance for the database. A Python script was used to perform this task. This script simply reads the file in a linear format, writes the association rules, and commits them to the database in a single transaction.

A performance bottleneck discovered during the execution of P_3 is somewhat related to the previous SQLite issues mentioned. In particular, with huge numbers of records, it is unwise to perform atomic commits for each record due to the overhead. However, the opposite is also true. Committing a massive block in one transaction can induce a severe performance penalty. To counter this, the script was modified to commit to the database once every 10,000 records. The total time required to execute P_3 was approximately 4.8 hours.

In testing, allowing the use of rules with low support values, as well as combining a subset of single-value support items (for only single word pairs) was attempted in order to test its effectiveness.

After all of this was done, a basic web interface was created that provides access to the database for predictions. It chooses the highest predictive value from the rule mining set and presents it to the user in a simple format. The site was coded in HTML5 and JavaScript. A PHP script serves as an API endpoint from which the page retrieves JSON data.

B. Experimental Results

The website may be accessed here:
`http://cse891.justinmrkva.com`.
If the site fails to load, please contact the author by email.

Once the website was completed and the PostgreSQL database was put online, the final generator was tested. A suggestion list in the form of the most likely 10 candidate completions is presented in a list. Entering the word "This" generated the sentence "This is the only thing that matters at the end of a long day" by using some of the likely candidate phrases. Using only the top results for the seed word "How do I" produces "How do I get a follow from you". Alternately, "How would I" produces "How would I know that I love you". Clearly, the system appears fully capable of producing reasonably comprehensible English sentences with only a minimum amount of input.

The system is not perfect; for example, the word "The" generates "The best thing I have ever tried to get" but then continues to form the sequence "The best thing I have ever tried to get me to go to the gym". Clearly, attempting to form longer phrases can result in strange results. In part, this is because the current level of data analysis only looks at the previous 3 words. A more robust system would look at a longer sequence of words and favor generating longer resulting phrases. For a proof of concept, however, it seems to work surprisingly well.

C. Discussion

The difficulties in analyzing language stem from its inherent complexity. The goal of an algorithm such as this one is to provide a large, robust dataset by using crowdsourced information to find patterns.

The most obvious improvement to this system would be the use of a larger span of words. In this version, only up to three words are taken into account. In a more robust system, a larger number of words would be used, and results would be weighted more strongly if they were based on more words. Additionally, this system does not deal with punctuation or termination.

Potential uses for this kind of language engine include mobile device predictive entry and grammar checkers. Although it is an expensive approach computationally, optimization may be able to make it more approachable by applying online algorithms appropriately.

V. CONCLUSION

This serves as a proof of concept that association rule mining on a crows-sourced data set can provide a relatively robust form of sentence completion and natural language analysis.