Scrubbing The Web For Association Rules, An Application In Predictive Text

Justin Lovinger, Iren Valova

Computer and Information Science Department,
University of Massachusets Dartmouth, North Dartmouth, MA, USA

Abstract

Modern smartphones have led to an explosion of interest in predictive text. Predicting the next word that a user will type saves precious time on the compact keyboards that smartphones use. By leveraging the vast amounts of text data available on the Internet, we can easily gather information on natural human writing. We can then use this data with association rules to efficiently determine the probability of one word appearing after another given word.

In this paper, we explore the gathering of text data from online social media. We also examine the use of association rules for predictive text, and develop an algorithm that can quickly and efficiently generate rules for predictive text. The results of the presented algorithm are compared to Google's Android keyboard.

Keywords association rules, web scrubbing, predictive text analysis

1 Introduction

The Internet is full of information. While some of this information is presented directly for the user, much of it is hidden between the lines, and must be extracted and processed before becoming useful. In this paper we look at the process of gathering and extracting word associations from the Internet. With the association rules (AR) data mining technique, the vast quantities of text that exists on the web can be harnessed to create an extensive database of word associations. With such a database, the process of predicting words becomes trivial.

Simply put, AR is a technique that takes a dataset of items and discovers associations between them [1]. Association rules have been successfully utilized in many areas of everyday life, including healthcare [2], link analysis, bank fraud, etc. As our goal is to discover associations between words, AR is a natural fit. AR is a well studied and optimized process, which allows for efficient parsing of the large amounts of data that can be gathered from the web. Typically, the association rule mining involves two steps: frequent patterns discovery and extraction of interesting information. The first step in the more traditional algorithms, such as Apriori, however, eliminates the order of items arriving for processing. This poses an obvious problem as our goal is to predict the next word, e.g. "are" in "How are you?". There have been recent modifications to the traditional association mining algorithms. While the changes to the frequent pattern in [3] are more drastic than we need, the authors produce an effective algorithm for the purpose. In this work, we modify the traditional Apriori algorithm to fit the needs of the predictive text mining. Therefore, we need rules discovering

as many words as possible, but the order of the word presentation also must be kept into account. The algorithm is presented in Section 4.

Predictive text takes words that have already been typed, and predicts the next word or words that a user will type. Predictive text can also refer to the prediction of one word, given a partial sequence of characters, but this paper focuses on next word prediction. In the past, predictive text was primarily used with languages that have large character sets [4, 5]. Today, the prevalence of smartphones, and their small digital keyboards, has made predictive text a very important aspect of typing in any language.

As with many predictive systems, large quantities of data are necessary for a predictive text system to handle the variety of words and phrases that people use. In this paper, we leverage the internet to gather this data, and use AR to process it efficiently.

2 Ease of Use

The Internet contains 3.3 billion pages. Each page filled with countless words. It is, without a doubt, the largest source of text in the world. Additionally, the Internet is always changing as fads and language change. It is not only vast, but also continually up to date. These factors, as well as many others, have made the Internet a popular target for data mining [6, 7, 8]. It is a perfect resource for discovering word associations.

In order to harness this resource, we utilize the popular website Twitter. Twitter allows users to send brief public messages, and they provide an API that allows a third party to easily plug into this stream of messages. Twitter's 255 million active users send 500 million messages daily, which allows a third party to gather massive amounts of text in minutes. Additionally, because Twitter is largely used by average people, it allows us to gather word data that is unbiased by professionalism. This is an advantage when trying to predict the words that average people will use.

Although the fast paced nature of the Internet allows for rapid data gathering, it can also lead to some odd associations being formed. While generating association rules from a live stream of Twitter messages, we discovered some odd and out of place word associations. One particularly memorable instance is the inclusion of the word "catfish" in a very large number of word associations. A small amount of research discovered a TV series named "Catfish" that was airing at the same time we were gathering our data. This lead to us unwittingly creating associations with a short trend. To minimize this effect, we gathered data in many small intervals over longer periods of time, rather than all at once. This causes small, individual, trends to be diluted by the large amounts of text that does not follow the trend, but severely slows down the data gathering process.

Another oddity we discovered is the local dialect that some internet communities have, and the ways that this lingo can find its way into text data. Since Twitter has a 140 character limit for all messages, many users use abbreviations and short words. Therefore, our text data from Twitter largely lacked longer, more sophisticated words, and contained a large number of abbreviations. Additionally, the brief messages that Twitter caters to encourages a lack of consideration when submitting messages, and leads to poor grammar and spelling in many messages.

To avoid the unique complications that data mining a social media site such as Twitter presents, we instead explore the website Blogger.com. Blogger is a traditional blog website that hosts a vast variety of personal webpages full of word data. By scrubbing blog posts, we still gather word data from average people, while avoiding time sensitive associations and the restrictions of a 140 character limit. Our Blogger scrubber starts with Blogger's featured blogs page. For each of these featured blogs, we extract the text in every post of every page of the blog. This process can continue virtually indefinitely, due to the huge number of featured blogs that Blogger has catalogued, and

because we are not concerned with time sensitive associations, we can gather data as quickly as possible.

3 Processing the Text

Before association rules can be created, our stream of text must be converted into a format that makes sense for AR; a process commonly known as text refining [9]. This involved standardizing the text, and converting it into word pairs. Text standardization allows us to ignore irrelevant differences between words, such as capitalization and the use of apostrophes, and eliminate text that does not form words. This is done by making all text lowercase, and stripping numbers and special characters other than periods, commas, question marks, and exclamation marks. The result is a string that consists of individual words, end of line punctuation, and nothing more. With text standardized, we can now convert it into the word pairs that will make up our dataset. Every set of two adjacent words in our string forms a word pair. For example, given the string, "Alpha Beta Charlie", our word pairs are, [Alpha, Beta] and [Beta, Charlie]. The result is a dataset where every row is a set of two adjacent words. Words after a new line are considered part of a new set of text. This prevents associations from the end of one paragraph and the beginning of the next.

4 The Algorithm

The popular Apriori algorithm [10] is utilized widely for extracting frequent itemsets from transactions gathered into a database. Frequent itemsets are items that occur frequently enough together among the transactions, therefore, judging them to be creating a possible pattern. The frequency is measured with the support level as determined by the user. The support is formalized as

$$\textit{Support}(\text{coverage}) \text{ of } A \Rightarrow B : P(A, B) = \\ (\#\text{ of transactions containing both } A \text{ and } B)/(\text{total } \#\text{ of transactions})$$

The next step in the association rule mining process is the generation of interesting rules based on the frequent itemsets. There are a number of parameters associated with the process one can use to tune the algorithm's focus on various elements. Next, comes the need to generate rules based on the discovered itemsets. Before analyzing the rules, they are judged based on probability measures (usually) for fitness with the problem being studied. This measure is defined as confidence:

$$\textit{Confidence}(\text{accuracy}) \text{ of } A \Rightarrow B : P(B|A) = \\ (\#\text{ of transactions containing both } A \text{ and } B)/(\#\text{ of transactions containing } A)$$

Although widely used and very tempting, adopting the Arpiori algorithm results in rules that are difficult to use for predictive text. This is because the Apriori algorithm does not respect the order of items in itemsets. For example, given the itemsets $\{A, B, C\}$, $\{A, B\}$, and $\{A, C\}$, the Apriori algorithm will likely produce the rule $B \rightarrow A$. However, A is never actually followed by B in our example. To be useful for predictive text, we want the rule $A \rightarrow B$, but not $B \rightarrow A$, because B is often followed by A, but not vice versa.

Additionally, although AR is generally about discovering a small number of highly interesting associations in a large dataset, in order to properly perform predictive text, we must have rules to cover as many words as possible. Otherwise, without a word in our rules database, we will not be able to offer a suggestion. Without needing to cull uncommon items, the concept of a support

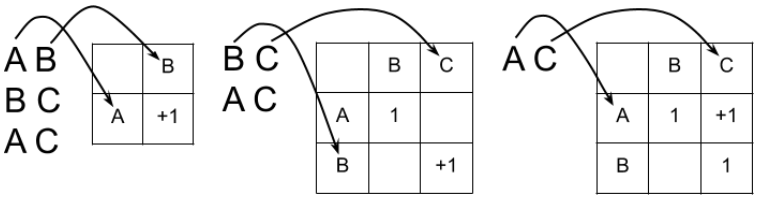

Figure 1: Generating the Hash Table

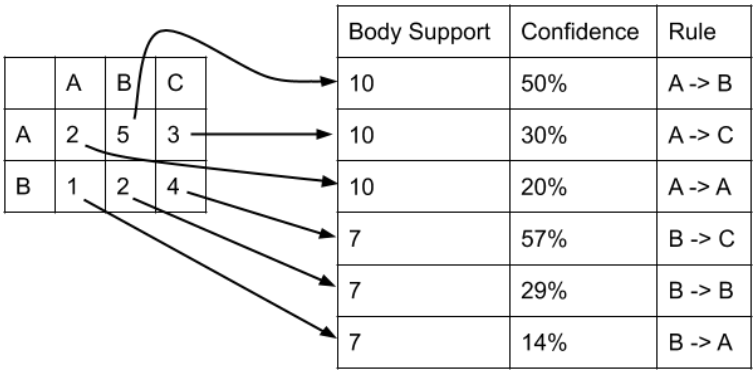

Figure 2: Generating the Rules

threshold is unnecessary for us. Additionally, it is better to have low confidence rules to cover some words, than no predictions at all following the word. Therefore, the traditional value-based confidence threshold is less useful than a truncation method, which can give us a constant number of rules for every word in a rules body. To this end, we have developed a simple AR algorithm that respects the order of items, has no support threshold, generates a maximum number of rules for every body item, and is optimized for datasets with two items in every transaction.

Our algorithm utilizes a two phase strategy. During the first phase a single pass of the dataset is performed. During this pass, a two dimensional hash table is generated as illustrated in Figure 1. Hash tables have found frequent use in AR [10, 11], and this structure is proven especially effective for two-item itemsets [12]. The first word in a transaction (base word) is the table row. The second word in a transaction (next word), is the table column. Each cell contains the number of times a next word occurs for a given base word, or the support. This value can easily be converted into the rules confidence. During the second phase, a single pass of the hash table is performed, one row at a time. This allows us to reorganize the hash table into a list of rules shown in Figure 2. Each rule consists of the table row as the body, and the table column as the head. The support for each rules is obtained from the value stored in the cell. The confidence for each rule can optionally be calculated after all rules for a given base word are computed, by dividing the support for the rule by the total support for a base word. Before the rules for a given base word are added to the rule database, they are truncated by taking the n rules with the greatest support, and discarding the rest. For our tests, n is 3. The result is a database of rules where every base word has between one and three rules, and each of these rules have the greatest confidence, given the base word.

Since word datasets can be arbitrarily large, and predictive text benefits from large amounts of data, memory usage is our primary concern when generating AR for this purpose. To this end our algorithm can easily be modified to handle arbitrarily large datasets. First the word dataset must be externally sorted. External sorting allows for a file to be sorted without storing the entire file in memory. Now that the dataset is sorted, it is read into memory one base word at a time.

Since a sorted dataset will have each base word stored in a contiguous block, we can guarantee that all instances of a base word are in memory by scanning the dataset until another base word is encountered. Our algorithm is performed on this one base word, generating a hash table with a single row, and the resulting rules are written to a file. This process is repeated until the end of the dataset is encountered.

4.1 Performance Characteristics

Given a dataset with n transactions, our algorithm requires $2n$ passes if every word pair is unique. This is the worst case scenario; the first pass is used to generate the hash table, and the second pass iterates over a hash table of the same size of the dataset. Given a best case scenario where every word pair is the same, our algorithm requires n passes, because our hash table will have only 1 cell. A typical dataset should lie somewhere in the middle. Regarding space complexity, our hash table requires one cell for every unique word pair in the dataset. As such, the memory usage of our algorithm scales with the number of unique word pairs in the dataset, up to a worst case of one cell for every word pair. However, if the algorithm is configured for large datasets as outlined, the limiting factor will be the first word in the pair (base word) with the largest number of unique second words (next words). This is because the hash table can consist of only a single row, which is then purged from memory, and replaced.

4.2 Using the Rules

With our database of rules fully created, we can now use these rules to predict words based on words already typed. To do this, we take the last word a user has typed, and fetch from our database the rules with the same base word. This gives us a list of rules with a potential next word as the head of each rule. In order to provide accurate predictions, these rules can be sorted by confidence, with the most confident rules being the most likely. Since confidence is a measure of how often a particular rule occurs, given the body of the rule, it is an accurate estimation of the probability of one word given another. To this end, a data structure consisting of a hash table that maps the base word to a list of rules, which are sorted in descending order of confidence, is effective for fast lookup and extraction of rules.

5 Results

Our blog scrubber is used to gather 120mb of text. This text is then processed into 230mb of word pairs. Finally, our association rules algorithm is used to generate 18mb of rules. A small subset of these rules is presented in Table 1.

For comparative purposes, we pitch our own association rules to next word predictions in Google's default Android keyboard, as of Android version 4.4.4. Our predictions are extracted from our AR by examining the next words for a given base word, and ordering them by confidence. Google's predictions are extracted by typing the base word with Android's default keyboard, and examining the three next word predictions. For the Android tests, no words occur before the base word, and personalized predictions are disabled. The results can be seen in Table 2.

These 10 samples are generated with our base words with the greatest support. This comparison shows that, of these 10 samples, our first predictions matches Google's first prediction 80% of the time, and our second prediction matches Google's second prediction 40% of the time. As such, with adequate data, our predictive method is comparable to Google's predictive text implementation.

Table 1: Rules

Body Support	Confidence %	Rule		
1,005,145	12.52	.	→	.
1,005,145	7.08	.	→	i
1,005,145	4.90	.	→	the
883,127	8.57	,	→	and
883,127	4.79	,	→	i
883,127	4.06	,	→	but
849,481	1.04	the	→	first
849,481	0.93	the	→	same
849,481	0.91	the	→	most
511,815	4.80	and	→	the
511,815	4.29	and	→	i
511,815	2.62	and	→	a
473,110	3.44	a	→	few
473,110	2.62	a	→	little
473,110	1.74	a	→	lot

Table 2: Comparison of Predictive Text

Base Word	Our Predictions			Google's Predictions		
	First	Second	Third	First	Second	Third
the	first	same	most	new	other	way
and	the	i	a	i	the	then
a	few	little	lot	few	bit	lot
to	the	be	make	the	be	see
of	the	a	my	the	this	a
i	am	was	have	am	have	can
in	the	a	my	the	a	this
for	the	a	me	the	your	a
is	a	the	that	a	the	not
that	i	the	is	i	the	is

6 Conclusions

In this paper, we have explored methods of gathering text data from social media for the purposes of predicting the next word a blogger is likely to type. In order the develop a predictive system, we propose a modified Apriori algorithm to convert the text into rules. Finally, we compare the resuts from the proposed system to the predictive text in Android's default keyboard.

The Apriori algorithm modifications are necessitated by the nature of our approach, which eliminates the need for calculating support by utilizing hash tables, and scales well for large database applications (which covers virtually all blogs and other forms of social media). While, at a first glance, the memory requirements of the proposed algorithm may seem demanding, we have found a solution to that problem as well.

In short, the proposed algorithm is very fast, runs in linear time, and can be configured to use very little memory. By comparing our resulting rules with the ones generated by the predictions that Google provides in their Android keyboard we validate the excellent performance of the proposed approach.

References

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, 1993.

[2] Jesmin Nahar, Tasadduq Imam, Kevin S Tickle, and Yi-Ping Phoebe Chen. Association rule mining to detect factors which contribute to heart disease in males and females. *Expert Systems with Applications*, 40(4):1086–1093, 2013.

[3] Ansel Y Rodríguez-González, José Fco Martínez-Trinidad, Jesús A Carrasco-Ochoa, and José Ruiz-Shulcloper. Mining frequent patterns and association rules using similarities. *Expert Systems with Applications*, 40(17):6823–6836, 2013.

[4] Manoj Kumar Sharma, Sayan Sarcar, Pradipta Kumar Saha, and Debasis Samanta. Visual clue: an approach to predict and highlight next character. In *2012 4th International Conference on Intelligent Human Computer Interaction (IHCI)*, pages 1–7. IEEE, 2012.

[5] Phavy Ouk, Ye Kyaw Thu, Mitsuji Matsumoto, and Yoshiyori Urano. A word-based predictive text entry method for khmer language. In *2008 IEEE International Conference on Information Reuse and Integration*, pages 214–219. IEEE, 2008.

[6] Kshitija Pol, Nita Patil, Shreya Patankar, and Chhaya Das. A survey on web content mining and extraction of structured and semistructured data. In *2008 First International Conference on Emerging Trends in Engineering and Technology*, pages 543–546. IEEE, 2008.

[7] Jaroslav Pokorny and Jozef Smizansky. Page content rank: an approach to the web content mining. In *Proceedings of the IADIS International Conference on Applied Computing*, volume 2, pages 22–25, 2005.

[8] Tao Jiang, Ah-Hwee Tan, and Ke Wang. Mining generalized associations of semantic relations from textual web content. *IEEE transactions on knowledge and data engineering*, 19(2):164–179, 2006.

[9] Ah-Hwee Tan et al. Text mining: The state of the art and the challenges. In *Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases*, volume 8, pages 65–70. sn, 1999.

[10] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[11] Fengjun Han and Nini Rao. Mining co-regulated genes using association rules combined with hash-tree and genetic algorithms. In *2007 International Conference on Communications, Circuits and Systems*, pages 858–862. IEEE, 2007.

[12] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu. Using a hash-based method with transaction trimming for mining association rules. *IEEE transactions on knowledge and data engineering*, 9(5):813–825, 1997.