HARVARD | BUSINESS | SCHOOL

9-118-015
REV: DECEMBER 12, 2018

SRIKANT M. DATAR
CAITLIN N. BOWLER

Tamarin App: Natural Language Processing

Martha Jay worked in sales for a software company specializing in financial reporting software, FinRep. As part of its strategic growth plan, FinRep had recently acquired a small company that developed a phone app for online marketers: Tamarin SEO (Search Engine Optimization). The Tamarin app served up metrics on website performance and sales to a phone specific dash. Access to metrics on page rank, web traffic, conversion, sales, and retention rates allowed sales managers to maintain close connections to the online marketing team while out on sales visits. Jay was eager to shift roles within the company, and prepared to interview with the Tamarin SEO marketing division. The hiring manager advised her to be prepared to discuss the fundamentals of the product, as well as some of the challenges it presented to the marketing team.

Marketing professionals warmed to Tamarin SEO almost immediately, garnering good sales and excellent name recognition. Jay's company loved the product and had quickly acquired it, but the internal marketing team now had to deal with the annoying – but real – challenge the word "tamarin" posed for assessing sentiment when they monitored social media for references to the company. The original founders believed the nimble, canopy-dwelling South American tamarin symbolized the energized mobility their app enabled in users. The imagery differentiated Tamarin SEO from its competitors' generic brands, but a significant number of people referred to "tamarins" in non-app related tweets. The marketing team could pull in as many as 2,000 tweets with references to tamarins in a week, and only a small slice of those would be in reference to the app. The team used a basic natural language processing (NLP) technique to sort out those tweets that were about the Tamarin app from those that were about other references to tamarins, including the animal and a band; then they could focus on sentiment.

Jay set out to learn the basics of NLP and filtering. The Tamarin app development team was planning to incorporate sentiment analysis into the app's future functionality; NLP would power this feature. More immediately, she would understand how the marketing team distinguished between app and non-app related tweets.

Natural Language Processing: A Conceptual Overview

NLP was a branch of computer science concerned with enabling computers to derive meaning from human ("natural") language input. Although computers carried out instructions communicated

Professor Srikant M. Datar and Research Associate Caitlin N. Bowler prepared this case. This case is not based on a single individual or company but is a composite based on the author's general knowledge and experience. Funding for the development of this case was provided by Harvard Business School. HBS cases are developed solely as the basis for class discussion. Cases are not intended to serve as endorsements, sources of primary data, or illustrations of effective or ineffective management.

through constructed languages (C++, Basic, etc.), they struggled with the idiosyncrasies of human languages.

The first applications of NLP were in the 60s: ELIZA was a simulation of a psychotherapist, written by Joseph Weizenbaum. To the input "My head hurts," ELIZA might have provided a simple response, for example, "Why do you say your head hurts?" Applications of NLP were now everywhere: (1) Google Mail automatically removed your spam, (2) Google Translate could translate text from a variety of languages, and (3) many different retailers now incorporated chat bots into their online websites. More recently, voice recognition software had been widely deployed in phone and other consumer electronics.

The Bag of Words Approach

From the point of view of the computer, words had no inherent recognizable meaning. Rather, the computer viewed words as *strings* of letters arranged in a particular sequence.[a] For a classification exercise, such as spam filtering, sentences could be conceptualized as a *bag of words* where the computer treated pages of text as a collection of unordered words; words in turn were called "tokens." Thus to prepare text for processing, the programmer stripped away all concept of object, subject, adjective etc., as well as sentence structure. She just cared about the actual words. For instance, consider the two "bags of words" below that are completely equivalent from the computer's point of view and would be treated exactly the same:

- People's Republic of Cambridge
- of Republic Cambridge People's

These bags of words still retained two characteristics of a human language: (1) capitalization and (2) punctuation. To prepare the text, the programmer eliminated capital letters because they did not change the meaning of most words in a significant way. She eliminated punctuation, everything from periods and apostrophes to commas, semi-colons, etc., because the tokens (e.g. People's) had to be fully isolated to be recognized as "the same" across hundreds of tweets no matter whether the token was at the end of a sentence or not. Here are the previous bags of words processed this way.

- peoples republic of cambridge
- of republic cambridge peoples

Overview of a Bayesian Classifier

Bayes' Rule

Reverend Thomas Bayes articulated this eponymous rule almost 250 years ago. He might be surprised at the outsized role it plays in spam filtering. The rule is thus:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)},$$

where A and B are events and $P(B) \neq 0$.

[a] A *string* is any finite sequence of characters—i.e., letters, numerals, symbols and punctuation marks. The length of a string is often an important characteristic.

- $P(A)$ and $P(B)$ are the probabilities of observing A and B without regard to each other.
- $P(A \mid B)$, a conditional probability, is the probability of observing event A given that B is true.
- $P(B \mid A)$ is the probability of observing event B given that A is true.

Bayes' rule is a key pillar in one of two major schools of "probability interpretation." More casual students of statistics are likely familiar with the *frequentist* school of interpretation, which views probability measures as a "proportion of outcomes" for real, physical processes. In such processes (systems), such as rolling dice, a given type of event tends to occur at a persistent rate, or "relative frequency," over many trials. Members of this school practice evidence- or outcome-based statistics.

In contrast, the *Bayesian* school of interpretation views probability as a measure of the "degree of belief" one has about a proposition—before *and after* accounting for available evidence. As one accumulates more evidence about a proposition, one can update the probability measures accordingly. One application is that programmers can write surprisingly effective algorithms for classifying discrete pieces of text when the absolute frequency of texts within separate classes out in the world is not actually known.

The Approach in Practice

To dig in, Jay considered a set of 300 tweets that all contained the word "tamarin." (Of the tweets, 150 were app-related and 150 were not.) The objective would be to build a model that could determine whether a new tweet belonged to the set "app" or the set "not-app." As with all classifiers, Jay needed a training set for each of the two classes. To assemble the training sets, Jay sub-divided that set into two sets: one contained tweets that refer to the app (set "app"); the other contained tweets that refer to something else (set "not-app").

The algorithm would use Bayes' rule to assign class labels (e.g. "app," "not-app") to new observations (tweets) from outside the training set, based on the class labels from these two subsets of tweets. The key to the approach was that any Bayes classifier assumes that the value of a particular token (word) was independent of the value of any other token, given the class variables (e.g. "app," "not-app").

Consider a simple example. Suppose Jay was trying to determine if the word apple referred to a fruit or not. A fruit would be considered an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considered each of these features—color, shape, size—to contribute *independently* to the probability that the word apple referred to a fruit (or not), regardless of any possible correlations between the color, roundness, and diameter features.

Jay began thinking about how to apply Bayes' rule to the Tamarin app problem. The algorithm would need to look at the two subsets of tweets to determine the probabilities of a tweet being either about the app or about something else, given the words/tokens in the two sets of tweets. These two probabilities for the training sets were defined as:

$$p(\text{app} \mid \text{word 1, word 2, word 3, ...})$$

$$p(\text{other} \mid \text{word 1, word 2, word 3, ...})$$

A naive Bayes model classifies a new tweet based on which of these two classes is most likely given the words. In other words, Jay would conclude that the tweet was about the Tamarin app, if:

$$p(\text{app} \mid \text{word 1, word 2, word 3, ...}) > p(\text{other} \mid \text{word 1, word 2, word 3, ...})$$

This decision rule, which picked the class that was most likely given the words, was called the *maximum a posteriori rule (MAP rule*).[b]

Building a Classifier

Jay set out to build her own simple classifier to ensure that she really understood the mechanics of this process.

Preparing the data

Jay listed out the steps she would take to transform the discrete tweets in her excel workbook into the two *bags of words* she would use to generate probabilities for the two training sets.

1. Create two training sets. Assign each of 300 tweets to one of two bags of words: app or not-app (150 tweets in each). *Human judgement was essential to the construction of these two training sets.*
2. Transform tweets into tokens.
    a. Scrub tweets of capitalization and punctuations
    b. For each dictionary, identify every unique token (word)
    c. Filter out *stop words*. Every language had what were termed "stop words," or words with very little "lexical content." Examples of stop words might be "instead" or "anyway." In English many of the stop words were short—"a," "and," "the" were all three letters or less. To build out the probability table she would have to drop these words, because they appeared so frequently in both sets that their inclusion would only add noise.
    d. She would then have the final list of tokens for the app-related set—the *bag of words*.

Jay built a table to keep track of the calculations for the next steps of the process. See **Table 1**.

3. Count token frequencies.
    a. List each token that appeared in the bag of words (Column a).
    b. Count the frequency of each token from within the bag of words (Column b).
    c. Apply *additive smoothing* to the frequency column. Additive smoothing was a technique to mitigate the impacts of $p = 0$ when calculating conditional probability. Specifically, she would use it to deal with rare words. The technique required her to add an additional "1" to the frequency count for each token (Column c).[c]

[b] See Forest Provost and Tom Fawcett, *Data science for business* (Sebastopol: O'Reilly, 2013): 234-239 for a thorough explanation of the probability theory underlying the Naive Bayes model.

[c] Rare words cause problems when calculating probabilities. Consider a tweet about Tamarin SEO that included the word "olfactory." That would be a rare occurrence, and so it is possible that based on past data in the training set, one or both classes may not have seen this word—that is, "olfactory" would not be in the bag of words. This happens a lot on Twitter in regards to shortened URLs. Every new tweet of a URL might have a different, never-seen-before encoding. In the case of "olfactory," we would assume:

$$p(\text{"olfactory"} \mid \text{app}) = 0; \text{ which leads to: } p(\text{"olfactory"} \mid \text{app}) * p(\text{word} \mid \text{app}) * p(\text{word} \mid \text{app}) \ldots = 0$$

Olfactory effectively "zeros out" the entire probability calculation. To deal with this we assume that we have seen "olfactory" before; we give it a value of "1." We do that for all rare words. But if we do it to the rare words, that gives them the same

4. Calculate *P*(Token | App). This was the frequency count with additive smoothing (column c), divided by the total number of tokens (with additive smoothing) in the bag of words.

5. Calculate Natural Log (*P*). Take the natural log of the probability *ln(P)*:
   ln(0.000829876) = -7.094234846. Calculating natural logs would make it easier to calculate joint probabilities later in the process and avoid dealing with very small numbers. Jay did these calculations for all words in the tweets about Tamarin Apps.

Table 1 Token Counts, Frequencies, and Probabilities—App-related Tweets

Token #	Token	Frequency Count	With Additive Smoothing	Probability *P* (Token | App)	Natural Log (probability)
	(a)	(b)	(c) = (b) + 1	(d) = (c) ÷ 2,410	(e) = ln(d)
1	friday	1	2	0.000829876	-7.09423
2	#drupal	3	4	0.001659751	-6.40109
3	@tamarinapp	13	14	0.005809129	-5.14832
4	bounce	6	7	0.002904564	-5.84147
5	changes	1	2	0.000829876	-7.09423
6	data	2	3	0.001244813	-6.68877
…	…	…	…	…	…
826	zapier	1	2	0.000829876	-7.09423
		Grand Total	**2,410**		

Source: Casewriter.

To get a better handle on the distribution of token frequencies and the probabilities within the bag of words, Jay created a summary table. Jay repeated this entire process for the *non-app* tweets to produce the probabilities of each of those tokens in the *non-app* training set.

A Small Test Run

Jay worked through a small test run of the process.

- In **Step 1** she went through the full process to calculate the log probabilities for each token in each small bag of words (step 1a, app; step 1b, non-app).
- In **Step 2** she calculated the log probabilities for a *new tweet*, using both the app token probabilities and the non-app token probabilities.
- Finally, in **Step 3**, she summed the probabilities and classified the tweet based on which total probability was higher—app or not-app.

frequency as words we have seen, if only once. So, we add "1" to those words, giving them a frequency of "2." To side step this problem, which would ripple across all frequencies, we add "1" to the frequency of *every* token in the bag of words. This technique is called *additive smoothing*. Source: John Foreman, *Data Smart: Using Data Science to Transform Information into Insight*, (Indianapolis, IN : John Wiley & Sons, 2014).

Step 1a: *Calculate log probabilities for tokens in the app-related bag of words.*

She considered two app-related tweets:

App-Related Tweets

→ just love tamarin monitoring service
→ just need help with tamarin password reset

Table 2 that follows presents the tokens for the two app-related tweets above. It provides the count for each token (column b), adds 1 to account for rare words (column c), calculates basic probability based on the total token count (column d), and then calculates the natural log (probability) for each token (column e). Step 2 will draw on the values in the natural log (probability) column.

Table 2 Probabilities of App-Related Tweets

Token #	**Token**	**Frequency Count**	**With Additive Smoothing**	**Probability** *P* (Token | App)		**Natural Log** (probability)
	(a)	**(b)**	**(c) = (b) +1**	**(d) = (c) ÷ 22**		(e) = ln(d)
1	just	2	3	3/22	0.136364	-1.99243
2	love	1	2	2/22	0.090909	-2.39790
3	tamarin	2	3	3/22	0.136364	-1.99243
4	monitoring	1	2	2/22	0.090909	-2.39790
5	service	1	2	2/22	0.090909	-2.39790
6	need	1	2	2/22	0.090909	-2.39790
7	help	1	2	2/22	0.090909	-2.39790
8	with	1	2	2/22	0.090909	-2.39790
9	password	1	2	2/22	0.090909	-2.39790
10	reset	1	2	2/22	0.090909	-2.39790
	Total	**12**	**22**			
All other words		0	1	1/22	0.045455	-3.09104

Source: Casewriter.

Note that the row "all other words" stands for words that will show up in the test set, but which are not present in the training set.

Step 1b: *Calculate log probabilities for tokens in the non-app-related bag of words.*

Other Tweets

→ spark tamarin acapella group very good
→ tamarin mexican pride

Table 3 that follows considers the tokens for the two non-app tweets above.

Table 3 Probabilities of Non-App-Related Tweets

Token #	**Token**	**Frequency Count**	**With Additive Smoothing**	**Probability** *P* (Token | App)		**Natural Log** (probability)
	(a)	**(b)**	**(c) = (b) + 1**	**(d) = (c) ÷ 2,410**		(e)= ln(d)
1	spark	1	2	2/17	0.117647	-2.14007
2	tamarin	2	3	3/17	0.176471	-1.73460
3	acapella	1	2	2/17	0.117647	-2.14007
4	group	1	2	2/17	0.117647	-2.14007
5	very	1	2	2/17	0.117647	-2.14007
6	good	1	2	2/17	0.117647	-2.14007
7	mexican	1	2	2/17	0.117647	-2.14007
8	pride	1	2	2/17	0.117647	-2.14007
	Total	**9**	**17**			
All other words		0	1	1/17	0.058824	-2.83321

Source: Casewriter.

Step 2: *Calculate the overall probabilities for the full tweet.*

Jay then looked at a test tweet. See **Table 4** for the probabilities of each word.

Test Tweet

→ tamarin monitoring help with password good

Table 4 Comparing Probabilities of Test Tweet

Token #	**Token**	**Frequency Count**	**App Token** ln(probability) x # of Times word occurs	**Other Token** ln(probability) x # of Times word occurs
1	tamarin	1	-1.99243	-1.73460
2	monitoring	1	-2.39790	-2.83321
3	help	1	-2.39790	-2.83321
4	with	1	-2.39790	-2.83321
5	password	1	-2.39790	-2.83321
6	good	1	-3.09104	-2.14007
Total			**-14.67507**	**-15.20751**

Source: Casewriter.

Jay relied on Bayes' rule to add up the log probabilities for each token in the tweet, to determine the overall probability that the tweet was (a) about the app or (b) not about the app. Jay worked through the logic and assumptions by which this comparison was made.

Using Bayes' rule,

$$p(\text{A} \mid \text{B}) = p\,(\text{A}) \times p(\text{B} \mid \text{A}) \,/\, p(\text{B}) \qquad \text{or} \qquad p(A|B) = \frac{P(B|A)\times P(A)}{P(B)}$$

$$p(\text{app} \mid \text{word 1, word 2,…}) = p(\text{app}) \times p(\text{word 1, word 2, …} \mid \text{app}) \,/\, p(\text{word 1, word 2, ……})$$

Similarly,

$$p(\text{not-app} \mid \text{word 1, word 2,…}) = p(\text{not-app}) \times p(\text{word 1, word 2, …} \mid \text{not-app}) \,/\, p(\text{word 1, word 2, ……})$$

In the test sample (and in making predictions), Jay was interested in comparing

$$p(\text{ app} \mid \text{word 1, word 2, ...}) \quad \text{and} \quad p(\text{not-app} \mid \text{word 1 , word 2 , ...})$$

The denominator on the right-hand side was exactly the same and cancels out so Jay only had to compare:

$$p(\text{app}) \times p(\text{word 1, word 2, …} \mid \text{app}) \quad \text{and} \quad p(\text{not-app}) \times p(\text{word 1, word 2, …} \mid \text{not-app})$$

Assuming that the probabilities of these words being in the tweet were independent of each other,

$$p(\text{word 1, word 2, …} \mid \text{app}) = p(\text{word 1} \mid \text{app}) \times p(\text{word 2} \mid \text{app}) \times p(\text{word 3} \mid \text{app}) \times \ldots\ldots$$

and

$$p(\text{word 1, word 2, …} \mid \text{not-app}) = p(\text{word 1} \mid \text{not-app}) \times p(\text{word 2} \mid \text{not-app}) \times p(\text{word 3} \mid \text{not-app}) \times \ldots\ldots$$

Jay wondered if this was a reasonable assumption because words were not independent of one another in a document. Once one saw the first few words in a sentence, one could guess what some of the next words might be. At the same time, Jay recognized that it was not the absolute value of this calculation that she cared about. Her only interest was in comparing the probability calculations for the app and not-app cases. She wondered if it was fine to make this assumption too while doing the comparison. Jay now had to compare:

$$p(\text{app}) \times p(\text{word 1} \mid \text{app}) \times p(\text{word 2} \mid \text{app}) \times p(\text{word 3} \mid \text{app}) \times \ldots\ldots \quad \text{and}$$

$$p(\text{not-app}) \times p(\text{word 1} \mid \text{not-app}) \times p(\text{word 2} \mid \text{not-app}) \times p(\text{word 3} \mid \text{not-app}) \times \ldots\ldots$$

Over time, if the probability of a tweet about the app and not-app was about the same, Jay had to compare the following two terms:

$$p(\text{word 1} \mid \text{app}) \times p(\text{word 2} \mid \text{app}) \times \ldots\ldots \quad \text{and} \quad p(\text{word 1} \mid \text{not-app}) \times p(\text{word 2} \mid \text{not-app}) \times \ldots\ldots$$

Taking logs in order to get to numbers that were easier to calculate, this equaled:

$$ln\ p(\text{word 1} \mid \text{app}) + ln\ p(\text{word 2} \mid \text{app}) + ...$$

Similar calculations for the other (non-app) tweets gave:

$$ln\ p(\text{word 1} \mid \text{not-app}) + ln\ p(\text{word 2} \mid \text{not-app}) + ...$$

Jay considered each tweet in the test sample. For the first tweet she looked at all words in the tweet. Using the logs of the probabilities for each of these words calculated in **Table 2**, she calculated:

$$ln\ p(\text{word 1} \mid \text{app}) + ln\ p(\text{word 2} \mid \text{app}) + ...$$

Similarly, using the logs of the probabilities for each word in the first tweet **Table 3**, she calculated:

$$ln\ p(\text{word 1} \mid \text{not-app}) + ln\ p(\text{word 2} \mid \text{not-app}) + ...$$

She then summed the probabilities for app and not-app.

Step 3: *Classifying a new tweet*

Test Tweet

→ tamarin monitoring help with password good

The *ln* probability for the app ($\mathbf{-14.67507}$) was greater than the *ln* probability for not-app ($\mathbf{-15.20751}$), so she classified the test tweet as a tweet about the app.

Seeing how the naïve Bayes analysis proceeded, Jay felt a little unclear about why the analysis was done in the way it was. Why did the analysis go from $p(\text{app} \mid \text{word 1, word 2, word 3……..})$ to flipping things around to $p(\text{word 1} \mid \text{app})$, $p(\text{word 2} \mid \text{app})$, ……? Was it not possible to do a similar analysis directly to determine the $p(\text{app} \mid \text{word 1, word 2, word 3……..})$?

Sentiment Analysis

Jay could now describe how the marketing department's app/non-app classifier worked. However, she was well aware that this was really just a first step toward the department's real objective: monitoring the quality and tone of comments users and reviewers were making about Tamarin SEO.

Given the volume of tweets, the marketing team wanted to classify the Tamarin app tweets even further according to some set of meaningful dimensions. She did not yet have the marketing team's level of domain knowledge, but she could imagine some of the issues they would want to flag within the tweets. This was an important challenge and one that she would be excited to dig into if she were hired to join the team.

Jay decided to review some of the original tweets in the app training set. From this set she wanted to get a sense of the range of tweets related to the Tamarin app that were appearing in the Twitter-sphere and what distinguished them from each other. For example, complaints versus questions—how were they different in terms of word choice, tone, etc. See **Exhibit 1** for the collection of app-related tweets Jay reviewed and **Exhibit 2** for the collection of non-app related tweets she reviewed. Would it be possible for her to categorize them herself in a meaningful and actionable way?

She did a review and identified several patterns in the tweets that she thought would be meaningful to the marketing team.

A brief foray into the literature suggested to Jay that there were similarities between the filtering exercise she had just completed and the basics of sentiment analysis. For instance, sentiment analysis used a more sophisticated version of the bag of words called a *lexicon*. A lexicon contained entries with

data about words and word *stems*[d] and was much more sophisticated than the simple frequency table from the filtering example. For each word, the lexicon might contain data about parts of speech, spelling variants, inflectional variants, and basic syntax. Most notably, lexicons assembled by human researchers were characterized as gold standard.[1]

The other component that distinguished the lexicon from a simple bag of words was the inclusion of a *semantic orientation score*. The paper she read described *polarity-based* scores, which simply meant that based on a scale from 1 to 9 each word could fall into one of three value categories: positive, negative or neutral. Using this kind of system, for each word in the lexicon a researcher would assign a score from 1 to 9, where < 5 was negative, 5 was neutral, and > 5 was positive. If a classifier used this lexicon, for each chunk of text it analyzed, it would reference the lexicon to draw the assigned scores for each word in the chunk. It would then compute the average to assign an overall score to that chunk of text. Someone would then face the task of interpreting that score.[2]

With this new research in hand, Jay walked into the meeting ready to join the team.

[d] A word stem was the most basic component of a set of related words. For example, the words fishing, fished, and fisher all contain the word stem *fish*.

Exhibit 1 Sample of App-related Tweets from that Training Set

	Tweets
1	@ankeshk +1 to @mailpro We use MailPro for marketing emails and their Tamarin SEO app for txn emails... @sampad @abhijeetmk @hiway
2	@devongovett I'm using Tamarin, but been saving issues with some domains getting blocked with no bounce message. Very hard to debug.
3	@devongovett Tamarin seems pretty cheap (by MailPro)
4	@frankioh @sinue mmm no. Tamarin y Sendgrid son + bien p/emails transaccionales. Mailpro es + bien p/mkt. Admon de listas de dist. y tal.
5	@kennydude @devongovett Yeah, Tamarin is well worth a look.
6	@tamarin Realised I did that about 5 seconds after hitting send!
7	@tamarinapp tried refreshing, the link (line 184 on the homepage) goes here http://help.tamarin.com/forums/20689696-smtp-integration …
8	@tamarinapp we cannot even find out the last email sent to, as tamarin page crashes when enquiring
9	@tamarinapp yeap! that's what I meant throttling - throttling before it gets to Tamarin
10	@nathanbowser Mind submitting a request with details at http://help.tamarin.com ? We're glad to have a look!
11	@richaskew Can you give us some details about where you're seeing that? Submit a request at http://help.tamarin.com
12	@traskjd they 'migrate' everything to Tamarin?
13	@wesbos Tamarin is good but I've noticed a lack of attention to detail on @MailChimp's secondary properties (i.e. TinyLetter)
14	After looking at its website, not sure what @MailPro Tamarin does, but gosh darn its #logo is nice. #design pic.twitter.com/9OEGuXGJ4T
15	arrays would seem to be a portion of the solution. but its an area I don't know much about http://jsfiddle.net/tamarin/GAwTw/4/ … #javascript
16	From Coworker about using Tamarin: "I would entrust email handling to a Pokemon".
17	would like to send emails for welcome, password resets, payment notifications, etc. what should i use? was looking at mailgun/tamarin
18	ValidationError from tamarin with google app engine's urlfetch http://pyq.io/so/16260022 #python
19	progress, of a sort :/ http://jsfiddle.net/tamarin/GAwTw/2/ … I'm thinking an array might be needed, was trying to avoid :/ #javascript
20	Psyched that @MailPro just dropped the prices of tamarin (their SendGrid Amazon SES competitor). Ready to rock http://blog.mailchimp.com/unifying-mandrill-and-mailchimp-data/ …

Source: Casewriter.

Exhibit 2 Sample of Non-app-related Tweets from that Training Set

	Tweets
1	Spark tamarin theme #nerdatwork
2	@alicegreennn_ but how come you didn't have a tamarin
3	@tamarinband @j_smedley is around here somewhere... He'd know the old drummer.
4	@tamarinband great show!
5	@charlie29598 #GreatJob #Tamarin
6	@tamarin Cheers for the free proofing service - always good to have an extra set of eyes, although the GameSkinny editors are pretty sharp.
7	love those tamarins! so cute!

Source: Casewriter.

Endnotes

[1] "Once More, With Feeling: Draws and Drawbacks of Sentiment Analysis," June 30, 2016, post on blog "University of Michigan: Digital Project Studio," https://digitalprojectstudio.wordpress.com/2016/06/30/once-more-with-feeling-draws-and-drawbacks-of-sentiment-analysis, accessed September 2005.

[2] C.J. Hutto and Eric Gilbert, "VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text," Association for the Advancement of Artificial Intelligence, 2014, http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf.