Generating Twitter Replies Based on User Location

Nick Wilson
The University of Texas at Austin
Austin, TX USA
`njwilson@cs.utexas.edu`

Nazneen Rajani
The University of Texas at Austin
Austin, TX USA
`nrajani@cs.utexas.edu`

Abstract

This paper discusses the detailed implementation of our Twitter bot, *SpellBound*. When a user tweets to *SpellBound*, it fetches the user's location and generates candidate replies relevant to the user's location and her past tweets. It then ranks the generated replies to obtain the one that most resembles a tweet written by a genuine person. In this way, *SpellBound* uses various machine learning and natural language processing techniques to post tweets as a way to communicate like a human does.

1 Introduction

Our Twitter bot responds to users with automatically generated replies targeted at the user's interests and location. Rather than retrieving content written by others and posting it from the bot, we take the approach of creating a language model customized to the user and using it to generate several unique candidate replies. These replies are then filtered and ranked to pick the best one. Sample interactions with our bot show some promising results, but the ratings from the human evaluation we conducted show that our results fall far short of human standards.

The code described in this report can be found with the "ANLP-Course-Final-Report" tag of our GitHub repository[1].

2 Location Resolver

Our bot depends on the ability to determine the physical location (latitude and longitude) of the user tweeting to it. To locate a user, we first check if the incoming tweet was geotagged and already includes exact coordinates. Since the large majority of users do not geotag their tweets, we fall back to locating the user based on the optional free-form location field in their profile. The contents of this field are used as a query to the GeoNames[2] API (with the AND operator) to search over all fields in the database. The results are filtered such that they include only populated places and then the highest-ranked location is used.

While this approach works quite well if the user has a standard location like "Austin, TX" in their profile, there is plenty of room for improvement. For example, more general indicators of location such as "Greater Portland Area" will not return any results. Furthermore, many users do not set their location field or do not set it to the name of a real place (Hecht et al., 2011). That said, it is still frequently a valuable resource for determining a user's location and is sufficient for our needs. Our bot uses "Austin, TX" as the default location if the user's location can not be resolved.

3 Generating Candidate Replies

In this section we discuss SpellBound's implementation for generating candidate replies. This procedure is triggered every time a user tweets to the bot.

3.1 Discovering Local Trends

Using the location resolver described in Section 2, we attempt to find the location of the user that tweeted to us. We then query the Twitter API to get a list of trending topics near the location and randomly pick one to be the subject of our reply.

3.2 Language Model

In order to generate candidate replies, we use an aggregate bigram language model (Saul and Pereira, 1997). It consists of a weighted combination of three models: an offline model, a user model, and a trend model. Each of these are discussed in detail below.

[1]`http://github.com/ut-anlp-bot/tshrdlu`

[2]`http://www.geonames.org/`

The offline bigram model was created from 1000 random tweets from the Twitter sample stream. The model was generated and stored offline and is loaded every time the reply generator is triggered. This model is mainly for adding content diversity to the aggregate model.

The user bigram model is created using the tweeting user's past tweets. The model uses at most 200 past tweets authored by the user. This helps in generating a model relevant to a user's interest and/or what the user has tweeted about in the past. To assist with development and testing of the bot, there is a provision to set the user whose tweets are retrieved to create this model. In order to set a user, the bot can be sent a tweet:

```
Set User: put_screen_name_here
```

The bot will then respond to future tweets with replies targeted towards the named user.

The trend bigram model is created by searching on the local trend identified as described in Section 3.1. This generates a model relevant to the local trend.

The tweets used to create the models have a special start token '∧' and a stop token '$$' inserted which allows the language model to determine the probability of start and end of a reply respectively. The final aggregate model is formed by combining the aforementioned bigram models using weights 0.2, 0.4, and 0.4 respectively. This aggregate model is later used to generate candidate replies.

A key challenge faced while developing the bigram model based on trends was that the bot framework we use was starting a Twitter stream for monitoring the user's Twitter feed, but our original implementation also required using the Twitter stream and filtering it based on the trending topic near the user. Since Twitter does not allow multiple streams, we resorted to using the search API for tweets related to the trend. The main limitation of this approach is that searches count against the Twitter rate limit[3] and limit the number of tweets we can retrieve to build the language model. Future work may be able to overcome this limitation by finding a way to have multiple Twitter streams open at the same time. This is discussed in Section 7.

[3]`https://dev.twitter.com/docs/rate-limiting/1.1`

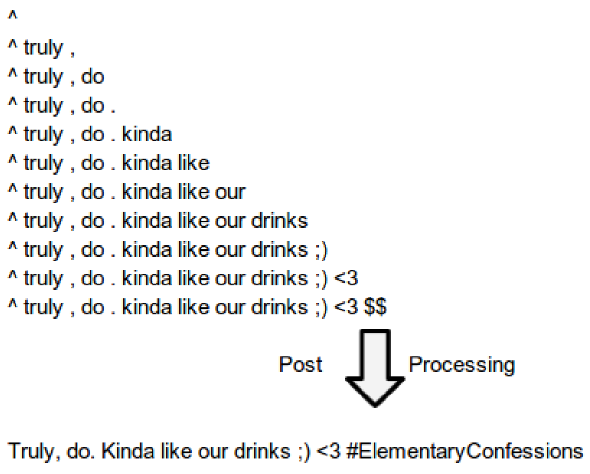
^
^ truly ,
^ truly , do
^ truly , do .
^ truly , do . kinda
^ truly , do . kinda like
^ truly , do . kinda like our
^ truly , do . kinda like our drinks
^ truly , do . kinda like our drinks ;)
^ truly , do . kinda like our drinks ;) <3
^ truly , do . kinda like our drinks ;) <3 $$
Post
Processing
Truly, do. Kinda like our drinks ;) <3 #ElementaryConfessions

Figure 1: Overview of generating a candidate reply.

3.3 Candidate Replies

Using the aggregate bigram model developed in Section 3.2, we generate candidate replies for a time period of 20 seconds using random sampling. This ensures that we generate a variety of different candidates albeit the highly probable replies are generated frequently. We allow the candidate replies to start and end naturally symbolized by the start and end tokens respectively. If the generated candidate reply does not already contain the hashtag from the trending topic, we append it at the end of the reply.

3.4 Post Processing

In this step we process the candidate replies to resemble a real tweet. The post-processing includes capitalizing the first letter after punctuation where appropriate, stripping extra spaces between tokens, capitalizing 'i' when it comes by itself or with an apostrophe as in "i'll". These filtering steps are primarily done with regular expressions. Thereafter, we filter the replies that exceed the character limit (because we allow them to end naturally, at times they may get very long) and make a list of distinct candidate replies. Figure 1 shows a sample candidate reply generation and post-processing step.

4 Ranking Candidate Replies

Most of the candidate replies generated from our language model do not look like tweets a real person would write. In order to sort the candidates and pick the best one, we trained a classifier offline that is designed to distinguish between real and fake tweets. The confidence scores from the classifier are used to rank the candidates based on how real they are predicted to be.

4.1 Generating Data

The training corpus consists of 1400 real tweets and 1400 fake tweets that were created in batches. For each batch, we picked a random user and a random trending topic. Real tweets were collected from the user's timeline and the trending topic. Fake tweets were generated using the same technique described in Section 3. The tweets were shuffled and split into a training, development, and test set containing 2000, 400, and 400 tweets respectively. Each set has an equal number of real and fake tweets.

4.2 Features

We extract the following types of features after tokenizing the tweets with Twokenize (O'Connor et al., 2010):

Skip (Gappy) Bigrams. A skip bigram is a pair of tokens in order, possibly with other tokens between them (Goodman, 2000). They are useful for capturing long-distance dependencies between words with sparse data. Consider the sample tweet "the cat eats fish." We use two types of skip bigrams. The first type encodes the number of tokens in the gap (e.g., "$the + \_ + \_ + fish$"). The second type encodes all gap sizes equally (e.g., "$the + \_\_\_ + eats$" and "$the + \_\_\_ + fish$"). We use gaps containing 1 to 5 tokens for the first type and 1 to 50 tokens for the second type.

Duplicate N-Gram Fraction. Fraction of the tweet's n-grams (between 1-gram and 5-grams) that are duplicates. For example, the tweet "I think I can" contains four unigrams, one of which is a duplicate.

Number of Tokens. The total number of tokens.

Stopword Fraction. The fraction of words that are stopwords.

Trigrams and Quadrams. Sequences of 3 and 4 tokens.

4.3 Results and Analysis

We used Nak[4] to train the model with the L2-regularized logistic regression solver in LIBLINEAR (Fan et al., 2008). After optimizing the regularization parameter on the development set, the model obtains an overall accuracy of 83.5 and an average F_1 score of 83.5 (83.33 for the fake class and 83.66 for real) on the held-out test set.

Analyzing the results of classification, we notice that the classifier's performance is mainly affected by candidate replies that are a lot like the real tweets. For example, the candidate reply "Never before, promos and bailey combined for myself. #ThingsICantLiveWithout" was incorrectly labeled; it was fake but mistaken as real. On the other hand an example of a real tweet classified as fake was "when did you meet Taylor? #ChristianMingleUsernames". Looking at these and other similar examples of mis-classification we can conclude that the incorrectly labeled tweets were highly ambiguous and difficult to classify even by human judgment.

While our model was evaluated on its ability to distinguish between real and fake tweets, it is important to note that it is not used to classify any real tweets in our final application. Instead, it is used to rank a set of fake tweets based on how real they seem. An interesting option for future work would be to investigate the use of learning to rank algorithms such as SVM-RANK (Joachims, 1999).

5 Examples and Analysis

In this section, we present examples of responses generated by SpellBound and analyze them.

5.1 Examples

This section discusses examples of tweets generated and ranked by our Bot. We set the user to *Bill Gates* for the following example by tweeting to the bot:

```
Set User: billgates
```

Once the user is set, we tweet to the bot and get a reply as follows:

```
USER: Hey, how are you doing today?

BOT: ``I hate you turned to conquer
with my window in the ballet. They
follow fredo #YoureBeingJudged
```

[4]http://github.com/scalanlp/nak

The bot was tweeted at from Austin, TX and one of the trending topics at the time this example was created was *#YoureBeingJudged*. Based on the aggregate model that combined the offline model, the model generated using Bill Gate's past tweets, and the model developed by searching on *#YoureBeingJudged*, SpellBound generated several candidate replies and used the classifier to rank them and select the best one.

Looking at the tweet posted by the bot to the user, we notice that the word "window" is relevant to the set user which is *Bill Gates* in this case. It also has words relevant to the trend and some random tokens. The above generated reply ranks highest with a difference of 0.58 between the confidence scores of the reply being real and fake. This can be contrasted with the candidate reply that scores -0.2:

```
Want' why text me hables o d
#YoureBeingJudged
```

Here is another example of a reply posted by our bot when the set user was "*jasonbaldridge*" and the local trend randomly selected was "*#ElementaryConfessions*"

```
BOT: Djing in silence, take our
record to death of yourself. 3916
riviera drive. That you <3 enjoy
your dm #ElementaryConfessions
```

The classifier ranking implementation discourages the bot from posting a reply that has either too much or too little punctuation and/or other tokens.

5.2 Analysis and Interpretation

From the aforementioned examples, we observe that the responses generated are of poor quality and do not resemble tweets that are generated by a human. The main reason for poor response is the language model used. The offline bigram model was created using a small number of random tweets from the Twitter stream with the purpose of adding more diverse bigrams to the language model, however, it adds a lot of noise to the generated candidate replies. Although the user bigram model uses a user's recent tweets for generating replies, it may not reflect the user's interests and thus may not lead to relevant tweet generation. The trend bigram model is made using Twitter search API and thus is made using only around 10 tweets. Although the trend model has more weight, the bigrams from this model are rare in the final reply and thus the trending topic almost always needs to be appended at the end.

We discuss some methods of resolving the issues mentioned above in Section 7.

6 Human Evaluation

Two human subjects not associated with the project participated in our human evaluation. The subjects were asked to evaluate each tweet on a scale of 1 (worst) to 5 (best) according to five measures:

1. **Coherence.** Tweet is well-connected; understandable; about a single topic.
2. **Intelligibility.** Tweet is able to be understood; makes sense.
3. **Human-ness.** Tweet feels natural; written by a human.
4. **Relevance – User.** Tweet is relevant to the user's interests.
5. **Relevance – Trend.** Tweet is relevant to the trending topic.

Table 1 shows the results of the user evaluation for each measure. We observe that the ratings are really quite poor. In fact, looking at the original data, none of the individual ratings was higher than 2. The highest-scoring measure was humanness with an average score of 1.40. The best-rated tweet had an average score of 1.5:

```
" haha, eat a neck tattoo so I
mean to to understand that hour.
#ILoveMyMomBecause
```

The worst-rated tweet had an average score of 1.00:

```
Fun to moa to california
area is doing homework, have.
#thingsthatirritateme when will
get jealous.  #ThankYouJesusFor
```

Using Spearman's rank coefficient (rho) to evaluate the degree of agreement between our human raters, we find that the scores are actually slightly negatively correlated ($rho = -0.12$) indicating a strong lack of agreement[5]. One possible explanation for this may be the difficulty in

[5]Note, however, that Spearman's rank coefficient does not capture the fact that both raters agreed that the tweets were all very bad (scoring a 2 or less on a scale of 1-5) relative to human standards.

Measure	Mean Rating
Coherence	1.15
Intelligibility	1.20
Human-ness	1.40
Relevance – User	1.10
Relevance – Trend	1.20

Table 1: Average user ratings along five different measures.

distinguishing between the quality of tweets that are all largely nonsensical. Another possibility is that the scoring criteria used in the evaluate was not sufficiently well-defined or the raters were not well-trained.

Additional evaluation strategies may be useful for evaluating tweet generation in the future. For example, we could measure fluency with the cloze task (Taylor, 1953) where single tokens in a tweet would be replaced with a blank and raters would attempt to fill in the blank. A higher rate of correct blank-filling implies a more fluent tweet.

We could also introduce some automatic evaluation metrics such as those used for evaluating machine translation systems such as BLEU (Papineni et al., 2002). These may be useful for evaluating how similar our generated candidates are to real tweets from the user or trending topic. Automated metrics, in addition to providing more feedback on our final system performance, would allow us to iterate more quickly and evaluate possible improvements while developing our tweet generator.

7 Future Work

In this section we discuss how our bot can be improved to generate more relevant responses as well as other methods which could help improve our implementation of SpellBound.

7.1 Improved Reply Generation

First, the offline bigram model needs to be trained on more data so that its purpose is fulfilled. It would also be nice to classify the tweets used for the offline model into general categories like sports, news, politics, etc. A separate model could be created from each category so that it would help generate more relevant replies to the user. Topic based language models try to take advantage of the fact that different topics will have different kinds of words (Bellegarda, 2004).

Second, the user bigram model could be trained on the user's interest rather than her recent tweets. This could be achieved by extracting the user's interests from her Twitter profile and interactions with other users and by creating topic models from her recent tweets.

Finally, the trend bigram model could be created by filtering on Twitter stream for trending tweets rather than using search API and thus would not be restricted by rate limiting. Also, instead of using a random trend close to the user, we could use trends that have high degree of overlap with the user's interest and thus be more relevant. This would be done using cosine similarity between the trending topics and the user's topics of interest. For a user whose tweets are geotagged, our location resolver would have high confidence and thus instead of fetching tweets based on trends in the area, SpellBound could make recommendations for popular nearby places by looking up geotagged Wikipedia articles.

While forming the aggregate model, instead of using fixed weights, a better technique would be to use Stacking which is a type of ensemble learning. To help develop a personality for SpellBound, we could maintain a session with the user and generate responses based on the recent interaction with the user. This can be achieved by fetching the interaction tweets and creating a language model from them. (Chu et al., 2010) observe the difference between humans and bots in terms of tweeting behavior and tweet content and use them as features in classifying human and bot accounts on Twitter. We would also like to experiment with more with their features and incorporate more human-like features in our bot.

Because of the nature of bigram models, it is not always possible to generate a reply that is relevant and also grammatically correct at the same time. Thus we would like to evaluate the performance of our bot by generating replies using higher order n-gram language models created from a much larger corpus of tweets. We would also like to identify the limitations of our generated responses and if the quality of generated responses is below a certain predefined threshold, use the search API instead to find a relevant response (like the original Tshrdlu bot). Exploring the quality of tweets using a template based technique to generate a response is also something we would like to experiment with.

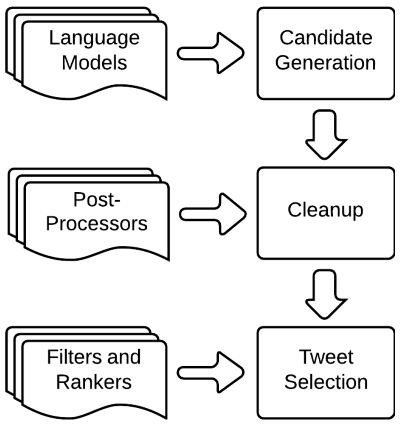

Figure 2: General tweet generation pipeline.

7.2 General Tweet Generation Pipeline

Our tweet generation implementation could be generalized to create a simple framework with pluggable components for others to build on (Reiter and Dale, 2000). Figure 2 shows a sample architecture. First, one or more language models are combined by weighting each one as we do with our aggregate model. Next, one or more post-processing steps are applied in a preconfigured order to clean them up and remove artifacts from the first stage. Finally, one or more filters or rankers are applied to remove tweets from consideration and to surface the best one. If all rankers report scores on a common scale, each one could be weighted to prefer the scores from a particularly ranker to suit the goals of the application.

A pluggable architecture such as this would allow developers to easily get started with generating tweets and share components. By allowing components and weights to be modified at runtime, bots could easily adapt based on the goals of the bot. For example, if the bot is having an ongoing conversation with a user, we might want to give a larger weight to a ranker that measures how relevant each candidate is to the conversation.

8 Conclusion

In this paper we presented SpellBound, a Twitter bot that generates a response relevant to the tweeting user's interests and her location. Our bot uses a classifier for ranking tweets in order to obtain a response that most resembles a tweet composed by a human. To achieve this, we discussed the features on which our classifier is trained. Finally, we analyzed our results and proposed solutions to address the challenges faced. We envision applications of our work on automated systems such as online help, personalized service, or information acquisition.

References

Jerome R Bellegarda. 2004. Statistical Language Model Adaptation: Review and Perspectives. *Speech communication*, 42(1):93–108.

Zi Chu, Steven Gianvecchio, Haining Wang, and Sushil Jajodia. 2010. Who is Tweeting on Twitter: Human, Bot, or Cyborg? In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 21–30. ACM.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

J. Goodman. 2000. A Bit of Progress in Language Modeling. Technical report, Microsoft Research, 56 Fuchun Peng.

Brent Hecht, Lichan Hong, Bongwon Suh, and Ed H. Chi. 2011. Tweets from Justin Bieber's heart: the dynamics of the location field in user profiles. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 237–246, New York, NY, USA. ACM.

Thorsten Joachims. 1999. Advances in kernel methods. chapter Making large-scale support vector machine learning practical, pages 169–184. MIT Press, Cambridge, MA, USA.

Brendan O'Connor, Michel Krieger, and David Ahn. 2010. TweetMotif: Exploratory Search and Topic Summarization for Twitter. In William W. Cohen, Samuel Gosling, William W. Cohen, and Samuel Gosling, editors, *ICWSM*. The AAAI Press.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ehud Reiter and Robert Dale. 2000. *Building natural language generation systems*. Cambridge University Press, New York, NY, USA.

Lawrence Saul and Fernando Pereira. 1997. Aggregate and mixed-order Markov models for statistical language processing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 81–89. Somerset, New Jersey: Association for Computational Linguistics.

Wilson Taylor. 1953. Cloze Procedure: A New Tool for Measuring Readability. *Journalism Quarterly*, 30:415–433.