A Novel Social Search Assistant for Twitter

Ashley Edwards
Department of Computer Science
Georgia Institute of Technology
Atlanta, GA, 30332
aedwards8@gatech.edu

Himanshu Sahni
Department of Computer Science
Georgia Institute of Technology
Atlanta, GA, 30332
himanshu@gatech.edu

Abstract

Twitter is a popular social network where users frequently post short messages and engage in conversations with other users. This provides a large amount of information about the user’s interests and social preferences and has been a lucrative research area. In particular, research in Social Search, the act of posing questions to and receiving answers from one’s social network, has gained interest. However, the large database of information that results from users’ Twitter interactions is not presently leveraged for aiding in social search. In this paper, we introduce CoBot, a novel social assistant and data mining agent for Twitter. CoBot gathers social statistics about users and their followers, allows them to query about their social interactions, and assists with Social Search by recommending knowledgeable followers and providing relevant tweets. By using social statistics about users and learning from feedback about recommendations, CoBot is more amenable to ‘social queries’ than conventional search engines.

1 Introduction

Twitter’s interface allows for a natural domain for Social Search, a term coined for users that use social networks to ask questions and receive answers [1, 2, 3]. Twitter users often pose many different types of questions to their followers, such as those relating to recommendations, opinions, factual knowledge, and social topics [3].

One reason for this is that users of Twitter often trust their followers to be able to answer their questions, particularly those relating to opinions or recommendations [1, 3]. However, many queries in Twitter remain unanswered or unnoticed by a user’s followers. Because users are constantly receiving real-time updates from the people that they follow, it may be difficult to notice any queries placed. Furthermore, Twitter relationships are not necessarily bidirectional, that is, if A follows B, it does not mean that B follows A. Therefore, even if a user poses a question to their followers, this does not imply that all of their followers will be able to see the query [1]. However, it is much more likely that a question will be answered by someone if it is directed to them, even if they are not following the user posing the question [3].

In this paper, we introduce CoBot, an autonomous Twitter user that acts as a social assistant for his followers. CoBot collects statistics about his network of followers. Based on these statistics, he attempts to respond to queries with recommendations of friends of the user that will likely know the answer. CoBot is able to answer such queries by leveraging the Twitter feed of the user and their friends. CoBot is able to learn from feedback he receives after providing a recommendation. Furthermore, CoBot responds to queries that a user may have about their usage of Twitter.

The paper is organized as followed: In Section 2, we discuss work relating to social search and information retrieval. We formally introduce CoBot and his underlying algorithm in
In Section 4, we will show both quantitative and qualitative results of our work and in Section 5 we discuss these results and conclude.

2 Related Work

CoBot is inspired by similar work performed by Isbell et al. [4]. CoBot was introduced in LambdaMOO, which is an active online community with hundreds of users. The goal of this research was to make CoBot a part of a community that people frequently interacted in. The authors used statistics about user interactions as a model for modifying the agent’s behavior. Reinforcement Learning was used to train CoBot’s responses to the actions directed at him, with rewards coming from human feedback. Our work introduces CoBot to Twitter. We formulate a different learning problem which uses statistics gathered about CoBot’s followers to guide responses to user queries.

There has been much work describing the mechanics of social search in Twitter. Paul et al. [1] describe three common questions that are asked in Twitter: rhetorical, factual knowledge, and polls. They use Amazon’s Mechanical Turk to label a set of tweets as being a question or not a question. The authors then find how many responses these tweets received. The results show that less than 20% of the questions received an answer. Morris et al. discuss some reasoning behind why humans are motivated to answer certain questions [3]. They find that most people answer to be altruistic, or because they are an expert in the field. Furthermore, the authors find that close friends are more likely to answer a user’s questions. We consider these motivations in our formulation of CoBot.

In general, we are interested in extracting information from tweets. Therefore, our motivation is related to information retrieval, but not in the traditional definition. Rather than receiving documents that are relevant to a social search, we instead retrieve relevant people that may be able to answer the query. This is similar to the work by Sahay et al. [5]. The authors introduce a preliminary framework for a conversational search agent. This agent was capable of recommending people that may have been able to answer a question in a personal health forum. However, a user’s similarity to a recommended person was not taken into account because interactions in health forums are generally brief. User closeness/similarity is an important feature of our framework.

Our work most closely relates to the work by Horowitz et.al describing the Aardvark system [2]. Aardvark was an autonomous agent that was capable of directing users to people in their extended social network who might know the answer to the user’s query. The system retrieved information about the user from Facebook, Twitter, and other means of indirect communication. The agent was capable of learning from three forms of feedback from requested question answerer: muting a topic, declining to answer a topic, and receiving negative responses about the answer. We will explain how such signals from the environment are useful to our framework. While the motivation behind this work is similar to ours, the approach is very different. The Aardvark interface that the user interacts with is separate from the one they would use on a daily basis, such as Facebook or Twitter. The authors implemented a chat bot that was used solely for question and answering. By using Twitter for our UI, we introduce a more convenient medium. CoBot is able to directly respond to user queries that are sent from Twitter. Furthermore, users in Aardvark were asked to state which topics they felt comfortable answering. The framework was formalized as a question/answer site. However, although Twitter is a good source for social search, this is not its main purpose. Therefore, we choose to learn which topics people have knowledge of. Nevertheless, Aardvark introduces a clear structure for designing a system capable of recommending people, which we will expand upon in Section 3.

3 Method

We will now describe the underlying structure of CoBot, which is motivated from the structure derived in the Aardvark System [2].

3.1 Data Collection and Topic Extraction
In order for CoBot to intelligently answer queries, we first collect data about users and their followers.

### 3.1.1 Data Collection

CoBot’s answers to queries are based solely on a user’s followers. Before collecting data about a user, he or she must allow CoBot to follow them.

We collect data by using the Twitter API. Every tweet is logged in “generating user” and “target user” log files as a tuple containing both user IDs and the message text, as shown in Fig. 1.

![Tuple for logging user data](image)

These logs are updated in real time for all users CoBot follows and for each user’s followers. If a tweet is not directed at a specific person, targetID is null. Along with collecting data, we also generate statistics about the user and their followers, such as the number of interactions two users have had. The user can query CoBot about information generated from these statistics, as shown in Table 1. These statistics are an important component for CoBot. They are used by the learner, described in Section 3.3, to recommend people who appear knowledgeable about a subject. The statistics are similar to those in [4]. However, we replace “verbs” with “topics”, and statistics are generated from users in Twitter, as opposed to in LambdaMoo.

<table>
<thead>
<tr>
<th>Query Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tell me about “topic”</td>
<td>Tell user which of their Twitter friends tweet about and have tweets directed to them that involve “topic”</td>
</tr>
<tr>
<td>Tell me about me</td>
<td>Tell user what they tweet about the most and what their friends most often tweet to them about</td>
</tr>
<tr>
<td>Who spams</td>
<td>Tell user which of their friends generate the most topics and have the most topics directed to them</td>
</tr>
<tr>
<td>Relate to “follower”</td>
<td>Tell user about their similarity to one of their followers</td>
</tr>
</tbody>
</table>

### 3.1.1 Topic Extraction

The statistics that we collect do not contain the entire message text from a tuple. Instead, we extract topics that seem relevant to the text. This ensures that we can find users that know about a particular concept. For example, if a user tweets about “Starbucks” and “Java Bean,” then the Coffee topic counter within the user’s set of statistics will be incremented by 2.

We used three different APIs for extracting topics from a tweet: Open Mind API [6], Open Calais API [7], and Alchemy API[8]. These APIs use machine learning to determine the semantic meaning of text documents.

Once we retrieve these topics, they are then stored in a Java user object saved on disk. The list of topics directed at a user and generated by a user are stored separately to increase speed of retrieval at query time. Statistics about the user and their followers are also stored in this user object.

### 3.2 Query Analysis

The next component of CoBot is analyzing queries. This involves determining which type of
question the user is asking. We programmed in keywords for each of the types of query statistics. For example, if a query contains “relate” then CoBot knows that the user is querying about statistics on their relationship with one of their followers. We extract the topics from the message text in this query, and then send the information to the learner.

3.3 Learning and Recommendation

A core functionality of CoBot is to provide relevant recommendations of users and past tweets of answers to social queries posed by users. As such, CoBot is a novel information retrieval agent because it is biased by users’ relationships to individuals in their social networks. A perceptron learning algorithm is used to determine weights on features pertinent to social information retrieval. These weights are unique for each user and are updated by feedback to CoBot provided by the user posing the query. The features are themselves chosen from the statistics CoBot keeps about each user. The features we decided to use are shown in Table 2.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset</td>
<td>Always set to 1</td>
</tr>
<tr>
<td>Tweeted To</td>
<td>Count of tweets sent to a particular ‘target’</td>
</tr>
<tr>
<td>Tweeted From</td>
<td>Count of tweets received from ‘target’</td>
</tr>
<tr>
<td>Query Similarity</td>
<td>Dot product of query and target’s issued tweets frequency vectors</td>
</tr>
<tr>
<td>User Similarity</td>
<td>Dot product of user and target’s issued tweets frequency vectors</td>
</tr>
<tr>
<td>Target Tweets</td>
<td>Total count of tweets issued by target</td>
</tr>
<tr>
<td>Target Received Tweets</td>
<td>Total tweets directed towards target</td>
</tr>
<tr>
<td>Target Total Topics</td>
<td>Total count of unique topics tweeted about by target</td>
</tr>
<tr>
<td>Target Received Topics</td>
<td>Total count of unique topics in tweets directed towards target</td>
</tr>
<tr>
<td>Neighbor Similarity</td>
<td>Similarity between user and target’s most similar follower</td>
</tr>
</tbody>
</table>

Some of these features deserve a deeper explanation. A feature vector is always constructed at query time and is unique for a <user, target, query> tuple. The ‘user’ in this tuple is the user issuing the query, ‘target’ is a Twitter follower of ‘user,’ and ‘query’ is a list of topics extracted from the query posed by ‘user’. This feature vector is uniquely determined for all followers in the list of followers of the ‘user’. Similarity values are calculated by taking dot products between two frequency vectors. Frequency vectors are high dimensional vectors with the term frequency of each topic in the topic dictionary. Term frequency is defined as below:

\[
\text{tf}(t, d) = \frac{f(t, d)}{\max\{f(w, d) : w \in d\}}
\]

Here \(t\) is the term, \(d\) is the document containing it, and \(w\) is the entire set of documents. The query frequency vector is constructed by calculating the term frequencies of every possible topic from the query and then normalized by the maximum frequency of a topic by the user who issued the query. Unfortunately, this vector is large and sparse. Thus, to efficiently compute it, we only store and compute the frequency of topics present in the query and store them alongside the topics themselves as tuples. When calculating the similarity, we iterate through the frequency vectors and search for a match in the target frequency vector. The formula for similarity is hence:
The reader may note that this formula is analogous to cosine similarity except that it is not normalized. We believe this will work better in a sparse setting and is easier to calculate. The downside is that it is not comparable between different users and queries. We take advantage of the fact that it can be compared to different targets if the former two are held constant.

The overall learning architecture is as follows:

1. CoBot constantly collects statistics on its followers and their followers
2. A user may pose a natural language query at any time, which CoBot parses into a topic list.
3. CoBot provides the best recommendation from the user’s followers based on the features from Table 2 and the weights stored for the user.
4. The user independently verifies if the recommendation was useful or not and provides CoBot appropriate feedback with the use of a few chosen keywords. This feedback is then translated into -1 for misclassification and 1 for correct classification.
5. CoBot then updates the users feature vector according to the perceptron rule:

\[
    w_i = w_i + y_i x_i, \forall i \in \text{features}
\]

It should be noted here that a user may train CoBot to retrieve recommendations based on their own tastes and preferences. This may defy conventional information retrieval approaches because users may choose to completely neglect query relevance and concentrate only on the social features.

3.4 User Interface

We designed CoBot to be an autonomous agent on Twitter. Users query CoBot by sending him a tweet, and he sends the user a response in the form of a tweet. We believe that this is more convenient than using another medium to find an answer to a question, and gives a more natural component to social search. In general, we wanted the queries to be conversational. We even made CoBot more human-like by giving him chat abilities.

4.2 Results

We will first show the results from statistics querying. Then we show the results from recommending people to answer questions.

4.1 Statistics Querying

We will show qualitative results for querying on statistics. The statistics are used to provide users with information about their interactions on Twitter, and therefore there are no quantitative results to report.

Fig. 2 shows an artificial conversation that was used to generate statistics that could be queried about. One of the authors (CoBotUser3) also spent some time conversing with CoBot to attempt to “spam” Twitter with topics. Fig. 3 shows the results from querying CoBot after these artificial conversations took place.
**C0b0tUser1**: I’m really sad that football season will be over soon! I don’t know what I’m going to watch now!!!

**C0b0tUser2**: @C0b0tUser1 ah, don’t worry. Football isn’t that great anyway! You should watch basketball instead...

**C0b0tUser1**: @C0b0tUser2 No, football is great! You’re just scared that UGA is going to beat Tech next week!!!

**C0b0tUser2**: @C0b0tUser1 Anyways.. are you going to watch the MTV music awards???

**C0b0tUser1**: @C0b0tUser2 Probably not. I think they’re showing the Lion King on ABC and, to be honest, Mufasa trumps Katy Perry anyday!!!!

**C0b0tUser2**: @C0b0tUser1 Mufasa never even sang anything in the Lion King…

**C0b0tUser1**: @C0b0tUser2 Exactly…

---

Figure 2: Simulated conversation in Twitter

---

**C0b0tUser1**: @c0b0t tell me about myself

c0b0t: @C0b0tUser1 You tweet most about: [the lion king %: 9.52, facility %: 4.76, sport %: 4.76]  
c0b0t: @C0b0tUser1 People tweet you the most about: [invention %: 4.76, fieldofstudy %: 4.76, awarddiscipline %: 4.76]

**C0b0tUser2**: @c0b0t tell me about "sports"

c0b0t: @C0b0tUser1 @C0b0tUser2 These used sports the most: @C0b0tUser1 @C0b0tUser3  
c0b0t: @C0b0tUser2 These had other people tweet them about sports the most: @c0b0t @C0b0tUser1

**C0b0tUser2**: @c0b0t tell me about "awarddiscipline"

c0b0t: @C0b0tUser2 These used awarddiscipline the most: @C0b0tUser2  
c0b0t: @C0b0tUser2 These had other people tweet them about awarddiscipline the most: @C0b0tUser1

**C0b0tUser2**: @C0b0tUser2 The people who spam the most are: @C0b0tUser3

---

Figure 3: Results from querying CoBot on statistics

---

### 4.2 Recommending

Rather than having a user study, we generated five artificial users for evaluating CoBot’s recommendation abilities. We had a list of topics {Georgia Tech, Tacos in Atlanta, Health, Car, Football} and searched the Twitter public stream for tweets containing these topics. We generated a corpus for each topic from these tweets. Then, we created artificial user logs based on manually generated probabilities for each topic, as shown in Table 3.

---

Table 3: Content probabilities for each user
That is, for each user X, we iterate through each topic corpus and add the current content to the user's log of data with P(Topic). For example, User 1's data log will contain content from about 0% of the Georgia Tech Corpus, 10% of the Taco's in Atlanta Corpus, 20% of the Health Corpus, and so forth. The purpose of this was that some users would “tweet” more about a particular subject than others and we could use this information for validating our results.

After generating logs for each of the artificial users, we created queries and issued them from User 1, then evaluated the recommendations produced by the perceptron. Table 4 shows the similarity scores between User 1 and each other user:

**Table 4: Similarity scores between User 1 and each other user**

<table>
<thead>
<tr>
<th>User</th>
<th>Similarity Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.93</td>
</tr>
<tr>
<td>3</td>
<td>2.05</td>
</tr>
<tr>
<td>4</td>
<td>1.93</td>
</tr>
<tr>
<td>5</td>
<td>1.77</td>
</tr>
</tbody>
</table>

It is evident that users that tweet about similar topics (as shown from the probabilities in Table 3) receive high similarity scores and vice versa.

The top most tweeted topics extracted from User 1 are shown in the following (topic, count) pairs:

- (RT, 26)
- (car, 19)
- (tacos, 8)
- (football, 7)
- (Atlanta, 5)
- (health, 4)
- (football season, 4)
- (atlanta, 3)
- (health care, 3)
- (Australia, 3)

In the next experiment, the following queries were simulated for User 1:

Query 1: "I am feeling sick, I need to go to the hospital. Can you suggest me one please"
Query 2: "I feel like eating mexican food, can you suggest me a good taco restaurant please"

In both cases, the perceptron recommended User 2 as the appropriate follower after 2 iterations. It should be noted that query 2 is actually more relevant to user 2 while results of query 1 may have been generated due to social features outweighing the query similarity features.

**5 Discussion**

Section 4.1 shows the results from querying CoBot for statistics. As we noted, these results are qualitative. We see that CoBot is able to answer queries about a user’s interactions. As shown in Section 4.2, this means that we have a good base for recommending relevant people for answering queries. However, these results are somewhat preliminary. In order to truly evaluate the system, we will need to perform user studies with real people.

Nevertheless, we have created an agent, CoBot, that is capable of easily answering social queries. There are many different directions that we can take this research in. First, as noted...
by Paul et al. [1] and Morris et al. [3], there are different forms of queries that users can ask in social search. We can take this into account when recommending people to answer a question. For example, we believe rhetorical questions, such as “How is everyone doing today?” may be directed to a user’s entire Twitter feed. However, more personal queries, such as “Does anyone want to go to the park today?” should probably be directed towards the user’s close friends. We could use this information when creating the weights for feature vectors. Users that are deemed “close” to the user should be weighted more heavily for personal queries and less for more public queries. Another way to extend this work would be for CoBot to answer the queries on his own based on answers that other users have given.

References


[6] Available at openmind.media.mit.edu

[7] Available at opencalais.com

[8] Available at alchemyapi.com