A Twitter Search Engine

Surya Teja Chavali
`chavali2@wisc.edu`

Houqi Li
`hli492@wisc.edu`

Abstract

We present a Search Engine for Twitter based on converting unstructured tweets into a relational table using NLP, and querying the result. We use one day's data on the archive tweetstream (arc,).

Our workhorse for parsing the tweets is Apache Spark (Zaharia et al., 2012), with SQL queries being run on Spark SQL (Armbrust et al., 2015).We implement our project on the Databricks Community Edition, a free resource for learning the basics of cloud computing.

Keywords: Spark, Query Answering, Twitter

1 Introduction

With the rise of microblogging portals like Twitter[1], an increasingly large amount of content is being generated online. Social media is easily one of the most important sources of information in the modern age. It is able to provide real-time data about ensuing trends as well as general information available in other repositories online. It can be viewed as a crowd-sourced repository of data along with relevance information, which is reflected in threads of tweets, and trending topics. Crucially, traditional data sources such as Wikipedia[2] lack such (temporal)relevance information, and partially lack information relevant to a particular topic as well(it can be argued that a larger number of people contributing to Twitter would decrease the amount of excluded relavant information).

However, there are a few an important issues impeding the direct use of this information:

- Twitter data is *unstructured*, that is, it is not organized in the ingestible form of a relational table. This is true of Wikipedia as well.
- Twitter data about any topic is *distributed* across multiple(thousands) of small *tweets*, rather than being in one location, such as a Wikipedia article.
- Twitter data is generated at a much larger rate than that of any traditional Knowledge Base.
- Twitter data is *noisy* and can be irrelevant to the topic under consideration - as opposed to the data in Wikipedia, which is largely relevant to the topic on account of a strict peer-review system.
- Underlying Twitter is a *social network graph* of users - based on the relationship of 'following'.
- Tweets also often contain rich features like 'hashtags' and user location information, all of which can be used for information retrieval and ranking.

In the face of these challenges, it is essential to be able to retrieve relevant content from Twitter in a systematic manner. More importantly, traditional methods of information retrieval and ranking such as the Inverted Index Search Engine based on TF-IDF (Manning et al., 2010) and PageRank (Page et al., 1999) are unable to make use of the additional information provided by Twitter, such as social network and temporal information.

2 Problem Statement

We wish to design a search engine tailored for tweets, by exploiting the specific information they provide. Our end goal is to take as input a query specified by a keyword search, of the form `action:drink object:cola` and return a ranked list of tweets relevant to that query.

This is accomplished by firstly transforming unstructured tweet data into a structured relational table, and querying that table for relevant tweets.

[1]https://www.twitter.com
[2]https://en.wikipedia.org

Formally, suppose $\mathcal{U}$ is a set of users, and $\mathcal{G}(\mathcal{U}, \mathcal{E})$ is a social network graph on these users. Consider a corpus of tweets $\mathcal{T}$, where each tweet $t = (u, \tau, c)$ comprises of a user ID $u \in \mathcal{U}$, a timestamp τ, and content $c \in \Sigma^*$. Let $\mathcal{T}^!$ denote the set of all orderings of all subsets of $\mathcal{T}$.

The goal is to answer a query $q = (u, s)$ where $u \in \mathcal{U}$ is a user and s is a query string, by returning an ordered list of relevant tweets $\mathcal{T}_q^k \in \mathcal{T}^!$ so as to maximize a utility metric

$$M : \mathcal{U} \times \Sigma^* \times \mathcal{T}^! \to \mathbb{R}^{\geq 0}$$

where $u \in \mathcal{U}, s \in \Sigma^*, \mathcal{T}_q^k \in \mathcal{T}^!$

3 Related Work

A large amount of recent work has gone into analyzing data from Twitter. Twitter data has been used for various NLP tasks, including entity recognition and sentiment analysis, as graph analytics. (O'Connor et al., 2010) proposes a system for topic summarization for tweets. (Pak and Paroubek, 2010) proposes a method for collecting a corpus for sentiment analysis and opinion mining.

Some work on twitter-specific search is to be found as well. (Teevan et al., 2011) performs a comparison between user behaviors with Twitter search versus web search. (Jeong et al., 2013) builds a crowd-powered question answering service based on tweets. (Halberg and others, 2012) proposes a PageRank-like algorithm for measuring user influence from the social network graph of Twitter. (Nagmoti et al., 2010) shows how to rank results from twitter search.

4 The Solution

The idea is to create a database with a schema that allows user to enter queries that specify `Subject, Predicate, Object` of a tweet. The system then evaluates the query and outputs the result for this query. We want to build a system similar to the basic version of the Twitter advanced search engine.

Here is one example test query when we enter: "Subject:I Predicate:eat Object:pizza". It will be compiled in SQL as the following query:

```
SELECT * FROM TWEETS
WHERE SUBJECT LIKE 'I'
AND PREDICATE LIKE 'eat'
AND OBJECT LIKE 'pizza'
```

5 Approach

We divide the task to three steps which are included in subsections. We first collect tweets, then clean tweets, extract tuples, store them in a database and answer queries. After these three steps, we execute test queries and use ranking measures to evaluate the results. We use Apache Spark (Zaharia et al., 2012) as our system of choice, owing to its scalability, and the fact that Spark code runs on clusters big and small, with no changes required. For relational queries, we use Spark SQL (Armbrust et al., 2015).

5.1 Data collection

We gathered the Twitter data from Internet Archive. This is an authoritative source for tweets. The data we acquired is Spritzer version of Twitter data dated on 2018-10-01 (arc,). It is a random sample of 1% of tweets for that day. The size is 52 GB with 417633 users.

5.2 Data cleaning and tuple extraction

For data cleaning, we used a tweet cleaner that parses raw tweet data in JSON format into a form ingestible by Spark. This is stored in a Spark SQL DataFrame, with schema as below

$$(userId, userName, userScreenName, \\ userFollowersCount, userFriendsCount, \\ userFavouritesCount, userFollowing, \\ coordinates, pace, retweetCount, \\ entities, urlRank)$$

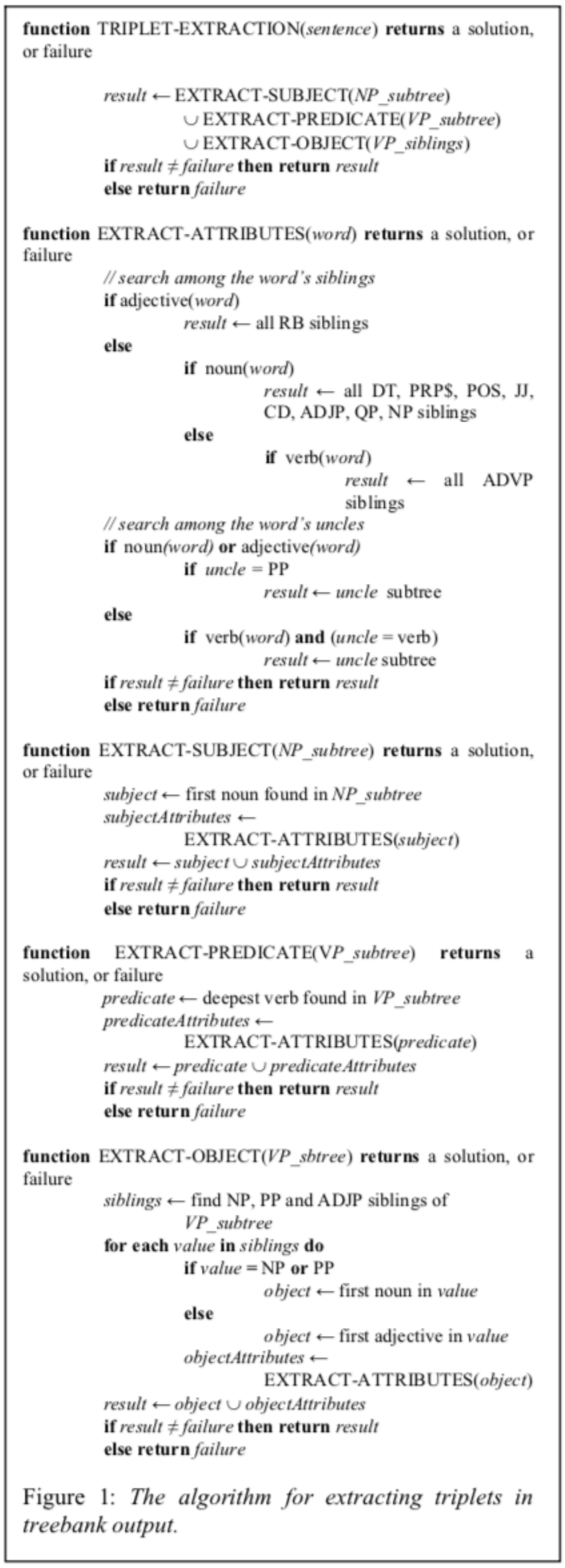
For NLP part, we used SpaCy (Honnibal and Montani, 2017) package in Python to extract triplets into `(tweetId, subject, predicate, object)`.

Our approach involved three steps:

- Split a tweet into its constituent sentences using spacy's sentence splitter.
- Generate a parse tree for the sentence using spacy's internal deep-learning based parser
- Use the parse tree to generate triplets as in the Algorithm in figure 1.
- Lemmatize the verbs in the sentence so that they are ingestible. Lemmatization of a verb converts it into the present tense from other forms.

- Insert triplet into a table with schema $(tweetId, subject, object, predicate)$

Our tuple extraction is based on (Rusu et al., 2007).



```
function TRIPLET-EXTRACTION(sentence) returns a solution, or failure
        result ← EXTRACT-SUBJECT(NP_subtree)
                ∪ EXTRACT-PREDICATE(VP_subtree)
                ∪ EXTRACT-OBJECT(VP_siblings)
        if result ≠ failure then return result
        else return failure

function EXTRACT-ATTRIBUTES(word) returns a solution, or failure
        // search among the word's siblings
        if adjective(word)
                result ← all RB siblings
        else
                if noun(word)
                        result ← all DT, PRP$, POS, JJ, CD, ADJP, QP, NP siblings
                else
                        if verb(word)
                                result ← all ADVP siblings
        // search among the word's uncles
        if noun(word) or adjective(word)
                if uncle = PP
                        result ← uncle subtree
        else
                if verb(word) and (uncle = verb)
                        result ← uncle subtree
        if result ≠ failure then return result
        else return failure

function EXTRACT-SUBJECT(NP_subtree) returns a solution, or failure
        subject ← first noun found in NP_subtree
        subjectAttributes ←
                EXTRACT-ATTRIBUTES(subject)
        result ← subject ∪ subjectAttributes
        if result ≠ failure then return result
        else return failure

function EXTRACT-PREDICATE(VP_subtree) returns a solution, or failure
        predicate ← deepest verb found in VP_subtree
        predicateAttributes ←
                EXTRACT-ATTRIBUTES(predicate)
        result ← predicate ∪ predicateAttributes
        if result ≠ failure then return result
        else return failure

function EXTRACT-OBJECT(VP_sbtree) returns a solution, or failure
        siblings ← find NP, PP and ADJP siblings of
                VP_subtree
        for each value in siblings do
                if value = NP or PP
                        object ← first noun in value
                else
                        object ← first adjective in value
                objectAttributes ←
                        EXTRACT-ATTRIBUTES(object)
        result ← object ∪ objectAttributes
        if result ≠ failure then return result
        else return failure
```

Figure 1: *The algorithm for extracting triplets in treebank output.*

Figure 1: Triplet Extraction Algorithm

5.3 Ranking the results

For ranking results, we implemented a paper (Nagmoti et al., 2010) to rank the results by several different metrics listed below.

– *FollowerRank:* The FollowerRank of a user a is defined as

$$\mathrm{FR}(a) = \frac{i(a)}{i(a)+o(a)}$$

with $i(a)$ being the in-degree of a, i.e. the number of followers of a, and $o(a)$ being the out-degree of a, i.e. the number of users followed by a.

The intuition for this is that a user who has more followers in proportion to the number of people they follow, is 'influential' and hence their tweet should be ranked higher.

– *LengthRank:* The LengthRank of a tweet t w.r.t. a query q is defined as

$$f_{LR}(t,q) = \frac{l(t)}{\max\limits_{s\in\mathcal{T}_q^k} l(s)}$$

with $\mathcal{T}_q^k$ the set of top-k tweets returned for query q, and $l(t)$ and $l(s)$ the length of tweet t and s respectively.

The intuition for this is that a longer tweet is potentially more informative. The denominator is a normalization factor to keep the LengthRank between 0 and 1.

– *URLRank:* Let t be a tweet and q a query, then the URLRank of t w.r.t. q is defined as

$$f_{UR}(t,q) = \begin{cases} c & \text{if } t \text{ contains a URL} \\ 0 & \text{otherwise} \end{cases}$$

where c is a positive constant. In our ranking, we let c equals to 2 because the paper shows empirically it works the best when $c = 2$.

URLRank accords importance to tweets with a URL, because the URL may point to a source of relevant information.

5.3.1 Combining the ranking measures

Following (Nagmoti et al., 2010), we add up the various ranking measures to generate a score for the relevance of a tweet. In general, we could take a linear combination of these measures, but it is unclear as to how to set the weights for this combination. The ranking measures f_{FLR} and f_{FLUR} for tweets are defined as

$$f_{FLR}(t,q) = f_{FR}(t,q) + f_{LR}(t,q)$$
$$f_{FLUR}(t,q) = f_{FLR}(t,q) + f_{UR}(t,q)$$

for all $t \in \mathcal{T}$ and $q \in \mathcal{Q}$.

6 Evaluation

We evaluate our system on 7 classes of possible queries:

– subject only, object only, verb only present

– two out of three present

– all three present

We selected top 3 most frequent tuples in each category and form 21 test queries. This is because we needed enough potential results for each query to meaningfully evaluate our ranking. We used Twitter advanced search as the ground truth and compared our results with results in that search engine.

Our metric of evaluation is precision@5, which corresponds to the number of relevant results present in the top 5 search results. We check if our top 5 results feature in top 500 results of Twitter advanced search. We do this because we only have 1% of the data.

The result is shown in the table below.

Average p@5	0.05
Max p@5	0.4
Min p@5	0.0

6.1 Explaining the results

These results are far from impressive because of a multitude of reasons.

- **Size of Dataset.** The size of our Twitter stream was tiny, and it was highly likely that many of the relevant and important tweets were not present in the dataset. We had computational constraints from the limitations of the Databricks Community Edition.

- **Untuned NLP Models.** We used out-of-the-box NLP models for tuple extraction. This was a suboptimal choice, but we did not have the ability to fine-tune them on account of not having sufficient labeled ranking data.

- **Few meaningful search queries.** The only search queries we could use on our system were those which had a sufficiently large number of rows to rank. Given that the top ranking rows(for which our results are shown) were highly 'generic', with 'i love you' being the most frequent triple, we could not hope for a semantically meaningful ranking of tweets containing 'i love you'. We tried to look at the 80^{th} and 90^{th} percentile of frequent tuples, but these gave nearly zero precision, and are hence not reported here. Lower percentiles had too few tuples for a meaningful ranking performance.

- **Machine Learned Ranking beats static ranking every time.** We had hoped to build a non-machine-learned baseline for ranking tweet search competetive with a learned one. This was an uphill task, but it has been observed that the marginal gain of complex ML models is small when compared against baselines with well-tuned hyperparameters.

7 Challenges

One interesting and challenging aspect of this project is the use of machine learning. Machine learning is used to implement tuple extraction. We do not train models from scratch due to the scope of a class project. Instead, we want to find a proper framework with well-trained models that works with Spark. There are a few options include Stanford Parser, OpenNLP, Link Parser and Minipar in literature (Rusu et al., 2007). However, we spent hours implementing the initial setup and found them incompatible with our project. We tried other open-sourced tools such as ReVerb but it is based in Java, so it does not work under Spark. We also experimented with NLTK (Loper and Bird, 2002), Stanford CoreNLP (Manning et al., 2014) and SparkNLP (spa,) but ran into interfacing issues with Spark. Finally, we decided to implement the machine learning and the NLP module using SpaCy (Honnibal and Montani, 2017).

8 Future steps

There are several steps we can take to further this project, including larger datasets, better test queries to evaluate the results, and more tables to cover more relationships.

8.1 Larger datasets

It is likely we can see better results and higher accuracy if we increase total amount of entries in the database. That is to say, we can improve the ranking if we extract more tuples out of larger datasets. Further, we will be able to get more meaningful search queries to evaluate our system on.

8.2 More tables

Currently, the system generates the same result for the same query, which means the ranking process doesn't involve user information. The idea is to create a database with multiple tables that can represent followers and people each user follows, so we can allow user to enter queries such as "*userId: 123, subject: pizza*". Notice when an user enters the query, the user can specify a $userId$, which will be used along with relationships associated with this $userId$ to rank all results and return the one with the highest ranking. The output will be different for each user. That is to say, we can expand the ranking measures to cover more aspects, such as the social relationships of users.

References

archiveteam-twitter-stream-2018-10. `https://archive.org/details/archiveteam-twitter-stream-2018-10`. Accessed: 2019-05-05.

Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. 2015. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1383–1394. ACM.

Victor Halberg et al. 2012. Tweetrank: An adaptation of the pagerank algorithm to the twitter world. *Search Engines and Information Retrieval*, pages 1–11.

Matthew Honnibal and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*.

Jin-Woo Jeong, Meredith Ringel Morris, Jaime Teevan, and Dan Liebling. 2013. A crowd-powered socially embedded search engine. In *Seventh International AAAI Conference on Weblogs and Social Media*.

Edward Loper and Steven Bird. 2002. Nltk: the natural language toolkit. *arXiv preprint cs/0205028*.

Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. 2010. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

Rinkesh Nagmoti, Ankur Teredesai, and Martine De Cock. 2010. Ranking approaches for microblog search. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Volume 01*, pages 153–157. IEEE Computer Society.

Brendan O'Connor, Michel Krieger, and David Ahn. 2010. Tweetmotif: Exploratory search and topic summarization for twitter. In *Fourth International AAAI Conference on Weblogs and Social Media*.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Alexander Pak and Patrick Paroubek. 2010. Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, pages 1320–1326.

Delia Rusu, Lorand Dali, Blaz Fortuna, Marko Grobelnik, and Dunja Mladenic. 2007. Triplet extraction from sentences. In *Proceedings of the 10th International Multiconference Information Society-IS*, pages 8–12.

spark-nlp, johnsnowlabs. `https://github.com/JohnSnowLabs/spark-nlp`. Accessed: 2019-05-05.

Jaime Teevan, Daniel Ramage, and Merredith Ringel Morris. 2011. # twittersearch: a comparison of microblog search and web search. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 35–44. ACM.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association.