Who to follow on Twitter given topics, using TF-IDF and PageRank

GROUP 7

Anton Lindqvist antoli@kth.se
Anna Lindelöf anna@kth.se
Thiago Lobo thiagol@kth.se
Helder Martins helder@kth.se
Casper Renman casperr@kth.se

Abstract

1 Introduction

With the increase of information available on the internet, information retrieval is an important topic of today. The popularity of social media, such as Facebook and Twitter, has contributed to the explosion of data and a demand for large growing databases. Recommender systems perform retrieval of information deemed relevant or of interest to a user, using relevance feedback, user ratings, user history or other information which may help determine the user's interests. A few topics of the research on content-based recommender systems are documents [7] [6] and user recommendation [9] systems. These aim to quickly extract and present relevant data to a user from large data. Therefore it is essential for enhancing the user's experience on social media or on commercial sites.

1.1 Objective and Specifications

The objective of this paper is to implement a web application of a twitter user recommender system. The application should, given a topic, recommend twitter users to follow. The specifications are as follows. The topic is represented as nouns or adjcetives, or a combination of both. The web application is only required to run locally and the data is modelled in a graph database. Users are ranked using pageranking and tf-idf scores. Lastly, potential synonyms to the words in the search query are generated and presented to the user as a suggestion.

2 Related work

In a previous work, [9], by Subercaze et al and first published in 2015, a twitter user recommender system was also developed which treats the user's combined tweets as the user's representation. However, the extracted keywords of the tweets are not only nouns, but also verbs or adjectives. A co-occurrence matrix is then computed and its values are used to build a graph of the extracted words as the nodes and the probabilities of occurring together with another word as the edges. This representative graph of the user is hashed and compared to previous hashes of other users in order to find the k most similar users to follow. In contrast to this paper, an important focus of the work by Subercaze et al. is scalability, and thus they claim that their system achieves a better performance than using the tf-idf algorithm.

A graphical database it not used, instead it is the hashes of the twitter users that are stored for comparison.

Another previous work is the open source word2vec toolkit published by Google and heavily based on the works [3] and [4] by Mikolov et al. The toolkit provides functionality for word embedding and aims to learn meanings of words using machine learning, which is described in more detail in the Background section. This work is highly relevant to this paper since the generation of synonyms makes use of this toolkit by training a model on a chosen dataset and extracting similar words from the trained model.

3 Method

3.1 Crawling Twitter

The Twitter API [10] was used to fetch tweets related to a limited set of hashtags (Appendix A). The crawler collected 10Gb of data which the group decided was sufficient for the use-case.

3.2 The Neo4j database

Neo4j is a graph database that removes the need to explicitly define a schema for the relationships between entities. It has efficient techniques for storing graphs, making it suitable for storing large amounts of data with many relationships. Neo4j allow us to represent the full structure of the Twitter database as a graph, where each different entity is a *node* and the relationship between the nodes an *edge*. Figure 1 details the nodes and the edges of our recommender engine.

The base node of the graph is the *Tweet*, which is a free-text short message sent through Twitter. Every tweet is posted by a *User*, which is represented by the *post* relationship. A tweet may *mention* another user with the @ special character, and it may also *tag* a topic with #, represented by the node *Hash*. To this basic schema derived directly from Twitter, we added the node *Word* which are all the parsed words in the free text of the Tweet. This node is linked to the rest of the graph through the *contains* and *discusses* relationships, with a property specifying the amount of times this relationship happens for every tweet and user. This new entity allows us to easily represent the user as a *bag-of-words* document of every word that

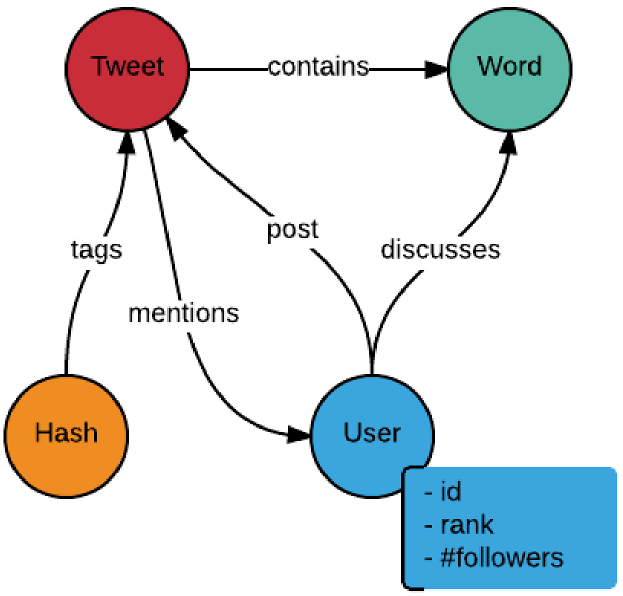


Figure 1: Graph schema used for the Twitter data.

the user discusses in all of the user's tweets, thus allowing us to map the recommendation problem to the standard techniques used on information retrieval.

3.3 Tweet text processing

The topics of the tweets are extracted by parsing their freetext and finding nouns and adjectives. That was an empiric decision. This is done using string processing alongside the Natural Language Toolkit (NLTK) [2] which provides interfaces in Python for classification, tokenization and stemming.

3.3.1 Cleaning tweets

A tweet can contain hyperlinks, hashtags, mentions and other symbols. These are removed in order to properly parse the text of the tweet. Specifically, words starting with *#, @, & or http* are ignored. A few other words that commonly occur in a tweet were also ignored as they would not contribute to the cause. These are *don't, i'll, retweet and rt.*

3.3.2 Extracting topics

First, letters are lowercased and the text is tokenized, then words shorter than three characters are removed, after which ignored symbols are removed. Finally, we use NLTK to Part of Speech-tag [8] each term so that we can pick only `NN`s (nouns) and `JJ`s (adjectives) and stem these terms, which are returned as a list.

3.4 PageRank

One of the most well-known ranking and scoring measures is called PageRank [5]. Made famous by Google in late 90's, its main idea is to use the auxiliary information, mainly the *link structure*, present in the World Wide Web as an *authority measure* of the web pages contained within. Representing the web as a graph were each node is a web page and the edges the links between a page and another, it is intuitive to see that nodes with higher number of *inlinks* (that is, the number of links arriving into a node) are of higher importance than the ones with no inlinks at all, just like a scientific article which is cited by several different sources, for example.

3.4.1 PageRank in the Twitter graph

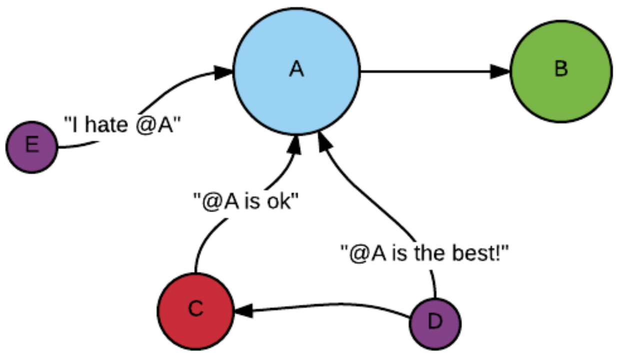


Figure 2: PageRank applied to the Twitter database. Every user is a node, while every mention in tweets is an edge. The size of the node is its relative rank among others.

Although the original PageRank algorithm was modeled with focus on the World Wide Web, its method could be applied to any problem which can be modelled as a graph. Specifically for Twitter, one could see each *user* of the platform as a node and every *mention* in the tweets of a user to another as a link. In the same way that web pages with high number of inlinks have a higher rank, users that are mentioned frequently will be considered more relevant for our recommendation engine, this process can be more clearly seen in Figure 2. Note that we actually do not analyse the content of the tweet, so tweets with positive or negative sentiment will have the same importance for ranking, one could think of it being a "any publicity is good publicity" kind of model.

The original PageRank algorithm considered following an outlink with equal probability among all the possible links. That is reasonable with the unstructured meta information available in the Web today, but is intuitive to reason that, with more information about these users, different probabilities could be applied to each one of them, depending on the task that we have at hand. For a user recommender engine, our approach used the *number of followers* as a good measure of importance. That is, users with high number of followers will be jumped to with higher probability in the random walk,so their score will be naturally higher. Our engine implemented both methods for evaluation, and the results are reported in the Experiments section.

3.4.2 PageRank Monte Carlo

The standard implementation of the PageRank computation is done via a method called power iteration, which involves finding the largest eigenvector of a transition matrix $\mathcal{P}$ composed of the transition probabilities between every web page of the World Wide Web, a process which is done over several iterations until convergence. This method, although popular and still used today by Google, has its drawbacks mainly regarding the speed of convergence, several passes may be needed until the desired precision is obtained. In our approach we explored a relatively new method, which utilize *Monte Carlo algorithms* to estimate the score of the nodes of the graph. As proposed by Avrachenkov et al. [1], the idea is that, if we sample the web page after a sufficient large amount of random walks, a probability distribution could be calculated with an acceptable degree of precision and with a faster convergence rate. While the power iteration method may require more than 50 iterations for an acceptable ranking to be reached, Avrachenkov et al. proposed method has ranks for the import pages after one iteration only.

Of the several different algorithms proposed, our engine implements the *Monte Carlo complete path*, which is detailed in the Algorithm 1. For every user in the Twitter database, we start a random walk beginning in that user and ending when the user is bored of following mentions. We keep track of the total steps of all random walks and how many times each user was visited. A new user is selected to be followed in the walk from all the users the user mentions, which can be done by applying equal probabilities to each one of them or with increased chance for higher number of followers. If a user does not mention anyone, we consider it a *sink* and jump to any other user in the database with the same method. After every user has been at the beginning of the random walk for a set number of walks, we calculate the user rank by dividing the number of times each user was visited over all random walks with the total steps taken.

Algorithm 1 PageRank Monte Carlo, complete path

```
 1: procedure PageRank
 2:     for all walks do
 3:         for all user in users do
 4:             $username \leftarrow user['username']$
 5:             $bored \leftarrow False$
 6:             while $\neg bored$ do
 7:                 $totalSteps \leftarrow totalSteps + 1$
 8:                 $userSteps['username'] \leftarrow userSteps['username'] + 1$
 9:                 $mentions \leftarrow getUserMentions(username)$
10:                 if $mentions \in \emptyset$ then
11:                     $username \leftarrow getRandomUser(users)$
12:                 else
13:                     $username \leftarrow getRandomUser(mentions)$
14:                 $bored \leftarrow isUserBored()$
15:     for all user in users do
16:         $username \leftarrow user['username']$
17:         $ranks['username'] \leftarrow userSteps['username'] \div totalSteps$
18:     return $ranks$
```

3.5 TF-IDF

In order to also consider terms in the search algorithm *TF-IDF* was used, which is a well known solution to the problem of matching (in a ranked way) documents modelled as *bags-of-words*. Each document (including the input query) is represented by a vector of scores, each of which related to one of the possible terms in our dataset. The scores are calculated as follows: $tf_{w,d} * log_{10}(\frac{N}{df_w})$ where $tf_{w,d}$ is the number of times term w appears in document d, N is the total number of documents and df_w is the number of documents term w appears in. Then, *cosine-similarity* is used to compute how close the query is to each of the documents. A link between a *User* node and a *Word* node maps directly to a $tf_{w,d}$ score. The final procedure can be seen in Algorithm 2.

Algorithm 2 TF-IDF in a Graph Database

```
 1: procedure TF-IDF
 2:     $scores \leftarrow \emptyset$
 3:     $sizes \leftarrow \emptyset$
 4:     for token ∈ query do
 5:         $users \leftarrow$ query(users that discuss 'token')
 6:         $df \leftarrow$ length(users)
 7:         $count \leftarrow$ # of occurences of 'token' in 'query'
 8:         $wtq \leftarrow count * log_{10}(\frac{length(documents)}{df})^2$
 9:         for user ∈ users do
10:             $tf \leftarrow$ query(# of times 'user' discusses 'token')
11:             $scores[user] \leftarrow scores[user] + wtq * tf$
12:             $sizes[user] \leftarrow$ query(# of words discussed by 'user')
13:     for user ∈ scores do
14:         $scores[user] \leftarrow \frac{scores[user]}{sizes[user]}$
15:     return $sort(scores)$
```

3.6 Final Score

After retrieving the sets of *PageRank* and *TF-IDF* scores for all users that discuss any query term, they are normalized to zero-average and unitary variance so that they can be mixed together by means of a parameter, α. Then, the final score, for each user u, is: $s_u = \alpha * \bar{s_{p_u}} + (1 - \alpha) * \bar{s_{t_u}}$.

3.7 Word2vec - Generating Synonyms

Word embeddings, also referred to as distributed representations of words, is a method for generating representation of words which capture their meaning and relationships to other words. This is achieved by having each word represented by a real vector, and therefore maps each word into a multi-dimensional continuous space. In accordance to the distributional hypothesis, words which are used in similar contexts, and therefore have similar vector representations, should be mapped close to each other in the vector space. One of the most basic and widely used models is the Continuous Bag of Word (CBOW) model as described in [3], which takes a context of words as input and computes the most likely target word in that context. The context words are encoded as 1-of-V vectors, where V is the size of the vocabulary, and these vectors are then averaged into one final context vector to be multiplied by the trained weight matrix of the network [3].

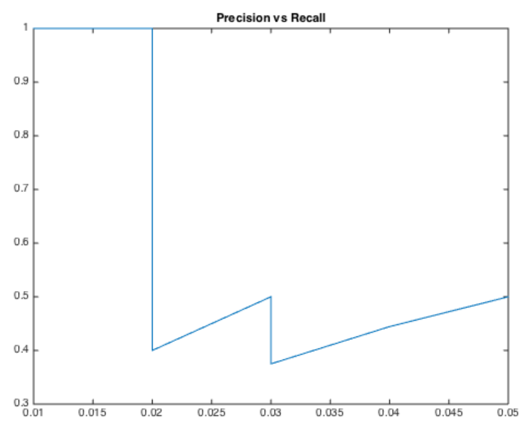
The model is trained using the word2vec toolkit by Google Inc. The dataset used is the text8 file, a 97 567 kB file available from the same source as the toolkit. For the training the CBOW model was run with a word vector size of 200. The window was set to 5, hierarchical softmax was used and 1 thread was run.

To extract the synonyms from the trained model, the cosine and the vocab methods were called on the loaded model. The input words and the output synonyms were also cleaned in the same way as the tweets in order to gain more consistency in relation to the database.

4 Experimental results

In both of our experiments, a toy query, "usa election president politics " was used. We first tried only TF-IDF and then a combined retrieval with rankType = TODO, both of which were compared to combined retrieval with rankType = TODO. We graded the top ten results of each retrieval as relevant or not and plotted a precision vs recall graph for each of them.

4.1 TF-IDF only



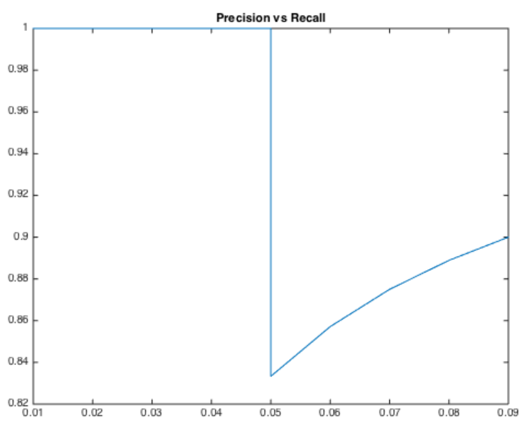


Figure 3: Precision vs Recall for TF-IDF only (left) and combination with $\alpha = 0.5$ and rankType = TODO (right).

As expected, a combination of TF-IDF with PageRank performs better than just the first algorithm. The problem is that our database does not contain *all* the tweets of each user, so the bag of words model is only precise among the crawled topics. If, for instance, a user posts a single tweet about politics and we only crawl that tweet for that user, it's going to appear really well-ranked for TF-IDF while, in reality, it shouldn't. When PageRank is added to the equation we exploit the graph structure of Twitter and take the authority of the aforementioned user in consideration, which highly improves our Precision vs Recall curve, as can be seen in Figure 3.

4.2 PageRank variant

4.3 Evaluation

Summary of what alpha should be and why.

5 Evaluation of the result

6 Summary and Conclusions

References

[1] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM Journal on Numerical Analysis*, 45:890–904, 2007.

[2] Steven Bird. Nltk: the natural language toolkit. pages 69–72, 2006.

[3] Tomas Mokolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[4] Tomas Mokolov, Kai Chen, Greg Corrado, Jeffrey Dean, and Ilya Sutskever. Distributed representations of words and phrases and their compositionality, 2013.

[5] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. *World Wide Web Internet And Web Information Systems*, 54:1–17, 1998.

[6] Simon Philip, P.b. Shola, and Abari Ovye. Application of content-based approach in research paper recommendation system for a digital library. *International Journal of Advanced Computer Science and Applications ijacsa*, 5, 2014.

[7] C. Porcel, J.m. Moreno, and E Herrera-Viedma. A multi-disciplinar recommender system to advice research resources in university digital libraries. *Expert Systems with Applications*, 36:12520–12528, 2009.

[8] Helmut Schmid. Probabilistic part-of-speech tagging using decision trees. 12:44–49, 1994.

[9] Julien Subercaze, Christophe Gravier, and Frédérique Laforest. Real-time, scalable, content-based twitter users recommendation. *Web Intelligence WEB*, 14:17–29, 2016.

[10] INC Twitter. Twitter api, 2016.

7 Appendix

A Hashtags

#FeelTheBern, #Bernie2016, #BernieSanders, #NotMeUs, #Bernie, #UniteBlue, #StillSanders, #NYPrimary, #WIPrimary, #ImWithHer, #Hillary2016, #HillaryClinton, #Hillary, #Trump2016, #MakeAmericaGreatAgain, #TrumpTrain, #Trump, #tcot, #AlwaysTrump, #TeamTrump, #WakeUpAmerica, #ccot, #TeaParty, #DonaldTrump, #PJNET, #elections2016, #vote, #cir, #USlatino, #AINF, #Latinos, #GOP, #2016Election.