Institut für Informationssysteme
Abteilung für Verteilte Systeme
VU Advanced Internet Computing (184.269)

Project Topic 4 – Analysing Big Data

Introduction

Your task in this project is to develop an application for analysing relationships in a social network. Assume that you are operating a clone of the popular Twitter Web service via status.net[1]. As your users are not paying in money for your service (it is free as in "free beer"), you need to follow the age-old Web 2.0 idiom "the user is the product" to monetize. In this vain, your task is to mine the various relationships of users in your system, so as to be able to send them better targeted advertisements.

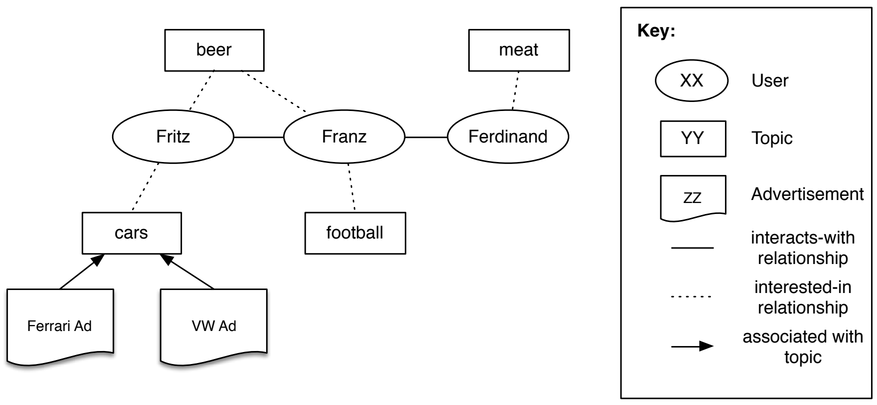

Figure 1: Basic Model of Interactions and Interests

Assume a basic model of interactions and user interests as exemplified in Figure 1. Essentially, there are users which you think are interested in given topics. These relationships are established by analysing the content of the users' tweets. Additionally, users interact with other users. Different types of relationships can exist in Twitter, e.g., users can follow other users, can communicate with them via replies, can retweet other users' tweets, can "favorite" tweets, or can block other users. Define which interactions are important for you, and whether you want to weight connections. You can assume that users are also interested in topics that their "friends" are interested in, even though maybe to a lesser degree. This property of social networks is known as "homophily" [3]. Finally, you know about ads or campaigns that are associated with topics. Your goal is to find out which ads to suggest to which user based their own and their friends' interests.

The core of your project should be to build up and analyse a database of "big data" [2], i.e., a data store that is ultimately too large to process using traditional database technology. To this end, you should build up an infrastructure consisting of a combination of various NoSQL databases [1], which allows you to quickly access user data and do on-the-fly graph analysis. More concretely, your solution should integrate (1) a traditional SQL database for managing users and user metadata, (2) a document store for storing the actual ads and campaigns, and (3) a graph

[1] `http://status.net`

database such as Neo4J[2] for storing interactions between users. Integrate all these different data management approaches in a web application for ad recommendation. You can use your choice of programming language (e.g., Ruby on Rails, or Java JSF) to implement this web application.

Outcomes

The expected outcomes of this project are two-fold: (1) the actual project solution, (2) two presentations of the results.

Project Solution

The project has to be hosted on a Git repository hosted by the Distributed Systems Group [3]. You will get instructions how to apply for such a repository at the lab kickoff meeting. Every member of your team is assigned a separate Git account. We will check who has contributed to the source code, so please make sure you use your own account when submitting code to the repository. Further it is required to provide an easy-to-follow README that details how to deploy, start and test the solution.

For the submission (see below), you also have to create a virtual machine (Linux or Windows as a vmdk file), which hosts your implemented solution preconfigured with some test data and a short README how to access your implementation. There are different options how to provide the virtual machine: Online, via an external hard drive or on a CD/DVD. For the deadline, see below.

Presentations

There are two presentations. The first presentation is during the mid-term meetings, and covers at least the tasks of Stage 1 (see below), the second presentation is during the final meetings and contains all your results. The actual dates have been announced in TISS.

Every member of your team is required to present in either the first *or* the second presentation. Each presentation needs to consist of a slides part and a demo of your tool. Think carefully about how you are actually going to demonstrate your tool, as this will be part of the grading. You have 20 minutes per presentation (strict). We recommend to use only a small part of the presentation for the slides part (e.g., 5 minutes) and to clearly focus on the presentation of your results.

Grading

A maximum of 60 points are awarded in total for the project. Of this, up to 30 points are awarded for the tool (taking into account both quality and creativity), 20 points are awarded for the presentations (taking into account content, quality of slides, presentation skills, and discussion), and 10 points are awarded for individual accomplishments during the presentations and contributions towards the implementation work – the latter includes teamwork aspects.

A strict policy is applied regarding plagiarism. Plagiarism in the source code will lead to 0 points for the particular student who has implemented this part of the code. If more than one group member plagiarizes, this may lead to further penalties, i.e., 0 points for the tool.

Deadline

The hard deadline for the project is **January 31st, 2016**. Please submit deployment instructions and the presentations as a single ZIP file via TUWEL. The submission system will close at 18:00 sharp. The only exception is the virtual machine, which has to be provided during office hours on **February 1st, 2016** (from 16:00-18:00).

[2] `http://www.neo4j.org`
[3] `http://hyde.infosys.tuwien.ac.at`

Test Cloud Infrastructure

If you need (additional) Cloud resources to deploy and test your solution, please make use of Amazon Web Services[4] (AWS). AWS provides free resources to students, so we recommend that you sign up for these credits at `https://aws.amazon.com/education/awseducate/`. If for some reason you cannot sign up (e.g., since you do not own a credit card) or the free credits are not sufficient, please send an email to `aic@dsg.tuwien.ac.at`; we will then provide you with user credentials. However, it is more convenient if you get your own account and credits.

Stage 1

In the first stage, the focus should be set on architecting your solution, setting up your basic infrastructure, and getting a significant amount of test data into your system. At the end of Stage 1, you should have the basic model from Figure 1 implemented.

Tasks:

1. Start by designing an architecture of your solution. Choose a relational database management system, a document store, and a graph database, and decide which programming language you will use to "glue" those technologies together.

2. Populate your databases with the test data. Use the Twitter API[5] to retrieve a non-trivial amount of real data for testing (a few gigabytes at least). Make sure to monitor specifically high-traffic accounts, so as to get highly connected tweets. Start this task early, as you will need to monitor Twitter over multiple days to get a significant data set of interconnected tweets.

3. You will need to enrich the Twitter data set with additional information, such as ads and campaigns. Define topics and populate your document store with ads matching these topics. Then analyse your Twitter data to find out which users are interested in which topics, and store this information in your graph database. You can use a very simple algorithm to mine interests from tweets, e.g., you can assume every user that mentioned a certain topic or product more than x times is interested in it (clearly, finding out what matters to people is more difficult in practice, see for instance [4]).

4. Further, mine connections between users (*interacts-with relationships*) from your Twitter data set, and store them in your graph database as well.

Stage 2

In Stage 2, it is now time to actually analyse the database you have built up in Stage 1. Build a web application that allows to trigger the following queries. Queries should be executed in real-time, and should be parameterizable from the web application:

1. Which users are the most influential persons in your data set? Influential persons do not only have many followers, they also trigger many retweets and favourites.

2. Which users are interested in a broad range of topics, which are more focused? Keep in mind that simply counting topics may be too limiting, as users that tweet a lot will naturally also mention more topics. Porting basic concepts from the "term frequency-inverse document frequency" (or similar approaches) may help you here.

[4]`http://aws.amazon.com`
[5]`https://dev.twitter.com/docs/api`

3. Suggest concrete ads from your database to a given user based on her *existing* interests, i.e., topics the user is already mentioning actively.
4. Suggest concrete ads from your database to a given user based on her *potential* interests, i.e., topics the user does not actively mention at the moment, but which are her connections (and their friends, and their friends, . . . , with decreasing importance) are interested in.

Tasks:

1. Build a web application using a technology of your choice that serves as the frontend of your database. The web application should visualize the database in a meaningful way, and allows to trigger and parameterize the queries mentioned above.
2. When implementing the queries, make sure to keep performance in mind. Queries should be executed "in the database" to the extent possible. Make sure to specifically use the capabilities of your graph database to execute efficient queries on your connected data.

References

[1] Jing Han, E. Haihong, Guan Le, and Jian Du. Survey on NoSQL database. In *6th International Conference on Pervasive Computing and Applications (ICPCA 2011)*, pages 363–366. IEEE, 2011.

[2] Adam Jacobs. The pathologies of big data. *Communications of the ACM*, 52(8):36–44, 2009.

[3] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27(1):415–444, 2001.

[4] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. Twitterrank: finding topic-sensitive influential twitterers. In *Third ACM International Conference on Web Search and Data Mining (WSDM '10)*, pages 261–270. ACM, 2010.