Universität Innsbruck

Department of Computer Science
Research Group Quality Engineering

MASTER THESIS

The Title

Martin urek

Supervisor: Ass.-Prof. Dr. Michael Felderer

Innsbruck, June 2, 2018

Contents

Introduction

Proposed framework

proposedFramework

Social Media

Social media is a term referring to online communication channels meant for social interaction, content-sharing and collaboration. Over time, this term became widely used, its exact definition somewhat blurred and if really wanted, most of the today's websites could be labeled as social media. In context of this thesis, I refer to social media in its original meaning. To be considered a social media platform, most of following features usually need to be fulfilled:

- **User accounts** - platform allows users to create and run their own accounts that they can log into. These are online representations of their owners and serve as a too to reach and interact with other users.
- **Profile pages** - pages which represent an individual, might it be a real person, group of people or company. It should contain several personal information about the user like bio, profile picture or other personal data.
- **Friends, followers, groups** - list of accounts whose owners have some form of a relationship or common interest with the user.
- **News feeds** - Area where all new content from other connected entities appears.

Even if a platform fulfills these requirements, it doesn't necessarily have to be classified as social networking platform as pointed out in Haewoon Kwak's paper[1].

Potential of social media data mining

Social media are changing the way that information is passed across societies and around the world.[2] Among many other potentials of social media, there is a huge amount of data generated on daily basis. These data carry lots of real world data and if used correctly, can offer deep insight into almost any area. The process of analyzing these data and searching for repetitive reoccurring patterns with goal of predicting future trends is also called social media mining. Successful mining can not only save money and time spent on getting the data in more traditional way like surveys but can also provide crucial factor in planning or decision making of businesses. Although internet is one big hole and contains lot of false facts and desinformation, there is indication that social networks tend to favour valid information over rumours.[3]

Getting data

As you will have a chance to see, big social networking platforms started realizing that data they own are a golden egg and getting raw full data from them got much more difficult than it was in the early years of social media age.

Twitter: To get the data from Twitter, I first tried to use the Twitter API but I very early got to know that Twitter is well aware of the worth of their data. They don't provide tweets older than 2 weeks what basically made their API inapplicable for my purposes. The only way how to get historical Twitter data is actually:

- collect them over time
- buy them from Twitter
- buy them from other companies who collect Twitter dumps over time

Therefore to obtain my data, I had to use the Twitter Search Api. To do this I used the project (https://github.com/Jefferson-Henrique/GetOldTweets-python) which simplifies work with Search Api to several basic command line calls. Using this technique, I managed to get the significant amount of data needed for all my tasks in this thesis.

Facebook: I originally planned to use Facebook statuses as a big part of analyzed data. Sadly, this was not possible since Facebook Graph API doesn't allow post searching feature. There was this option till early 2014 with Facebook API 1.x versions but since Graph API has been introduced, there's no way how to make Facebook application send requests to 1.x versions of API. At first, application created before 2014 were still working on top of the early Api versions and it has been maintained but over time, all the applications were migrated to 2.x Api versions. There's currently to public way how to freely get the Facebook posts data. The other types provided by the Api are for example user, page, event, group or place.

Reddit: Despite that reddit doesn't offer big amount of user data in OSS projects subreddits, I thought getting and working with Reddit could increase the variety of users and the data which I will be working with. To get the data I used Python PRAW library used to directly work with Reddit Api.

Stack overflow: To extract the questions about the OSS projects of my interest I once again used provided Api.Python module called StackApi offers a way how to communicate with various Stack Exchange Api endpoints - answers, badges, comments, posts, questions, tags and users. Stack Exchange is limited on 30 requests per second what caused the process of getting the data to take much more time since program execution needed to be regularly stopped to avoid the SO throttle violation and from it resulting penalization. At first I intended to use the type posts which returns both, questions and answers, but later I've realized how huge amount of data SO contains. The average count of questions using one of the examined projects names as a tag was around 150 000. Because of this, I had to find out how to filter just the questions with higher probability of talking about bugs. Since the questions on SO do not have any labels which I could use to my advantage like I did with Git issues, I decided to keep just the questions which mention a word bug. This still left me with considerable big dataset of X questions to work with. Out of all properties of questions retrieved from questions endpoint, I decided to store and further work just with several of them mainly its title and body since these two provide most of the semantic meaning.

Open-source projects

There are loads of OSS projects nowadays and it turned out to be pretty interesting process to choose the correct ones for my project. To make sure chosen projects fit into my work and fit all my needs I defined several requirements which needed to be fulfilled at least to some extent:

- Project needs to be widely used and well-known. This ensures there will be enough data on social media about it what will result in the less biased final results.
- Project has to have accessible bug tracking system or Git repository with list of known issues. This will provide the data for pairing the social media data with their corresponding bug items.
- Ideally, projects could be from the same area to avoid coincidentally choosing an outlier project from some either popular or unpopular field.

After considering these three points, I ended up choosing several open source web development frameworks as my projects of interest. Project of my choice are NodeJS, AngularJS, EmberJS, VueJS for frontend technologies and Laravel, Symfony and CakePhp for backend PHP technologies. Some frameworks like Django, Meteor, React have been left out because of their misleading names would require lot of additional work to filter out data unrelated to the actual frameworks. For example, when tested Django framework, most of the twitter data referred to movie "Django Unchained".

Other very interesting group of OSS project to examine are cryptocurrencies. Being a very hot topic these days, I've decided to work with some of the most popular cryptocurrency repositories as well. These were Bitcoin, Ethereum, Litecoin, Dash and Ripple.

Sentiment analysis

Sentiment analysis and opinion mining is the field of study that analyzes people's opinions, sentiments, evaluations, attitudes, and emotions from written language.[?] It's also known as emotion AI or opinion mining. The main and basic task of this field is to correctly classify the polarity of a particular text and evaluate whether is it positive, negative or in some cases neutral. It can be used in almost any situation where data need to be analyzed for its sentiment aspect - for example product reviews, online discussions or social media content.

Sentiment analysis can be divided into several standalone steps:

1. **Data collection** - involves all the substeps required to gather user-generated content from any source. Surveys, blogs, various forums and social media. These all contains huge amount of real people from real world with real experiences. These data are always expressed completely different - using slang, shortcuts, internet language or generally just being used in different context.

2. **Data preprocessing** - this step represents cleaning the data. Data preprocessing can often have a significant impact on generalization performance of a supervised ML algorithms [?] and if there is much irrelevant information and data are unreliable, then knowledge discovery during the training phase is more difficult. It removes irrelevant content which could potentially lead to bad and incorrect results. This is a very delicate step because it manipulates the raw data and if not done correctly, it can easily change results.

3. **Sentiment detection** - this step basically stands for training of the classifier. Subjective feelings and expressions are highlighted, emphasized and retained while objective information like facts are discarded and ignored.

4. **Sentiment classification** - subjective expressions are classified.

5. **Output interpretation** - graphical presentation of obtained results. Time and sentiment can be analyzed to construct various charts, graphs, timelines and many other metrices.

History

Cross-correlation between releases and sentiment change

When working with time series, we often want to determine whether one series causes changes in another. To find this relationship, measuring a cross-correlation and finding a lag is one way how to do it. Lag represents when change in one data series transfers to the other several periods later.

To ensure a cross-correlation calculation makes sense, first I have to determine, whether are the data stationary. A stationary time series is one whose properties do not depend on the time at which the series is observed[4]. More precisely, if y_t is a stationary time series, then for all *s*, the distribution of *$(y_t,, y_{t+s})$* does not depend on *t*.

To determine whether my data are stationary, I've used the Dickey-Fuller test method of tseries package in R. Results can be seen in the table 0.1 and 0.2

Stationarity test of web frameworks sentiment data		
Framework	Dickey-Fuller	p-value
NodeJS	-2.6775	0.2964
AngularJS	-3.883	0.0199
EmberJS	-4.0783	0.0199
VueJS	-3.438	0.0646
CakePHP	-3.480	0.04847
Laravel	-2.57	0.3431
Symfony	-4.3979	0.01

Table 0.1: Stationarity test of sentiment

Stationarity test of web frameworks release count		
Framework	Dickey-Fuller	p-value
NodeJS	-2.896	0.205
AngularJS	-2.547	0.353
EmberJS	-3.297	0.0802
VueJS	-2.158	0.511
CakePHP	-3.224	0.08915
Laravel	-2.368	0.425
Symfony	-2.218	0.488

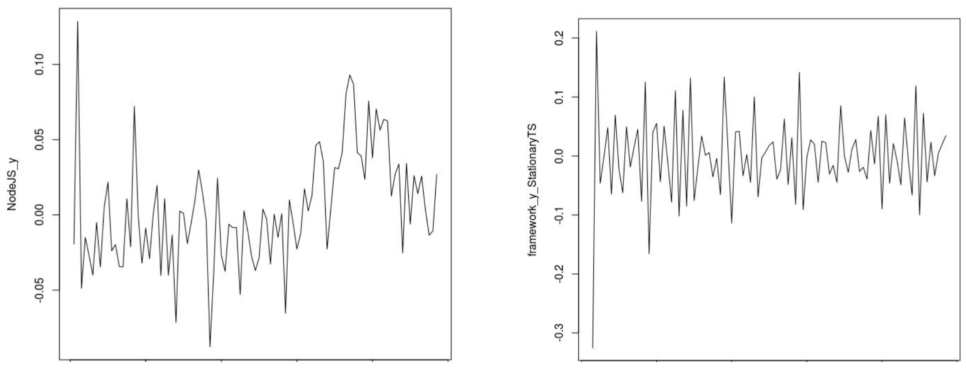
Table 0.2: Stationarity test of release counts

As we can see, p-values are always higher than 0.05 what indicates non-stationarity of the data, therefore I can't calculate the cross-correlation on them in this state. To transform non-stationary data into stationary, 2 approaches can be used. These are differencing and transforming. I've taken data series and differenced the values in listing 1. I've executed both, seasonal differencing and stationary differencing although seasonal probably was not needed because the data should not be dependant on the season.

Listing 1: Used differencing method in R

```
Differencing <- function(x,y)
{
 framework_x_seasdiff <- diff(x,differences=1)  # seasonal differencing
 framework_x_Stationary <- diff(framework_x_seasdiff, differences= 1)
 framework_y_seasdiff <- diff(y, differences=1)
 framework_y_Stationary <- diff(framework_y_seasdiff, differences= 1)
 return(list(framework_x_Stationary,framework_y_Stationary))
}
```

New differenced values do appear to be stationary in mean and variance, as the level and the variance of the series stays roughly constant over time. Sentiment for NodeJS before and after differencing can be seen in Figure 0.1



(a) Before differencing (b) After differencing

Figure 0.1: NodeJS monthly sentiment values

Same procedure needed to be done with the "number of releases per month" data and afterwards. Then, cross-correlation could be executed. For this task I've used ccf method in R which implements Pearson's correlation calculation method. Results for all 7 OSS projects can be seen in Figure 0.2

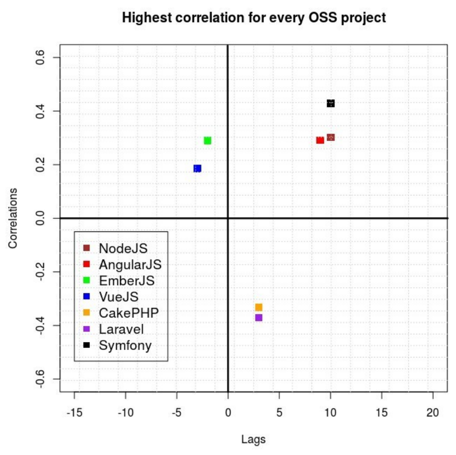


Figure 0.2: Highest correlations for every OSS project

Results interpretation: As we can there is no general pattern. Maximal project correlations happen to occupy 3 of 4 possible quadrants. Each quadrant represents a different relationship between number of releases and sentiment change.

- **I. Quadrant**(Positive correlation + positive lag) - Increase of release count increases a sentiment

- **II. Quadrant**(Positive correlation + negative lag) - Increase of sentiment increases a release count

- **III. Quadrant**(Negative correlation + positive lag) - Increase of release count decreases a sentiment

- **IV. Quadrant**(Negative correlation + Negative lag) - Increase of sentiment decreases a release count

Also, in Figure 0.3 are the results without making the data series stationary. It's obvious that making the data stationary has a big impact on the results.

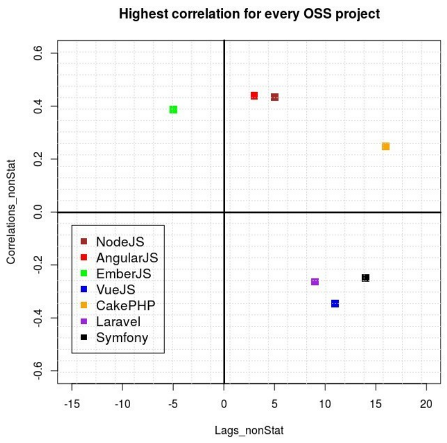


Figure 0.3: Highest correlations for every OSS project (non-stationary)

Commits count within releases

Initially, I thought that to modify project to take into account a size of the release (amount of commits) will be pretty straightforward task. It actually was straightforward, but as always I've encountered several unexpected problems on the way.

I intended to extend my previously used method which uses Git Api tags endpoint to get the release dates. Unfortunately I wasn't able to find number of commits in the returned objects. JSON object returned from API has following structure:

```
{
  "url": X,
  "assets_url": X,
  "upload_url": X,
  "html_url": X,
  "id": X,
  "tag_name": X,
  "target_commitish":X,
  "name": X,
  "draft": X,
  "author":{},
  "prerelease": X,
  "created_at": X,
  "published_at": X,
  "assets":[],
  "tarball_url": X,
  "zipball_url":X,
  }
```

I've done some extra searching but didn't want to spend extra time so I've decided to go the way I knew will work. Instead of using Api to get the commit counts, I've crawled Github UI page of each release and extracted information directly from page source code.

Each release details page provides information how many commits behind the current HEAD the commit is. The difference in this number between two following releases represents count of new commits for a release. Results of simple tabular substraction with spreadsheet formula needed to be manually corrected because projects often release several branches parallel and therefore substraction from the previous release was not always the correct one.

Eventually, I got correct number of commits for every release and could execute the same cross-correlation analysis described in the previous chapter, but this time instead of releases count, I've explored relationship between sentiment and commits count. One possible flaw in the commit count data are the pre-releases. I treated them as normal releases because they do offer new features but those very same commits are then counted in the official relaeses later on.

After getting the data ready I performed a stationarity test for commit counts. Sentiment values ar the same as before with count of releases. Results can be seen in table 0.3

Stationarity test of web frameworks commit counts		
Framework	Dickey-Fuller	p-value
NodeJS	-7.0239	0.01
AngularJS	-2.547	0.3531
EmberJS	-3.2764	0.0831
VueJS	-2.9748	0.1886
CakePHP	-3.655	0.03283
Laravel	-2.919	0.2084
Symfony	-4.8461	0.01

Table 0.3: Stationarity test of commit counts

I see that there are again several data series (AngularJS, EmberJS, VueJS, Laravel + NodeJS because of unstationarity of sentiment data) which are not stationary so exactly as before with release counts, I had to transform the data. After that, Pearson's cross correlation was calculated. Results for all 7 OSS projects can be seen in Figure 0.4

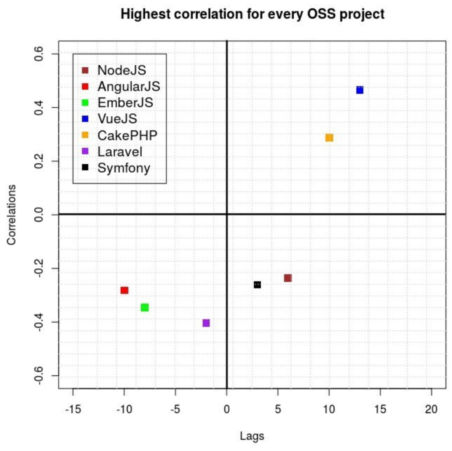


Figure 0.4: Highest correlations for every OSS project

If I would skip the step of making the data stationary, results would again look completely different 0.5.

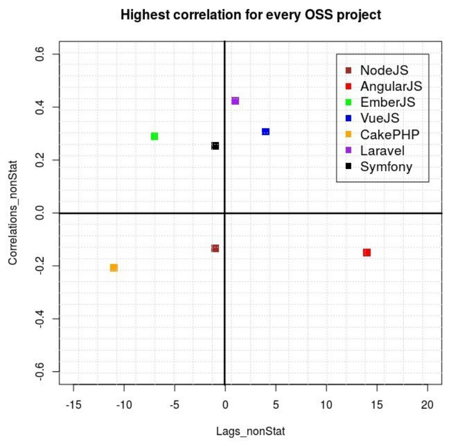


Figure 0.5: Highest correlations for every OSS project (non-stationary)

Pairing bugs

pairingbugs

The content goes here...

Bibliography

[1] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM, 2010.

[2] Antony Mayfield. What is social media. 2008.

[3] Carlos Castillo, Marcelo Mendoza, and Barbara Poblete. Information credibility on twitter. In *Proceedings of the 20th international conference on World wide web*, pages 675–684. ACM, 2011.

[4] Rob J Hyndman, George Athanasopoulos, Slava Razbash, Drew Schmidt, Zhenyu Zhou, Yousaf Khan, Christoph Bergmeir, and Earo Wang. forecast: Forecasting functions for time series and linear models, 2013. *R package version*, 5.