Universität Innsbruck

Department of Computer Science
Research Group Quality Engineering

MASTER THESIS

The Title

Martin urek

Supervisor: Ass.-Prof. Dr. Michael Felderer

Innsbruck, May 27, 2018

Contents

Introduction

Proposed framework

proposedFramework

Social Media

socialMedia

Open-source projects

ossProjects

Sentiment analysis

sentimentAnalysis

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[1]. 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

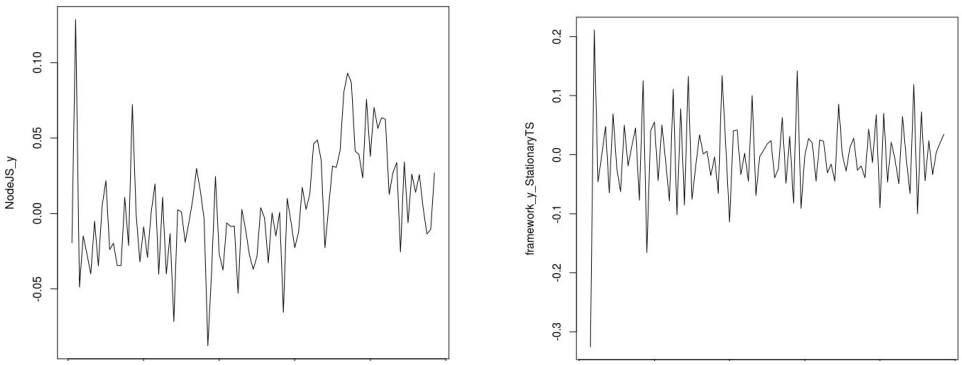
Table 0.2: Stationarity test of release counts

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

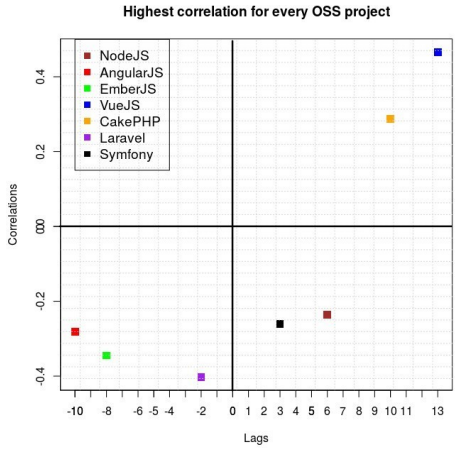


Figure 0.2: Highest correlations for every OSS project

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.

Pairing bugs

pairingbugs

The content goes here...

Bibliography

[1] 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.