Managing open-source projects' manpower efficiently:

Forecasting the number of daily issues on GitHub repositories

https://github.com/Repo-liveliness-forecasting

Aditya Utama Wijaya
NTHU – IMBA - 104077429
adityautamawijaya@gmail.com

Wen Lee
NTHU – CS -104062515
katyprogrammer@gmail.com

Cindy Soh
NCTU – GMBA - 0453037
hi.cind@gmail.com

Renaud Jollet De Lorenzo
NTHU – ISA -104065436
renaudjollet@gmail.com

V K Sanjeed
NTHU – IMBA -104077453
sanjeed27@gmail.com

This project has been conduct for the Business Analytics using Forecasting course given at National Tsing Hua University (NTHU) Institute of Service Science (ISS) by Professor Galit Shmueli.

Executive Summary

More business and government organizations developing IT solutions now use open-source repositories because of their reliability and rapid development. The bedrock of an open-source project is the community that uses, maintains, and creates new applications from it. This is because the more people who can see and test the code, the more likely any flaws will be caught and fixed quickly. Therefore, it becomes crucial for the foundation hosting the repository, to manage the massive number of issues submitted by users on a daily basis. Thus, forecasts of upcoming issues are valuable to open-source foundations that need to manage their manpower and resources to resolve issues efficiently,

Open-Source repositories are highly tested and maintained pieces of software that are used in most IT projects to hasten development. We collected secondary data - number of daily issues and commits - from five such repositories on Github, then forecast their respective number of daily issues for the next three weeks using a variety of time series forecasting techniques. By evaluating the predictive performance metrics, and forecast time-plots, we selected the best forecasting techniques for each repository to generate ensemble forecasts. Our forecasts are on average about 16% more accurate than the seasonal naive benchmark, and capture the important elements in the series including trend, day-of-week seasonality and autocorrelation.

We recommend our forecasting method to repositories with higher volume of daily issues, as for them, our 16% greater accuracy translates to a large number of issues going unresolved on a daily basis. Foundations should allocate more manpower to a repository with higher forecasted issues in the upcoming three weeks.

I. Problem Description

GitHub is the most popular project hosting platform today which allows developers to collaborate on open-source or private repositories. Since GitHub is an open-source platform highly dependent on the community who will report and solve issues as well, community contribution is critical. There are companies that will build a repository for certain projects on GitHub and let millions of users to contribute on it to produce a more robust project. In general, open-source software development will work according to this cycle:

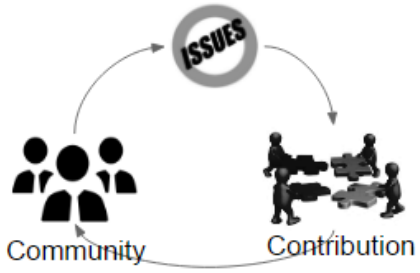


Figure 1. The Open-Source Software Development Cycle

When new developers join the community they may first encounter issues, like a non-working or missing feature and report them through Github issues. The community or the host company developers solve these issues, creating a feedback loop that makes the project more robust and accessible, and grows the community. However, as unsolved issues pile up, the feedback loop is disrupted since users detect there is no contribution ecosystem anymore. This leads to community leaving the repository, and hence the need for a company (or foundation on GitHub) to allocate sufficient manpower in advance to those repositories with more reported issues.

Thus, our goal is to forecast the number of issues on certain repositories three weeks ahead, as a proof-of-concept that our method can help companies allocate their manpower efficiently in advance. The rationale for a three weeks ahead forecast is that it costs time for one programmer to switch from one project to another. Instead of pulling a programmer from one task to another daily, it's better to allocate them to the repository three weeks before in order to warm up before the storm of issues arrives.

II. Data Description

Our data source is the GitHub API Repository statistics and we collect the total number of issues authored by the contributors per day. We use the last 52 weeks (1 year) of issues data from the five different repositories from more than 19 Million active repositories on GitHub. We choose large repositories maintained by large companies or foundations that allocate professional developers to maintain it. Below are the sample data of the five repositories (see Appendix A for repository descriptions):

Table 1 Five Repositories Sample Data

Date	*Spark* *No. of issues*	*Swift* *No. of issues*	*Julia* *No. of issues*	*Node* *No. of issues*	*TensorFlow* *No. of issues*
2016/12/09	16	26	8	23	29
2016/12/10	8	16	5	9	18
2016/12/11	3	21	7	9	11
2016/12/12	10	27	9	13	12
2016/12/13	13	20	19	17	25
2016/12/14	11	24	14	12	27
2016/12/15	15	35	14	24	20

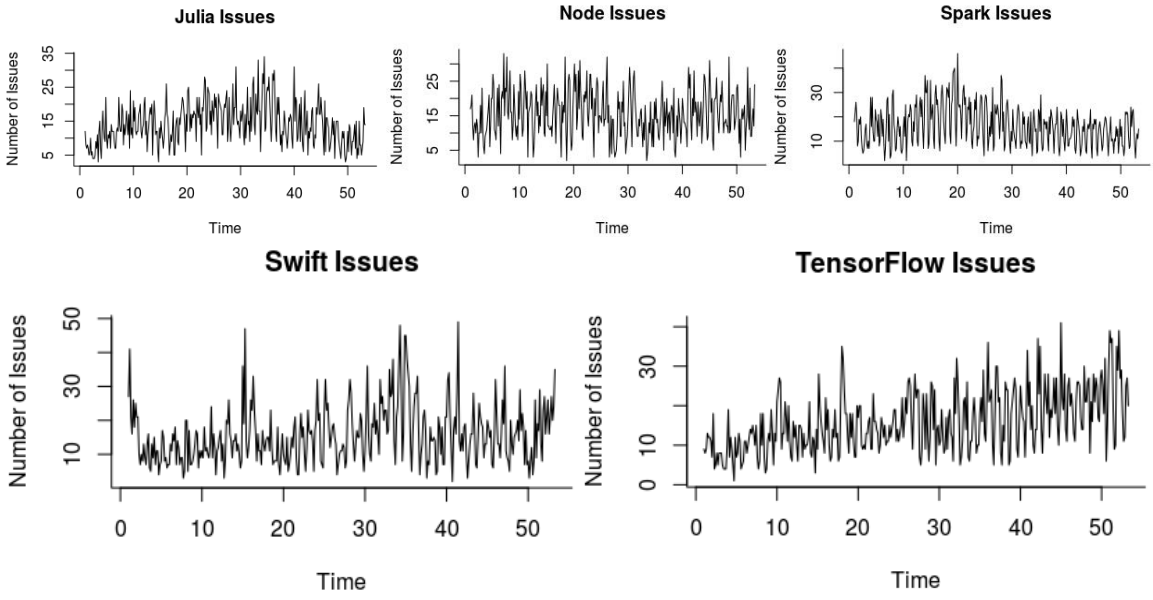


Figure 2 Time plots of five series' Number of Issues

III. Data Preparation

We extracted the number of issues on Github API from the five repositories, then aggregated the data from unix timestamp to daily timestamp. Time-series components1 (trend, seasonality, noise) were determined by visual inspection of the issues time plots. We applied training period: + 315 days; validation period: 21 days, then experimented with all the forecasting methods that fit to the time-series (see flowchart in Appendix B). Additionally, we applied roll-forward training period so we have 28 samples with different training period and validation period with the same length – 21 days (see Appendix C). Performance metrics such as RMSE, MAPE, and validation period forecast error plots were determined using forecast package in R (see Appendices D and E for our codes).

IV. Forecasting Solution

In terms of RMSE and MAPE scores in validation period, the Ensemble method (assembling between Holt Winters and Multiplicative linear regression) consistently ranks among the top three performing methods for all five series (see Appendix F). Additionally, the ensemble method can improve on the seasonal naive benchmark at least by ~4% and on average ~16%.

V. Time-plots with future forecasts

Figure 3 shows forecasts for the 21 days' period in blue as well as the forecast intervals in a lighter blue shades, across training and validation periods.

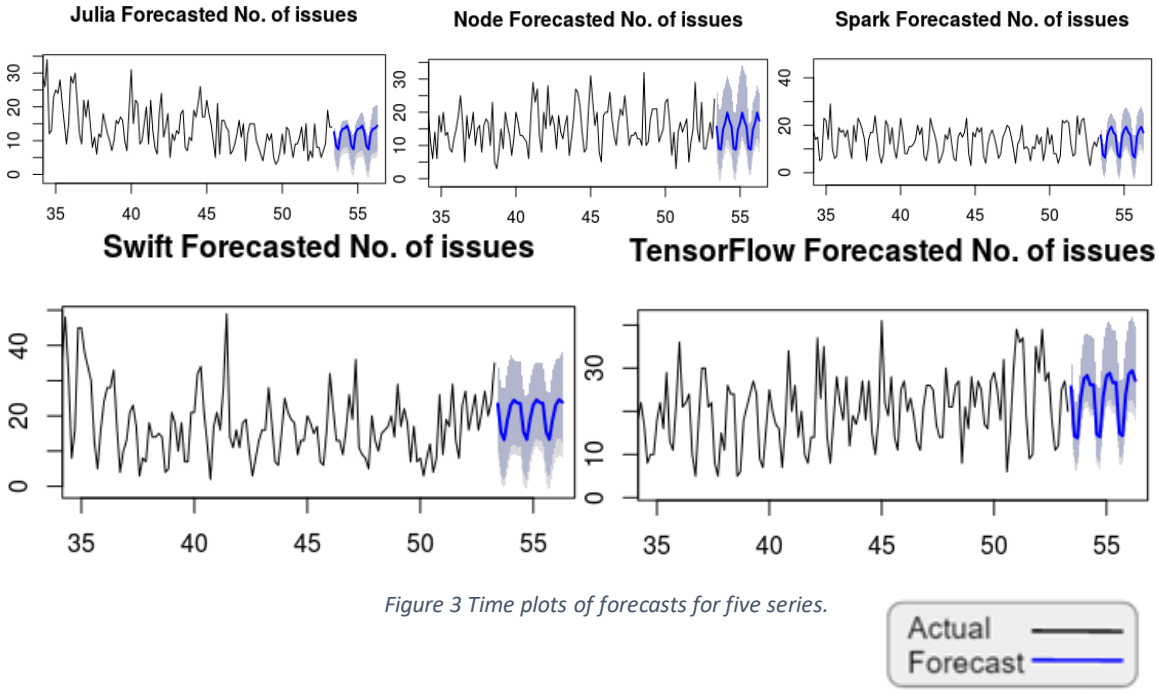


Figure 3 Time plots of forecasts for five series.

[1] Removed the outlier data on Dec 1st on Node.Js time series

VI. Conclusion

To summarize, we learn that clearly defining the business goal and forecast goal in the first place is critical. Clearly determining the time-series components for each series is necessary to determine the most optimum forecasts method.

Table 2 Recommendations

Advantages	*Drawbacks*
Our model can improve by 16% against seasonal naive forecasts Tried out all of the forecasting methods that can generate a more optimal forecast results. Our model can capture the trend and seasonality, as well as autocorrelation to determine whether a series is a random walk or not.	There are large prediction intervals that might not produce most optimal point forecast. Low daily issues on 5 repositories might not produce a significant impact on how many people to be allocated

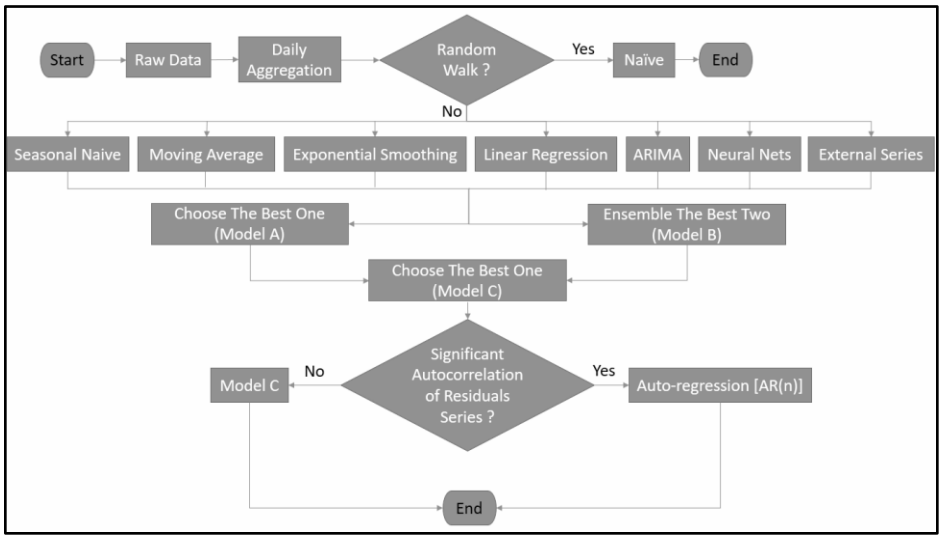
Table 3 The Chosen Methods for Each Repository

	Nodjes	*Spark*	*Tensor flow*	*Swift*	*Julia*
Trend / Seasonality	Seasonality, no trend	Seasonality, no trend	No seasonality, trend	No seasonality, trend	Trend and seasonality
Chosen Methods	Ensemble of Holt-Winters and Multiplicative Linear Regression				

VI. Appendix

Foundation / Corporation	*Repository*	*Description*
Apache	Spark	Large-scale data-processing engine
Apple	Swift	Programming language for iOS / OSX
NumFOCUS	Julia	Programming language for Data Analysis
Node.js	Node	Programming language to use Javascript on server-side
Google	TensorFlow	Machine-learning framework

A. Data Description



B. The Steps of Data Forecasting

C. Create 28 training and validation samples

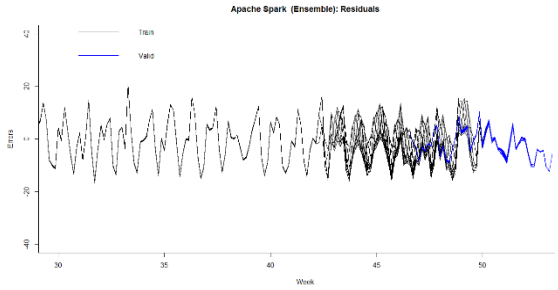
D. The code for getting the daily issues and commits: https://github.com/Repo-liveliness-forecasting/github_api

E. The code for pre-processing the time series, trying out different methods and plotting out the prediction and residuals: https://github.com/Repo-liveliness-forecasting/github_forcasting

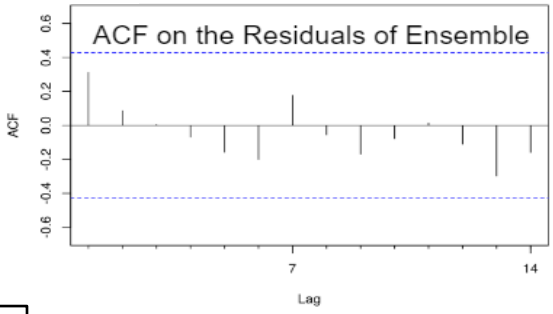
	Tensorflow		Spark		Julia		Swift		Nodejs	
	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE
Naïve	7.74	48.38	8.78	57.87	6.64	42.81	8.60	53.42	9.13	58.05
	10.00	47.87	7.13	60.10	5.39	65.10	10.22	82.14	7.97	60.43
Seasonal Naïve	7.13	43.18	6.59	36.95	6.44	39.95	8.95	54.37	9.01	52.53
	7.45	30.80	5.22	40.58	4.74	50.77	10.49	87.74	7.76	53.50
Holt Winter's	5.23	35.27	5.41	32.32	4.75	31.67	6.72	41.83	6.37	40.27
	5.80	26.26	**4.64**	**34.78**	**3.94**	**47.21**	8.26	71.94	5.75	38.80
Moving Average	6.81	38.75	6.87	39.76	6.22	36.53	8.69	52.01	8.30	52.62
	9.69	39.10	6.09	46.89	7.66	91.64	13.61	115.13	10.29	73.11
Linear Regression (Additive)	5.25	35.14	6.03	35.38	5.48	39.40	7.24	44.74	6.44	42.42
	6.28	27.92	6.19	51.18	6.34	87.43	8.78	89.72	**5.58**	35.17
Linear Regression (Multiplicative)	5.21	32.05	6.13	32.96	5.59	36.79	7.36	39.34	6.58	38.14
	5.84	**26.07**	5.50	44.55	5.42	74.00	8.39	79.51	5.72	**33.31**
Neural Networks	5.47	38.70	5.43	34.43	4.81	34.28	6.26	42.80	6.15	43.85
	8.20	38.35	5.71	55.55	5.80	82.66	**8.00**	**67.87**	7.61	63.91
External Info	5.15	34.85	5.27	31.68	5.00	34.53	6.91	42.49	6.40	42.01
	6.63	29.66	5.33	39.73	6.72	92.15	9.05	92.34	5.61	35.67
Ensemble	8.91	61.13	10.58	79.39	6.79	43.72	9.95	67.83	8.85	63.78
(Holt Winter's + Mult. Linear Reg.	**5.79**	26.13	4.90	39.12	4.47	58.69	8.25	75.25	5.62	34.81
Improvement (Snaive -> Ensemble)	**22%**	**15%**	**6%**	**4%**	**6%**	**16%**	**21%**	**14%**	**28%**	**35%**

F. *Applied Methods and Performance metrics.* Notes: RMSE/MAPE values for Training and Validation periods can be found in each method's first and second rows respectively. Values highlighted in Pink indicate top three RMSE/MAPE scores for each series.

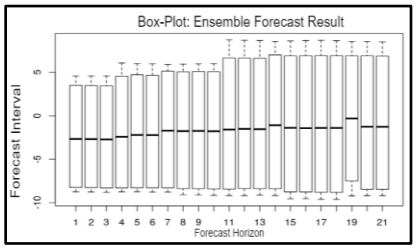
(i)



(ii)



(iii)



Amongst the five time-series, the Apache_Spark series shows a more consistent result across methods. (i) shows the residuals plot over training and validation period of the series. We also fit the ACF on the residuals of the ensemble as it performed the best among others (ii). In (iii) as the forecast horizon expands, the forecast interval also widens.

G. Apache Spark forecast result details (i) Residual plot, (ii) ACF plot, (iii) Box – Plot: Forecast intervals of Ensemble method forecast results.