Data Driven Project Management: Predicting the Development Time

Marko Prelevikj
Faculty of Computer and Information Science
University of Ljubljana
ID: 63130345, Email: mp2638@stuent.uni-lj.si

***Abstract*—We are helping project managers with the issue of time estimation. To do so, we modelled the development time based from project tasks from a real-world case. Several ML methods were trained on the model data out of which SVM delivered the best results. We explored the model with the SHAP local explainability method to rank the model features by their importance. With the feature ranking, we helped the project managers with their time estimation task.**

1. Introduction

The project manager's (PM) main task is to break down the project they oversee into smaller tasks which are not very complex. Once the project is broken down into pieces the PM needs to answer the following questions for each task: (1) how much time the task is going to take to develop; and (2) which project member is the best fit for the task.

In this paper we are focusing on the task of estimating the time required to develop a given task. We use data provided from a company's JIRA portal, which keeps record of the project's tasks. We made 4 different models of the time required to develop a task, where we changed the time span of development time, ranging from all time down to a maximum of 10 days. We evaluated the variations of the model using 4 distinct methods: Naive Bayes, Random Forests, SVM, all provided by Scikit-Learn [1], and additionally XGBoost [2]. In the end we uncover which are the most important features of the task which should be considered when estimating the development time for which we used the SHAP [3] local explainability method.

2. Model data

The data used to build the model was extracted from a company's JIRA portal. We are taking into account all the tasks which have been resolved and have at some point been in development. The total number of such tasks amounts to 2935. All of the tasks are described by their categorical features: *type*, *priority*, *components* of the project they affect, and *labels* which are specific for the project. Due to their high cardinality, the values of the *components* and *labels* features have been filtered such that there are only left values which have at least 50 entries in the dataset. The filtered values are used in their one-hot-encoded form to reduce the complexity of the model. Additionally, we take into account the *number of linked issues* which is a discrete feature denoting the number of tasks which are in a relation with the observed task.

Table 1: Dataset characteristics.

Statistic	All	1Q	1M	10D
count	2935.000	2902.000	2775.000	2451.000
mean	205.252	149.210	96.165	55.762
std	678.103	293.573	132.257	57.019
min	2.000	2.000	2.000	2.000
25%	15.000	15.000	13.000	11.000
50%	49.000	48.000	44.000	33.000
75%	148.000	143.000	121.000	83.000
max	15003.000	2260.000	754.000	228.000

Due to improper usage of JIRA, there is some noise in the data which causes a strong bias towards low values of the predicted time to develop. To reduce this effect we have filtered out all tasks which have development time lower than $2h$. To further reduce the variance in our dataset we have decided to limit the upper bound of the development time. We have done so in 4 different stages to measure the effect of the variance on our model: 1) there is no upper bound, 2) the upper bound is 1 quarter ($\approx$ 90 days), 3) the upper bound is 1 month ($\approx$ 30 days), and 4) the upper bound is 10 days. The summary of the datasets is shown in Table 1, where the first column associates to the general statics of all the data at our disposal. From Table 1 we can observe that the majority of the tasks ($\approx 83\%$ of them) had their development done in less than 10 days.

3. Quality evaluation

We evaluated the variations of the model data using 4 distinct methods: Naive Bayes, Random Forests, SVM, and XGBoost. For each of these methods we did not tune any of the available parameters in order to improve the performance, but rather used recommended parameters available in the online documentation of the method implementations respectively [1], [2]. The observed statistics of model quality are Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and R^2. The evaluations of the listed methods are shown in Table **??**.

Table 2: Performance of the listed methods on the variations of the dataset.

Dataset	Method	RMSE	MAE	R^2
All	XGBoost	**791.428**	254.770	-0.042
	Naive Bayes	948.649	420.036	-0.498
	Random Forest	840.027	256.399	-0.174
	SVM	792.344	**191.267**	-0.045
1Q	XGBoost	**286.835**	160.587	0.018
	Naive Bayes	577.520	389.602	-2.982
	Random Forest	361.223	179.258	-0.558
	SVM	302.772	**124.272**	-0.095
1M	XGBoost	**138.034**	92.616	-0.048
	Naive Bayes	255.083	213.834	-2.579
	Random Forest	178.280	109.402	-0.748
	SVM	143.368	**78.249**	-0.131
10D	XGBoost	**56.406**	43.847	0.041
	Naive Bayes	120.824	106.642	-3.398
	Random Forest	77.243	56.629	-0.797
	SVM	60.460	**41.976**	-0.101

Our goal is to reduce the MAE and RMSE measures which express the fitness of the predicted values to the target values. We can observe this as the variance across the datasets decreases. For each of the dataset variants we had at least one method which has a lower MAE value than the mean value of the data. And both of the measures are monotonously decreasing as the variance in the data decreases, which implies that they are correlated.

The R^2 measure expresses how much we have improved over the mean value of the model. The built model has failed to show that there is any improvement over the mean model. This can be seen from Table **??** where the R^2 model is either very close to 0 or negative in some of the cases. This suggests that we cannot get a very accurate prediction of the target variable, *hours of development*. One of the reasons why is the low number of samples we have at our disposal for training and testing. However, the low R^2 value does not prevent us from exploring which features are the most important [4], [5].

4. Model Explainability

We inspected the feature importance using a local explainability method which provides a clear explanation of the model output to the end user, i.e. which of the features contributed to the final decision and to what extent. In our case, we used the SHAP [3] method, which visualizes the decisions the underlying model has made using force plots, presented in Figure **??**.

The force plots visualize the magnitude of the forces of each feature values which combined together provide the final output of the model. For an example, we can take the visualization from Figure 1b where the `issueType=1` is the main factor for prolonging the duration of the development, whereas the `numberOfLabels=0` is the main factor for reducing it. We can observe that `analytics=1` is consistently delaying the development in both of the shown examples. This kind of information is very insightful for PMs and it makes their job much easier.

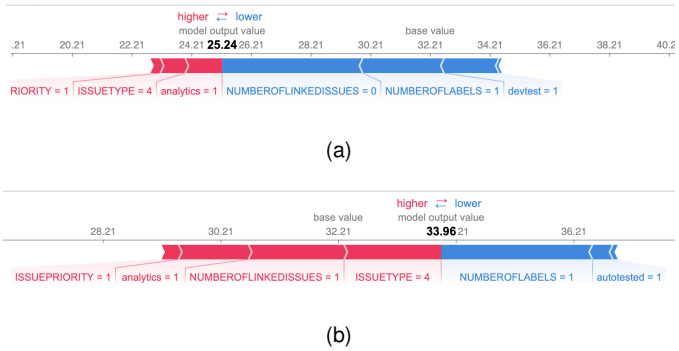

Figure 1: Force plots visualizing the decision of the model. The visualized values are log-probabilities.

By combining all instances of the model together (e.g. averaging them) we can calculate the global impact of each feature. In our case the most impactful features based on the model trained by the SVM method are: *number of linked issues*, *issue type*, *number of labels* and *issue priority*. In combination with individual observations, these are the main pieces of information which PMs need to account for when estimating the development time of a particular task.

5. Conclusion

In conclusion, we addressed one of the biggest problems PMs are facing on a daily basis. To do so, we built a model based on the data stored in JIRA, which consists of the the task priority, type, the components of the product it affects, their meta labels and the number of other tasks it depends on. The original data has a lot of noise, which we reduced by limiting the maximum development time we were taking into account. We trained a number of different methods on the purified model, out of which SVM provided the best results. Using the built model we were able to identify the most important features of the underlying tasks which affect the development time and therefore ease the task of time estimation for the PMs.

References

[1] F. Pedregosa and e. a. Varoquaux, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[2] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[3] S. M. Lundberg and e. a. Erion, "From local explanations to global understanding with explainable ai for trees," *Nature Machine Intelligence*, vol. 2, no. 1, pp. 2522–5839, 2020.

[4] "Regression Analysis: How Do I Interpret R-squared and Assess the Goodness-of-Fit?" library Catalog: blog.minitab.com. [Online]. Available: https://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit

[5] M. B. Editor, "How to Interpret a Regression Model with Low R-squared and Low P values." [Online]. Available: https://blog.minitab.com/blog/adventures-in-statistics-2/how-to-interpret-a-regression-model-with-low-r-squared-and-low-p-values