

FIT3162

Final Project Report

Designing software effort estimation model using machine learning techniques

Jovan Ong Shung Jiet

29274311

jong0016@student.monash.edu

Benjamin Marc Wijayaratne

30955246

bwij0006@student.monash.edu

Wong Zi Qi

30897548

zwon0015@student.monash.edu

Team: MCS17

Supervisor: Dr Golnoush Abaei

Word count: 7567

Table of contents

Table of contents	2
1. Introduction	4
2. Project Background	5
2.1 Background	5
2.2 Literature Review	5
2.2.1 Introduction	5
2.2.2 Content	6
2.2.2.1 Ensemble Learning	6
2.2.2.2 Hyperparameters Tuning	8
2.2.3 Updated Synthesis Matrix	9
2.2.4 Conclusion	10
3. Outcomes	11
3.1 What has been implemented	11
3.1.1 Dataset retrieval	11
3.1.2 Data preprocessing	11
3.1.3 Development of local code	11
3.1.4 Evaluation of other ML models	12
3.1.5 Web application integration	12
3.1.6 Hyperparameter tuning	13
3.2 Product delivered	13
3.3 How are requirements met	14
3.4 Justification of the decisions made	16
3.4.1 Dataset preprocessing	16
3.4.1.1 Preprocessing method	16
3.4.1.2 Removing year end feature of traditional dataset	16
3.4.2 ML algorithm	16
3.4.3 Web application	17
3.4.3.1 Project methodology	17
3.4.3.2 Preset feature field and warning messages	17
3.5 Discussion of all results	17
3.5.1 Performance of ML model	17
3.5.1.1 Performance Evaluation Methodology	17
3.5.1.2 Results evaluation	18
3.5.2 User interface	18
3.6 Limitations of project outcomes	19
3.6.1 Dataset	19
3.6.2 ML model	19
3.6.3 Performance optimization	19
3.7 Discussion of possible improvements and future works	20

3.7.1 Pre-processing and feature extraction	20
3.7.2 Preset web application feature fields	20
3.7.3 Further ML algorithm research	21
3.7.4 Further Hyperparameter tuning research	21
4. Methodology	21
4.1 Design	21
4.1.1 Research Process	21
4.1.2 System Architecture	23
4.1.3 Comparison to Initial Plan from Project Proposal	25
4.1.4 Frontend Integration	25
4.2 Software and Tools Used in Implementation	26
5. Software Deliverables	28
5.1 Summary	28
5.2 Software Qualities	30
5.2.1 Robustness	30
5.2.2 Security	30
5.2.3 Usability	30
5.2.4 Scalability	30
5.2.5 Documentation and Maintainability	31
6. Critical Discussion	31
7. Conclusion	32
8. Appendix	33
8.1 Code appendix	33
8.2 Team member contributions	34
9. References	34

1. Introduction

This is the Final Report for the team MCS 17 for our FIT3162 (Computer Science Project) Semester 2, 2022. In this report, we will be recording and documenting what we have been able to accomplish and how we have been able to accomplish these things throughout our two semesters together as a team.

All of our team members and our project supervisor are listed on the cover page of this report. We have never worked together before this project, but we came together at the beginning of 2022 to work on this project when we were randomly assigned to this group together. Although things started slow, we seemed to have endless chemistry and boundless synergy that allowed us to fuel each other into achieving just about everything we set forth to do. Call it a stroke of luck or a twist of fate, regardless we have made it and we have done what we set forth to accomplish. This report is a formal documentation of everything that we have managed to accomplish throughout these two semesters.

2. Project Background

2.1 Background

Software development is a rather lucrative industry currently in this modern day and age and has become a rather rapidly growing industry with millions of companies globally with millions of projects from millions of customers all over the world.

With things being in this state, it has become rather difficult for software developers as they struggle to choose between several different projects available and are ultimately unable to tell just how much would have to be committed in order to complete such a project and hence will be unable to determine how much they should gain or how much value such a project may garner.

Furthermore, the accuracy of estimation is essential for reducing the likelihood of a project failure. As software projects become more complex, traditional parametric models and statistical methods often fail to depict the correlation between the project features and the software effort (Suresh Kumar et al., 2021). Machine Learning techniques have been proposed to help tackle the accuracy issue, as they possess effective learning capabilities to adapt to a wide range of variations that a software project is subject to (Das et al., 2015).

This is where our project comes in. The objective of our project is to produce an application that is able to predict just how much effort is required of a software development company in order to be able to complete a project, and is more effective than state-of-the-art models using machine learning techniques. This would enable companies to properly budget their time and also properly value their projects according to how much effort must be put into it against how much the customer is willing to pay for it. Our project also aims to build a web-based tool with a user interface that prioritises user-friendliness by having a customizable template to reflect on project management approaches.

2.2 Literature Review

2.2.1 Introduction

In the past decades, extensive studies have been put into applying various machine learning (ML) approaches to improve the accuracy of effort prediction. One of the challenges faced in the ML aspect is to define which effort estimators can achieve the best results. Any effort estimator may move up or down in the ranking since comparisons of estimators are conducted based on modified conditions. From the previous literature review, we have discussed the effectiveness of Ensemble Learning in resolving this limitation through the integration of multiple learning algorithms. With ensemble learning, the software effort estimation process can be improved to yield better accuracy in effort prediction (Kocaguneli et al., 2012).

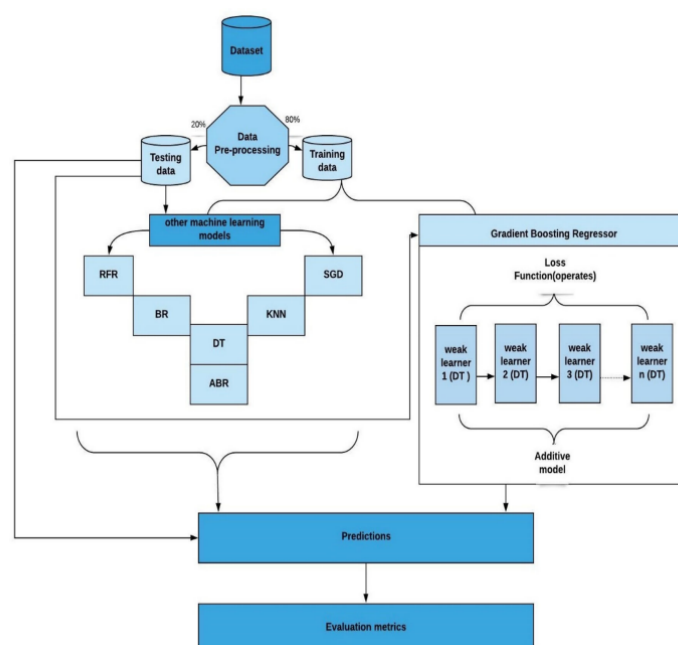
In this literature review, we hope to re-examine the relevant studies that we referenced in implementing our system.

2.2.2 Content

2.2.2.1 Ensemble Learning

Recently, extensive attention from software effort estimation communities has been on ensemble learning techniques as they consistently outperform single learning techniques. Groups of ML algorithms are trained and integrated to carry out a similar operation, thus enhancing predictive performance.

One of the ensemble learning algorithms introduced was Gradient Boosting regressor (GBR) published by Suresh Kumar et al. (2021). The proposed GBR, which is an abstraction of gradient boosting employs an optimised loss function and DT as its weak learners to enable the addition of outputs, the appending of the next immediate model outputs, and the rectification of the residuals in the predictions. In this study, they compared various algorithms, such as Stochastic Gradient Descent (SGD), K-nearest neighbour (KNN), Decision Tree (DT), Bagging regressor (BR), Random Forest regressor (RFR), AdaBoost regressor (ABR) with their proposed Gradient Boosting regressor (GBR). Simulations were carried out by using COCOMO'81 and CHINA datasets. The statistical results concluded that the proposed GBR outperformed the other models in terms of lower MAE, MSE, and RMSE, and higher R2 overall.



But

Fig 2.2.2.1.1: Proposed framework by Suresh Kumar et al. (2021)

In agile software development projects, precise effort estimation can aid in sprint planning, which produces the best outcomes. Zia et al. (2012) presented a regression model that bases its estimation on estimation on User Stories. The data on which they executed are from 21 agile software development projects. The regression model was built to accommodate most of the characteristics of Agile approaches, especially Adaption and Iteration, to tackle the various challenges that Agilests confront.

An improved agile model was presented by Satapathy and Rath (2017) who applied the story point approach proposed by Zia et al. (2012) along with ML algorithms to estimate the effort required for building software projects using Agile methodology. An attempt had been made to improve the results of the story point approach using DT, SGB, and RF algorithms over the same story point dataset. Validation and comparison of the outcomes were conducted with the regression model proposed by Zia et al. (2012). They found that the SGB technique outperformed other ML techniques and the existing regression model based on three metrics, MMRE, MdMRE, and PRED (x).

As discussed above, the predictive performance of models can further be enhanced with the use of ensemble learning. A similar study was published by Malgonde and Chari (2019), who proposed an ensemble-based model that produces accurate effort estimation in agile software development projects to aid in sprint planning. They experimented with seven predictive algorithms, comprising Bayesian network (BN), Ridge Regression (RR), Artificial Neural Networks (ANN), SVM, DT, KNN, and Ordinary Least Squares Regression (OLS), to find the one that provides the better accuracy. Despite their testing, neither approach consistently outperformed the others. They, therefore, suggested an ensemble-based model which employs RR, ANN, SVM, DT, and KNN as its components. Simulations were executed on data from 24 agile software development projects. The results revealed the ensemble-based model outperformed other approaches in terms of MAE and MBE.

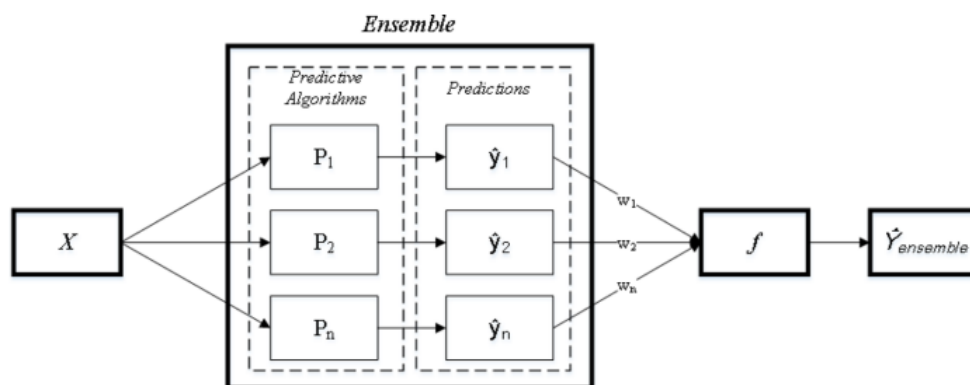


Fig 2.2.2.1.2: Ensemble architecture proposed by Malgonde and Chari (2019)

2.2.2.2 Hyperparameters Tuning

Any machine learning models need to be tuned so that we can adapt them to the problem at hand, thus enhancing the models' performance. According to Feurer et al. (2019), a machine learning model's ability to produce more accurate results depends heavily on its ability to tune its hyperparameters.

Palaniswamy and Venkatesan (2021) proposed a methodology that employed a stacking ensemble. Stacking ensemble uses a meta-learner to integrate diverse fundamental machine learning algorithms. The hyperparameter tuning of base learners and a meta learner was then conducted using PSO and Genetic Algorithms (GA). These algorithms search through the immense hyperparameter space to determine the optimal hyperparameter values for the ensemble model. The authors selected four base learners for the stacking ensemble which were LR, MLP, RFR, and ABR. The authors also employed SVR as the meta learner with radial basis function (RBF) kernel. The evaluation metrics used were MAE, MMRE, and PRED(x). If MMRE is less than 0.25 and PRED (25) is greater than 0.75, the model would then be considered satisfactory in terms of accuracy. Besides, two sets of experiments were carried out on the ISBSG dataset to forecast the accuracy of software effort estimation, namely without hyperparameters tuning, and with hyperparameters tuning using PSO and GA individually. They found that the stacking ensemble with hyperparameters tuning outperformed the ensemble without tuning in terms of lower MAE, lower MMRE, and higher PRED(x), and the PSO method yielded slightly better performance than GA in terms of accuracy.

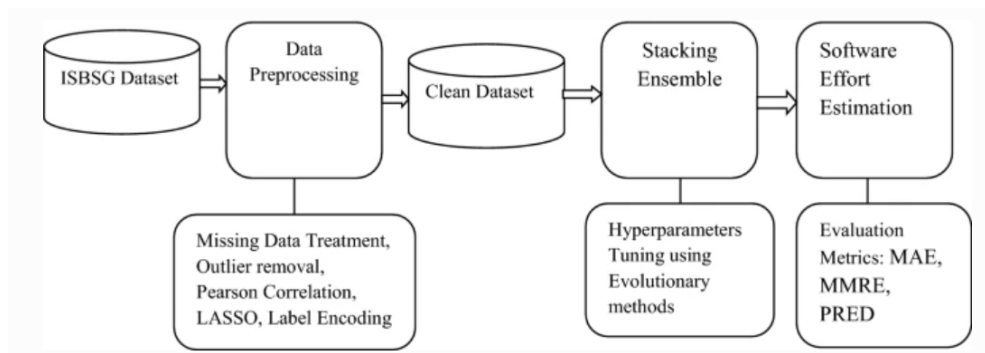


Fig 2.2.2.2.1: Process diagram proposed by Palaniswamy and Venkatesan (2021)

2.2.3 Updated Synthesis Matrix

Author	Dataset	Methodology	Evaluation Metric
Suresh Kumar et al. (2021)	1) COCOMO'81 2) CHINA	SGD, KNN, DT, BR, RFR, ABR, GBR	1) MAE 2) MSE 3) RMSE 4) R^2
Zia et al. (2012)	21 software projects developed by 6 software houses	Regression	1) MMRE 2) PRED(x)
Satapathy and Rath (2017)	21 software projects developed by 6 software houses	DT, SGB, RF	1) MMRE 2) MdMRE 3) PRED (x)
Malgonde and Chari (2019)	24 agile software development projects	Predictive model: BN, RR, ANN, SVM, DT, KNN, OLS Ensemble model: SVM, ANN, KNN, DT, RR	1) MAE 2) MBE 3) RMSE
Palaniswamy and Venkatesan (2021)	ISBSG	- Stacking ensemble - Base learners: LR, MLP, RFR, and ABR - Meta learner: SVR - Hyperparameter tuning: PSO, GA	1) MAE 2) MMRE 3) PRED(x).

Table 2.2.3 Updated Synthesis Matrix

2.2.4 Conclusion

Based on our findings, we acknowledge that there exist various successful and accurate approaches to our topic. We can also conclude that ensemble learning, which outperformed other ML models in the research discussed above, is currently the best approach for predicting software effort in the ML aspect. Moreover, it is also observed that the ensemble model with hyperparameters tuned yields better predictive performance than the one without. In our project, we will present a comparative study between 4 ensemble-based models, including BR, RFR, ABR, and GBR. The model that performs the best will be selected for hyperparameters tuning using cross-validation with the grid search method. Hopefully, with the use of our software effort estimation tool, companies or project managers can accurately estimate software project effort, resulting in better project planning, effective resources use, better contract commitments, and contented and productive project teams.

(refer to Literature Review from Project Proposal)

3. Outcomes

3.1 What has been implemented

3.1.1 Dataset retrieval

The very first step the team has taken into the implementation stage was dataset retrieval. As the project requirement was to implement software effort estimation for both traditional and agile type of projects and the type of features required to compute the estimation of software effort of both project methodologies differ, several different datasets were required for training and testing the models for both project methodologies. After thorough research, the team has decided the main dataset for traditional project methodology is the Desharnais dataset that is open-source and widely used for software effort estimation for traditional projects as suggested by De Carvalho et. al. (2021). As for agile project methodology, the team initially decided on JIRA datasets but due to the dataset not being publicly available, the team has implemented training and testing on the dataset that was used by Zia et al. (2012).

3.1.2 Data preprocessing

The datasets retrieved as stated from the section above are then preprocessed as the next step of the team's implementation stage. For the traditional project methodology, the dataset is manually preprocessed to remove any invalid data that may heavily impact the effort estimated for the projects. Then, the team also decided on the features to be extracted from the dataset and decided to remove a feature that is not a requirement for users to input. As for the agile project methodology, the dataset is also manually preprocessed for the removal of invalid and missing data. The team have implemented such preprocessing methods to ensure that the final model will be able to train on the datasets with no errors and to also be able to estimate the software effort required as accurately as possible.

	A	B	C	D	E	F	G	H
1	TeamExp	ManagerExp	YearEnd	Length	Effort	Transactions	Entities	PointsNonAdjust
2	1	4	85	12	5152	253	52	305
3	0	0	86	4	5635	197	124	321
4	4	4	85	1	805	40	60	100
5	0	0	86	5	3829	200	119	319
6	0	0	86	4	2149	140	94	234
7	0	0	86	4	2821	97	89	186
8	2	1	85	9	2569	119	42	161
9	1	2	83	13	3913	186	52	238
10	3	1	85	12	7854	172	88	260
11	3	4	83	4	2422	78	38	116

Figure 3.1.2. Sample of preprocessed traditional dataset.

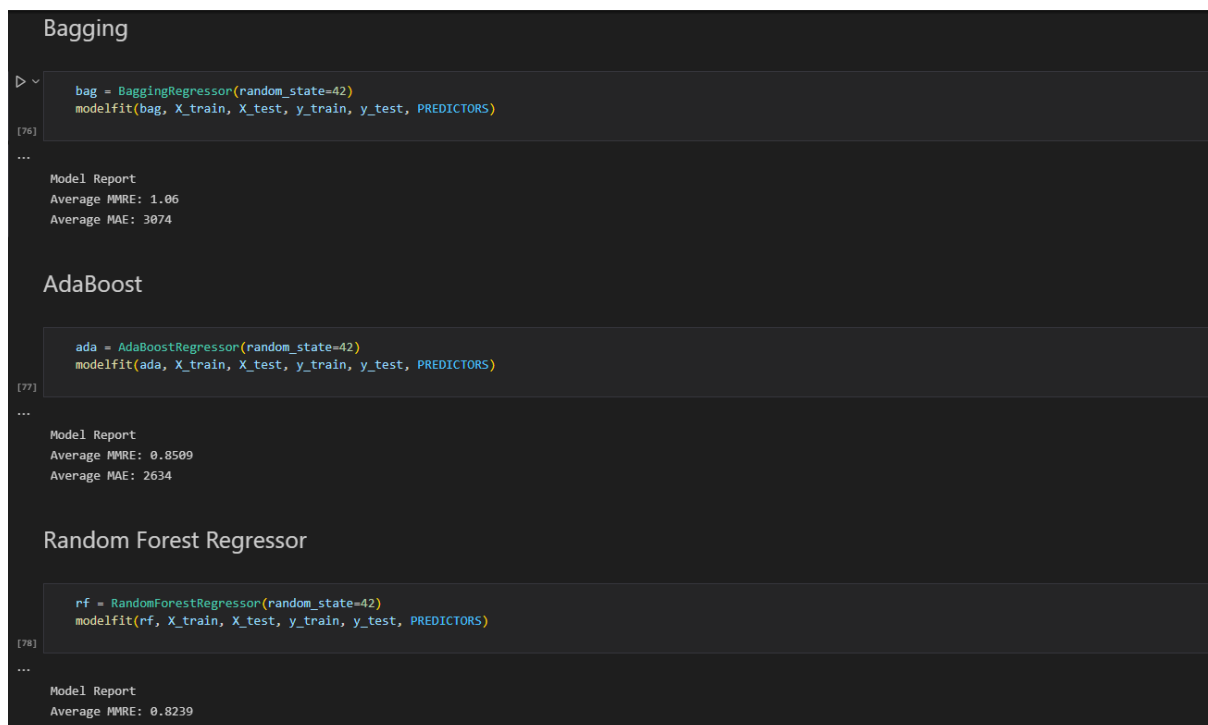
3.1.3 Development of local code

As the datasets are preprocessed and ready for training and testing for effort estimation models, the team implemented the main body of code for testing and training a model. Due to the implementation of software effort estimation for traditional and agile methodologies, two

separate code files were required, each for one of the two project methodologies. First, the dataset will be split based on the features, where the features that are for estimating the required software effort and the effort required are split into two separate datasets, for both methodologies. Then, the team implemented the splitting of the datasets such that 70% of the datasets will be used for training and 30% will be used for testing the model. The team then implemented the initial model, which is a random forest regression model as the data is continuous. After performing basic parameter tuning, the team then trained and tested the initial model, and was then able to successfully evaluate the accuracy of the model as well as allow input for estimating required software effort.

3.1.4 Evaluation of other ML models

The team was then able to implement other machine learning algorithms for comparison with the initial machine learning model. The other models that the team has implemented are widely used state-of-the-art ensemble learning methods for software effort estimation, such as Adaboost, Bagging and gradient boosting. By running the same training and testing methods with the same datasets on the ensemble algorithms above, each model was then evaluated with the same accuracy metrics. Finally, the models were used for comparison based on the evaluated accuracies and the final chosen model was selected, which is still the random forest regression model.



```
Bagging

bag = BaggingRegressor(random_state=42)
modelfit(bag, X_train, X_test, y_train, y_test, PREDICTORS)

[76]

...

Model Report
Average MMRE: 1.06
Average MAE: 3074

AdaBoost

ada = AdaBoostRegressor(random_state=42)
modelfit(ada, X_train, X_test, y_train, y_test, PREDICTORS)

[77]

...

Model Report
Average MMRE: 0.8509
Average MAE: 2634

Random Forest Regressor

rf = RandomForestRegressor(random_state=42)
modelfit(rf, X_train, X_test, y_train, y_test, PREDICTORS)

[78]

...

Model Report
Average MMRE: 0.8239
Average MAE: 2455
```

Figure 3.1.4 Sample of various models being evaluated.

3.1.5 Web application integration

Finally, the team then implemented a front-end web based application for the software effort estimation application. The fundamental requirements for the web application were implemented, where users are able to select the project methodology which is traditional or

agile, enter variables that affect required effort, and to be able to visualise the estimated effort. Furthermore, the web application is designed proportionally and simplistic for increased user satisfaction.

3.1.6 Hyperparameter tuning

After completing the application allowing users to predict software effort, hyperparameter tuning was further implemented to achieve better accuracy results on estimating the required effort for a software project. As the random forest regression model was used for the final selected model, the hyperparameter tuning was only implemented onto the final selected model. By implementing the grid search tuning method, the team have implemented hyperparameter tuning on 5 main tree-based hyperparameters of the model to further optimisation based on the trained dataset.

```
Optimal tree-parameters:

n_estimators: 40
min_samples_split: 8
min_samples_leaf: 3
max_depth: 5
max_features: 'sqrt'/2

Results

rf_tuned = RandomForestRegressor(bootstrap=True,n_estimators=40,max_depth=5,min_samples_leaf=3,min_samples_split=8,max_features='sqrt',random_state=42)
modelfit(rf_tuned, X_train, X_test, y_train, y_test, PREDICTORS)

Model Report
Average MMRE: 0.7722
Average MAE: 2326
```

Figure 3.1.6 Tuned random forest regression model.

3.2 Product delivered

The first main product in this project being delivered is the entire python code base. The files in the code base are agile_model.ipynb, trad_model.ipynb and app.py. The first 2 ipynb files are the files for training and testing the ML algorithm, as well as for accuracy evaluation of the algorithms. The app.py file is the main python file that computes the effort estimation for the web application and returns the results to the web application. Further details are discussed at [section 5.1](#).

The second main product in this project being delivered is the software effort estimation web application. The web application is rather simple, where users will first be shown the option to choose the project methodology of choice, and then based on the chosen option, users will then be able to enter the project variables for each respective feature field. Upon entering all feature fields with valid inputs, users can then click on the predict software effort button to see the effort required for their projects. Figure 3.2.2 shows how the web application will look like when users first click on the link.

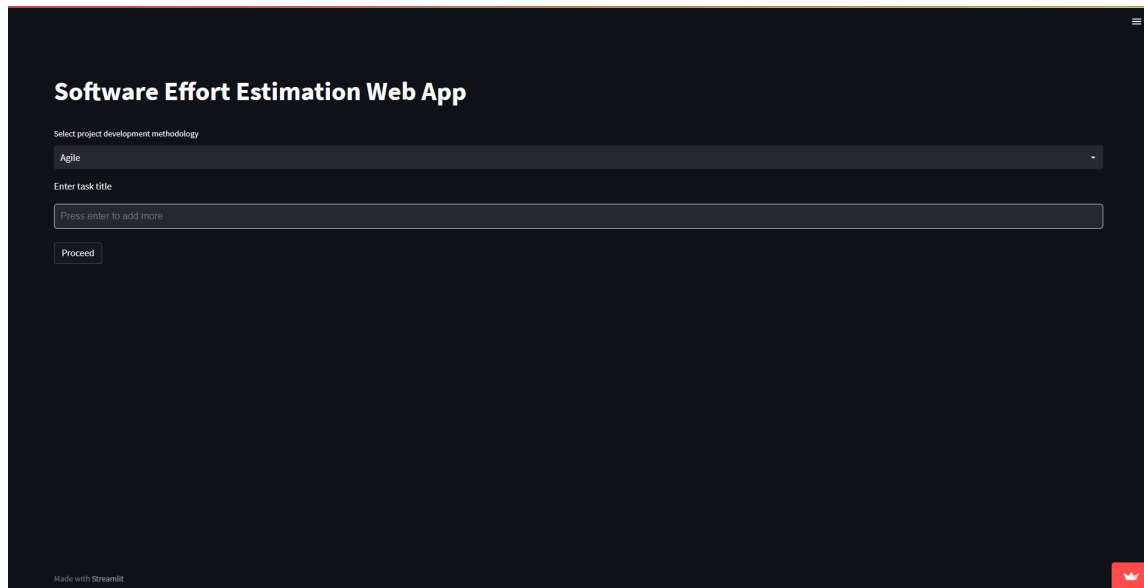


Fig 3.2.2: Look of the software effort estimation web application

3.3 How are requirements met

<i>ID</i>	<i>Requirements (Functional or Non Functional)</i>	<i>Assumption(s) and/or Customer Need(s)</i>	<i>Category</i>	<i>Source</i>	<i>Status</i>
001	Successful modelling	ML techniques used are examined and verified in the industry	Functional	Project proposal	In Progress
002	Correctly conducts experimental evaluation using different measures	Evaluation measures used are examined and verified in the industry	Functional	Project proposal	In Progress
003	Produces appropriate visualisation for comparison purposes	Visualisation is appropriate for different types of results	Functional	Project proposal	In Progress
004	Selects optimal and suitable model as final model	Final model is appropriately chosen from the built models	Functional	Project proposal	In Progress
005	Final model correctly computes software effort estimation	Effort estimation computed is validated by industry benchmark	Functional	Project proposal	In Progress
006	Produces a customizable template to reflect on the type of projects, development method, resources need, etc.	The template takes user input	Functional	Project proposal	In Progress

007	Displays a solution panel that presents a summary of our estimates	The unit measurements for effort and cost depend on the dataset and algorithm implementation	Functional	Project proposal	In Progress
008	The system is robust enough to handle invalid inputs	Feedback to user is displayed instead of triggering crash	Non-Functional	Project proposal	In Progress
009	The system is user-friendly	Proper rounding for result of estimation	Non-Functional	Project proposal	In Progress

Table 3.3.1 Requirements Traceability Matrix

Table 3.3.1 Shows the requirements traceability matrix from the initial project plan. Table 3.3.2 will show how the team has met the requirements.

ID	How the requirements are met
001	The team was able to implement an initial model using a random forest regressor from sklearn, train and test it with the preprocessed datasets and was able to estimate software effort.
002	After training a successful model, the model is used to predict the training dataset, then the absolute difference between the predicted values and the test values is calculated, which is then used to calculate the MRE, MMRE and MAE metric values.
003	The MRE, MMRE and MAE metric values for each model will be displayed after computing it, hence the accuracy and efficiency of each model can easily be visualised.
004	After running all models and evaluating the models with the metrics described, manual comparisons were made from each model's visualisations for accuracy. Then, the chosen final model will be exported for the web application.
005	As the models are being evaluated, all of the models were able to predict the required software effort with no errors. Hence the model that was selected to be the final model that is exported will also have no errors predicting the required software effort.
006	The web application is designed to handle different types of project methodologies, and is reflected by the app.py local application. In the local application, the code handles input by users for choosing the selected project methodology, and will display the template for each respective methodology based on the user's selection.
007	After the app.py file for the web application utilises the selected model for predicting the required software effort, the results will be sent to the web application for display.

008	In the app.py file for the web application, when coding for the fields to be displayed on the web app, the min and max value for each respective field (if applicable) was stated, and if the code catches any invalid inputs, it will display to the user that their value is invalid.
009	The web application was designed to be simple and user-friendly, while still ensuring that fundamental requirements for the web application are still present in the web application, hence the application is user friendly.

Table 3.3.1 Requirements Traceability Matrix

3.4 Justification of the decisions made

For readability, the justification of the decisions made throughout the project will be split into sections based on the timeline of what was implemented.

3.4.1 Dataset preprocessing

3.4.1.1 Preprocessing method

Preprocessing of datasets was done manually to ensure that any invalid data that may heavily impact the results of the software effort estimation application are not present in the dataset. The reason preprocessing was done manually was also because the team can also evaluate the importance of each feature to be extracted and gain a better understanding of the datasets overall, and to also ensure that the model is not trained on outlier data.

3.4.1.2 Removing year end feature of traditional dataset

After preprocessing the Desharnais dataset, each feature in the dataset is evaluated by the team based on importance in affecting the required effort of a software project, and have found that the feature year end is redundant. The feature is for the year when the project ended, and at the time of software effort estimation, users will not know the year that the project will be ending hence the removal of such data fields is necessary to ensure that accuracy of the estimation is maintained.

3.4.2 ML algorithm

The machine learning algorithms that were implemented and evaluated were 4 ensemble machine learning algorithms, which are the Bagging, Gradient Boosting, AdaBoost and Random Forest Regression models. The reason we chose ensemble machine learning algorithms is because through research, ensemble machine learning algorithms have been proven to be far more reliable and effective as compared to other models which we have initially planned to evaluate as well, such as extreme learning machine (ELM) models. Furthermore, not much research has been done on other machine learning algorithms, and the main article that the team is basing the project on utilises ensemble machine learning algorithms, hence the implementation of Ensemble machine learning algorithms.

3.4.3 Web application

3.4.3.1 Project methodology

The reason the application was implemented for both traditional and agile methodologies is because projects nowadays don't always follow traditional or waterfall methodology, hence the option to allow agile methodologies was included to allow a larger base of users to utilise our application for software effort estimation. But in the initial project plan, the idea of the web application was to fit both traditional and agile project feature fields into a singular page for comparison of effort required for the same project but different methodologies. We have changed that because it makes the web application less user-friendly, and may make it much more messier and difficult to understand. Hence, a separate feature field page is implemented for increased user satisfaction.

3.4.3.2 Preset feature field and warning messages

The application was implemented so that the feature fields automatically are set a value, and if a user tries to input invalid values, the input will not be accepted and the user will be returned with a warning message for invalid values. This decision makes it so that no matter what the user's input, the web application will not allow the user to attempt to predict software effort with invalid values. This also makes the application much more user friendly as it will allow users to understand where and which inputs were invalid, and will help them correct the input information.

3.5 Discussion of all results

3.5.1 Performance of ML model

3.5.1.1 Performance Evaluation Methodology

In terms of performance evaluation, we had used two different metrics in order to evaluate the performance of each model we had designed. Firstly was MMRE or mean magnitude or relative error, which is calculated by the formula:

$$RE = (\text{Absolute Error} / \text{True Value})$$

With RE being relative error.

We had also made use of MAE, or mean average error. Which was calculated using the formula:

$$MAE = \frac{\sum_{i=1}^n |x_i - x|}{n}$$

Where n is the sample size and x refers to the true result and x_i is the predicted result.

3.5.1.2 Results evaluation

Traditional model		
ML Algo Name	Avg MAE	Avg MMRE
Bagging	3074	1.06
Gradient Boosting	2691	0.9017
AdaBoost	2634	0.8509
Random Forest	2455	0.8239
Tuned Random Forest	2326	0.7772
Agile model		
ML Algo Name	Avg MAE	Avg MMRE
AdaBoost	5.286	0.0869
Bagging	2.429	0.04472
Gradient Boosting	1.888	0.043
Random Forest	1.56	0.03048

Table 3.5.1.2 Results of algorithms

Table 3.5.1.2 displays the results of testing the algorithms and their respective accuracies. As we can see, for both traditional and agile methodology, the average MMRE and average MAE for random forest regression model is better than the other ensemble machine learning algorithms. For MMRE and MAE, the lower the value, the higher the accuracy in predicting software effort. And because random forest is the best resulting algorithm, further hyperparameter tuning is implemented onto the model to attempt to further improve its accuracy and performance.

As we can see for the traditional models, the hyperparameter tuned version of the random forest regression model returns a significantly lower MMRE and MAE value compared to the originally implemented random forest regression model.

3.5.2 User interface

Overall, the feedback that the team has received for the software effort estimation web application is good and satisfactory. Despite not having step by step instructions located on the web application, users found that it is still easy to navigate and with the help of a good web application structure and warning messages for invalid inputs, users can easily estimate

the effort required to complete a software project. Also, as mentioned in [section 5.2.2](#), the security aspect of the web application is robust and there will be no major security issues with our web application.

3.6 Limitations of project outcomes

When working on the project, we found that there were some minute details that limited the progress and effectiveness of the project. The below sub sections will indicate the limitations that the team have discovered for the duration of the project.

3.6.1 Dataset

For the current implementation of our software effort estimation application, both traditional and agile methodologies were trained and tested on a singular dataset only, which is the Desharnais dataset and a singular agile dataset respectively. This is a limitation to our project because the feature fields that users are required to input to predict required software effort may be much more specific, which implies that users must always have data information that is specifically suited for our web application ready for effort estimation, and any other data information that our web application does not require that could be important to a software project's required effort will not be utilised which can be bad. Furthermore, the current datasets that were used are somewhat small, which can heavily impact the training and testing of the machine learning algorithms leading to a lower accuracy, and does not allow for much space for hyperparameter tuning, which can be shown for the agile dataset as hyperparameter tuning was not done on the dataset as it is too small.

3.6.2 ML model

For the current implementation of our software effort estimation application, the finalised machine learning algorithm that is implemented is the random forest regression model that is an ensemble machine learning. Although the chosen machine learning algorithm is able to predict software effort somewhat accurately as compared to various other models we have evaluated, we believe that the machine learning model can be further improved to yield better results. The current implemented model works well and implementing grid search hyperparameter tuning on the model also helps improve the accuracy of the model further, though the team still believes that the model may be too simple to accurately handle far complex data features if a different dataset was used.

3.6.3 Performance optimization

Before discussing the performance results that were achieved, we want to mention that prior to this project, members of the team had little to no prior experience in handling hyperparameter turning on machine learning algorithms. Hence, we want to emphasise that the hyperparameter tuning that was implemented as of recent is of basic understanding only and based on minimal research on grid search tuning methodology. Due to this, we believe

that the hyperparameter tuning that was implemented for the random forest regression model could be significantly improved to yield better accuracy results.

Overall, the limitations did not affect much of the outcome of the project, and may only require further research for slight improvements.

3.7 Discussion of possible improvements and future works

In this section, we will discuss possible further improvements and future works that can help improve the web application for software effort estimation as a whole, as well as combat the limitations of the project as discussed in the section above.

3.7.1 Pre-processing and feature extraction

The first possible improvement that could be made to the application is for automated preprocessing of datasets. There are several positives to such an implementation, and will be further discussed in this section. One of the positives of this implementation is that users can upload their own datasets to train and test our models with. Doing this will allow users to perhaps upload their own datasets that may be befitting of the data information of their own software project at hand, hence the effort required to complete their software projects can be accurately predicted based on the dataset they have uploaded instead. Furthermore, automating preprocessing the datasets and feature extraction will require less work for the team to manually go through the datasets and remove invalid values. Although there are tools out there that allowed the team to easily return all rows of data with invalid values, the team still had to manually define the values that are outliers and invalid to return the rows for the team to remove. With automated preprocessing and feature extraction, the team just has to upload the dataset, and the algorithm will automatically remove all rows of data with outliers and invalid data, and possibly even setting the invalid values with average values calculated from the datasets.

3.7.2 Preset web application feature fields

The next possible improvement that could be made to the web application is to have preset values in the information fields. When a user uses our application, they may not have all data information readily available to be input into the application's feature fields. By leaving the field at the lowest possible value, it may heavily affect the outcome of the effort prediction. Hence to combat this, by having a preset value in the feature fields in the web application, users don't always have to have all data information readily available to use our application. The preset values will be defined by averaging the values of that specific data field based on the training dataset, which makes the outcome way more realistic than leaving the value at the lowest possible.

3.7.3 Further ML algorithm research

The next possible improvement that could be made to the application as a whole is to do further research on machine learning algorithms. The team believes that the current result may be satisfactory and it could perform the basic functionality, which is to predict software effort required but the accuracy may not be as best as it could be. Hence, further research on far complex and state of the art machine learning algorithms may be beneficial and will yield far better results than before. Not only that, if the team ever decides to work on different datasets, the new and improved final machine learning algorithm will also be able to handle far more complex and larger datasets.

3.7.4 Further Hyperparameter tuning research

The last possible improvement that the team has decided on is to do further research on hyperparameter tuning. Hyperparameter tuning is an important step in machine learning algorithms that can heavily impact the accuracy of the algorithm, and being able to further implement a higher level of hyperparameter tuning will drastically improve the accuracy of the algorithm as compared to now. Also, implementing automated hyperparameter tuning is also a further improvement that can be made. According to Palaniswamy and Venkatesan (2021), automated hyperparameter tuning increases the accuracy of ML algorithms and reproducibility whilst decreasing the human work required to try out a variety of machine-learning model configurations.

4. Methodology

4.1 Design

4.1.1 Research Process

At the beginning of our project, we began by looking into information on any and all papers that had relation to our topic. We did this through the use of search keywords such as ‘machine learning’ and ‘software effort estimation’. This provided us with a narrow enough scope that we would only find papers that would be relevant to our research while keeping the scope wide enough to not allow any important papers to fall through and hence dodge our research.

From there the research would then be picked up by our respective team members, with each team member reading and analysing their papers to allow for maximum efficiency and for information sharing to become a commonplace thing. Each member would then compile their analysis in a table containing a few items:

1. Dataset Used - Information on what data set was used in the research and how it was obtained (especially if it was publicly available).

2. Publishing Year - The year in which the paper was published (To check the relevancy).
3. Methodology - Information regarding how the research was conducted including a summary of the steps followed by the researchers.
4. Evaluation Metrics - The measurements used throughout the research in order to evaluate results.
5. Futureworks - How the research could be improved over time or to make up for limitations.

This allowed for information to be condensed down into only the most important pieces of information and allowed for information to be shared among team members very quickly and efficiently.

4.1.2 System Architecture

Stage 2 of our project revolved around this particular chart which we had published in our initial project plan.

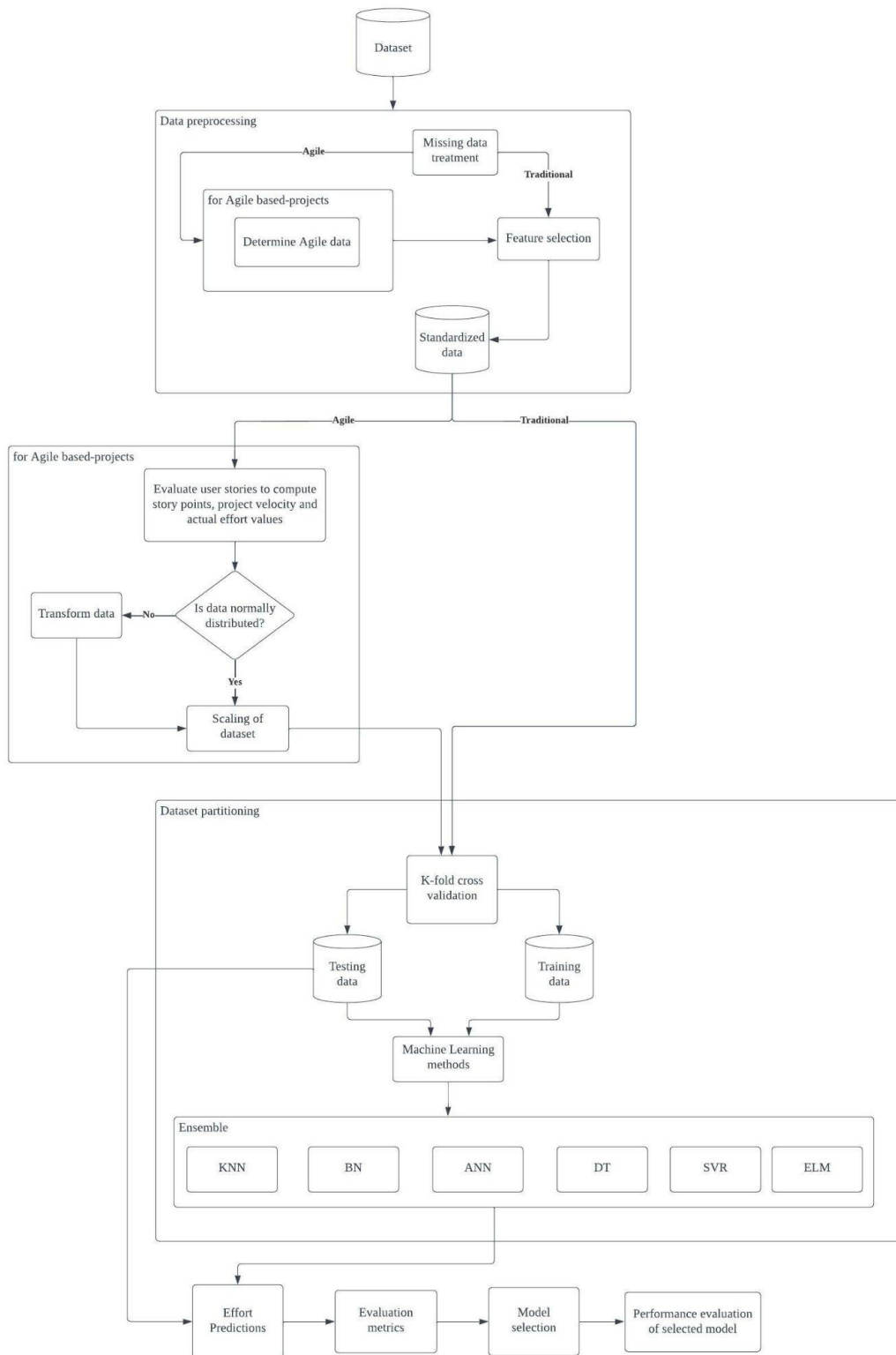


Figure 4.1.2 Backend Flow chart from project proposal

This chart summarises, for the most part, our process that we used for this project, however this chart only covers the data processing and model training and testing sections. Later on we would have to include integration of our model into the frontend part of our application. This was not accounted for initially but had to be done later on once the earlier sections were completed.

Data Preprocessing

In this project, we made use of many datasets, each of which had gone through data preprocessing which included:

1. Processing missing data:
Missing values could affect the overall accuracy of the model. Missing values are dealt with through dropping columns and rows with too many null values
2. Determining agile data (optional for agile projects):
Agile data is extracted from datasets through the analysis of the attributes and unique characteristics of agile projects.
3. Feature selection:
By removing irrelevant features, it allows for increased accuracy for the model. By using the computation of correlation coefficient matrix, any irrelevant features may be eliminated while any principal features are retained. Through this, the dataset becomes standardised.

Data taken from agile projects were later put through further processing which included:

1. Evaluation of user stories:
A user story refers to the very smallest unit of work in the agile framework. It is evaluated in order to determine principal parameters. These are collected along with the final velocity value as input arguments in order for machine learning models to calculate normalised efforts
2. Analysis of processed dataset:
The processed dataset based on story point approach is analysed statistically to remove any outliers that may potentially affect the overall accuracy by identifying the dataset's distribution, with any dataset that is not normally distributed going through a transformation process to ensure that it is normally distributed.
3. Transformation of data:
The dataset is transformed logarithmically into a normally distributed dataset.

Data Partitioning

Following processing of the data, the data is then partitioned or segregated into training and testing data sets and is then used to build and test ensemble machine learning methods. We had used and tested various ensemble machine learning methods, however we had decided that Random Forest was the best option. Other methods we had considered include: AdaBoost, Bagging, Gradient Boosting.

Hyperparameter Tuning

Furthermore, we had conducted tuning after testing and selection of our model in order to fully maximise and increase the accuracy of our model. This was done through the use of several tests which were used to determine the optimal parameters that we would be using for our final model. The tests led to a final model that had a significantly smaller MAE and MMRE and hence had an overall significantly better accuracy compared to before the hyperparameter tuning. This process was done through the guidance of an article prepared by Arashay Jain (2016) and making use of a technique known as grid search tuning.

4.1.3 Comparison to Initial Plan from Project Proposal

For the most part, the research portion of our project remained unchanged, this is mostly due to the fact that our research portion was mostly conducted before our project proposal had been submitted.

However, as for the implementation part of our project, we had initially forgotten to plan for creation of the front end of our application as well as integration of our model into the frontend of our application.

Our machine learning methods had also changed significantly as we had fully decided to lean towards purely different Ensemble Learning Methods (ELM) due to its efficiency. However we did attempt to implement multiple types of ELM, such as AdaBoost, Random Forest, and Bagging.

4.1.4 Frontend Integration

Following the previous steps, we had produced a working model within reasonable accuracy. This was then integrated into a frontend that had been previously created during the period in which we were creating and training the model. This frontend was created using the StreamLit library in Python.

It would allow for a smooth and neat frontend application page which could accept inputs and feed them into the model. This later allowed us to also ensure that our application looked aesthetically pleasing and so more appealing to users.

Our frontend would include a very nice and clean web page. On the web page there would be input boxes for each of the features in the traditional input, as well as sliders for certain inputs in the agile input as well. All of which are shown in the software deliverables (section 5).

Later on, the application would also be moved to a web platform through the use of the StreamLit public servers. This would allow for anybody to access the application through the use of a link to open it in their browsers, hence allowing them to use the application without having to install any dependencies or download the application.

It also allowed for any models to be preloaded in the server and so improved responsiveness as well as allowed for the dataset and model to be stored in the server and so decrease overall storage burdens and even allowed us to switch the model and dataset if need be.

4.2 Software and Tools Used in Implementation

No.	Category	Suggested	Potential Substitute
1	Software Library	Ensemble ML	AdaBoost
2	Programming Language Environment	Python v3.7 running on either VSCode or PyCharm	Any newer version of Python after v3.7
3	Project Management	Microsoft Teams	
4	Project Management	Google Drive	
5	Project Management	GitHub	Google Drive
6	Project Management	Project Libre	Microsoft Project
7	Hardware Requirements	<ul style="list-style-type: none"> - Intel Core i5 6th Gen - 16 GB RAM - NVIDIA GeForce GTX 960 	<ul style="list-style-type: none"> - Higher Gen accepted - Anything higher than 8GB accepted - Higher Gen Accepted
8	Data Storage	2GB	Higher accepted

Table 4.2.1 Initially Proposed Software Specification from Project Proposal

No.	Category	Software / Hardware Used
1	Software Library	SKLearn
2	Programming Language Environment	Python v3.7 running on either VSCode or PyCharm (Anaconda installation)
3	Project Management	Microsoft Teams
4	Project Management	Google Drive
5	Project Management	GitHub
6	Project Management	Project Libre
7	Hardware Requirements	<ul style="list-style-type: none"> - Intel Core i5 6th Gen - 16 GB RAM - NVIDIA GeForce GTX 960
8	Data Storage	2GB
9	Software Library	StreamLit

Table 4.2.2 Final List of Software/Hardware Specifications Used During Project

As can be seen above when comparing the two tables, there are indeed a few changes between them, with yellow indicating changes in what was used and green indicating a brand new requirement.

For SKLearn, we went with SKLearn over any other library as it was the simplest and most efficient method for us to implement multiple different types of machine learning methods without having to go out of our way to switch and change things.

StreamLit was added as it was what we had used to design the frontend of our application. It was also the service that we had used in order to implement our frontend as a web application as they had provided the servers for us to run it on. Hence, there are no requirements to run the application as it is all run on web servers.

5. Software Deliverables

5.1 Summary

The software deliverables comprise a web-based software effort estimation tool that allows users to accurately predict the effort required for completing projects to prevent slippage of timelines and over-budgeting issues. The interface of the web application is designed using an open-source Python library, Streamlit, that turns data scripts into interactive web applications.

Software Effort Estimation Web App

The screenshot shows the 'Agile' methodology selected. The task title input field contains 'microsoft testing' with a red 'X' icon and a prompt to 'Press enter to add more'. Below this is a 'Proceed' button. The main form area displays 'Task 1 : microsoft testing' and two sliders: 'Rate story size' (set to 1) and 'Rate user story complexity' (set to 1). At the bottom, there is an 'Enter team velocity' input field set to 0 and a 'Predict Software Effort' button.

Fig 5.1.1: UI for Agile projects

The above demonstrates the UI when users select Agile as the project development methodology. Users are required to key in one or many task titles. The program will prompt an error if no task title is entered. When users click the 'Proceed' button, a form will be generated.

Software Effort Estimation Web App

The screenshot shows the 'Traditional' methodology selected. The form contains six input fields with increment/decrement buttons: 'Enter team experience measured in years (-1 for zero experience)', 'Enter manager experience measured in years (-1 for zero experience)', 'Enter duration of the project in months', 'Enter number of basic logical transactions in the system', 'Enter number of entities in the systems data model', and 'Enter the size of the project measured in adjusted function points.' All fields are currently set to 0. A 'Predict Software Effort' button is located at the bottom left.

Fig 5.1.2: UI for Traditional projects

The above demonstrates the UI when users select Traditional as the project development methodology. Users are required to enter all input fields.

Predict Software Effort

Model Loaded Successfully!

Suggested effort: 26.8 months

Fig 5.1.3: Example output for Agile projects

The web application processes user input and displays output each time the Predict button is clicked. After submitting the form, input fields will not get removed until the page is refreshed. Users can modify an input field without having to re-enter all fields.

Besides, a graphical design representation of the system is delivered to allow easy visualisation of the structural aspects of our web-based tool.

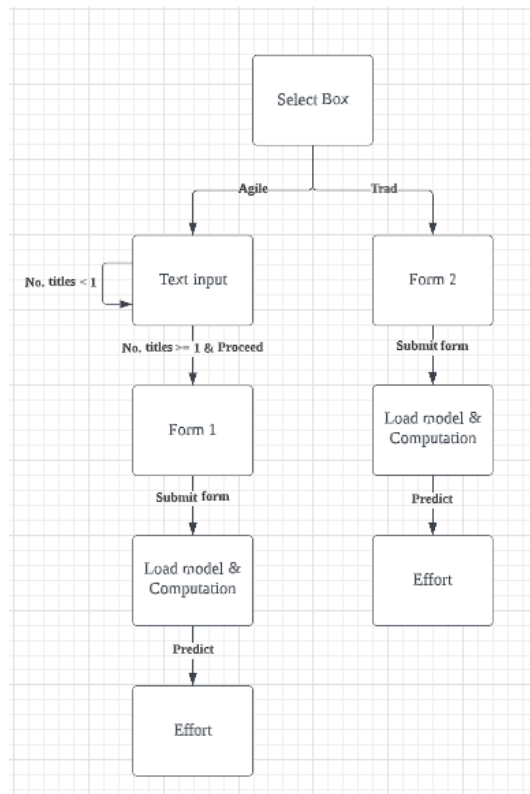


Fig 5.1.4: Graphical design representation

The last deliverable, the source code of algorithm implementation is composed of three parts, data preprocessing, model training and testing, and web application script (*See Appendix*).

5.2 Software Qualities

The following discusses how software qualities are assured in our project, as well as the shortcomings in the software.

5.2.1 Robustness

Robustness is assured when testing on both the backend and frontend systems by dealing with exceptional inputs. The web application script created in the backend system includes input validation for every input field. This will be deployed during input processing in the front end. The system's behaviour is observed to guarantee that the failures' effects are acceptable.

5.2.2 Security

The web application is built using the Streamlit library that provides best-in-industry security. According to Streamlit Docs. (n.d.), all-access and sign-ins to the infrastructure are conducted via an SSO provider, GitHub. Our sensitive data, such as authentication tokens, are encrypted. Only users with write access to the GitHub repository for our app can modify it in the administrative console. Moreover, the data is hosted and managed within the Google Cloud Platform (GCP), which continuously undergoes risk assessments through monitoring its data centres to ensure that they are compliant with the industry requirements. All the Streamlit servers are protected by firewalls, and all applications built using it are served entirely over HTTPS. The 256-bit encryption is employed to encrypt all information sent to or from Streamlit in transit over the public internet. Also, Streamlit uses third-party security tools to conduct penetration tests that examine vulnerabilities regularly. These factors have assured security quality in our project.

5.2.3 Usability

Usability is handled when testing on the frontend system by gathering users' feedback on their interactions with the application. One of the shortcomings of our application in terms of usability assurance involves finding appropriate representative users. Given the time constraints, we are unable to find experts who have backgrounds in related disciplines conducting the manual usability testing process. Instead, feedback was only gathered within our team.

5.2.4 Scalability

In order to make the application scalable and less resource-hungry, we ensure that memory-intense computations such as loading large machine-learning models or datasets directly into the scripts run only once using a cache-control option. However, our web application hosted by the employed infrastructure is only scalable up to a 1GB resource limit. The infrastructure only scales up the resources considerably higher than the community free tier for enterprise customers.

5.2.5 Documentation and Maintainability

All important functions and codes are commented to allow for easier understanding of the source code of our system. The source code is modularized so that they can be maintained more easily.

6. Critical Discussion

Overall, as we can see in [section 3.3](#), all of the fundamental functional and non-functional requirements have been met for the duration of this project. Due to the team closely following the project plan that was created in the previous semester, the requirements for the project as a whole have been met. Although all functional and non-functional requirements were met, the team ended up not implementing the extreme learning machine (ELM) machine learning algorithms to evaluate. This is due to the lack of time the team had to implement more machine learning algorithms, and also the lack of experience in said machine learning algorithms. Although much research has been done and literature reviews on ELM models, the team still could not grasp a further understanding of the modules and was not able to implement them, although attempts were made.

Although, due to miscommunication and some fall outs in the project management, some aspects of the deliverables were not able to be met on time, which caused the progress of the project as a whole to be delayed. But due to communication with members of the team and the supervisor, with due diligence the team was able to get back on track in the later weeks and implement the web application as planned. As part of the project management, the team was also able to communicate frequently with the supervisor which allowed the team to be able to understand the scope of all the deliverables of the semester which also helped with completing the project in a timely manner.

Without changing the implementation of the system architecture and also the way of implementing the web application using the streamlit libraries, the web application was up as planned. Although, there are some changes to the initial layout of the web application. The initial layout was planned to implement both traditional and agile methodology projects into one page for possible simultaneous comparisons, although during the implementation stage the team decided that it was not necessary to do so. The changes were made so that the user can only enter data information for one project at a time, one project methodology at a time. The user can then record down the returned required effort and load the application again to check effort for another project or if the same project but with different methodologies.

7. Conclusion

Overall, our team has managed to produce a project software estimation model that is able to predict software effort with reasonable accuracy. Not only that, but we have also managed to create an application to go along with it and is able to stand as a frontend. Altogether, it has been put together to become an application that could potentially be used by companies right now. The application has also been improved to the point where it is able to be used without any pre-installation or access dependencies as we have moved the application to a web server that is able to be run on any computer as long as it has an internet connection and a browser.

8. Appendix

8.1 Code appendix

```
Load Data Sources

# retrieve from https://www.researchgate.net/publication/268186219_An_Effort_Estimation_Model_for_Agile_Software_Development

data = {'Effort': [156,202,173,331,124,339,97,257,84,211,131,112,101,74,62,289,113,141,213,137,91],
        'Vi': [4.2,3.7,4.4,5.4,9.4,1.4,2.3,3.8,3.9,4.6,4.6,3.9,3.9,3.9,4.4,4.4,3.7,3.7],
        'D': [0.687,0.701,0.878,0.886,0.903,0.903,0.859,0.833,0.646,0.758,0.758,0.773,0.773,0.773,0.742,0.742,0.742,0.758,0.758],
        'V': [2.7,2.5,3.3,3.8,4.2,3.6,3.4,3.2,4.3,2.2,2.9,2.9,2.9,2.8,2.8,2.8,2.7,2.7],
        'Sprint Size': [10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10],
        'Work Days': [22,21,22,22,22,22,22,22,22,22,22,22,22,22,22,22,22,22],
        'Act:Time': [63,92,56,86,32,91,35,93,36,62,45,37,32,30,21,112,39,52,80,56,35]}

df_project = pd.DataFrame(data)

[3] ✓ 0.8s Python
```

```
Export dataset

# use original dataset to get user input since the processed data is scaled and output value is scaled as well.
df_project.to_csv('agile_dataset.csv', index=False)

[1] Python
```

Fig 8.1: Agile dataset preprocessing

```
Model Fitting

def modelfit(alg, X_train, X_test, y_train, y_test, predictors, printFeatureImportance=True):

    #Fit the algorithm on the data
    alg.fit(X_train, y_train)

    #Predict training set:
    Y_pred = alg.predict(X_test)
    diff = np.absolute(y_test-Y_pred)
    MRE = diff/y_test
    MMRE = np.mean(MRE)
    MAE = metrics.mean_absolute_error(y_test,Y_pred)

    #Print model report:
    print("\nModel Report")
    print("Average MMRE: %.4g" % MMRE)
    print("Average MAE: %.4g" % MAE)

[5] Python
```

```
# split data for testing and training
X_train, X_test, y_train, y_test = train_test_split(df[predictors], df[TARGET], test_size=0.3)

[6] Python
```

Fig 8.2: Parts of model training and testing source code

```
if __name__ == "__main__":
    submitted1 = False
    submitted2 = False

    # check if button is clicked
    if "button_clicked" not in st.session_state:
        st.session_state.button_clicked = False

    st.set_page_config(page_title="MCS17", layout="wide")
    st.title("Software Effort Estimation Web App")
    method_select = st.selectbox("Select project development methodology", options=["Agile", "Traditional"])

    if method_select == "Agile":
        # task titles
        task_title_lst = sttags.st_tags(label="Enter task title", text="Press enter to add more")
        # if proceed has not been clicked yet
        if st.button("Proceed", on_click=callback) or st.session_state.button_clicked:
            # prompt user error
            if not task_title_lst:
                st.error("Please enter a task title!")
                st.stop()
            # fill in form
            with st.form("form1"):
                c1, c2 = st.columns(2)
                story_size_lst = []
                complexity_lst = []
                # generate user stories based on the no of task titles entered
                for i, x in enumerate(task_title_lst):
```

Fig 8.3: Parts of web application script

8.2 Team member contributions

Front cover - Jovan Ong

Table of content - Benjamin Marc Wijayaratne

Introduction - Benjamin Marc Wijayaratne

Background - Wong Zi Qi

Outcomes - Jovan Ong

Methodology - Benjamin Marc Wijayaratne

Software Deliverables - Wong Zi Qi

Critical Discussion - Jovan Ong

Conclusion - Benjamin Marc Wijayaratne

Code listing in Appendix - Wong Zi Qi

Team member contribution in Appendix - Jovan Ong

References - Wong Zi Qi, Jovan Ong

Style and Presentation - Benjamin Marc Wijayaratne

9. References

Das, S., Dey, A., Pal, A., & Roy, N. (2015). Applications of artificial intelligence in machine learning: review and prospect. *Int J Comput Appl*, 115(9).

https://www.researchgate.net/publication/276178017_Applications_of_Artificial_Intelligence_in_Machine_Learning_Review_and_Prospect

Feurer, M., & Hutter, F. (2019). Hyperparameter Optimization. *Automated Machine Learning*, 3–33. https://doi.org/10.1007/978-3-030-05318-5_1

Glen, S. *Absolute error & mean absolute error (MAE)*. StatisticsHowTo.com: Elementary

Statistics for the rest of us. <https://www.statisticshowto.com/absolute-error/>

Glen, S. *Relative error: Definition, formula, examples*. StatisticsHowTo.com: Elementary

Statistics for the rest of us. <https://www.statisticshowto.com/relative-error/>

Jain, A. (2016, Feb 21). *Complete machine learning guide to parameter tuning in gradient boosting (GBM) in python*. Analytics Vidhya.

<https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/>

Kocaguneli, E., Tosun, A., & Bener, A. (2010). AI-Based Models for Software Effort Estimation. *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, 323–326. <https://doi.org/10.1109/SEAA.2010.19>

Malgonde, O., & Chari, K. (2018). An ensemble-based model for predicting agile software development effort. *Empirical Software Engineering*, 24(2), 1017–1055. <https://doi.org/10.1007/s10664-018-9647-0>

Palaniswamy, S. K., & Venkatesan, R. (2020). RETRACTED ARTICLE: Hyperparameters tuning of ensemble model for software effort estimation. *Journal of Ambient Intelligence and Humanized Computing*, 12(6), 6579–6589. <https://doi.org/10.1007/s12652-020-02277-4>

Satapathy, S. M., & Rath, S. K. (2017). Empirical assessment of machine learning models for agile software development effort estimation using story points. *Innovations in Systems and Software Engineering*, 13(2–3), 191–200. <https://doi.org/10.1007/s11334-017-0288-z>

Streamlit Trust and Security - Streamlit Docs. (n.d.). Retrieved October 21, 2022, from <https://docs.streamlit.io/streamlit-cloud/trust-and-security>

Zia, Z. K., Tipu, S. K., & Zia, S. K. (2012). An Effort Estimation Model for Agile Software Development. *Adv Comput Sci Appl*, 2(1), 314–324.

https://www.researchgate.net/publication/268186219_An_Effort_Estimation_Model_for_Agile_Software_Development

de Carvalho, H. D. P., Fagundes, R., & Santos, W. (2021). Extreme Learning Machine Applied to Software Development Effort Estimation. *IEEE Access*, 9, 92676–92687.

<https://doi.org/10.1109/access.2021.3091313>