

B827504

CS&AI

COC257

B827504

**EMPIRICAL COMPARISON OF ARTIFICIAL
INTELLIGENCE FOR SOFTWARE
EFFORT ESTIMATION**

by

Connor Campbell-Elliott

Supervisor: Dr T. Chen

*Department of Computer Science
Loughborough University*

May 2023

Table of Contents

Abstract	5
Acknowledgements.....	5
1 - Introduction	6
1.1 - Aims	6
1.2 - Objectives.....	6
1.3 - Statistical objectives.....	7
2 - Literature Review & Research	9
2.1 - Software Effort Estimation	9
2.2 - Machine Learning.....	9
2.3 - Study Design	11
2.3.1 - Study Objective	11
2.3.2 - Information Sources	12
2.4 - Surveys.....	12
2.4.1 - Summary	15
2.4.2 - Discussion	17
3 - Design & Development	20
3.1 - Python.....	20
3.1.1 - Anaconda	20
3.1.2 - Jupyter notebook.....	21
3.2 - Computer Spec.....	21
3.3 - Models	21
3.4 - Libraries.....	23
3.5 - Model training.....	24
3.6 - ScottKnottESD	28
3.7 - Datasets	28
3.8 - Feature Selection	30

3.9 - System Design	31
4 - Plan	32
4.1 - Requirements	32
4.2 - Software Engineering Methodology	32
4.3 - Gantt Chart	32
4.4 - Plan adjustments	35
4.4.1 - Challenges	35
4.5 - Risks	38
5 - Implementation	39
5.1 - Libraries	40
5.2 - Loading datasets	40
5.3 - Cleaning Data	41
5.4 - Exploratory Data Analysis (EDA)	41
5.4.1 - Feature Selection	42
5.5 - Models	48
5.5.1 - Hyper tuning	48
5.5.2 -Default	53
5.6 - Google Colaboratory	53
5.7 - ScottKnottESD	54
5.8 - Command line (CMD)	55
6 - Results	55
6.1 - CMD Interface:	56
6.2 - Tables of Results	57
6.2.1 - Feature Selection vs. Datasets	61
6.2.2 - Hyperparameters vs. Datasets	62
6.2.3 - Best Overall Performing Model	63
6.2.4 - Best Performing Dataset	64

6.2.5 - Worst Performing Model	64
6.2.6 - Worst Performing Dataset.....	65
7- Conclusion & Future work.....	65
7.1 - Personal Development.....	65
7.2 - Objectives met.....	66
7.3 - Hyperparameter Optimisation Advantages	67
7.4 - Hyperparameter Optimisation Limitations	68
7.5 - Models	68
7.6 - Datasets	68
7.7 - Cost benefit analysis	69
7.8 - User Interface	69
7.9 - CUDA (with TensorFlow)	69
7.10 - Cloud System	69
7.11 - Conclusion	70
8 - References.....	71
9 - Appendix.....	75
9.1 - Key words.....	75
9.2 - Running the program.....	76
9.3 - Additional program pictures	77

Abstract

With the increase in digitisation, there is a growing demand for software which has led to an increase in the number of projects however, these projects have less resources and time to be completed. Effort estimation is an approach used to estimate the effort required to complete these projects with regard to the given resources. The advancements in artificial intelligence (AI) have provided a different approach to effort estimation without the need of a human expert such as an estimator.

This report is the exploration of AI for software effort estimation (SEE) through the use of machine learning (ML). In this report, 5 models are created to retrieve data on past projects to give estimations on how much effort is required for that given project based on its attributes (features). The 5 models, Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), K-nearest neighbors (KNN) and Linear Regression (LR) will be optimised using hyperparameters and data will be filtered by using feature selection.

Through the use of these methods, it was concluded that the SVM was the most successful model returning highly accurate estimations, with an average R2 score of 0.77 and RMSE score of 2841. The data show that accuracy is proportional to the number of attributes (features) and item entries datasets included. The largest dataset, China, produced an average R2 of 0.93 and RMSE of 1501 across all 5 models.

Concluding the results, AI for SEE is a highly robust and accurate method for effort estimation, requiring a small amount of time for estimations. Someday effort estimation could replace or assist the role of an estimator when resources are scarce.

Acknowledgements

I would like to thank my supervisor Dr. Tao Chen for providing support along the way for this project.

I would like to thank the God I serve as well as my friends and family for their support through this academic year which has pushed me to ensure the completion of this dissertation and project.

I would specifically like to thank Zion Bassy-Maddison and Portia Laryea-Adu for their ongoing support too.

1 - Introduction

Modern day software projects are rarely linear and have many influences (Stakeholders) and variables which not only alter the process but consequently alter the finalised product. It is well reported that projects exceed their initial budgets due to factors just mentioned, which mainly affects the budget, causing a company to exceed their financial bounds causing a project to fail or be delayed and this has been witnessed first hand working within project management.

Cost overruns can be detrimental to companies and with a prediction of \$5889.9 billion forecasted to be spent over 2023 on IT projects (Gartner,2022) and with projects having a 35% success rate (The Standish Group, 2020) it is essential that companies carry out more care when proceeding with their estimations for software projects.

It is a crucial preparatory step whilst developing any software to have some degree of an understanding for the requirements it takes to build, and finish said software project. Expanding demand for software and short turnaround criteria has pushed the need for efficiency has become even more essential. It has been reported that post pandemic 67% of companies have accelerated digital projects (IBM,2021) thus software development across the board has become a larger market and has created pressure for software to be more pristine.

The objective of software effort estimation (SEE) is to accurately produce a robust estimation cost model which can then be used to predict software costs. From these, numerous methods of SEE have been developed and recently with the rise of Artificial Intelligence (AI) we are now able to produce methods which are able to make minimal assumptions to produce more accurate predictions using machine learning techniques.

1.1 - Aims

This project will imminently produce numerous models using machine learning (ML) which can be used with different datasets to estimate software effort. Using the data, software will make predictions on the effort required for a given project.

1.2 - Objectives

1. Create aims and objectives for the project.
2. Complete a literature review on software effort/cost estimation (SEE), Project failure, ML, Methods of implementing ML into software effort estimation.

3. Find datasets which could be used with the SEE model(s) created.
4. Decide on programming language and framework to use to implement SEE model(s)
5. Develop model(s) using different techniques that have been found from literature review and test these against datasets.
6. Compare the models and datasets against each other, answering the objectives defined when testing the models.
7. Evaluate the findings at the end of the project by assessing if all objectives set have been met using statistical data to confirm this.

1.3 - Statistical objectives

- The main purpose of a ML algorithm for software effort estimation is the ability to predict and therefore this needs to be at the forefront when considering if it is successful.
- A successful model will be able to predict a high percentage of actual values within a percentile. The percentile used will be pred (25) which will fall between 0-1, the closer the value to 1, the better model. This will be a check to see if predictions are within 25% of the actual value.

Together with an R^2 value, 'Any R^2 value less than 1.0 indicates that at least some variability in the data cannot be accounted for by the model' (Hamilton, Gchert, Simpson, 2015).

A value 0.65 would be classed as highly accurate as its confidently over halfway.

- Another way I look to measure performance is the use of root mean squared error (RMSE). RMSE is the square root of the average of squared differences between prediction and actual observation. RMSE will have a higher priority as it penalizes large errors contrary to MAE. RMSE is a negatively oriented scorer so the lower the score the better the model (Jj,2016). 'RMSE measures the average difference between values predicted by a model and the actual values' (SAP, 2020)
- MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. I will be measuring this for success and comparing it against models however my main focus will be directed towards R^2 . Just like RMSE, the lower the score the better the model. (Jj,2016)

- I would like to measure speed/time of models as they will differ between one another due to difference algorithms and datasets, so I will be recording in seconds how long each one takes to run.

For the literature review, I pose some general research questions:

Question		Reason
Q1	Which Algorithm is better in terms of accuracy?	<p>This is almost the pinnacle of the purpose of SEE. Accuracy helps optimisation of resources and is the main answer project managers look for and if it is not, it should be the first question asked.</p> <p>Underestimation leads to cost overruns and more pressure on meeting demands which can be extremely pernicious to a project.</p>
Q2	Which Algorithm rounds faster?	<p>Time is money, estimations need to be quick and provide answers as fast as possible. With working with larger datasets, the big o notation definitely comes into play as some algorithms will be optimised for smaller datasets and others will be for larger ones. This although will be specific to the datasets used for the SEE methods, if there is a median for results this would be preferred as it will offer more flexibility with working with varying datasets whilst still being able to provide answers.</p>
Q3	How robust is the algorithm?	Inconsistencies will invalidate a SEE methods authenticity. Methods

		should be robust and provide a somewhat consistent answer when working with similar sized datasets or variables without these estimations will be left with no backbone and seem random.
Q4	Are these approaches suitable for effort estimation?	Some SEE methods will be more long winded than others. Not all methods are as passive as others. For example, a machine learning method will do the work itself once data is input, whereas other older methods will need to be calculated manually.

Table of Proposed questions [1]

2 - Literature Review & Research

2.1 - Software Effort Estimation

SEE is driven by microeconomics and the strain experienced when developing a project with limited resources, as mentioned in Boehm (2008). The pursuit to develop more of an understanding and get ahead of the curve has led to the exponential growth in SEE methods we have today, specifically with the rise of AI. The use of AI meant that these methods could have new solutions which require less physical labour.

This study will explore SEE methods in regard to the implementation of AI and how this has helped to move forward the development of SEE.

2.2 - Machine Learning

AI was able to be used for SEE using machine learning, this allows machines to learn from data provided. Using data, which was used previously for other SEE studies, information can be used with mathematical functions to provide an estimate, just as previous studies would pre-ML.

The design and guideline of this review is entrenched on "Procedures for performing systematic reviews" by Kitchenham (2004) and Figure 1 shows how it was executed.

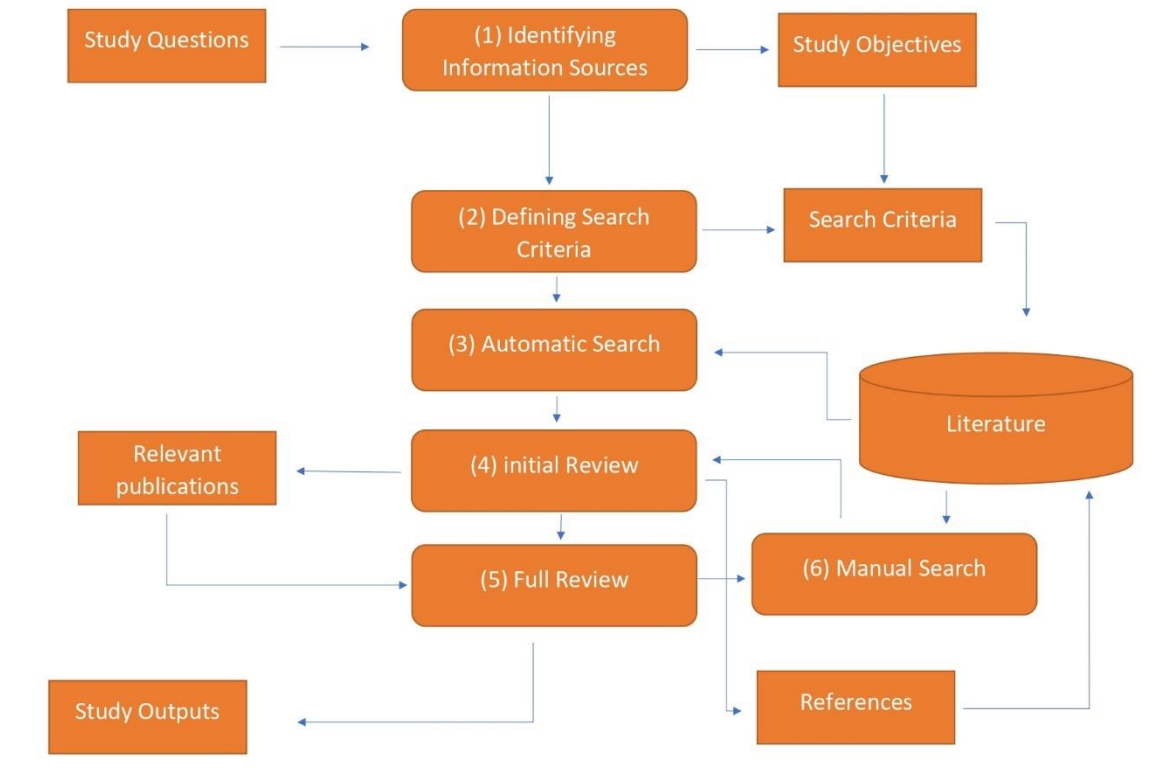


Figure Overview of literature review process [1]

Identifying information sources: sources of information was identified and filtered out of the journals, books and websites using the study questions.

Defining search criteria: My search criteria was defined from the study questions.

Questions listed were relative to most SEE studies, so for my search criteria anything relevant to SEE would be considered. To guarantee relevant and updated information due to the rapid changes in software engineering, studies from before 2005 will not be considered as it will not contain depth when it comes to machine/deep learning. To get a greater understanding for SEE, studies before this will be considered.

Study objectives: Study objectives were based on the questions I wanted to answer.

Objectives were to help identify things I should look to implement for myself, whilst the questions were more for comparison. These objectives were open, allowing me to have a greater understanding from these studies.

Automatic search: Automatic searches gave access to a wider range of libraries.

Specifically, the use of google scholar was used to find related articles which were based

on very close studies. This allowed different approaches to same or very closely related studies to be seen. ACM allowed access to surveys based on the questions proposed and the use of IEEE Explore articles. These sites provide authenticity which is particularly important when looking for surveys because validity is something that cannot be compromised.

Manual search: Manual search was used for references quoted in studies in articles seen that were relevant and gave more depth in understanding of what was wrote.

Initial review: With the initial report, the first thing to do would be to scan the paper for keywords associated with either questions or objectives. The abstraction of the article also gave great insight into what was being discussed. finally, I would look at references, if a reference that had already been saved in my list came up it would add more relevancy to the article giving more reason to conduct a full review.

Full review: In the full review close attention was paid to structure and would look specifically at the keywords or references mentioned in the initial review. I would also add references I had not seen before and manually search for these.

References: References played a big part in my searches as it helped to find other articles explaining the foundations of SEE, which otherwise may have not been found. Not only this but it also helped me to identify smaller reports which were just as important as the bigger ones in the list.

The logic for this review is to summarise findings on methods used for SEE to then propose a more balanced and robust method that will predict with high accuracy.

2.3 - Study Design

The study of this paper is formed from two questions: (1) a review of SEE method literature, (2) a survey performed alongside SEE methods.

2.3.1 - Study Objective

The objective of this study is to answer the proposed questions in table 1 but to also highlight the following aspects:

- what these SEE methods are particularly used for and if they meet the needs of the user(s)
- Ease of implementation
- Feature sensitivity
- Several types of implementations

2.3.2 - Information Sources

The analysis in this paper is based on two sources of information:

Literature Review: Any publicly published forms of literature which highlight and/or answer either of the objectives or questions. In this study, the foundations of SEE will be explored and the implementations and reasons behind them will also have light shone on them. Just like many things within this day and age, the best of the methods will never be publicly released but instead sold on for profit and utilised for future implementations. The of books, journals and dissertations will be at my dispose to explore this topic and provide myself with answers for future work.

Surveys: Surveys conducted by those looking for similar answers to me will provide a basis for my own research and will also give me answers for the objectives and answers I am looking for

2.4 - Surveys

Using the procedures laid out in “Procedures for performing systematic reviews” Kitchenham (2004) I embarked on with reviewing the surveys which I had approved were adequate for my objectives.

The following surveys were read:

- Survey 1: “State of the Practice in Software Effort Estimation: A Survey and Literature Review” - Trendowicz, Münch, Jeffery (2011)
- Survey 2: “Software Effort Estimation using Machine Learning Techniques”-Shivhare, Rath (2014)
- Survey 3: “A Baseline Model for Software Effort Estimation”- Whigham, Owen Macdonell (2015)
- Survey 4:” Multi-objective Software Effort Estimation” - Sarro, Petroziello, Harman (2016)

- Survey 5: “Data Mining Techniques for Software Effort Estimation: A Comparative Study” - Dejaeger, Verbeke, Martens, Baesens (2011)
- Survey 6:” Software Effort Estimation using Machine Learning Techniques” - Monika, Sangwan (2017)
- Survey 7:” Software Effort Estimation Accuracy Prediction of Machine Learning Techniques: A Systematic Performance Evaluation” – Mahmood (2021)
- Survey 8:” Features-Level Software Effort Estimation Using Machine Learning Algorithms” - Hammad, Alqaddoumi (2018)
- Survey 9: “Software Effort Prediction using Statistical and Machine Learning Methods” - Malhotra, Jain (2011)
- Survey 10: “Spectral Clustering Effect in Software Development Effort Estimation” - Silhavy, Silhavy, Prokopova (2021)

The surveys listed discussed numerous techniques and proposed different arguments towards developing a ML model for SEE. A lot of these models shared some of the same implementations which gives more emphasis on why it should be used. ANNs (Surveys: 2,3,5,6,7,8,9) were heavily discussed and compared amongst other models; with ease of implementation and understanding these models provided high accurate predictions which gave adequate solutions for SEE. LR (surveys:4,7,8,9) is the simplest form of ML techniques that can be used for SEE, being used as almost baseline for performance against the other models within the surveys. SVMs provided consistent results through all the surveys (5,7,8,9) it was compared in and had numerous ways of being tweaker and implemented; survey 5 used SVMs with least square regression as hybrid model. However, many studies looked to more niche techniques that were not as widely used. Survey 10 particularly looked at clustering and so the use of Stepwise Regression (SR), Special Clustering and Stepwise Regression (SRSC) and Stepwise Regression with Spectral Clustering and Categorical Variables (SRSCCV). On the other hand, survey 6 looked at other techniques such as Genetic Algorithm (GA) hybrids and Analogy. Survey 4 used CoGEE, GA-CI, GA-SAE, NSGAI-UO and concluded the survey with suggesting the use of tabu search. Survey 2 was the only survey to use naïve bayes, survey 3 provided a new baseline method (ATLM), survey 7 used an OLC implementation, survey 8 included the use of the K-star technique.

Datasets are not ubiquitous as many are private which caused the use of the same datasets being used within the surveys; this could be due to validity as these datasets are

somewhat standardised for the use of SEE. The main datasets used were USP05 (surveys: 2,5,8), COCOMO81 (surveys: 3,5,6), Desharnais (surveys: 3,4,5,7), Maxwell (surveys: 3,4,5,7), ISBSG (Surveys: 5,10). Two of surveys (1,7) used in house datasets which is less frequently used due to privacy as mentioned earlier. Survey 5 however, contained additional datasets that differed from all the other surveys, specifically, the Experience, ESA, Euroclear and Cocnasa.

Statistical measurements were also shared amongst surveys as the application of specific measurements allowed the survey to highlight advantages and limitations of models mentioned. RMSE was the mode of statistical measurements seen in the surveys, with surveys 2,4,9 and 10 using this as main measurement for model accuracy. The next most used measurement was PRED(X); survey 7,9 and 10 used 25 as the value for X. Joint last was MMRE (surveys:7,9) and MAE (surveys: 8,9). A couple of the other surveys used measurements that were not shared with the other surveys being mean absolute residual (survey 10), mean absolute percentage error (survey 10), sum of the squared error (survey 10) and Relative absolute error (survey 9).

The use of feature selection to improve models is mentioned in survey (2,7,9) but the techniques differ between surveys. Within survey 2 Rough set analysis is used to reduce features, whereas in survey 7 the use of Genetic Algorithm (GA) and Best Fit (BF) are proposed, and then in survey 9 the WEKA correlation-based feature selection tool was used.

Cross validation was mentioned in surveys 2,3,4,5,8,9,10 displaying the importance of the technique to estimate performance on unseen data. Leave-one-out cross validation was used throughout the surveys (2,3,4) as well as multiple-fold cross validation was also used (4,8,10), with survey 9 mentioning other techniques such as hold-out and K-cross validation.

To conclude many of the surveys suggested small improvements or things to look out for when considering implementing a SEE model. However, none did in as much depth as survey 1 did. Survey 1 compiled the following list for considerations when selecting a particular estimation model:

- Expert involvement: Method should not require expert involvement.
- Required Data: The method does not require copious amounts of measurement data of a specific type.

- Flexibility: The method provides context specific outputs, not tied to one specific estimation model
- Complexity: Minimal complexity (easier to understand)
- Support level: Support provided with method.
- Handling uncertainty: Handles uncertainty in outputs and inputs
- Comprehensiveness: The method can be used for different types of project activities on various levels of granularity
- Availability: The method is always applicable at any stage of the software cycle
- Empirical evidence: Validity is to be proven by theoretical and practical evidence.
- Informative power
- Reliability: it provides accurate, repeatable, precise outputs
- Portability: easily adaptable to another context.

2.4.1 - Summary

Summary of models

The table below (table 2) represents a summary of the surveys listed above.

Case study	Question(s) Answered	Objective(s) answered	Type of SEE model(s)	Datasets
Survey 1	None	All	None	In-house
Survey 2	All	All	Naïve bayes, ANN	USP05-FT, USP05-RQ
Survey 3	All	1,2,4	ATLM (Automatically transformed linear model), ANN, ABE-PSO	cocomo81, Desharnais, OrgAll, Maxwell
Survey 4	1,3,4	All	CoGEE, GA-CI, GA-SAE, NSGAII-UO, CBR1-3, LR and CART (Classification and Regression Trees)	China, Desharnais, Finnish, Maxwell, and Miyazaki
Survey 5	All	All	Ordinary least squares regression, OLS (Ordinary Least Squares) regression with log transformation, OLS regression with Box Cox (BC) transformation, Robust regression, Ridge regression, least median squares regression, MARS (Multivariate Adaptive Regression Splines), CART, Model tree, Multi-layered perceptron neural network, Radial basis function networks., Case-based reasoning, Least squares support vector machines.	ISBSG, the Experience ESA, USP05, Euroclear, Cocnasa, COCOMO81, Desharnais, Maxwell
Survey 6	All	3,4	ANN, Fuzzy System, Genetic Algorithm, Hybrid of (ANN + Fuzzy + GA), Analogy	NASA, COCOMO81
Survey 7	1,4	3,4	Adaptive Neuro Fuzzy, SVM, MLP (Multi-layered Perceptron), ANN, OLC, linear regression	Maxwell, Desharnais, in-house
Survey 8	1,3,4	2	ANN, SVM, K-star, Linear regression	USP05-TF

Survey 9	1,3,4	1	ANN, Decision Tree, Linear Regression, Bagging, SVM	China
Survey 10	1,3,4	2,3,4	SR, SRSC, SRSCCV, IFPUG	ISBSG

*Table of Summary of Surveys [2]***Datasets**

Most of the articles read maintained consistency in the processes they used, providing a foundation for anyone who looks to implement their own SEE model(s).

The datasets which were used amongst the papers were:

- China
- Desharnais
- Maxwell
- COCOMO81
- ISBSG,
- USP05-RQ
- USP05-T

These datasets provided a large amount of data and variables, allowing those to refine what they wanted to use, giving them more freedom and flexibility.

The datasets varied in size, ranging from 51-81 projects(small) to 1,000+ (large/very large); due to this, models were able to show their strengths with the range of the data provided by the datasets.

Validation

There were also consistencies in methods used to validate the data, these being:

- 10-fold-cross-validation
- hold-out
- K-cross validation
- Leave-One-Out-Cross-Validation
- 5-fold Cross Validation
- n-way cross-validation

with leave-one-out being the most used.

Validation helped models to be more accurate by allowing them to have different intervals for the amount of data they trained on.

Sample data also helped this too by using a small percent of the training data as testing data, allowing a higher accuracy to be obtained during training of the model. When the model is then used on testing data, this would result in a higher accuracy of predictions.

Measurements

Measurements were one of the most important things to consider when conducting testing as these were used to compare success and how robust models were.

The most used measurements for predictions were:

- Mean magnitude of error (MMRE)
- Mean relative error (MRE)
- Prediction percentage, specifically, PRED (25).

However, it should be noted that the use of mean absolute error (MAE) has now become more common for its ability to be unbiased (Towards over or underestimation). All 4 of these should be strongly considered.

2.4.2 - Discussion

Question	Summary
Q1	As thought before conducting my review, accuracy was the most crucial factor, and this was confirmed by all studies. The importance of such small numbers can have a massive effect in predictions e.g., MMRE being 0.15 vs. 0.30. that small difference pays a big cost in the long run and can prevent an under/over estimation for costs
Q2	The speed is important although not much of a dominant factor. Shivhare, Rath (2014) conducted 5000 epochs whereas others did not. hence there's different amounts of times to train the estimation. Some SEE models took this into consideration and tried to make them easier to run but some did not. In this case their models would be better to be ran off cloud or a system with a strong GPU to help speeds.

Q3	Robustness is as important as accuracy. If your SEE model can only produce consistent results with one dataset, it takes away from the validity which makes it less appealing to be used.
Q4	If your SEE model is very feature sensitive and takes a specific number of variables, then no matter how accurate it is, it may not be useful for SEE as data is normally varied e.g., it takes 16 variables, but a company's dataset only provides 13.

Table of Summary of questions [3]

Objective	Summary
What SEE methods are used for and if it meets the needs of the user(s)	Most methods were used for company needs and matched the needs of the user, sometimes outperforming state of the art techniques.
Ease of implementation	Baseline methods were easier to implement as they provided more flexibility and explanation to what exactly was happening in the background. Ease of implementation translate to complexity, and this is important as end users do not want something that they need to be heavily trained on to use according to Trendowicz, Münch, Jeffery (2011).
Feature sensitivity	Feature sensitivity is important as data varies. Reducing features to a smaller number normally accommodates for most data.
Different types of implementations	Different types of implementations can make a SEE method perform better e.g., SRSC compared with SR. it should be considered trying hybrids or

	adding more data validation techniques
--	----------------------------------------

Table of Summary of objectives.[4]

To conclude, somethings should be noted for any SEE model implementation:

- No true guideline answer on how to test SEE methods. Things such as validating data, the Jack-knife technique has no say about how many variables you should leave out. Some of the surveys read left 10 out whereas others left 1 out according to Whigham (2015). Leave-one-out is the best for results and should be implemented.
- Use 3 performance accuracy metrics: MMRE, PRED (25), MAE
- A benefit analysis would make a SEE model more appealing as it would make it cost beneficial.
- Decision Tree, ANN, Naïve bayes, SVM, or a hybrid of these should be considered for the next stages of implementation as they support machine learning.
- Focus on data quality which will lead to highly predictable attributes being evident.
- Ensemble methods are better than solo methods (Further backing for a hybrid model)
- Similar features should be filtered out as well as outliers to prevent performance decreases.
- Smaller data models are easier to learn from than larger ones.
- Selective classification
- Accurate estimation is seen as an issue.
- There are many variables for project overruns and why they are not properly addressed.
- Expert estimations are very close in accuracy to SEE models. However, SEE models are used because of consistency, they are not based off of guess-work or foggy memories.
- Graphs make complex information easier to understand and should be implemented when they can be.

With the questions I asked for the SEE methods I reviewed I would like to not only apply them to my work but also add to the questions I proposed.

The following questions I propose to my own implementation and further research:

Question	Reason
----------	--------

Q1	Which Algorithm is better in terms of accuracy?	Refer to table [1]
Q2	Which Algorithm rounds faster?	Refer to table [1]
Q3	How robust is the algorithm?	Refer to table [1]
Q4	Is my approach suitable for effort estimation?	Refer to table [1]
Q5	To what extent the algorithm sensitive to features?	This is also important. If a SEE method is more lenient to one feature than it, then becomes dependent on that which can have its advantages and disadvantages

Table of questions for implementation [5]

With these questions I hope to cover most questions that are considered when not only comparing a SEE model but when choosing a SEE model to use.

3 - Design & Development

3.1 - Python

The programming language is very relevant to the type of models you can implement as the libraries can ease the implementation and tuning of the algorithms. Libraries provide an efficient and straightforward implementation of a class/function without having to hard code it. Not only this but a wider range of statistical testing can be implemented due to access of libraries.

For me, the ideal programming language to use was Python. Python offered a range of different libraries and statistical testing and allowed the use of 'R' packages. The models that I found during research had numerous ways of being implemented into python and it was clear that people used python as their language to develop SEE models due to the breadth and depth of Algorithms, Statistical testing and machine learning libraries which showed its strength and reliability.

3.1.1 - Anaconda

Anaconda is a program which allows you to build separate python and R environments and allows packages to be easily installed and managed.

3.1.2 - Jupyter notebook

Jupyter notebook is web-based platform which allows python to be coded and ran online by using modules to separate code into blocks. This can be used under Anaconda which allows ease of libraries to be called and Python files to be micromanaged.

3.2 - Computer Spec

Computer spec needs to be noted as computation speed is proportional to how quick models can run. This can alter results, but the models are run relevant to each other as they use the same environment. The models will be run on my native system using google chrome (Jupyter notebook & Google Colaboratory).

My computer spec is:

- Intel(R) Core (TM) i7-4710HQ CPU @ 2.50GHz (Boosted to 3.2GHz)
- 8.00 GB (7.89 GB usable)
- 64-bit operating system, x64-based processor
- Nvidia GeForce GTX 850m

3.3 - Models

Upon conducting my research, I discovered that access to the highest performing models were private as they are the most profitable. This leaves models that are available to the public, and because of this the range I have access to is reduced and the accuracy these can achieve won't be on the same level of the state-of-the-art models, however I still have many methods at my hands.

Software effort estimation has several ways to be implemented within python. Models have parameters and can use different libraries to be built. Below are the models that interest me the most.

Artificial Neural Network (ANN)

An ANN works by taking inspiration from neurons in the brain linking with different neurons within different layers to come to a prediction.

I would like to use this as I have experience with this method previously.

Artificial neural networks offer great flexibility in layer and node and has good support via python using the 'MLPClassifier'.

Decision Tree (DT)

IBM stated that Decision Trees work by forming a root node with branches that spread out into decision nodes. These decide based on features given whether one option is better. The lead nodes represent all the possible outcomes from the dataset given

(<https://www.ibm.com/topics/decision-trees#:~:text=A%20decision%20tree%20is%20a,internal%20nodes%20and%20leaf%20nodes>, Accessed: 28/04/2023)

As the data I would be using would be continuous, I would have to consider the use of Regression Decision Tree (RDT). These work by continuously evaluating the property vector to lead to all possible outcomes (Myles,2004)

Python offers good support for decision trees too, through the library 'Decision Tree Regressor'.

Random Forest (RF)

Random Forests are based off regression trees with Liaw and Wiener (2004), stating that random forests change how the classification or regression trees are constructed.

Instead of using the best split amongst variables, splits are done by using the best among a subset of predictors randomly chosen at that node.

Random forest has support in Python as well in the form of the 'RandomForestRegressor' Library

Linear Regression (LR)

Linear regression seems to be the simplest of the methods mentioned with Mali (2023) saying that Linear Regression works by finding the linear correlation among variables, specifically the relationship between independent variables to the dependent variable.

Python offers support on linear regression once again through a library, specifically the 'LinearRegression' Library

Support Vector Machine (SVM)

Noble (2006) summarised, says that Support Vector Machines are algorithms that learn by example and have many uses in regard to classification and regression, stating that an SVM is a mathematical entity for maximising a particular mathematical function with respect the data given.

K nearest neighbours (KNN)

The basis of KNN is that local interpolation is used to predict the target using the targets associated of the nearest neighbours in the training set (<https://scikit-learn.org/stable/index.html> - Accessed: 28/04/23).

KNN is a recommended method of classification by Peterson (2009) saying that 'the first choices for a classification study when there is little or no prior knowledge about the distribution of the data' should be KNN. As python supports KNN through libraries it should be considered when implementing the model.

Hybrid Models (HM)

In the literature review there were many examples of hybrids being used with high success rates of accuracy with predictions (survey: 3,6), and they were done by taking the best parts of 1 or more algorithms and combining them.

These models would have to be built manually and I would have to specifically test combinations on the best performing models.

3.4 - Libraries

Libraries provide pre-written code that is often very efficient, stable, and reliable. Libraries contain built-in-modules that provide access to system functionality as well as standardised function solutions. (<https://docs.python.org/3/library/> - Accessed: 28/04/23)

NumPy

NumPy provides support for array and matrices, allowing them to be manipulated using functions that they provide.

NumPy will be used ideally in my program to figure out statistical performance measures like Root Mean Squared Error (RMSE).

Pandas

Pandas is a data analysis library which allows manipulation and analysis of data through their functions.

I specifically want to use Pandas for handling my datasets. This would be done using data frames which allow me to manipulate the datasets.

Matplotlib/Seaborn

Matplotlib is a plotting library which provides access to forming graphs and allows for the visualisation of data.

Seaborn is another library which helps with data visualisation and can be used alongside Matplotlib.

I look to use both of these together to produce some visualisation of what is happening with the data.

Timeit

Timeit will simply be used to record the speed/time of the algorithms so that we can compare on some level, the complexity of each model with their corresponding data.

SciPy

Many uses including optimisation, algebra, and special functions.

This will be used for loading my datasets as concluding my research some datasets are in the form of arff files whilst others are csv'. To add to this I'd also look to use Pearson's correlation to highlight correlation in datasets.

Scikit-learn (Sklearn)

Scikit -learn provides many different algorithm and functions for machine learning.

This library will be used for the different models as there is a range of different ones. Models can be tweaked and will provide parameters which can be adjusted to the users need.

Re

Used for regular expressions, I look to use this for my interface to make sure only certain options can be chosen.

3.5 - Model training

Models are built through Sklearn using TensorFlow and allow models to be built, learn, and execute efficiently. There are tools available to help with overfitting and specific adjustments with models and provide great flexibility in construction. This will allow the needs of the datasets to be accustomed for when testing and training the models.

Hyperparameters

Hyperparameter tuning is provided by Sklearn models and allow parameters to be set specifically to what is required but there are also default models which have a set standard of parameters.

Model performance can be hindered greatly depending on parameters but can also be improved. Estimators are used on datasets when parameters are not given and generalise what may be best for the model and datasets. Parameters when tuned however can provide that bit more of accuracy when needed for models.

Random State

Random state provides a stable and consistent randomisation on code that can fluctuate between runs. This specifically would be used for splitting my test and train data so that it always splits the same data when run and this can also be used for each algorithm when executing it so that the results can be consistently produced.

Test Size

Test size is related to the split of the datasets and allows a ratio between 0-1 to be set for how the data is partitioned for testing and training. The normal ratio is 0.33 but this will be adjusted at different times depending on the dataset.

GridSearchCV

Grid search is an exhaustive way of searching through parameters to find the optimum of the specifically given parameters.

I look to use this with all my models to hyper tune through the means of brute force using the most influential parameters. I can also set the number of folds the data goes through by defining a value for cv in the Grid Search.

Support Vector Machine (SVM)

SVMs have 11 parameters which can be set but the ones I specifically will highlight, and use are the following:

- Kernel – defaults to rbf but I specifically want to test 'linear', 'rbf', 'sigmoid'.
- Gamma – defaulted to 'scale' but also has the option of 'auto' so this will be tested.
- C – defaulted to 1. As C is the parameter which controls the trade-off between a low testing error and a low training error a broad range should be used. Ideally 1-10 would be good.

Decision Tree (DT)

Decision Trees has 12 parameters but the ones I will consider for hyper tuning are:

- **Max_depth** – defaults to 'none'. 'If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples' (Pedregosa,2011). This feature is interesting as it can vastly affect run times, specifically for medium to large datasets as the trees are not limited to a certain depth however It also allows all possible outcomes to be reached.
- **Min_samples_split** – minimum number of samples required to split an internal node.
- **Min_samples_leaf** - The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

K nearest neighbors (KNN)

KNN has 8 parameters and the ones I will be using for hyper tuning are:

- **n_neighbors** – default is 5 for the number of neighbors but I would like to test a range of 1-10
- **weights** – uniform is the default. But I liked to test 'distance' as this uses the inverse of the weight points distance.
- **metric** – Minkowski is the default, but I would like to use 'Euclidean' and 'Manhattan' as well.

Random Forest (RF)

RF uses 17 parameters but the only ones I am interested in tweaking are:

- **max_depth** – how far trees a trees length can be. The depth should be tested between 1-100
- **n_estimators** – the number of trees in the forest. This should be tested between 100-2000
- **min_samples_split** – default 2 but I would like to test 1-5.

Linear Regression (LR)

LR uses 4 parameters and I want to test 2 of these:

- **fit_intercept**.
- **n_jobs** – decides on how many processors are used. -1 = all processors, 1 the default

Loss Function

Loss functions plays a major part in the tuning of a models as this is how the models are adjusted. Minimising the loss output is what controls the direction of predictions and because of this, I made sure to put this at the forefront for the models I proposed.

Mean Absolute Error (MAE)

MAE is the mean magnitude of the difference between the measured value and the original value (doubtnut,2020).

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

Figure of MAE formula (Wikipedia,2023) [48]

Mean absolute error will be implemented using sklearn and will be used as a scorer for the models meaning the lower the score the better the model.

Pred (25)

Prediction (25%) Is a measure which many papers use to score accuracy of models. Pred (25) works by giving a ratio of how close the predicted values are to the actual values, in my models I will specifically be using predictions within 25%.

Mean Squared Error (MSE)

Mean square error measures how close a regression line is to a set of data points. Works by taking the mean of errors squared (Gupta,2023). This will be implemented using the sklearn.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

Figure of MSE formula (Wikipedia,2023) [49]

Root Mean Squared Error (RMSE)

Root mean squared error is the square root of MSE and will be used through sklearn.

R²

R² is a statistical measure that represents the proportion of the variance for a dependent variable that is explained by an independent variable in a regression model (Fernando,2023).

R squared is supported in python through sklearn and so I will be using this an indication for how successful model performance is.

$$R^2 = 1 - \frac{RSS}{TSS}$$

Figure of R² formula (Wikipedia,2023) where RSS = Sum of squares of residuals and TSS = Total sum of squares. [50]

Time

Using the python library 'Timeit' I will time how long it takes to run each model for each of the datasets used.

3.6 - ScottKnottESD

ScottKnottESD is a R package available through CRAN with the sole purpose of measuring effect size between techniques given the measured performance. Unlike the normal Scott Knott test, ScottKnottESD considers the magnitude of the difference of treatment means within a group and between groups and does not produce overlapping groups like post-hoc tests (Tantithamthavorn ,2019). ScottKnottESD is a necessity to the conclusion of the implementation as it will measure the algorithm across the board for each dataset grouping in order from highest to the lowest of what the best performing model was for the metric declared.

3.7 - Datasets

Datasets were found from various locations and came in different formats including .arff files and csv files. Below is a breakdown of the datasets I managed to find.

Size of datasets are defined by the following:

- Small 0 – 299
- Medium 300 – 599

- Large 600+

China – the China dataset is a medium size dataset containing 499 entries of projects with 17 independent features and 1 dependent feature with all continuous values.

(Venkataiah, Nagaratna, Mohanty, 2022)

Desharnais – The Desharnais dataset is a small size dataset containing 81 entries of projects with a small feature set of 10 independent variables and 1 dependent feature.

This data contains discrete and continuous data. (Esteves, 2018)

Maxwell – The Maxwell dataset is a small of the datasets containing 63 entries of projects however has the largest feature set of 26 independent variables and 1 dependent

variable. This data contains discrete and continuous data. (Venkataiah, Nagaratna, Mohanty, 2022)

Kitchenham – This dataset is a small dataset with 145 entries containing 9 independent features with 1 dependent feature. This data contains discrete and continuous data

(Tsunoda, 2017)

ISBSG – this dataset is the largest of them providing 952 entries with 10 independent features and 1 dependent feature, with both continuous and discrete data.

Finnish – this dataset contained 51 entries with the most features coming in at 43 independent features and 1 dependent feature. The data contained continuous and discrete data.

USP05 – this dataset contained 203 entries with 16 independent features and 1 dependent features. The data contained continuous and discrete data.

Dataset	Size	Entries	Independent features	Dependent features	Continuous Data	Discrete Data
China	Medium	499	17	1	Yes	No
Desharnais	Small	81	10	1	Yes	Yes
Maxwell	Small	63	26	1	Yes	Yes
Kitchenham	Small	145	9	1	Yes	No
ISBSG	Large	952	10	1	Yes	No
Finnis	Small	51	43	1	Yes	No
USP05	Small	203	16	1	Yes	No

Summary of datasets [6]

3.8 - Feature Selection

Feature selection derives from cleaning data and provides a buffer for overfitting and increasing speeds of algorithms, as the information fed from the datasets is reduced cutting the computational strain.

Features within datasets have relationships between each other. A change in one feature can have a high influence in the other feature it shares the relationship with. In my case the most important feature would be the 'Effort' and how this is affected by the other independent variables.

This, of course, can have its advantages and disadvantages, there is potential that accuracy of course will fall due to less information being at the hands of algorithms. What I look to do is provide my users with the choice to choose whether they would like to use the hyper tuned models or the baseline models which are defaulted parameters and use all the features available in the chosen dataset.

There are different ways to choose features, and these differ between ways of implementation and so below is a discussion of some of these methods which I will consider using.

Pearson Correlation

The Pearson correlation measures the strength of the linear relationship between two variables. It has a value between -1 to 1, with a value of -1 meaning a total negative linear correlation, 0 being no correlation, and + 1 meaning a total positive correlation. (Williams,2020)

As we will be measuring every feature to the rest of the features available python provides Pearson correlation that is able to process all features in the dataset.

Genetic Algorithm

Derived from Darwin's natural selection, Genetic algorithm is used for solving constrained and unconstrained optimisation problems using biological evolution.

(<https://uk.mathworks.com/help/gads/what-is-the-genetic-algorithm.html> - Accessed: 29/04/2023)

Genetic Algorithms are not only used for feature selection, but they can provide highly accurate results when configuration is tweaked. With a combination of using

Gridsearchcv and the decision tree classifier, results can be concluded about the most influential features.

Extra Trees

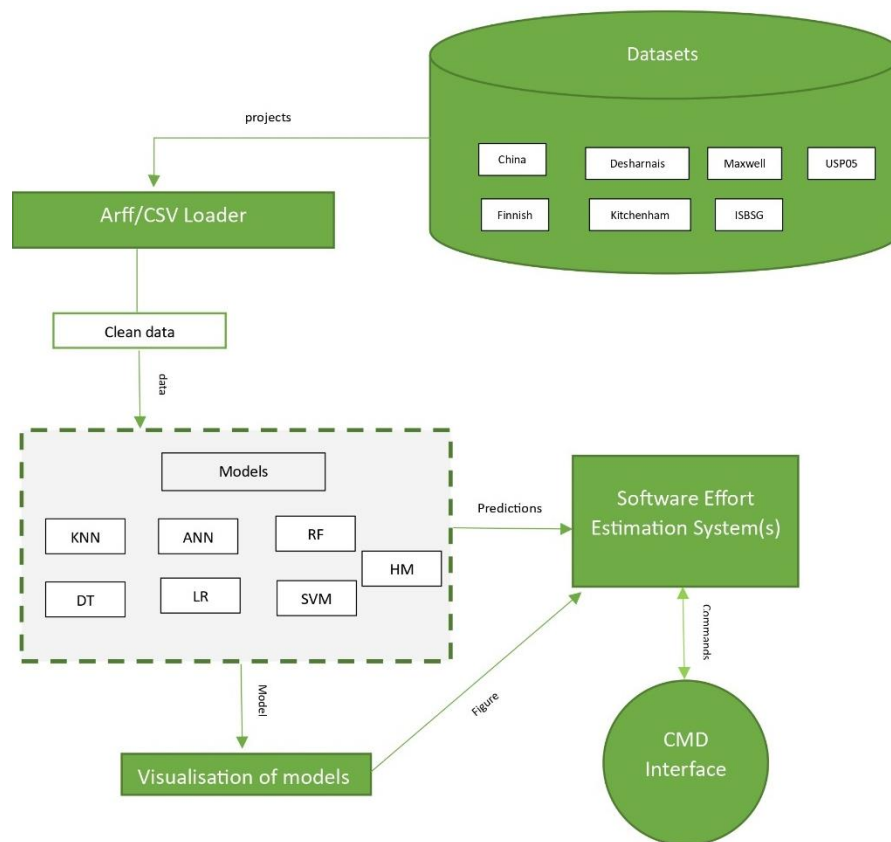
Like RF, Extra Trees differs by the construction of the DT. This works by providing at each test node with a random sample of k features from the feature set. Each tree then decides the best features to split the data by default on the Gini Index. (Gupta, 2020)

Python supports the implementation of Extra trees through sklearn, and I look to use it by splitting my data into independent and dependent features to find out the results.

Interface

Users should be able to interact with the system, running different models with different datasets through the use of a command line interface, which will provide options accordingly.

3.9 - System Design



4 - Plan

This following section is based off what I found during my research and literature review section.

4.1 - Requirements

#	Stakeholder	Requirement
1	Investor, Project managers	Machine learning model(s) which has the ability to predict the effort required for a project
2	Developer	The ability to tweak and add models, features, and other datasets to the program
3	Project managers	Data used is from real life projects that were completed.
4	Developer	Modularisation of models so they have the ability to be maintained and configured
5	Developer	The ability to view the accuracy of models

Table of requirements [7]

Project stakeholders:

- Project managers: The use of predicting projects with automation.
- Deep Learning Research: new ways of optimising models for greater accuracy of models
- Project manager: Reduced overhead when considering effort required for a project.

4.2 - Software Engineering Methodology

The timeline proposed below shows that the model(s) created need to have flexibility in development and be open to change; the agile methodology is suited to this.

Phases of development ensure the requirements of the models are curated to the objectives set.

4.3 - Gantt Chart

My initial workplan was in the form of a Gantt chart as it made information easier to understand. Each section was near enough followed one after the other, as one finished the other section started. Listed below in Figure 2 is my initial Gantt chart (created using LaTeX).

Proposed Gantt chart:

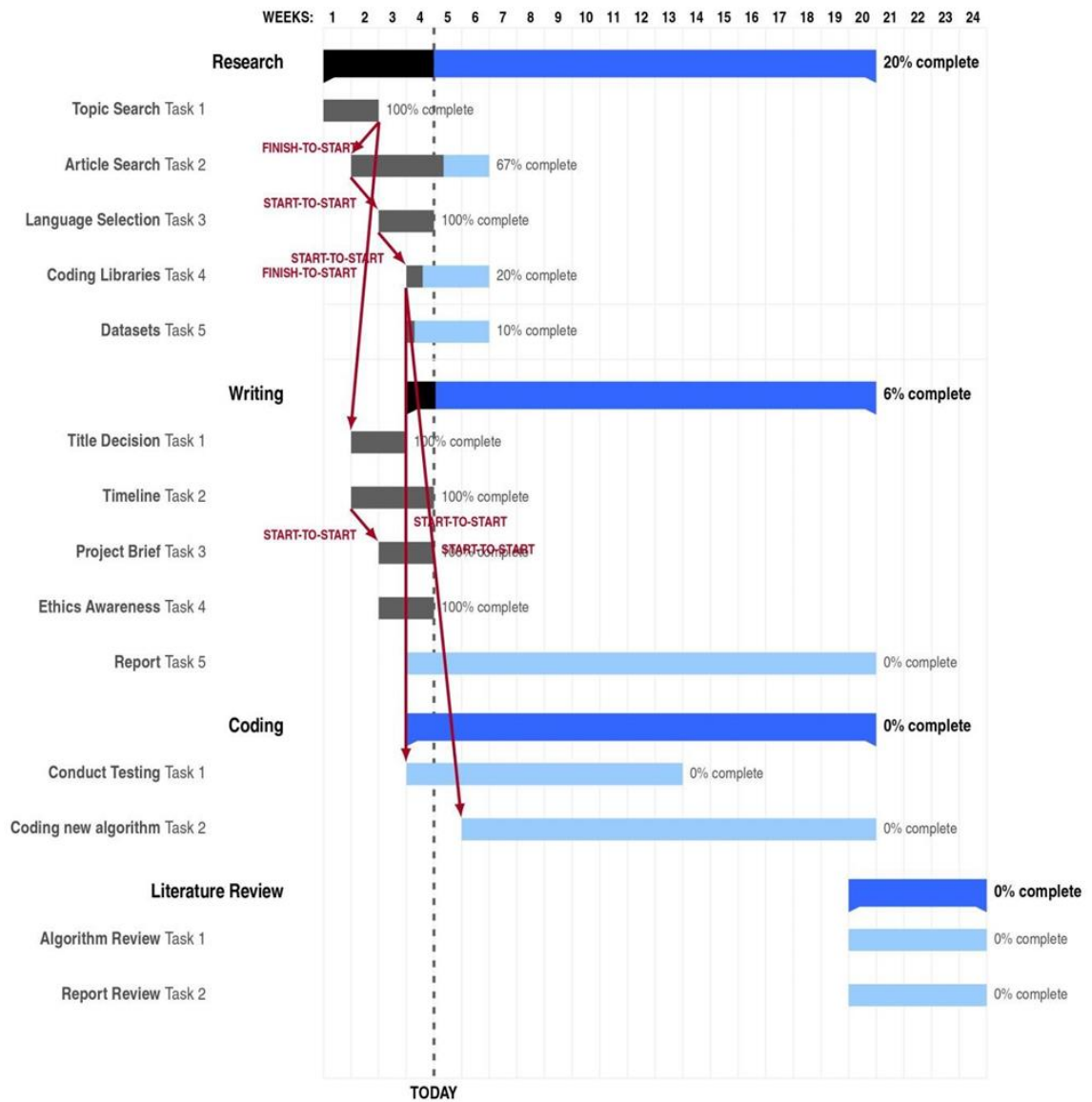


Figure of Initial Gantt Chart [3]

Upon conducting my research for my project, a revision to the plan was done (fig.3). As seen here some of the tasks needed to be completed that were running behind are shown by how filled the task bar is.

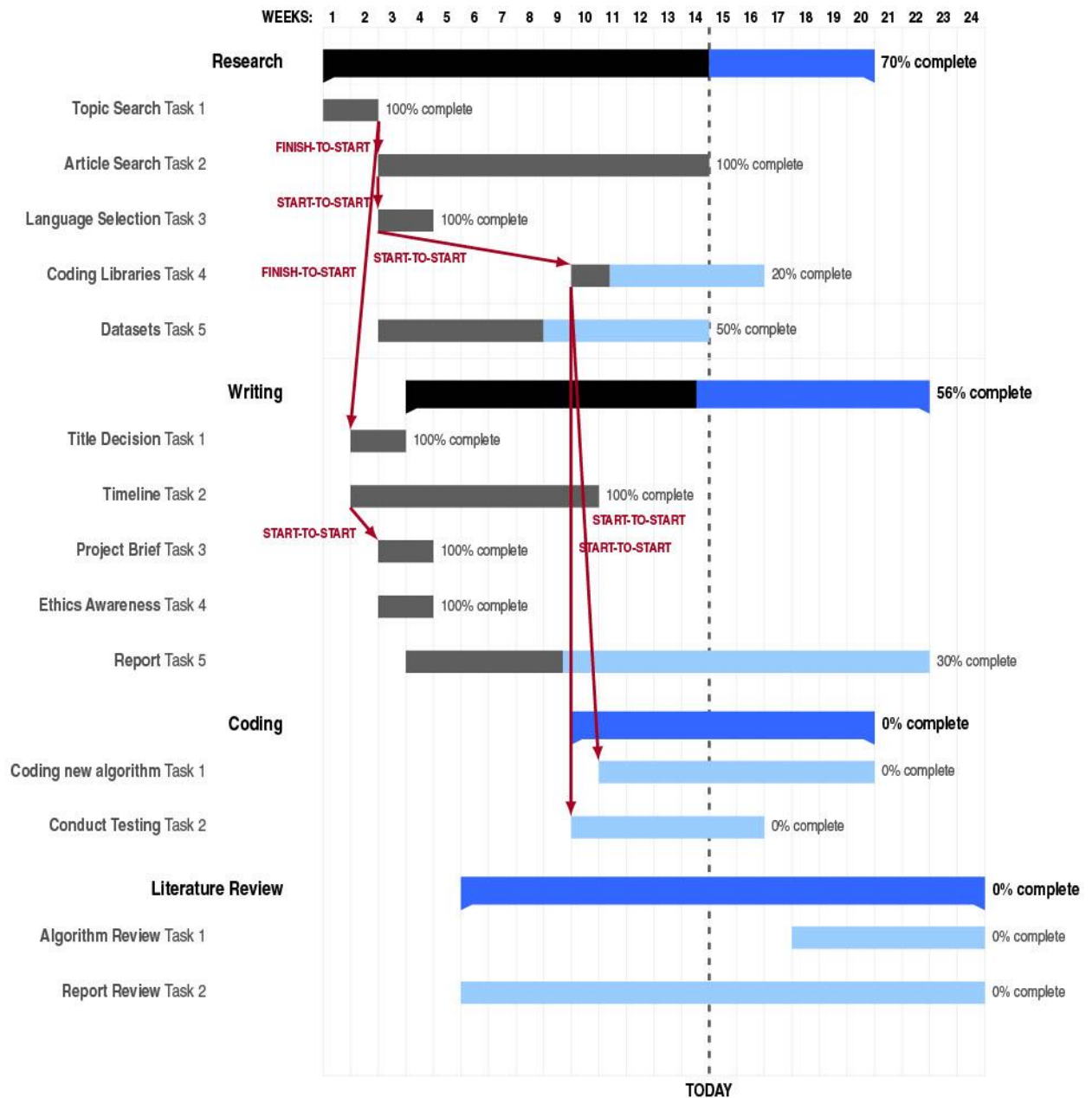


Figure of Revised Gantt Chart [4]

Task Descriptions:

Task	Description
Research: Topic Search	Developing depth on the topic at hand – software effort estimation and the models used to solve the problem at hand.
Research: Article Search	Finding articles which relate around the topic SEE
Research: Language Selection	The combination of the topic search and article search to conclude a language that is well

	supported and robust for the implementations I want
Writing: Title Decision	Deciding what the report would be about.
Writing: Timeline	How progress is going and the layout of the plan
Writing: Project Brief	Giving information for what is to come
Writing: Ethical Awareness	The ethics behind the project
Writing: Report	Putting together all the previous writing into one place to present
Coding: New algorithm	Bringing my findings during the research section to build a model(s) for SEE
Coding: Testing	Testing robustness of models
Literature Review: Algorithm Review	Reviewing the algorithms and the findings
Literature Review: Report Review	Reviewing the report and formatting

4.4 - Plan adjustments

Listed below are changes to the plan which occurred due to unforeseen circumstances or due to a preference being more preferred.

4.4.1 - Challenges

Along the way there were always going to be some challenges experienced those that I managed to find a work around for are mentioned during the implementation section however those that were much larger and affected the scope of the project are listed below.

Optimisation of models

Models are sensitive to features and parameters set below are some of the issues experienced through the use of the models that were listed above.

ANN/Hybrid Models

I Built my ANN model using Keras however the model did not seem to perform very well at all with any of the datasets and did not manage to get close to any predictions. Compared to the other models thus performed badly and so I decided that attention should be bought to the others.

Python does not offer many building blocks in terms of hybrid models. The framework for these models would have to be hard coded and not use any of Sklearns available libraries for ML algorithms. Within the timeline of the project laid out it would be hard to implement into building a hybrid model without taking away from the others, it was an

either-or situation and I thought that a wider range of models would be more appealing to studies.

Datasets

Access to datasets is restricted, and often are private; because of this access to alternatives are hard to come by. I managed to find one instance of all datasets that I had stated earlier apart from the Desharnais dataset. I was luckily enough to find a csv version which had the full number of features compared to its arff version.

The ISBSG dataset contained many projects, but a lot of these entries had null values for certain features. To get the most out of this dataset, data would need to be cleaned first before deciding on features that contained no null values and this would result in the most influential features being missed out. The next solution for this would be to remove entries that contain null values, and this would leave the dataset smaller than what I wanted. Along with containing discrete values I decided it would be more work to clean the dataset and so I decided I would not be using it.

Arff Formatting

Upon loading my Kitchenham dataset I experienced a problem of data not being loaded.

This could be due to file corruption or the format being slightly different to what the arff loader can handle. Regardless, this prevented me from being able to load the Kitchenham dataset, and due to restrictions, I could not find another version of this dataset.

```

I have two versions of Desharnais, I've imported both to compare the difference in formats

In [2]: china = arff.loadarff('china.arff') # Loads arff files from directory into variable
desharnais = arff.loadarff('Desharnais.arff')
isbgs = arff.loadarff('ISBGS.arff')
kitchenham = arff.loadarff('kitchenham.arff')
maxwell = arff.loadarff('maxwell.arff')
usp05 = arff.loadarff('usp05.arff')

.....
NotImplementedError                                Traceback (most recent call last)
Cell In[2], line 7
      3 desharnais = arff.loadarff('Desharnais.arff')
      5 isbgs = arff.loadarff('ISBGS.arff')
----> 7 kitchenham = arff.loadarff('kitchenham.arff')
      9 maxwell = arff.loadarff('maxwell.arff')
     11 usp05 = arff.loadarff('usp05.arff')

File ~\anaconda3\envs\ml\lib\site-packages\scipy\io\arff\_arffread.py:802, in loadarff(f)
     800     ofile = open(f, 'rt')
     801     try:
--> 802         return _loadarff(ofile)
     803     finally:
     804         if ofile is not f: # only close what we opened

File ~\anaconda3\envs\ml\lib\site-packages\scipy\io\arff\_arffread.py:835, in _loadarff(ofile)
     824 # XXX The following code is not great
     825 # Build the type descriptor descr and the list of converters to convert
     826 # each attribute to the suitable type (which should match the one in
     (... )
     829 # This can be used once we want to support integer as integer values and
     830 # not as numeric anymore (using masked arrays ?).
     831 if hasstr:
     832     # How to support string efficiently ? Ideally, we should know the max
     833     # size of the string before allocating the numpy array.
--> 835     raise NotImplementedError("String attributes not supported yet, sorry")
     837 n1 = len(attr)
     838 def generator(row_iter, delim=','):
     839     # TODO: this is where we are spending time (~80%). I think things
     840     # could be made more efficiently:
     (... )
     852     # Note, I have already tried zipping the converters and
     853     # row elements and got slightly worse performance.

NotImplementedError: String attributes not supported yet, sorry

```

Figure of Kitchenham error [5]

The arff loader has a preference to loading numerical values as loading maxwell, Desharnais and China datasets data was loaded in perfectly. Upon loading the remaining datasets (USP05, ISBSG and Finnish datasets) data was loaded in with leading 'b' values (due to byte conversion) and as these datasets contained discrete data they would need to be cleaned twice, removing the leading letter, and then hot encoding all values. I read solutions online which stated that data would have to be thoroughly cleaned and with the combined numbers of USP05, Finnish and ISGSB it would be near over 1,000 projects that need to be manually cleaned because certain inputs such as inline comments after 'meaningful' text like a feature name (Rajan, NobilisRex,2020)

- . This would be very time consuming and would take away from developing the models. So, I decided that what would be best is if I took the 3 datasets that loaded in and cleaned them.

Anaconda

ScottknottESD is an R package which is provided by CRAN. This specific package is not widely used and because of this it is not on anaconda's R package library, and this meant I would have to download it manually. ScottKnottESD is essential for final part of my code which is the comparisons of models with their respective performance metric.

Upon my attempts I tried to install this program through the anaconda prompt by first downloading the package to my 'R' folder within my environment and then I formed the skeleton of the package and from there I would build the package which allow me to use the package within my environment on Jupyter notebook. However, this was not the case, and I experienced my errors, due to the use of the more widely used R packages being available through the repository there is not much research online which could point me in the right direction to correct the errors I experienced.

This would result in me programming all my code within my Anaconda environment (via Jupyter notebook) and then have to transfer this to Google collab as this can support the installation of the package.

4.5 - Risks

As spoken before at the start of this report, risks play a big part in project success and if the project will run according to the initial timeline, because of this I thought it would be an obvious to include and highlight the potential risks of the project at hand.

Risks below will be marked by RAG (Red, Amber, and Green) status which is a combination of consequences and likelihood.

Machine Learning models susceptible to overfitting, this occurs when models become highly accurate when predicting results but are not able to predict unforeseen data e.g., can predict all train data accurately but fail to predict test to high margin. The workaround for this would be to incorporate parameters to avoid models becoming accustomed to training data. This is unlikely and has a moderate consequence, so it scores a 6 on the scale.

Predictions based on old datasets will not provide a modern-day standard – The datasets used (Desharnais, Maxwell and China) are all over decade old and may not be seen as a representation of modern-day project scope. However, the advancements of SEE studies are still using these datasets for their models/reports as standards of recording

effort have not changed much thus these datasets can still be used as a fair representation. It is likely this is possible, but the consequences are minor so its scores a RAG rating of 8.

Computational strain will be high for running models – At the highest level of strain 5 models will be ran across 3 datasets resulting in 15 outputs and over 3000 project entries. This will require a lot of power. How I look to deal with this is to individually tune algorithms in modules so that only a max of 3 datasets of 1 model will be ran and these will then have hyper tuned parameters which run the ideal set up, this will result in a bottleneck of finding models with better accuracy but the trade of it worth it for the efficiency and flexibility it offers to system using it. This is likely and the consequences are moderate, so the RAG score is 12.

		Consequence				
		Negligible 1	Minor 2	Moderate 3	Major 4	Catastrophic 5
Likelihood	5 Almost certain	Moderate 5	High 10	Extreme 15	Extreme 20	Extreme 25
	4 Likely	Moderate 4	High 8	High 12	Extreme 16	Extreme 20
	3 Possible	Low 3	Moderate 6	High 9	High 12	Extreme 15
	2 Unlikely	Low 2	Moderate 4	Moderate 6	High 8	High 10
	1 Rare	Low 1	Low 2	Low 3	Moderate 4	Moderate 5

Figure of a standard risk matrix (Kaya,2018) [6]

5 - Implementation

The aim of the project is to produce a model(s) that can before predictions for SEE on datasets provided and this section of the project outlines the practical implementation of what is discussed above. All aspects of what went into the model will be discussed in detail.

5.1 - Libraries

Library importation was done in the first module as this would prevent me from duplicating imports later in other modules and this also helped to keep code clean.

Libraries imported

The libraries imported below are to load data and to manipulate the data. using the manipulated data I will then plot this against a graph

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import timeit
import math
import re

from scipy.io import arff
from scipy.stats import pearsonr
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier, RandomForestRegressor
from sklearn.feature_selection import SelectBest, f_classif
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.svm import SVM
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score, confusion_matrix, mean_squared_error, mean_absolute_error, r2_score, make_scorer
from sklearn import tree, datasets
```

Figure of libraries used [7]

5.2 - Loading datasets

Using the arff loader I loaded in datasets to their respect variables and then assigned them to a panda data frame defining what the head of the data frame would be.

For the single csv file (Desharnais) the data was able to be directly imported to a data frame from pandas csv reader.

Loading Datasets

below I've loaded all the datasets from their corresponding files. for 'arff' type files I've had to load those twice. The first time, through an arff loader. The second time through pandas

I have two versions of Desharnais, I've imported both to compare the difference in formats

```
In [2]: china = arff.loadarff('china.arff') # Loads arff files from directory into variable
maxwell = arff.loadarff('maxwell.arff')

In [3]: chinaadf = pd.DataFrame(china[0]) # converts data 2-Dimensional
chinaadf.drop(columns=['Dev.Type', 'ID'], inplace=True) # null column

desharnaisdf = pd.read_csv('02.desharnais.csv', header=0) # Loads csv 2-Dimensional
desharnaisdf.drop(columns=['id', 'Project'], inplace=True)

maxwelledf = pd.DataFrame(maxwell[0])
```

Figure of datasets being imported, and headers being dropped [8]

Data was described to see exactly what was going on but also to see the memory allocation required for each dataset fig.9.


```

In [12]: chinadf.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 499 entries, 0 to 498
Data columns (total 19 columns):
 #   Column      Non-Null Count  Dtype  
---  -
 0   ID          499 non-null    float64
 1   AFP         499 non-null    float64
 2   Input       499 non-null    float64
 3   Output      499 non-null    float64
 4   Enquiry     499 non-null    float64
 5   File        499 non-null    float64
 6   Interface   499 non-null    float64
 7   Added       499 non-null    float64
 8   Changed     499 non-null    float64
 9   Deleted     499 non-null    float64
10   PDR_AFP     499 non-null    float64
11   PDR_UFP     499 non-null    float64
12   NPDR_AFP    499 non-null    float64
13   NPDU_UFP    499 non-null    float64
14   Resource    499 non-null    float64
15   Dev.Type    499 non-null    float64
16   Duration    499 non-null    float64
17   N_effort    499 non-null    float64
18   Effort      499 non-null    float64
dtypes: float64(19)
memory usage: 74.2 KB

```

Figure of China dataset [9]

5.3 - Cleaning Data

Data imported could contain null values(fig.10) which when ran through the models would be fatal either causing models to crash or would provide false data towards optimisers, so these needed to be removed. I only wanted data that would be used so I also dropped columns that would not be needed, specifically columns that contained any form of ID.

```

Cleaning Data
The data needs to be cleaned to remove null values as well as down sizing the features as they aren't needed.

In [4]: deshannaisdf.dropna() # removing rows which contain null values
chinadf.dropna()
maoie11df.dropna()

Out[4]:

```

	Syear	App	Har	Oba	Ifc	Source	Telouse	Nlan	T01	T02	...	T10	T11	T12	T13	T14	T15	Duration	Size	Time	Effort
0	92.0	2.0	2.0	1.0	2.0	2.0	0.0	3.0	4.0	3.0	...	5.0	4.0	4.0	4.0	4.0	5.0	18.0	647.0	8.0	7871.0
1	93.0	2.0	2.0	1.0	2.0	2.0	0.0	3.0	2.0	3.0	...	3.0	4.0	4.0	4.0	4.0	4.0	5.0	130.0	9.0	845.0
2	90.0	1.0	2.0	1.0	2.0	2.0	0.0	2.0	3.0	3.0	...	5.0	4.0	3.0	2.0	3.0	3.0	8.0	264.0	6.0	2330.0
3	88.0	3.0	2.0	1.0	2.0	2.0	0.0	3.0	2.0	2.0	...	4.0	5.0	4.0	3.0	2.0	3.0	16.0	1056.0	2.0	21272.0
4	88.0	2.0	2.0	1.0	2.0	2.0	0.0	2.0	3.0	3.0	...	4.0	3.0	4.0	5.0	4.0	4.0	12.0	383.0	4.0	4224.0
...
57	91.0	5.0	5.0	1.0	2.0	1.0	0.0	3.0	4.0	2.0	...	3.0	3.0	4.0	3.0	4.0	4.0	20.0	495.0	7.0	7105.0
58	90.0	3.0	3.0	1.0	2.0	2.0	1.0	4.0	2.0	3.0	...	3.0	4.0	3.0	2.0	4.0	3.0	15.0	822.0	6.0	6616.0
59	92.0	1.0	2.0	1.0	2.0	2.0	1.0	2.0	3.0	3.0	...	5.0	5.0	4.0	3.0	2.0	3.0	12.0	204.0	8.0	4620.0
60	90.0	3.0	3.0	1.0	2.0	2.0	1.0	4.0	2.0	3.0	...	4.0	5.0	5.0	1.0	5.0	4.0	15.0	615.0	6.0	7451.0
61	91.0	3.0	3.0	1.0	2.0	2.0	0.0	3.0	2.0	4.0	...	5.0	5.0	4.0	4.0	5.0	4.0	33.0	3843.0	7.0	36479.0

62 rows x 27 columns

Figure of Removing null values [10]

5.4 - Exploratory Data Analysis (EDA)

'Exploratory Data Analysis (EDA) is one of the techniques used for extracting vital features and trends used by machine learning and deep learning models in Data Science' (Madhugiri,2023)

The steps outlined in Madhugiri's (2023) article will be followed throughout my feature selection to make sure feature selection is done correctly. The steps outlined in the article were:

- Data collection
- Finding all variables and understanding them
- Cleaning the dataset
- Identify correlated variables.
- Choosing the right statistical methods
- Visualising and analysing results

5.4.1 - Feature Selection

In order to prevent overfitting and to increase speeds of algorithms I decided that the next stage should be feature selection. The use of 2 or more methods of feature selection to show integrity for the results and to show robustness for the methods used. If both methods proved to choose the same features, it would be more reassuring of the choices made.

The number of features chosen was relative to the number of features available, it needed to be enough to allow data to be trained to a high degree but not too much that the feature selection would not provide any advantage. It was decided that at least 60% of actual features (feature that are not ID or null columns) should be used. The following numbers were set:

- China – 10/15 features selected.
- Desharnais – 7/10 features selected.
- Maxwell 16/26 feature selected.

Pearson Correlation

Pearson Correlation has a shorthand implementation of its method and allowed as straightforward way of implementation by running '.corr' across the datasets and stating that all values input would be numerical.

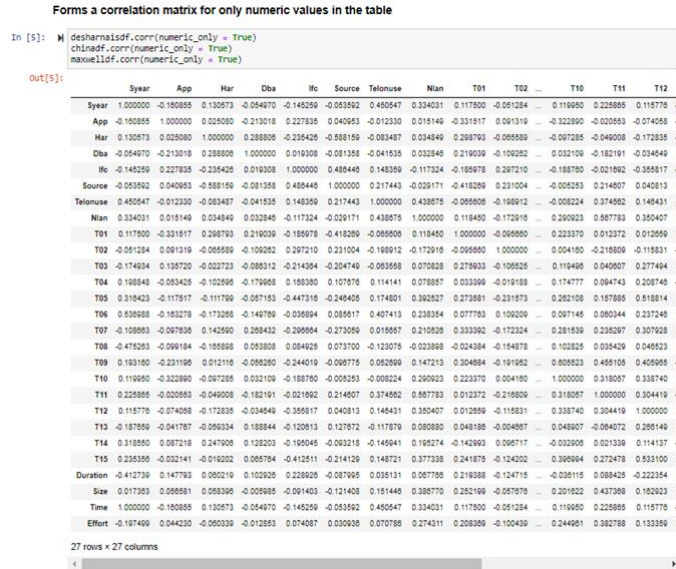


Figure of Pearson correlation [11]

I was able to visualise from the results from the feature selection which helps to summarise what exactly is happening, using the code in fig.12 I was able to plot a figure.

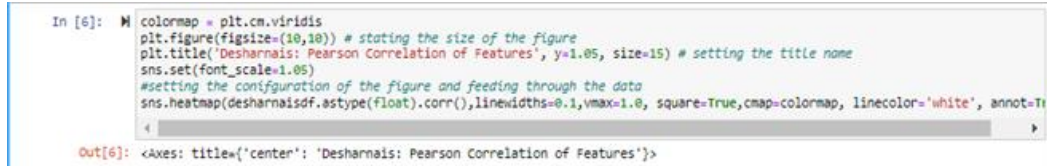


Figure of plotting Pearson Correlation [12]

From applying this code to each of the datasets I was able to visualise the results

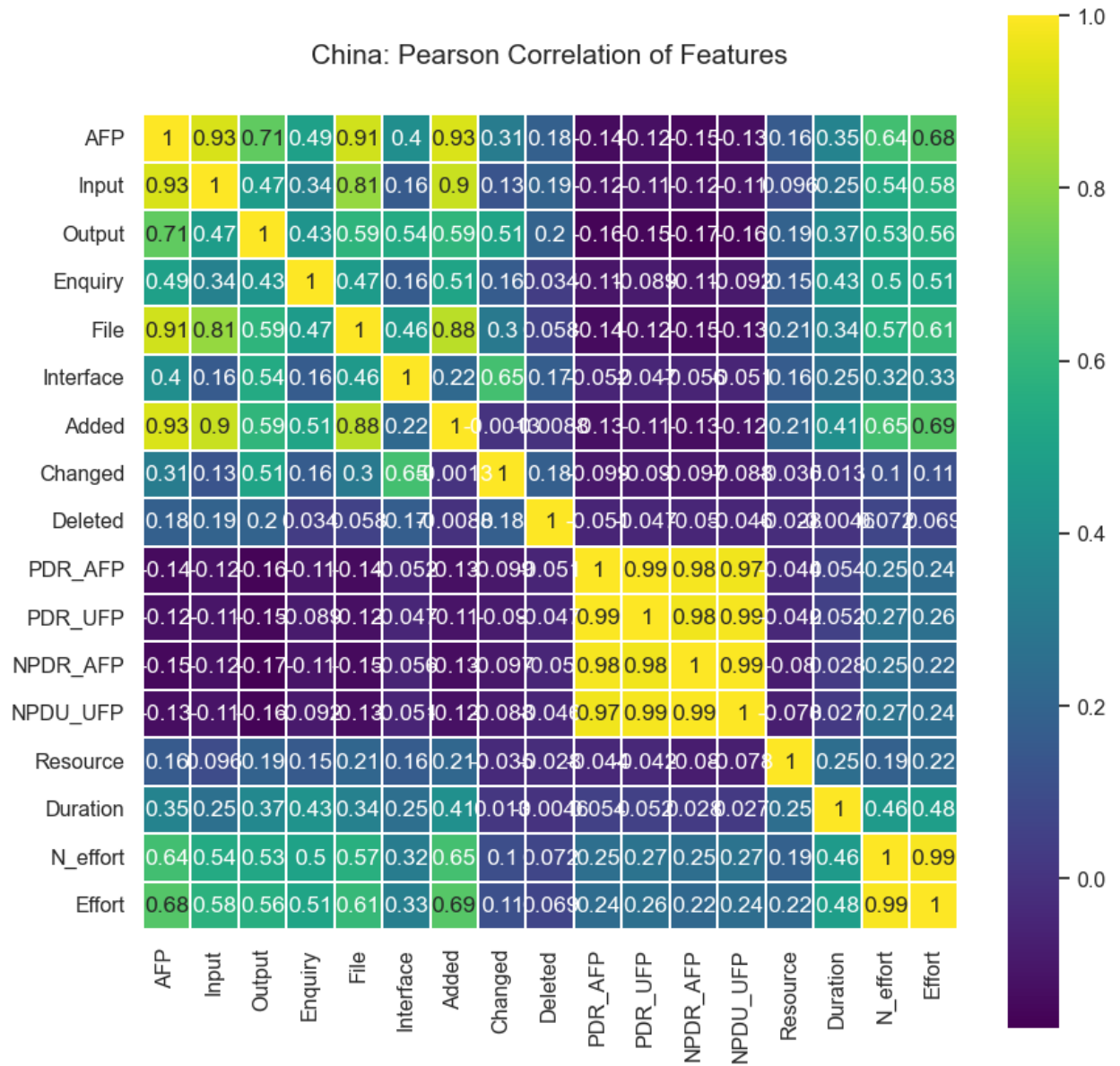


Figure of China's Pearson correlation [13]

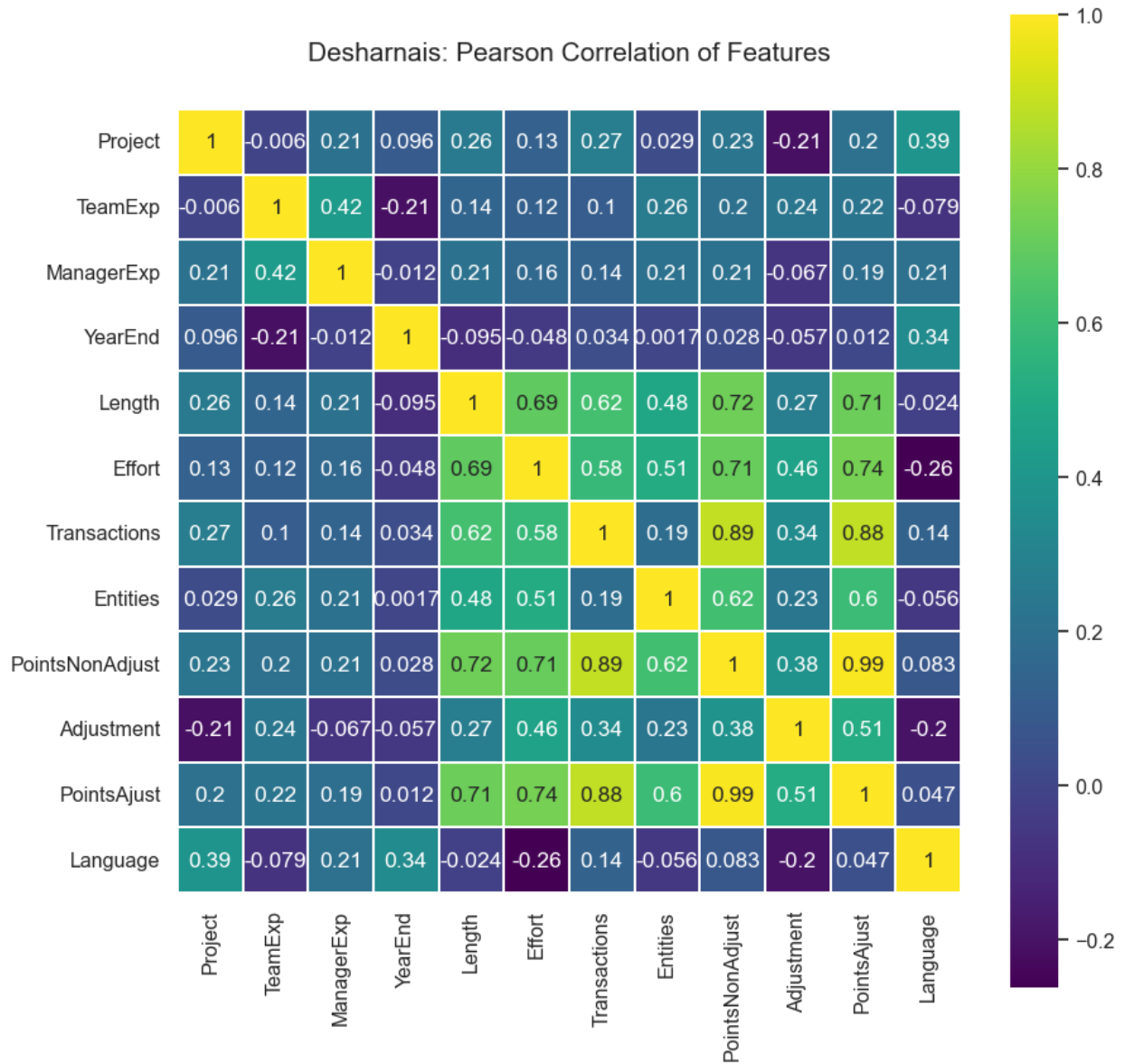


Figure of Desharnais' Pearson correlation [14]

Results from Pearson correlation suggest the following features:

- Length
- Transactions
- Entities
- PointsNonAdjust
- Adjustment
- PointsAjust

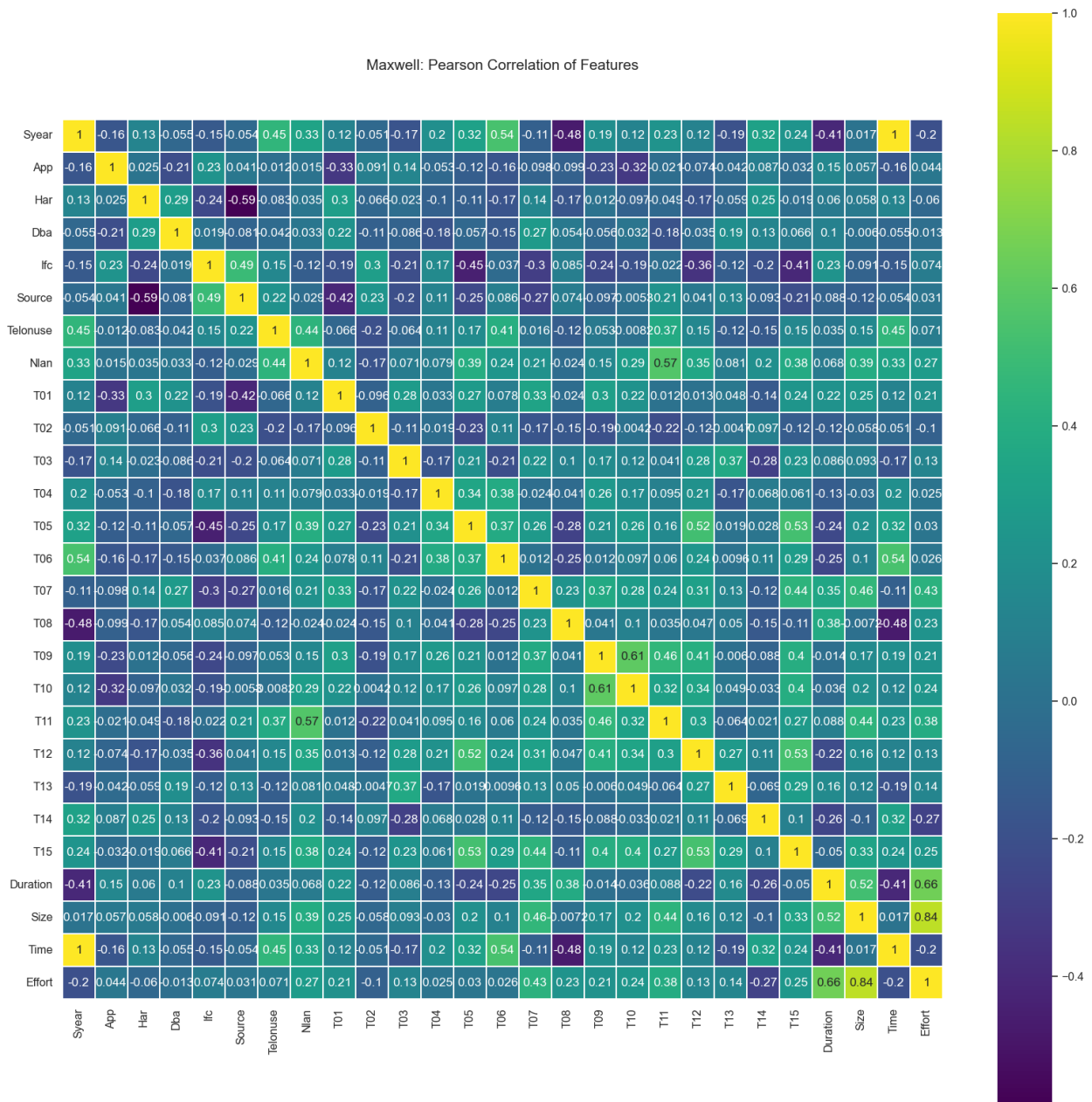


Figure of Maxwell's Pearson correlation [15]

A problem that was highlighted with these visualisations was that datasets with smaller feature sizes were easy to read but the maxwell diagram was quite complex and was not easy to read. To add to this the distinct features for selection were drowned out by the similar colours which showed how close the values were. This provided more emphasis on the reason for a second feature selection.

Extra Trees

Extra trees were used individually for each dataset, the process worked by loading the corresponding dataset, taking the independent feature columns, and assigning it to X and then taking the Effort column and assigning it to Y.

From there the data would be split using a test size of 0.33(how the data would be split proportionally to the test split), and a random state was used to keep consistency in the split.

The model would then be loaded with the classifier and once again random_state would be introduced. In this case the use of n_estimators to prevent the tree growing too big but also to make sure that the tree grew to a certain size. This was done to allow enough chance for features to be explored properly.

The model was run with training data and the most import features were selected by a defined number as shown in fig.16 (in this case it was the top 7 features) and ordered according to the highest priority.

Desharnais

```
In [9]: # Load the dataset
data = desharnaisdf
# Split the dataset into training and testing sets
X = data.iloc[:,desharnaisdf.columns != 'Effort'] # takes the values of all columns except effort
y = data.iloc[:, 4] # Location of the effort column
trainX, testX, trainY, testY = train_test_split(X, y, test_size=0.33, random_state=35)

model = ExtraTreesClassifier(n_estimators=100, random_state=32) # defines model and number of trees in the forest
model.fit(trainX, trainY) # fits the model to the training data
importances = model.feature_importances_ #extracts the most important features
topIndices = np.argsort(importances)[::-1][:7] # Gets the indices of the defined number of features
topFeatures = trainX.columns[topIndices] # assigns the columns with highest -> lowest influence

# Print the names of the top features
print(topFeatures)

Index(['Adjustment', 'Entities', 'Transactions', 'Length', 'Pointsajust',
       'Pointsnonadjust', 'YearEnd'],
      dtype='object')
```

Figure of Desharnais dataset being used for extra trees feature selection [16]

The model produced the same results as Pearson's correlation providing reassurance of the most influential features.

The features highlighted in each datasets results were then placed into X and Y variables accordingly for use later.


```

In [39]: chinaFeatures = ['Output', 'N_effort', 'Added', 'NPDU_UFP', 'NPDR_APP', 'APP',
                        'Input', 'PDR_APP', 'PDR_UFP', 'Enquiry']
chinaX = chinaFeatures
chinaY = chinaFeatures['Effort']

desharnaisFeatures = ['Entities', 'Adjustment', 'Transactions', 'PointsNonAdjust',
                    'PointsAdjust', 'Length']
desharnaisX = desharnaisFeatures
desharnaisY = desharnaisFeatures['Effort']

maxwellFeatures = ['Size', 'Duration', 'Syear', 'Time', 'Nlan', 'T08', 'App',
                  'T14', 'T11', 'T13', 'T07', 'T03', 'T10', 'T01', 'T06']
maxwellX = maxwellFeatures
maxwellY = maxwellFeatures['Effort']

```

Figure of Features being selected for each dataset [17]

5.5 - Models

All models were required to produce the performance metrics and so for each model the time, name, dataset used, Pred (25) value, MAE, MSE, RMSE and R^2 were returned so users could see how each model performed.

Apart from this all models differed in how they were created with the biggest part of this being hyper tuning.

5.5.1 - Hyper tuning

For every model and each dataset parameters were found with the help of Gridsearchcv and brute force. This was done by first setting very distanced intervals for parameters which took numerical input (e.g., max_depth) and then finding the optimal of that and then finding the local optimal around the previous optimal value (e.g., max_depth [20,30,40] with 30 being the optimal and then testing max_depth with [28,29,30,31,32] to find the local optimal)

For parameters which did not take numerical values I would use all the parameters available (e.g., 'metric': ['Euclidean', 'Manhattan', 'Minkowski']) to find the optimal between these.

Once the optimal parameters were set for each model, they were compiled into one module so that each one could be ran depending on what the user requested.

SVM

My SVM model took to the longest to find the optimal parameters for each dataset, specifically China. The time grew exponentially with increasing the range of C compared to the other parameters set and so I had to hyper tune this in more sections so that code did not take too long. As shown in fig.18,19 the difference of increasing the range of C from 1-5 to 1-7 Added an extra 18.5 minutes in compilation which is a significant

increase. The reason this took longer than other models was due to the optimal trade off of C.

The optimums for the other parameters were found quite quickly and meant that when using Gridsearchcv run times could be lowered by making sure that these were set to the one optimal value that was found.

```
In [29]: trainX, testX, trainY, testY = train_test_split(chinaX, chinaY, test_size=0.33, random_state=32)

parameters = {'kernel':('linear', 'rbf'), 'C':[1,2,3,4,5], 'gamma':('auto', 'scale')}

svr = SVR()
st = timeit.default_timer()
LinearSVC = GridSearchCV(svr, parameters, cv=2)
LinearSVC.fit(trainX, trainY)
elapsed = timeit.default_timer() - st
mse = mean_squared_error(testY, pred)
rmse = mse ** 0.5
r2 = r2_score(testY, pred)

print(f"Mean squared error: {mse:.2f}")
print(f"Root mean squared error: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
print("Elapsed time:", elapsed)
print("Best params hash: {}".format(LinearSVC.best_params_))
print(LinearSVC.score(testX, testY))

Mean squared error: 1752627.84
Root mean squared error: 1323.87
R-squared: 0.95
Elapsed time: 1014.4764262000003
Best params hash: {'C': 3, 'gamma': 'auto', 'kernel': 'linear'}
0.9511057619007525
```

Figure of SVM for China dataset with C = 1-5 [18]

```
In [30]: trainX, testX, trainY, testY = train_test_split(chinaX, chinaY, test_size=0.33, random_state=32)

parameters = {'kernel':('linear', 'rbf'), 'C':[1,2,3,4,5,6,7], 'gamma':('auto', 'scale')}

svr = SVR()
st = timeit.default_timer()
LinearSVC = GridSearchCV(svr, parameters, cv=2)
LinearSVC.fit(trainX, trainY)
elapsed = timeit.default_timer() - st
mse = mean_squared_error(testY, pred)
rmse = mse ** 0.5
r2 = r2_score(testY, pred)

print(f"Mean squared error: {mse:.2f}")
print(f"Root mean squared error: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
print("Elapsed time:", elapsed)
print("Best params hash: {}".format(LinearSVC.best_params_))
print(LinearSVC.score(testX, testY))

Mean squared error: 1752627.84
Root mean squared error: 1323.87
R-squared: 0.95
Elapsed time: 2133.8080007999997
Best params hash: {'C': 3, 'gamma': 'auto', 'kernel': 'linear'}
0.9511057619007525
```

Figure of SVM for China dataset with C = 1-7 [19]

DT

As for the decision Tree the same rigorous hyper parameter testing took place.

The option to visualise what was happening in terms of depth provided more guidance for when tweaking the model for each dataset as in shown in fig.22,23 the difference can be huge (please note the redder the result the least likely this path would be considered).

```
def dt(dataset):
    if dataset == "Maxwell":
        trainX, testX, trainY, testY = train_test_split(maxwellX, maxwellY, test_size=0.32, random_state=32)
        parameters = {'max_depth': [2], 'min_samples_leaf': [15]}

    elif dataset == "China":
        trainX, testX, trainY, testY = train_test_split(chinax, chinay, test_size=0.32, random_state=32)
        parameters = {'max_depth': [10], 'min_samples_leaf': [15]}

    elif dataset == "Desharnais":
        trainX, testX, trainY, testY = train_test_split(desharnaisX, desharnaisY, test_size=0.42, random_state=32)
        parameters = {'max_depth': [2], 'min_samples_leaf': [15]}
```

Figure of DT parameters [20]

Tree Figure

```
In [43]: fig = plt.figure(figsize=(25,20))
         _ = tree.plot_tree(chinadtr,
                           feature_names=None,
                           filled=True)
```

Figure of China's tree being plotted. [21]

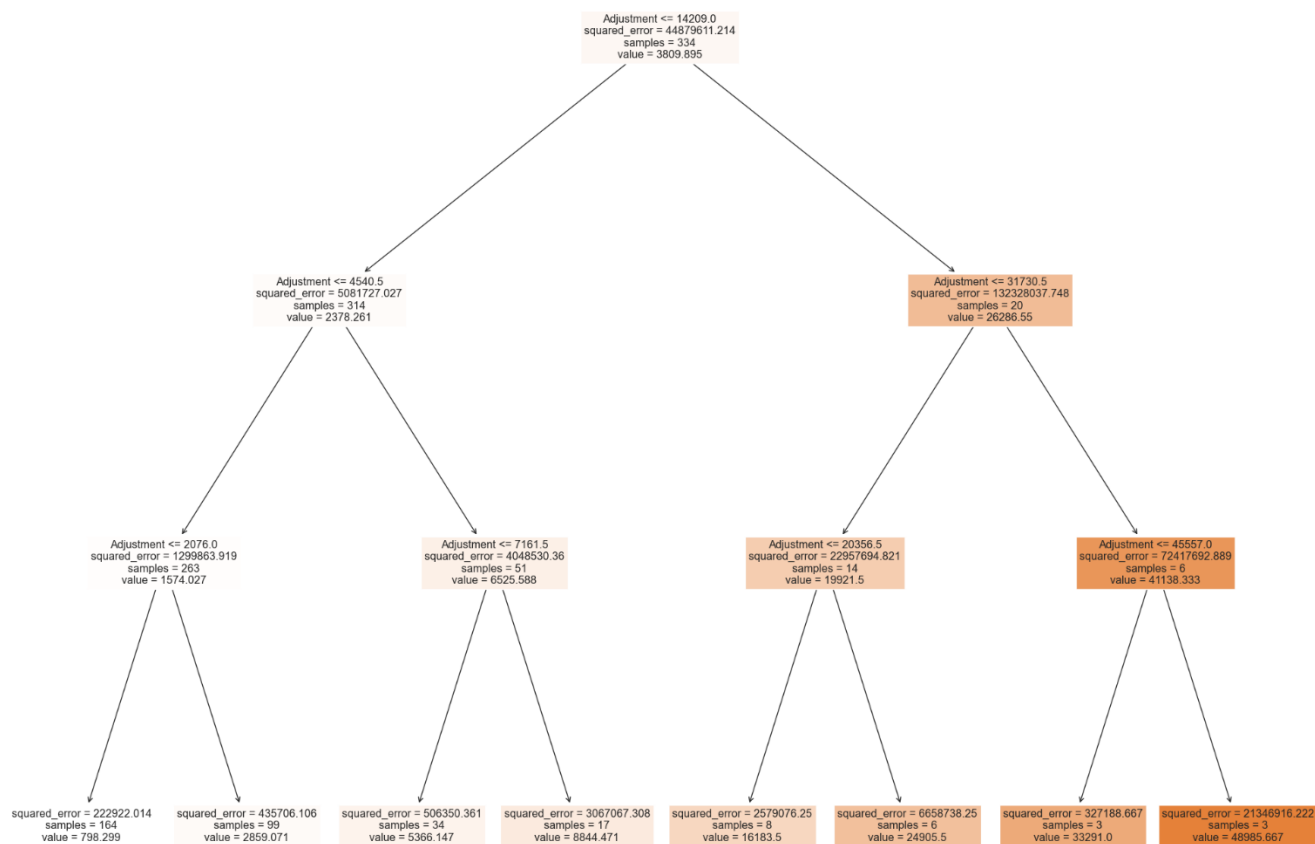


Figure of Chinas max_depth being set to 3 [22]

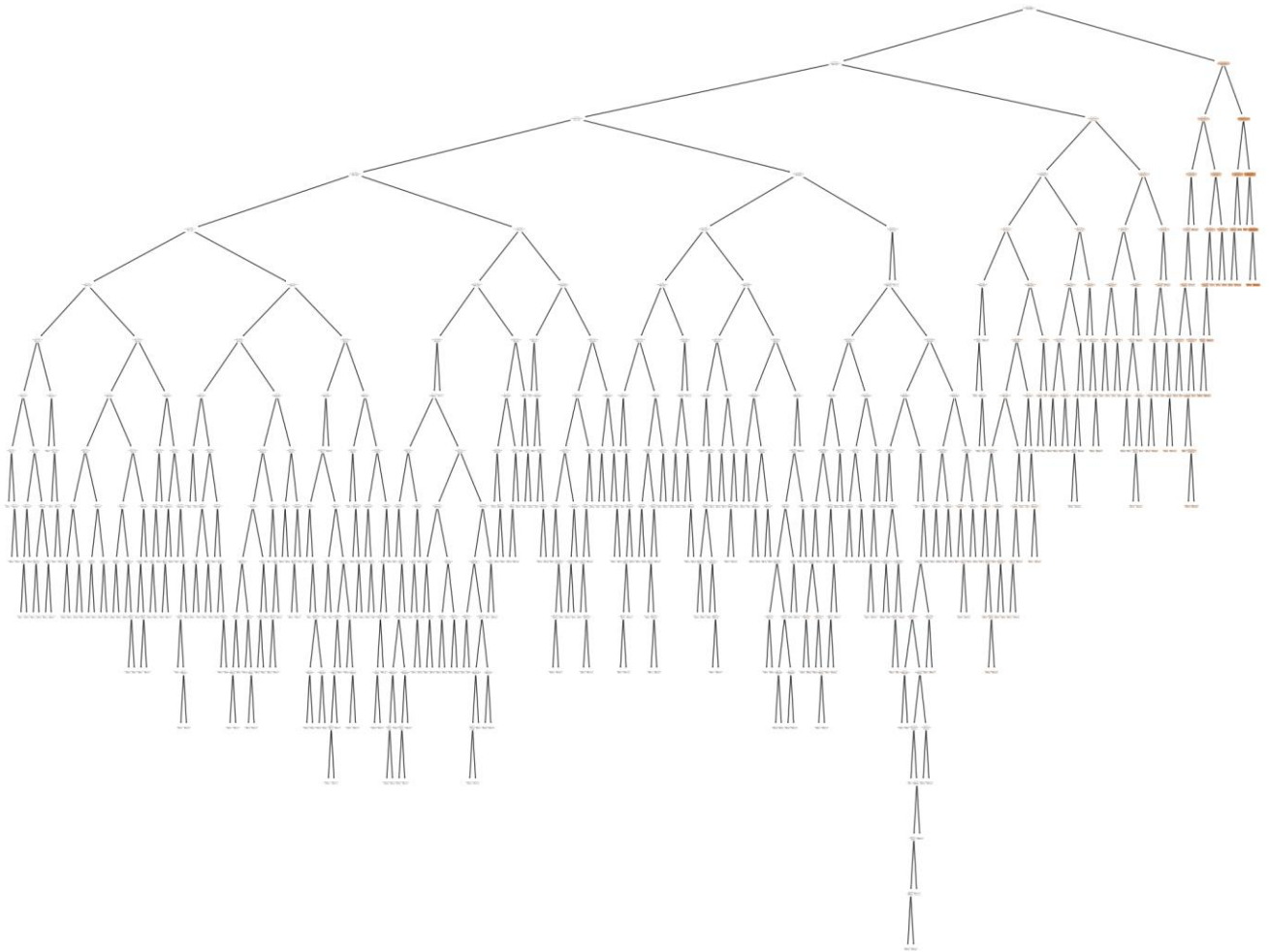


Figure of Chinas DT with no `max_depth` being set [23]

KNN

The parameter that had the most effect within this model was the `n_neighbors` parameter and this required the most testing followed by 'metric'.

```
def knn(dataset):
    if dataset == "Maxwell":
        trainX, testX, trainY, testY = train_test_split(maxwellX, maxwellY, test_size=0.32, random_state=20)
        parameters = {'n_neighbors': [2,3],
                      'weights': ['uniform', 'distance'],
                      'metric': ['euclidean']}

    elif dataset == "China":
        trainX, testX, trainY, testY = train_test_split(chinax, chinaY, test_size=0.32, random_state=32)
        parameters = {'n_neighbors': [4],
                      'weights': ['uniform'],
                      'metric': ['euclidean']}

    elif dataset == "Desharnais":
        trainX, testX, trainY, testY = train_test_split(desharnaisX, desharnaisY, test_size=0.42, random_state=32)
        parameters = {'n_neighbors': [2,3],
                      'weights': ['uniform', 'distance'],
                      'metric': ['euclidean', 'manhattan', 'minkowski']}
```

Figure of KNN parameters [24]

Random Forest

RF had a similar implementation to the DT. What differed however was that `n_estimators` required a lot of tweaking once the local optimal was found for the `max_depth`, with each model having around a difference of 300 between them.

Specifically for this model the Desharnais Dataset needed a higher `test_size` value to improve accuracy. `test_size` was set to 0.40 instead of the usual 0.32/33

```
def rf(dataset):
    if dataset == 'China':
        trainX, testX, trainY, testY = train_test_split(desharnaisX, desharnaisY, test_size=0.33, random_state=60)
        parameters = {'max_depth':[16], 'n_estimators':[445], 'min_samples_split':[2]}
    if dataset == 'Maxwell':
        parameters = {'max_depth':[12], 'n_estimators':[725], 'min_samples_split':[2]}
        trainX, testX, trainY, testY = train_test_split(maxwellX, maxwellY, test_size=0.33, random_state=30)
    if dataset == 'Desharnais':
        parameters = {'max_depth':[10], 'n_estimators':[112], 'min_samples_split':[2]}
        trainX, testX, trainY, testY = train_test_split(desharnaisX, desharnaisY, test_size=0.40, random_state=40)
```

Figure of RF parameters for each dataset [25]

Linear Regression

Linear Regression was the most straightforward to hyper tune as it has the least number of parameters and complexity of the algorithm compared to others is not as high.

It was quite surprising to see that both the Maxwell and Desharnais dataset gave better results when models did not fit a line intercept.

```
def lr(dataset):
    if dataset == 'China':
        trainX, testX, trainY, testY = train_test_split(chinaX, chinaY, test_size=0.33, random_state=32)
        parameters = {'fit_intercept':[True], 'n_jobs':[1]}
    if dataset == 'Maxwell':
        trainX, testX, trainY, testY = train_test_split(maxwellX, maxwellY, test_size=0.33, random_state=32)
        parameters = {'fit_intercept':[False], 'n_jobs':[1]}
    if dataset == 'Desharnais':
        trainX, testX, trainY, testY = train_test_split(desharnaisX, desharnaisY, test_size=0.33, random_state=32)
        parameters = {'fit_intercept':[False], 'n_jobs':[1]}
```

Figure of LR parameters for each dataset [26]

5.5.2 -Default

The default models used default parameters and had none set for them, they also used all the features available to the datasets.

5.6 - Google Colaboratory

Google Colaboratory is an online platform similar to Jupyter notebook but is hosted by google and provides access to a cloud computer which is able to run Python and R. The switch to Google Colaboratory was unforeseen but was needed to put the final additions on the statistical testing for my program.

Google provides a baseline system which have the following specs:

- CPU: Intel(R) Xeon(R) CPU @ 2.20GHz
- RAM: 12.7 GB

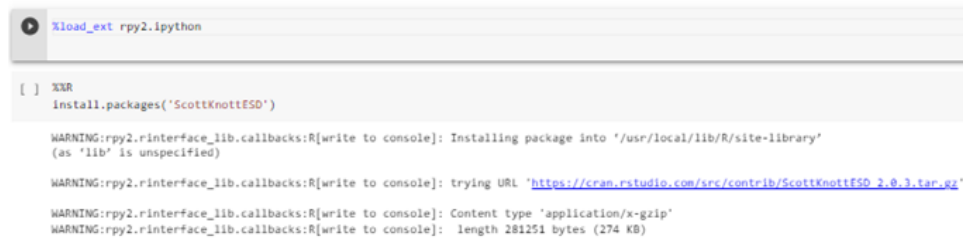
The stats taken from Jupyter Notebook were during testing and so is relevant to that environment. The comparisons in Google Colaboratory will be ran in that environment and so the results plotted will all be relevant to that environment making testing fair.

5.7 - ScottKnottESD

The ScottKnottESD package had to be installed and executed in Google Colaboratory as mentioned in section (Plan: Adjustments) I was unable to create an environment it was compatible with.

The following steps were taken to installing the R package:

- Load RPY2 (fig.27)
- Install ScottKnottESD(fig.27)



```
Xload_ext rpy2.Ipython

[ ] %R
install.packages('ScottKnottESD')

WARNING:rpy2.rinterface.lib.callbacks:R[write to console]: Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

WARNING:rpy2.rinterface.lib.callbacks:R[write to console]: trying URL 'https://cran.rstudio.com/src/contrib/ScottKnottESD_2.0.3.tar.gz'

WARNING:rpy2.rinterface.lib.callbacks:R[write to console]: Content type 'application/x-gzip'

WARNING:rpy2.rinterface.lib.callbacks:R[write to console]: length 281251 bytes (274 KB)
```

Figure of ScottKnottESD installation [27]

The next step was to use the table the package provides to input my data from the models depending on what metric was being used (fig.28)

```
if 2 in metrics:
    data = pd.DataFrame(
        {
            "SVM": [(SVM('Maxwell'))[1]], (SVM('China'))[1]], (SVM('Desharnais'))[1]]],
            "DT": [(dt('Maxwell'))[1]], (dt('China'))[1]], (dt('Desharnais'))[1]]],
            "RF": [(rf('Maxwell'))[1]], (rf('China'))[1]], (rf('Desharnais'))[1]]],
            "LR": [(lr('Maxwell'))[1]], (lr('China'))[1]], (lr('Desharnais'))[1]]],
            "KNN": [(knn('Maxwell'))[1]], (knn('China'))[1]], (knn('Desharnais'))[1]]],
        }
    )
    display(data)
    r_sk = sk.sk_esd(data)
    column_order = list(r_sk[3] - 1)
    ranking = pd.DataFrame(
        {
            "Model": [data.columns[i] for i in column_order],
            "rank": r_sk[i].astype("int"),
        }
    ) # long format
    display(ranking)
```

Figure of ScottKnottESD test for RMSE [28]

This produced the results from the table and then ranks them accordingly (fig.29).



Figure of ScottKnottESD ranking for RMSE (*Note SVM is the best achieving as it scored the lowest, highest scoring will be 1st (DT)) [29]

5.8 - Command line (CMD)

User interaction with the system was to be done through the command prompt. For this the choices the user could make were:

- Hyper tuned/Baseline (or both)
- Models (SVM, DT, KNN, RF, LR)
- Datasets (China, Desharnais, Maxwell)
- ScottKnottESD

The user has the choice to run 1 version of a model with one dataset to running both versions with all the models and datasets.

6 - Results

The goal of this project is to compare different implementations of AI for SEE and this was set against a set of objectives models should meet. Below are the results of each statistic that was measured. The most important statistic to focus on is R^2 followed by RMSE, MAE, and Pred(25), in the order given.

Models were run one after the other through the CMD interface for all the statistics together, these have been sorted and separated respectfully to what each tables statistic is. Once

the whole program was run, every statistic was run through the ScottKnottESD ranking which produced a ranking for each model. It should be noted that the ScottKnottESD ranks results by the highest value, which means statistics which indicate better performance for lower values (RMSE, MSE, MAE and Time) will rank the opposite way. This means the highest-ranking model scored in these is worse and the lowest ranking model scored is the best/better model. As for R^2 and Pred (25) these are ranked in the correct order due to the nature of the ScottKnottESD ranking.

The use of the versions with hyperparameters with all features and the baseline model are there only for testing and comparison for the main version which is the focus of this report; being the hyperparameter version with feature selection. The user is still able to use these versions however, they are not the better version as the results below will show.

6.1 - CMD Interface:

The role of the interface was to act as means for the user to interact with the system. This would be done through prompts for the user to input:

1. The version of the program the user wanted to use – (1) Hyperparameters with feature selection, (2) Hyperparameters without feature selection, (3) baseline model.
2. The AI model the user wanted to use – SVM, DT, RF, LR, KNN. All is an option for the user to run all the models and produce a ScottKnottESD ranking for their selected metric.
3. *If All is chosen, then metrics for running the ScottKnottESD are presented – R^2 , RMSE, MSE, MAE, Time, Pred (25)
4. The datasets the users want to run with these – China, Desharnais, Maxwell
5. The option to finish the program – 'Y', Enter key.

These options are interchangeable to the user and numerous combinations can be used to present different information. The user can then choose to finish the program by entering 'Y' or choose to run it again with a different input by pressing the enter key.

The results are printed to the CMD for the user to view.

6.2 - Tables of Results

Hyperparameters With Feature Selection						
Dataset:	SVM	DT	RF	LR	KNN	Avg. R ²
Maxwell	<u>0.761366</u>	0.300200	0.697760	0.568416	0.766743	0.618897
China	<u>0.952046</u>	<u>0.837992</u>	<u>0.948620</u>	<u>0.967067</u>	<u>0.967576</u>	<u>0.934660</u>
Desharnais	<u>0.587139</u>	0.026591	0.539506	0.562982	0.250387	0.393331
Avg. R ²	<u>0.766850</u>	0.388261	0.728629	0.699488	0.661569	
ScottKnottE SD Rank	<u>1</u>	3	1	2	2	
Hyperparameters With All Features						
Dataset:	SVM	DT	RF	LR	KNN	Avg. R ²
Maxwell	<u>0.765165</u>	0.305241	0.698366	-0.274820	0.766743	0.452139
China	<u>0.952692</u>	<u>0.844943</u>	<u>0.950386</u>	<u>0.959352</u>	<u>0.967168</u>	<u>0.934908</u>
Desharnais	<u>0.595455</u>	0.084633	0.546464	0.660802	0.249671	0.427405
Avg. R ²	<u>0.771104</u>	0.411605	0.638896	0.448444	0.661194	
ScottKnottE SD Rank	<u>1</u>	4	2	4	3	
Without Hyperparameters with All Features						
Dataset:	SVM	DT	RF	LR	KNN	Avg. R ²
Maxwell	-0.522449	0.381053	0.678738	<u>0.612502</u>	-0.342760	0.161417
China	<u>-0.133883</u>	<u>0.934547</u>	<u>0.950838</u>	<u>0.986071</u>	<u>0.967576</u>	<u>0.741030</u>
Desharnais	-0.080403	0.663346	0.287112	<u>0.667847</u>	0.250387	0.357658
Avg. R ²	-0.245578	0.659649	0.638896	<u>0.755473</u>	0.291734	
ScottKnottE SD Rank	4	2	2	<u>1</u>	3	

Table of R² for each model against its dataset [8]

Hyperparameters With Feature Selection						
Dataset:	SVM	DT	RF	LR	KNN	Avg. RMSE
Maxwell	<u>4093.375157</u>	6607.312492	5005.300611	5981.178212	5059.747517	5349.382798
China	<u>1326.598326</u>	<u>2573.176355</u>	<u>1429.804844</u>	<u>1086.501801</u>	<u>1090.840826</u>	<u>1501.384430</u>
Desharnais	<u>3103.308386</u>	4714.884726	3089.198134	3174.420555	3842.558545	3584.874069
Avg. RMSE	<u>2841.093956</u>	4631.791191	3174.767863	3414.033523	3330.715629	
ScottKnottESD Rank	<u>3</u>	1	2	2	2	

Hyperparameters With All Features						
Dataset:	SVM	DT	RF	LR	KNN	Avg. RMSE
Maxwell	4060.657205	7588.769969	5000.283893	10279.658817	5059.747517	6397.82348
China	1317.625834	2385.459679	1405.012389	1207.074747	1097.679107	1482.570351
Desharnais	3071.897649	4583.640569	3065.772248	2796.673626	3844.392639	3472.475346
Avg. RMSE	2816.726896	4852.623406	3157.022843	4761.13573	3333.939754	
ScottKnottESD Rank	3	1	2	1	2	
Without Hyperparameters with All Features						
Dataset:	SVM	DT	RF	LR	KNN	Avg. RMSE
Maxwell	8595.564600	7430.558689	8920.710703	9081.433713	4904.982791	7786.650099
China	6375.291924	2229.135817	1470.398212	744.465766	1090.840826	2382.026509
Desharnais	4613.121199	3173.693333	4021.300890	2767.476425	3842.558545	3683.630078
Avg. RMSE	6527.992574	4277.795946	4804.136602	4197.791968	3279.460721	
ScottKnottESD Rank	1	2	2	2	3	

Table RMSE for each model against its dataset [9]

Hyperparameters With Feature Selection						
Dataset:	SVM	DT	RF	LR	KNN	Avg. MSE
Maxwell	1.675572 X 10 ⁷	4.365658 X 10 ⁷	2.505303 X 10 ⁷	3.577449 X 10 ⁷	2.560104 X 10 ⁷	2.936817 X 10 ⁷
China	1.759863 X 10 ⁶	6.621237 X 10 ⁶	2.044342 X 10 ⁶	1.180486 X 10 ⁶	1.189934 X 10 ⁶	2.559172 X 10 ⁶
Desharnais	9.630523 X 10 ⁶	2.223014 X 10 ⁷	9.543145 X 10 ⁶	1.007695 X 10 ⁷	1.476526 X 10 ⁷	1.324920 X 10 ⁷
Avg. MSE	9.382035 X 10 ⁶	2.416932 X 10 ⁷	1.221351 X 10 ⁷	1.567731 X 10 ⁷	1.385208 X 10 ⁷	
ScottKnottESD Rank	4	1	3	2	2	
Hyperparameters With All Features						
Dataset:	SVM	DT	RF	LR	KNN	Avg. MSE
Maxwell	1.648894 X 10 ⁷	5.758943 X 10 ⁷	2.500284 X 10 ⁷	1.056714 X 10 ⁸	2.560104 X 10 ⁷	4.607073 X 10 ⁷
China	1.736138 X 10 ⁶	5.690418 X 10 ⁶	1.974060 X 10 ⁶	1.457029 X 10 ⁶	1.204899 X 10 ⁶	2.412509 X 10 ⁶
Desharnais	9.436555 X 10 ⁶	2.100976 X 10 ⁷	9.398959 X 10 ⁶	7.821383 X 10 ⁶	1.477935 X 10 ⁷	1.248920 X 10 ⁷
Avg. MSE	9.220544 X 10 ⁶	2.809654 X 10 ⁷	1.212529 X 10 ⁷	3.831660 X 10 ⁷	1.386176 X 10 ⁷	

ScottKnottESD Rank	<u>4</u>	2	3	1	3	
Without Hyperparameters with All Features						
Dataset:	SVM	DT	RF	LR	KNN	Avg. MSE
Maxwell	7.388373 X 10 ⁷	<u>3.465366 X 10⁷</u>	8.038503 X 10 ⁷	8.247244 X 10 ⁷	2.405886 X 10 ⁷	5.909074 X 10 ⁷
China	<u>4.064435 X 10⁷</u>	<u>2.160195 X 10⁶</u>	<u>2.074600 X 10⁶</u>	<u>5.542293 X 10⁵</u>	<u>1.189934 X 10⁶</u>	<u>9.324662 X 10⁶</u>
Desharnais	2.128089 X 10 ⁷	<u>6.492614 X 10⁶</u>	1.648205 X 10 ⁷	7.658926 X 10 ⁶	1.476526 X 10 ⁷	1.333595 X 10 ⁷
Avg. MSE	4.526966 X 10 ⁷	<u>1.443549 X 10⁷</u>	3.298056 X 10 ⁷	3.022853 X 10 ⁷	4.001405 X 10 ⁷	
ScottKnottESD Rank	1	<u>3</u>	2	2	3	

Table of MSE for each model against its dataset [10]

Hyperparameters With Feature Selection						
Dataset:	SVM	DT	RF	LR	KNN	Avg. MAE
Maxwell	<u>2781.682237</u>	4723.332963	3089.002408	4657.963739	3259.174603	3702.23119
China	<u>323.780918</u>	<u>771.544742</u>	<u>430.725994</u>	<u>325.295090</u>	<u>402.028125</u>	<u>450.674974</u>
Desharnais	<u>2414.812091</u>	3349.333333	2052.358020	2366.229380	2376.175704	2511.781706
Avg. MAE	<u>1840.091749</u>	2948.070346	1857.362141	2449.829403	2230.639452	
ScottKnottESD Rank	<u>3</u>	1	3	2	3	
Hyperparameters With All Features						
Dataset:	SVM	DT	RF	LR	KNN	Avg. MAE
Maxwell	<u>2733.033793</u>	4996.568594	3081.265688	7777.358118	3259.174603	4369.480159
China	<u>328.564268</u>	<u>849.503277</u>	<u>428.817747</u>	<u>378.459956</u>	<u>409.137500</u>	<u>478.896550</u>
Desharnais	<u>2394.968568</u>	3216.263690	2023.609686	2032.467731	2387.251746	2410.912284
Avg. MAE	<u>1818.855543</u>	3020.778520	1844.564374	3396.095268	2018.521283	
ScottKnottESD Rank	<u>2</u>	1	2	1	2	
Without Hyperparameters with All Features						
Dataset:	SVM	DT	RF	LR	KNN	Avg. MAE
Maxwell	5004.016803	4374.666667	3795.183810	6309.081255	<u>3297.495238</u>	4556.088755
China	<u>3311.605302</u>	<u>543.793939</u>	<u>439.261515</u>	<u>339.823637</u>	<u>402.028125</u>	<u>1007.302504</u>
Desharnais	2758.120753	2339.148148	2682.761111	2016.902395	<u>2376.175704</u>	2434.621622
Avg. MAE	3691.247619	2419.202918	2305.735479	2888.602429	<u>2025.233022</u>	
ScottKnottESD Rank	1	3	3	2	<u>4</u>	

Table of MAE for each model against its dataset [11]

Hyperparameters With Feature Selection						
Dataset:	SVM	DT	RF	LR	KNN	Avg. Time
Maxwell	0.415671	0.019357	7.485538	0.038051	0.071727	1.606069
China	209.914896	0.023688	5.938238	0.035494	0.023274	43.187118
Desharnais	0.047183	0.017804	1.157733	0.032959	0.197552	0.290646
Avg. Time	70.125917	0.020283	4.860503	0.035501	0.097518	
ScottKnottE SD Rank	1	4	2	3	3	
Hyperparameters With All Features						
Dataset:	SVM	DT	RF	LR	KNN	Avg. Time
Maxwell	0.176181	0.032083	6.443133	0.029809	0.061074	1.348456
China	356.506143	0.039937	8.033969	0.025617	0.021075	72.925348
Desharnais	0.120453	0.027682	0.936331	0.021478	0.141608	0.249510
Avg. Time	118.934259	0.033234	5.127811	0.025635	0.074586	
ScottKnottE SD Rank	1	4	2	5	3	
Without Hyperparameters with All Features						
Dataset:	SVM	DT	RF	LR	KNN	Avg. Time
Maxwell	0.002912	0.004228	0.138690	0.002772	0.1550	0.060720
China	0.018852	0.007430	0.356811	0.002524	0.017966	0.080717
Desharnais	0.003251	0.003692	0.137543	0.002404	0.144634	0.0583048
Avg. Time	0.008338	0.005117	0.211015	0.002567	0.105867	
ScottKnottE SD Rank	3	4	1	5	2	

Table of times for each model against its dataset (in seconds*). Please note a higher scoring time will mean the higher the ranking, the quickest model will be last. [12]

Hyperparameters With Feature Selection						
Dataset:	SVM	DT	RF	LR	KNN	Avg. Pred (25)
Maxwell	0.350000	0.150000	0.428571	0.238095	0.380952	0.3095236
China	0.968750	0.875000	0.969697	0.836364	0.943750	0.9187122
Desharnais	0.200000	0.185185	0.363636	0.259259	0.342857	0.437653
Avg. Pred (25)	0.506250	0.403395	0.584444	0.444573	0.555853	
ScottKnottE SD Rank	2	3	1	3	1	
Hyperparameters With All Features						

Dataset:	SVM	DT	RF	LR	KNN	Avg. Pred (25)
Maxwell	0.400000	0.190476	0.428571	0.142857	0.380952	0.308571
China	0.962500	0.856250	0.969697	0.751515	0.962500	0.900492
Desharnais	0.171429	0.200000	0.303030	0.407407	0.342857	0.284945
Avg. Pred (25)	0.511310	0.415575	0.567099	0.433926	0.562103	
ScottKnottE SD Rank	1	2	1	2	1	
Without Hyperparameters with All Features						
Dataset:	SVM	DT	RF	LR	KNN	Avg. Pred (25)
Maxwell	0.150000	0.238095	0.523810	0.047619	0.238095	0.239524
China	0.187879	0.915152	0.963636	0.757576	0.943750	0.753599
Desharnais	0.285714	0.444444	0.222222	0.370370	0.342857	0.333121
Avg. Pred (25)	0.207864	0.532564	0.569889	0.391855	0.508234	
ScottKnottE SD Rank	3	1	1	2	1	

Table of Pred (25) for each model against its dataset [13]

6.2.1 - Feature Selection vs. Datasets

The use of features selection arose to prevent overfitting by choosing only the most influential features as well as increasing the speed of models by reducing data input resulting in less computational strain. The downside of feature selection is that accuracy of models can drop due to having less data available. To ensure that this did not alter the accuracy of models too much, a comparison was run on the model with hyperparameters and feature selection against a model with hyperparameters all features. These models were given all the same parameters to make sure that the only difference was the feature set.

The conclusion of this comparison would solidify the use of the model with hyperparameters with feature selection, if for all models did not decrease more than 8% for their R^2 values.

The results shown in tables 8-13 showed that time taken for models to run were heavily impacted. One of the models that should be highlighted is the SVM. The composition of the SVM model means that for larger datasets such as the China dataset time taken to run the model can take at least 70% longer than the model with feature selection. This means reducing the data input for optimal parameters is key and this showed in the results with the hyperparameter version of the SVM with all features for China increased

in time from 3.50 minutes to 5.94 minutes which is an exponential difference and would not appeal to someone if they were to use this version with datasets of comparable size or larger. The assumption between using feature selection with hyperparameters and using all the features with hyperparameters is that the models with all features would exceed in terms of R^2 values and looking at the results this is only the case for two models being the SVM and DT with RF, LR and KNN providing better R^2 . The models which were outperformed by their all features counterpart had an increased R^2 value of 0.55% for the SVM model and 6% increase for the DT model, the increase was below the threshold defined at the start of the comparison therefore justifying the trade-off.

6.2.2 - Hyperparameters vs. Datasets

As stated, before in the Implementation section 5 models (SVM, DT, KNN, RF, LR) would undergo training and testing for 3 datasets (China, Desharnais, Maxwell).

Using Gridsearchcv models were provided with the datasets which had a range of different parameters for the optimisations. Through brute force local optimum for each model was found, producing better results than the start parameters given and to show this, models without parameters were compared to their counter parts as shown in tables 8-13

To confirm the integrity of results the local optimum hyperparameters were compared to each model's counterpart without the hyperparameters set and feature selection. The default models (no parameters and no feature selection) were run on all datasets acting as a baseline for metrics across the board (R^2 , RMSE, MSE, MAE, Time, Pred (25)). The results from this provided support for going down the route of hyperparameters as the only result that the default models outperformed the hyperparameters was time taken to run the models.

Results shown in tables 8-13 showed that models with default parameters did not provide the best metric results for the datasets. The trade off with speed whilst is a rather large increase, specifically the RF model had a 2203.39% increase, it is a jump from mere milliseconds to about 4.65 seconds which is bearable, with the biggest gap being 1 minute 10 seconds for the SVM model. To further add proof as to why hyperparameters with feature selection was the best choice is because the baseline model was returning negative R^2 values specifically for the SVM model, which meant the models fit worse than regression line resulting in the mean being better than using the regression line

(Fairly Nerdy, 2017). This would void the model for any predictions however, once hyperparameters were set this model performed the best out of them, with a 412.26% increase for R^2 value, which makes it a fair trade for accuracy over speed. To also help with the speed trade off the DT and RF model were limited in depth so that they did not run for longer than they needed too as shown in fig. 22, 23.

Pred (25) highlighted how accurate models were for when they predicted values (within 25%), with the overall average for all models being 0.50(table 13) which is good as it shows that when values aren't predicted exactly, half of them still fall between 25% of the actual value. Specifically, the RF model was able to perform better average providing a score of 0.58, with the KNN model coming second with a value of 0.56.

6.2.3 - Best Overall Performing Model

The best overall performing model was the SVM model. Despite models doing better than the SVM with certain datasets such as the KNN for R^2 (0.97) on the China dataset and the RMSE for LR (1086.5) on the China dataset; these models could not match consistency and high results for the other two datasets exhibiting that they are more tailored to larger datasets.

The SVM model produced the lowest average score results for RMSE(2841), MSE(9.38×10^6), MAE(1840) and the highest results for R^2 triumphing over the other models. This model was able to handle all datasets to high and consistent standard by producing an average R^2 of 0.76 for all datasets. This not only showed flexibility of the model as it handles high feature datasets like maxwell but also handles datasets with a lot of entries such as the China dataset.

The results given not only show that the variability in targets is explained to a impressively high level using the SVM model (R^2 value) for all datasets but also the average error between estimations an target values are also low(MAE). This would make the model a highly attractive option for any size of dataset or features.

The SVM model improved as more project data and features were input into the system and this can be shown through the results of 0.59 for Desharnais, increasing to 0.76 for Maxwell and then 0.95 for the China dataset. This trend shows that accuracy of the model is proportional to the size of the datasets and features available. This trend was

also shown through the Hyperparameter model without feature selection with the SVM model returning an R^2 score of 0.77 being the highest out all versions and models.

To confirm the best model a ScottKnottESD test which ranks effect size was used for all models. The results shown in tables 8-13 confirm that the SVM model is the best model, positioning the highest for R^2 (joint first with RF), RMSE (1st), MSE(1st), MAE(joint first with RF and KNN). As the ScottKnottESD ranking allows joint positions the averages were also included in the tables to add further context to the rankings and for the all of the 1st positions the SVM received it also had the best average scores.

6.2.4 - Best Performing Dataset

China achieved consistent scores across the board for every model and performed the best for every model too. China contains the most information on projects and because of this it allowed models to train to a higher level of accuracy than the other datasets. Achieving averages of 0.93 for R^2 , 1501 for RMSE, and 451 for MAE, the China dataset dominated in all the other classes.

The size of the dataset however also meant that it took the longest of all the datasets to go through and this is because it is over double the size of the others.

6.2.5 - Worst Performing Model

Unfortunately, not all models performed well and achieved high R^2 values. The model that ranked the lowest for R^2 and ranked last in all ScottKnottESD tests was the Decision Tree. Unlike other models which performed poorly for only one dataset e.g., Desharnais for the KNN model. Tweaks were made to the Decision Tree just as they were too the other models, but the Decision Tree performed poorly across the board producing a 0.39 average for R^2 to follow up with this the results of the ScottKnottESD concluded the evaluation that this model would not be fit to perform SEE.

The others although did not rank as highly as the SVM they showed promising results such as KNN & RF for Pred(25). The models of course are made to be anti-overfitting and low taxing on the CPU, because of this they will do better in generalisation and this can affect accuracy. The reason the Decision Tree was ranked as the worst was because it show it any highlights of good performance; coming last in every single metric (apart from time) even for the ScottKnottESD ranking.

6.2.6 - Worst Performing Dataset

Datasets are not equal, some contain large feature sets, some have many entries. This along with the values contained in the cells will alter how models perform with these datasets. In this case the worst performing model was the Desharnais dataset, the Desharnais dataset contains the lowest number of features, and this can account for the lower estimates. The dataset scored an average of 0.39 R^2 this means the models cannot explain more than 39% of the changes in effort values which would not be adequate for use on models.

7- Conclusion & Future work

This section of the report explores the significant findings through evaluating the models and datasets and future implementations to models and the program. The program will be set against the objectives and assessed on whether criteria was met, what worked well and what limited it.

7.1 - Personal Development

This project required skills in research, analysis, programming, planning and methodologies; this all together would produce this report. Learning more than 5 models of different machine learning techniques required a deep understanding for how they worked as they all have different foundations apart from DT and RF which share a similar background. To add to this choosing metrics to highlight the right qualities of model also required a great understanding. The use of latex to produce plans and graphs pushed for me to understand the way latex can be used alongside reports.

The datasets pushed my research and analysis skills. Datasets needed to be first found from a market that barely shares these publicly, after this the health and compatibility of each model was also to measured.

The program was produced with no prior knowledge of SEE, machine learning techniques (apart from ANN), Python libraries, and Jupyter Notebook/Google Colaboratory/Anaconda. As a result of successfully making myself relevant with the knowledge of these things the program also followed suit.

The experience gained through this project means I can take this knowledge and apply it to other studies such as image classification without having to be caught on so much information.

The set-out methodology allowed room for errors as well as mis planning my time and ultimately this proved to be the best thing for a project that underwent scope changes such as having to change platforms Jupyter notebook to Google Colaboratory.

7.2 - Objectives met

The objectives below were set during the research section of the report in order to access the models which were made.

Objective 1: Which Algorithm is better in terms of accuracy?

As discussed in the previous section the SVM was the most accurate model producing the best scores for 4/6 of the measured metrics.

Objective 2: Which Algorithm rounds faster?

The algorithm which rounded the fastest was the Decision Tree model, it took on average 0.02 seconds to process one dataset.

Objective 3: How robust is my algorithm?

All datasets used a random_state value, this means that for every run the same values were chosen to ensure consistency with implementing the data. This prevented variation between runs of the models ensuring that consistent results from the dataset are produced.

DT and RF allow the use of random_state too allowing further consistency with the results produced. As for the other models because the data being fed into the models is consistent the scores do not vary much (, max 0.1 R^2 variation) and so all models are robust.

Depending on the system running times may alter slightly and this is because of how your browser allocates memory, even though Google Colaboratory uses online resources.

Objective 4: Is my approach suitable for effort estimation?

Yes, specifically the SVM model. The models performed well across all datasets, taking the average of all the models, and then taking average of that, all the models produce on average 0.65 R^2 which is high for datasets that vary. Highlighting the SVM, if this were used in a modern-day project scenario an R^2 of 0.77 would be taken over a human

counterpart as human error would produce more inconsistencies when estimating software effort for numerous projects.

Objective 5: To what extent is the algorithm sensitive to features?

All models are somewhat sensitive to features as there is a drop of R^2 from Maxwell with the most features to Desharnais the least number of features, however this could be due to the difference in the number of projects in the dataset. Regardless, models like the SVM and RF are still able to produce R^2 scores greater than 0.70 showing that regardless of if its 17 or 26 features data is still able to produce consistent results.

As for the statistical objectives all were met. Every model presented had the ability to predict software effort, specifically the Maxwell and China datasets were able to predict to a highly accurate results (a score above 0.65 R^2), but the Desharnais dataset showed more variation for this. However, in terms of models all but the DT (0.39) achieved this by having a score R^2 score above 0.65: SVM (0.77), RF (0.73), LR (0.70), KNN (0.66).

Concerning the other statistical objectives, RMSE, MAE and Time were all evaluated for every model for every single providing more insight to how well a model did for each dataset.

7.3 - Hyperparameter Optimisation Advantages

The use of two methods for feature selection proved to be phenomenally successful, allowing the most influential features to be selected. Models that had feature selection performed on a similar level or out-did models that did not have feature selection (RF, LR) and if it did not, it did not decrease more than 8% for R^2 values. Looking at RMSE for these models the performed even close than R^2 with the SVM, RF, KNN models being under 1% in decrease for the scores, the DT was under 5% and the only model over 5% decrease was the LR, thus proving highly successful.

The speed as mentioned before in the results section was also a major highlight as the model with all features ran at least 5% slower with a max of 40% (for the SVM model).

Overfitting was helped using this method and it could be said that for certain datasets with a high R^2 like the China dataset could definitely have its features reduced even more to prevent overfitting and the speed bottleneck it would give certain models such as the SVM and RF.

7.4 - Hyperparameter Optimisation Limitations

Poorer performing models would perform better with all their data. The Desharnais dataset performed better with all its features compared to when using feature selection and for certain models such as the LR model the model increased by 30.75% to 0.66 when using all features which would have made the LR method more appealing and gave the Desharnais data set a better score overall.

7.5 - Models

Models as mentioned in the results section were tailored to not overfit and not cause a lot of strain on the computer. Done through feature selection and not including GPU models this was able to be attained, however, due to the fairness all models needed to have either hyperparameters or not. This affected some models worse than others (specifically the DT & LR) if there was chance to run a mixture of feature selection and hyperparameters between datasets and models the overall performance would've been higher. This is shown in tables 8-13 where default models or models with all the features available did exceedingly well compared to their hyperparameter (with feature selection) models did.

There are plenty of ML models that could have been used including hybrids, gradient boosting and ANNs to name a few. Some of the advantages of using more than 5 models: would allow more comparison between techniques, would allow a more optimal technique that could work better with smaller datasets such as Desharnais dataset to be found. There also could be a technique that was quicker and provide more parameters to tweak allowing greater customisation.

7.6 - Datasets

Although scarce, I managed to find 6 datasets (China, Desharnais, Maxwell, Kitchenham, ISBSG, USP05). Using only half (China, Desharnais, Maxwell) due to complications with cleaning and importing them, but if these datasets were able to be used, I could have tested my models with larger datasets such as ISBSG allowing the true flexibility of models such as the SVM to be evaluated.

Datasets would allow greater context for when a certain model may be used, as one model may perform better on larger datasets.

7.7 - Cost benefit analysis

A cost benefit analysis of the models presented could be carried out to determine if the best model (the SVM) would be economically viable, this way it could be attractive for companies to use this over human estimation or other methods of SEE.

7.8 - User Interface

The user interface presented was an CMD interface which allows the user to interact with the system, but this could be greatly improved with graphical user interface (GUI) this would allow data to be presented in a more appealing way because a layout would be created.

The use of figures would be able to be used more such as the ones that were produced early for the DT (fig. 22 & 23) this could be presented in a better format and would allow users to save these to their systems with the click of a button.

7.9 - CUDA (with TensorFlow)

The use of a GPU can increase the speed models run exponentially, reducing the time taken.

CUDA is a system developed by Nvidia and can be used with TensorFlow so models such as ANN can be run with much more complexity at quicker speeds.

Not all the models I used however has support, only the SVM, DT, LR and KNN have support meaning I may have to implement another model such as ANN instead of RF to make sure that it is all fair.

The use of an LSTM model could prove to provide remarkably high R^2 due to the memory capabilities however this runs extremely slow on normal systems due to the amount of computation strain but in combination with CUDA these speeds would be greatly reduced allowing the ease of use for it.

7.10 - Cloud System

A cloud system could provide more resources, which would result in a quicker running system. Google colab provides custom system would allow me to set my CPU, GPU, and RAM and then through using the cloud(GCP) I could trigger the algorithms to run automatically for when a dataset is uploaded, essentially automating the entire system.

7.11 - Conclusion

Modern day approaches to developing software is becoming more streamlined, with less resources and because of this time needed for expert guides to assess software effort is becoming harder to find. With the use of AI, ML can be used as a substitute for SEE providing better results at quicker speeds without the need to be hire someone specifically for this role cutting costs in one area which then can used in another.

Specifically, from the models assessed in this report the SVM would be the most appropriate model to use in a modern-day situation due to its robustness and accuracy with varying datasets. The use of this model could have a positive impact on companies spend and could be used through the whole program as data changes giving essentially real time estimates of software cost, increasing efficiency. This would help a company to have a lower risk assessment when looking into projects and would add much more clarity to what is to be expected at the end of the project.

This contributes to the discussion of ML within industries and how can be used as an advantage. Due to the ambiguity between the project information the model was given, it can be concluded that the model can be used for a wide range of datasets meaning it can be applied to multiple industries making it ubiquitous.

Solutions explored within this report required a notable amount of research and understanding to implement them. Everything from the framework used for models down to type of methodology used had chosen with care and reason. Although the models produced did not out do the models which are state-of-the-art, it did provide more evidence towards the use of ML for SEE and how It can be helpful in grand scheme of things. A foundation was shown on and how to build on models, respectively.

The accomplishments of this project showed that SEE could be predicted to high accuracy. R^2 scores of 0.97 were achieved for both the LR and KNN model for the China dataset which tell us that only 0.3% of efforts within the dataset did not fit the model. To add to this the KNN model also had 0.94% score for Pred (25) meaning that 314 of the project entries efforts were predicted within a 25% percentile range.

The next stage for this system would be automation, with this companies could simply upload their data to a cloud storage and the data would be cleaned, features selected and models ran on the datasets. With just implementing these slight changes the system would be able to handle most inputs and give highly accurate results; jobs like an

estimator could be completely replaced with the use of this program and ironically save money by cutting costs on hiring someone to minimise costs for projects.

8 - References

- [1] Sakpal, M. (2022) Gartner forecasts worldwide government IT spending to grow 6.8% in 2023, Gartner. Gartner. Available at: <https://www.gartner.com/en/newsroom/press-releases/2022-12-12-govt-it-spending-forecast-2023#:~:text=Modernization%20of%20Legacy%20Systems%20and,%2C%20according%20to%20Gartner%2C%20Inc.> (Accessed: January 4, 2023).
- [2] Arora, R. (2021) IBM: Digital Transformation in manufacturing 2021, Manufacturing Digital. The Manufacturer. Available at: <https://manufacturingdigital.com/smart-manufacturing/ibm-digital-transformation-manufacturing-2021> (Accessed: January 4, 2023).
- [3] Shivhare, J. and Rath, S.K. (2014) "Software effort estimation using Machine Learning Techniques," Proceedings of the 7th India Software Engineering Conference on - ISEC '14 [Preprint]. Available at: <https://doi.org/10.1145/2590748.2590767>.
- [4] Johnson, J. and Mulder, H. (2020) Endless Modernization - ResearchGate. The Standish Group. Available at: https://www.researchgate.net/profile/Hans-Mulder-2/publication/348849361_Endless_Modernization_How_Infinite_Flow_Keeps_Software_Fresh/links/60132878299bf1b33e30c29e/Endless-Modernization-How-Infinite-Flow-Keeps-Software-Fresh.pdf (Accessed: January 5, 2023).
- [5] Dejaeger, K. et al. (2011) "Data mining techniques for software effort estimation: A comparative study," IEEE Transactions on Software Engineering, 38(2), pp. 375–397. Available at: <https://doi.org/10.1109/tse.2011.55>.
- [6] Sarro, F., Petrozziello, A. and Harman, M. (2016) "multi-objective software effort estimation," Proceedings of the 38th International Conference on Software Engineering [Preprint]. Available at: <https://doi.org/10.1145/2884781.2884830>.
- [7] Trendowicz, A., Münch, J. and Jeffery, R. (2011) "State of the practice in software effort estimation: A survey and literature review," Software Engineering Techniques, 4980, pp. 232–245. Available at: https://doi.org/10.1007/978-3-642-22386-0_18.

- [8] Kitchenham, B., 2004. "Procedures for performing systematic reviews". Keele, UK, Keele University, 33(2004), pp.1-26.
- [9] Whigham, P.A., Owen, C.A. and Macdonell, S.G. (2015) "A baseline model for software effort estimation," ACM Transactions on Software Engineering and Methodology, 24(3), pp. 1–11. Available at: <https://doi.org/10.1145/2738037>.
- [10] Mahmood, Y. et al. (2021) "Software effort estimation accuracy prediction of Machine Learning Techniques: A systematic performance evaluation," Software: Practice and Experience, 52(1), pp. 39–65. Available at: <https://doi.org/10.1002/spe.3009>.
- [11] Monika and Sangwan, O.P. (2017) "Software effort estimation using Machine Learning Techniques," 2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence [Preprint]. Available at: <https://doi.org/10.1109/confluence.2017.7943130>.
- [12] Hammad, M. and Alqaddoumi, A. (2018) "Features-level software effort estimation using machine learning algorithms," 2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT) [Preprint]. Available at: <https://doi.org/10.1109/3ict.2018.8855752>.
- [13] Malhotra, R. and Jain, A. (2011) "Software effort prediction using statistical and Machine Learning Methods," International Journal of Advanced Computer Science and Applications, 2(1). Available at: <https://doi.org/10.14569/ijacsa.2011.020122>.
- [14] Silhavy, P., Silhavy, R. and Prokopova, Z. (2021) "Spectral clustering effect in software development effort estimation," Symmetry, 13(11), p. 2119. Available at: <https://doi.org/10.3390/sym13112119>.
- [15] Usharani, K., Ananth, V.V. and Velmurugan, D. (2016) "A survey on software effort estimation," 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT) [Preprint]. Available at: <https://doi.org/10.1109/iceeot.2016.7755665>.
- [16] Walkerden, F., Jeffery, R. An Empirical Study of Analogy-based Software Effort Estimation. Empirical Software Engineering 4, 135–158 (1999). <https://doi.org/10.1023/A:1009872202035>

- [17] K. Molokken and M. Jorgensen, "A review of software surveys on software effort estimation," 2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings., Rome, Italy, 2003, pp. 223-230, Doi: 10.1109/ISESE.2003.1237981.
- [18] Myles, A.J. et al. (2004) "An introduction to decision tree modeling," Journal of Chemometrics, 18(6), pp. 275–285. Available at: <https://doi.org/10.1002/cem.873>.
- [19] Liaw, A. and Wiener, M., 2002. Classification and regression by randomForest. R news, 2(3), pp.18-22.
- [20] Mali, K. (2022) Everything you need to know about linear regression! Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/> (Accessed: April 28, 2023).
- [21] Noble, W.S. (2006) "What is a support vector machine?" Nature Biotechnology, 24(12), pp. 1565–1567. Available at: <https://doi.org/10.1038/nbt1206-1565>.
- [22] <https://docs.python.org/3/library/> - Accessed: 28/04/23.
- [23] Peterson, L.E., 2009. K-nearest neighbor. Scholarpedia, 4(2), p.1883.
- [24] <https://scikit-learn.org/stable/index.html> - Accessed: 28/04/23.
- [25] Venkataiah, V., Nagaratna, M. and Mohanty, R. (2022) "Application of chaotic increasing linear inertia weight and diversity improved particle swarm optimization to predict accurate software cost estimation," International Journal of Electrical and Electronics Research, 10(2), pp. 154–160. Available at: <https://doi.org/10.37391/ijeer.100218>.
- [26] Esteves, T. (2018) Desharnais dataset, Kaggle. Kaggle. Available at: <https://www.kaggle.com/datasets/toniesteves/desharnais-dataset> (Accessed: April 29, 2023).
- [27] Tsunoda, M. (2017) Kitchenham, Zenodo. Zenodo. Available at: <https://zenodo.org/record/268457#.ZEKrznbMKUk> (Accessed: April 29, 2023).
- [28] Williams, B. et al. (2020) "Data-driven model development for cardiomyocyte production experimental failure prediction," Computer Aided Chemical Engineering, pp. 1639–1644. Available at: <https://doi.org/10.1016/b978-0-12-823377-1.50274-3>.

- [29] Gupta, A. (2020) ML: Extra tree classifier for feature selection, GeeksforGeeks. GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/ml-extra-tree-classifier-for-feature-selection/> (Accessed: April 29, 2023).
- [30] Hamilton DF, Gchert M, Simpson AH. Interpreting regression models in clinical outcome studies. Bone Joint Res. 2015 Sep;4(9):152-3. Doi: 10.1302/2046-3758.49.2000571. PMID: 26392591; PMCID: PMC4678365.
- [31] Jj (2016) Mae and RMSE - which metric is better? Medium. Human in a Machine World. Available at: <https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d> (Accessed: April 29, 2023).
- [32] Gupta, A. (2023) Mean squared error: Overview, examples, concepts and more: Simplilearn, Simplilearn.com. Simplilearn. Available at: <https://www.simplilearn.com/tutorials/statistics-tutorial/mean-squared-error#:~:text=The%20Mean%20Squared%20Error%20measures,it%20relates%20to%20a%20function.> (Accessed: April 30, 2023).
- [33] Doubtnut (2020) How is absolute error different from mean absolute error? doubtnut. doubtnut. Available at: <https://www.doubtnut.com/question-answer-physics/how-is-absolute-error-different-from-mean-absolute-error-415572522> (Accessed: April 30, 2023).
- [34] Wikipedia (2023) Coefficient of determination, Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Coefficient_of_determination (Accessed: April 30, 2023).
- [35] Wikipedia (2023) Mean absolute error, Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Mean_absolute_error (Accessed: April 30, 2023).
- [36] Wikipedia (2022) Mean squared error, Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Mean_squared_error (Accessed: April 30, 2023).
- [37] Fernando, J. (2023) R-squared: Definition, calculation formula, uses, and limitations, Investopedia. Investopedia. Available at: [https://www.investopedia.com/terms/r/r-squared.asp#:~:text=R%2Dsquared%20\(R2\),variable%20in%20a%20regression%20model.](https://www.investopedia.com/terms/r/r-squared.asp#:~:text=R%2Dsquared%20(R2),variable%20in%20a%20regression%20model.) (Accessed: April 30, 2023).

- [38] Tantithamthavorn, C. et al. (2019) "The impact of automated parameter optimization on defect prediction models," IEEE Transactions on Software Engineering, 45(7), pp. 683–711. Available at: <https://doi.org/10.1109/tse.2018.2794977>.
- [39] Rajan, V. and NobilisRex (2020) Python Loadarff fails for string attributes, Stack Overflow. Stack Overflow. Available at: <https://stackoverflow.com/questions/60704360/python-loadarff-fails-for-string-attributes> (Accessed: April 30, 2023).
- [40] Kaya, G.K. (2018) good risk assessment practice in hospitals, Research Gate. Research Gate. Available at: https://www.researchgate.net/publication/323570642_Good_risk_assessment_practice_in_hospitals (Accessed: April 30, 2023).
- [41] Boehm, B.W. (2008) Academic Staff Websites Directory | EMU academic staff directory, Eastern Mediterranean University. Available at: <https://staff.emu.edu.tr/alexanderchefranov/Documents/CMPE412/Boehm1981%20COCOMO.pdf> (Accessed: May 1, 2023).
- [42] Pedregosa, F. et al. (2011) Learn, scikit. Available at: <https://scikit-learn.org/stable/index.html> (Accessed: May 1, 2023).
- [43] Madhugiri, D. (2023) Exploratory Data Analysis (EDA): Types, tools, process, KnowledgeHut. Available at: <https://www.knowledgehut.com/blog/data-science/eda-data-science> (Accessed: May 1, 2023).
- [44] Fairly Nerdy (2017) What is R squared and negative r squared, Fairly Nerdy. Available at: <http://www.fairlynerdy.com/what-is-r-squared/> (Accessed: 10 May 2023).
- [45] SAP (2020) Predictive Factory Online Help, SAP help portal. Available at: <https://help.sap.com/docs/r/41d1a6d4e7574e32b815f1cc87c00f42/3.1/en-US/c2f0326ebc89450f83f106529733d60e.html> (Accessed: 11 May 2023).

9 - Appendix

9.1 - Key words

Machine Learning – ML

Software Effort Estimation – SEE

Prediction within 25% of actual value – Pred (25)

Mean absolute error – MAE

Mean squared error- MSE

Root mean squared error – RMSE

Support Vector machine – SVM

Decision tree – DT

Regression Decision tree - RDT

Linear regression – LR

K nearest neighbors – KNN

Random Forest – RF

Hybrid Models – HM

Artificial neural network – ANN

Artificial Intelligence – AI

Exploratory Data Analysis – EDA

R - A programming language for statistical computing and graphics supported

Fig. – Figure

Genetic Algorithm - GA

Best Fit – BF

Optimal Linear Combining Method – OLC

Automatically Transformed Linear Baseline Model – ATLM

9.2 - Running the program

The program is too be used in Google Colaboratory.

1. Once the program is loaded in the RPY2 version needs to be downgraded (Code provided for this). Once this has been finished the runtime needs to be restarted
2. Upload the dataset files to the runtime. The specific files are named 'china.arff', '02.desharnais.csv' and 'maxwell.arff'
3. The next step is to install the 'ScottKnottESD' package.
4. After this all code can be ran one after the other.
5. The last line of code calls the program 'make_predictions()' which allows the user to interact with the system.
6. The inputs into the CMD only takes numerical inputs followed by a comma. Any other input will give an error crashing the program.
7. To use the ScottKnottESD ranking test users will need to choose option '6' when the model choice is to be made, from here a list of metrics to test will be given.
8. To test the other model's users can enter the corresponding number (followed by a comma for multiple models)
9. After this the user will have the choice of choosing out of the 3 datasets.

9.3 - Additional program pictures

	SVM	DT	RF	LR	KNN
0	5004.016803	4374.666667	3795.183810	6309.081255	3297.495238
1	3311.605302	543.793939	439.261515	339.823637	402.028125
2	2758.120753	2339.148148	2682.761111	2016.902395	2376.175704

```

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/pandas2ri.py:
    for name, values in obj.iteritems():
        Model rank
0    SVM      1
1    LR       2
2    DT       3
3    RF       3
4    KNN      4

```

Figure of Default Models ScottKnottESD ranking for MAE [30]

	SVM	DT	RF	LR	KNN
0	7.388373e+07	3.465366e+07	8.038503e+07	8.247244e+07	2.405886e+07
1	4.064435e+07	2.160195e+06	2.074600e+06	5.542293e+05	1.189934e+06
2	2.128089e+07	6.492614e+06	1.648205e+07	7.658926e+06	1.476526e+07

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/pandas2ri.py:54: |
 for name, values in obj.iteritems():
 Model rank

0	SVM	1
1	RF	2
2	LR	2
3	DT	3
4	KNN	3

Figure of Default Models ScottKnottESD ranking for MSE [31]

	SVM	DT	RF	LR	KNN
0	0.150000	0.238095	0.523810	0.047619	0.238095
1	0.187879	0.915152	0.963636	0.757576	0.943750
2	0.285714	0.444444	0.222222	0.370370	0.342857

/usr/local/lib/python3.10/dist-packages/rpy2/robj
 for name, values in obj.iteritems():

	Model	rank
0	RF	1
1	DT	1
2	KNN	1
3	LR	2
4	SVM	3

Figure of Default Models ScottKnottESD ranking for Pred (25) [32]

	SVM	DT	RF	LR	KNN
0	-0.052249	0.381053	0.678738	0.612502	-0.342760
1	-0.133883	0.934547	0.950838	0.986071	0.967576
2	-0.080403	0.663346	0.287112	0.667847	0.250387

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/p
 for name, values in obj.iteritems():

	Model	rank
0	LR	1
1	DT	2
2	RF	2
3	KNN	3
4	SVM	4

Figure of Default Models ScottKnottESD ranking for R^2 [33]

	SVM	DT	RF	LR	KNN
0	8595.564600	7430.558689	8920.710703	9081.433713	4904.982791
1	6375.291924	2229.135817	1470.398212	744.465766	1090.840826
2	4613.121199	3173.693333	4021.300890	2767.476425	3842.558545

```

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/pandas2ri.py:
  for name, values in obj.iteritems():
    Model rank
0    SVM    1
1     RF    2
2     DT    2
3     LR    2
4    KNN    3

```

Figure of Default Models ScottKnottESD ranking for RMSE [34]

	SVM	DT	RF	LR	KNN
0	0.002912	0.004228	0.138690	0.002772	0.001550
1	0.018852	0.007430	0.356811	0.002524	0.017966
2	0.003251	0.003692	0.137543	0.002404	0.144634

/usr/local/lib/python3.10/dist-packages/rpy2/robjects
 for name, values in obj.iteritems():

	Model	rank
0	RF	1
1	KNN	2
2	SVM	3
3	DT	4
4	LR	5

Figure of Default Models ScottKnottESD ranking for Time [35]

	SVM	DT	RF	LR	KNN
0	2733.033793	4996.568594	3081.265688	7777.358118	3259.174603
1	328.564268	849.503277	428.817747	378.459956	409.137500
2	2394.968568	3216.263690	2023.609686	2032.467731	2387.251746

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/pandas2ri.p
 for name, values in obj.iteritems():
 Model rank

0	LR	1
1	DT	1
2	KNN	2
3	RF	2
4	SVM	2

Figure of Hyperparameter Models with all features ScottKnottESD ranking for MAE [36]

	SVM	DT	RF	LR	KNN
0	1.648894e+07	5.758943e+07	2.500284e+07	1.056714e+08	2.560104e+07
1	1.736138e+06	5.690418e+06	1.974060e+06	1.457029e+06	1.204899e+06
2	9.436555e+06	2.100976e+07	9.398959e+06	7.821383e+06	1.477935e+07

```

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/pandas2ri.py:54:
    for name, values in obj.iteritems():
        Model rank
0    LR    1
1    DT    2
2    KNN   3
3    RF    3
4    SVM   4

```

Figure of Hyperparameter Models with all features ScottKnottESD ranking for MSE [37]

	SVM	DT	RF	LR	KNN
0	0.400000	0.190476	0.428571	0.142857	0.380952
1	0.962500	0.856250	0.969697	0.751515	0.962500
2	0.171429	0.200000	0.303030	0.407407	0.342857

```
/usr/local/lib/python3.10/dist-packages/rpy2/robj  
for name, values in obj.iteritems():  
    Model rank
```

0	RF	1
1	KNN	1
2	SVM	1
3	LR	2
4	DT	2

Figure of Hyperparameter Models with all features ScottKnottESD ranking for Pred (25) [38]

	SVM	DT	RF	LR	KNN
0	0.765165	0.305241	0.698366	-0.274820	0.766743
1	0.952692	0.844943	0.950386	0.959352	0.967168
2	0.595455	0.084633	0.546464	0.660802	0.249671

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/
 for name, values in obj.iteritems():
 Model rank

0	SVM	1
1	RF	2
2	KNN	3
3	LR	4
4	DT	4

Figure of Hyperparameter Models with all features ScottKnottESD ranking for R^2 [39]

	SVM	DT	RF	LR	KNN
0	4060.657205	7588.769969	5000.283893	10279.658817	5059.747517
1	1317.625834	2385.459679	1405.012389	1207.074747	1097.679107
2	3071.897649	4583.640569	3065.772248	2796.673626	3844.392639

```

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/pandas2ri.py:5
  for name, values in obj.iteritems():
    Model rank
0    DT    1
1    LR    1
2    KNN   2
3    RF    2
4    SVM   3

```

Figure of Hyperparameter Models with all features ScottKnottESD ranking for RMSE [40]

	SVM	DT	RF	LR	KNN
0	0.176181	0.032083	6.443133	0.029809	0.061074
1	356.506143	0.039937	8.033969	0.025617	0.021075
2	0.120453	0.027682	0.936331	0.021478	0.141608

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/p
 for name, values in obj.iteritems():

	Model	rank
0	SVM	1
1	RF	2
2	KNN	3
3	DT	4
4	LR	5

Figure of Hyperparameter Models with all features ScottKnottESD ranking for Time [41]

	SVM	DT	RF	LR	KNN
0	2781.682237	4723.332963	3089.002408	4657.963739	3259.174603
1	323.780918	771.544742	430.725994	325.295090	402.028125
2	2414.812091	3349.333333	2052.358020	2366.229380	2376.175704

```

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/pandas2ri.py:
    for name, values in obj.iteritems():
        Model rank
0    DT    1
1    LR    2
2    KNN   3
3    RF    3
4    SVM    3

```

Figure of Hyperparameter Models with feature selection ScottKnottESD ranking for MAE [42]

	SVM	DT	RF	LR	KNN
0	0.35000	0.150000	0.428571	0.238095	0.380952
1	0.96875	0.875000	0.969697	0.836364	0.943750
2	0.20000	0.185185	0.363636	0.259259	0.342857

```

/usr/local/lib/python3.10/dist-packages/rpy2/robj
for name, values in obj.iteritems():
    Model rank

```

0	RF	1
1	KNN	1
2	SVM	2
3	LR	3
4	DT	3

Figure of Hyperparameter Models with feature selection ScottKnottESD ranking for Pred (25) [43]

	SVM	DT	RF	LR	KNN
0	0.761366	0.300200	0.697760	0.568416	0.766743
1	0.952046	0.837992	0.948620	0.967067	0.967576
2	0.587139	0.026591	0.539506	0.562982	0.250387

/usr/local/lib/python3.10/dist-packages/rpy2/robjects
 for name, values in obj.iteritems():
 Model rank

0	SVM	1
1	RF	1
2	LR	2
3	KNN	2
4	DT	3

Figure of Hyperparameter Models with feature selection ScottKnottESD ranking for R^2 [44]

	SVM	DT	RF	LR	KNN
0	4093.375157	6607.312492	5005.300611	5981.178212	5059.747517
1	1326.598326	2573.176355	1429.804844	1086.501801	1090.840826
2	3103.308386	4714.884726	3089.198134	3174.420555	3842.558545

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/pandas2ri.py:54: FutureWarning: Iterating over a dict will return the values by default. Please use obj.items() to iterate over a dict.

	Model	rank
0	DT	1
1	LR	2
2	KNN	2
3	RF	2
4	SVM	3

Figure of Hyperparameter Models with feature selection ScottKnottESD ranking for RMSE [45]

	SVM	DT	RF	LR	KNN
0	0.415671	0.019357	7.485538	0.038051	0.071727
1	209.914896	0.023688	5.938238	0.035494	0.023274
2	0.047183	0.017804	1.157733	0.032959	0.197552

/usr/local/lib/python3.10/dist-packages/rpy2/robjects
 for name, values in obj.iteritems():
 Model rank

0	SVM	1
1	RF	2
2	KNN	3
3	LR	4
4	DT	5

Figure of Hyperparameter Models with feature selection ScottKnottESD ranking for Time [46]

	SVM	DT	RF	LR	KNN
0	1.675572e+07	4.365658e+07	2.505303e+07	3.577449e+07	2.560104e+07
1	1.759863e+06	6.621237e+06	2.044342e+06	1.180486e+06	1.189934e+06
2	9.630523e+06	2.223014e+07	9.543145e+06	1.007695e+07	1.476526e+07

/usr/local/lib/python3.10/dist-packages/rpy2/robjects/pandas2ri.py:54:
 for name, values in obj.iteritems():
 Model rank

0	DT	1
1	LR	2
2	KNN	2
3	RF	3
4	SVM	4

Figure of Hyperparameter Models with feature selection ScottKnottESD ranking for MSE [47]