Bachelor's Thesis Computing Science

Radboud University Nijmegen

Investigating the impact of GitHub repository features on popularity

---

Author:
Andrei-Radu Milicin
s1076192

First supervisor/assessor:
Dr. Mairieli Wessel

Second assessor:
Dr. Twan van Laarhoven

June 10, 2024

Abstract

GitHub is currently the largest and most utilised open-source repository hosting platform with integrated Git version control, CI/CD pipelines, AI auto-completion and task management. Because of GitHub's scale and importance in the developer community, discovering what makes GitHub repositories popular is crucial in understanding the expectations of the programming world. Multiple studies have been conducted on the topic but they fail to encompass some GitHub features that may relate to repository popularity, they do not discuss all effects releases have on the number of stars, nor do they explore the importance of these features in predicting the number of stars. This thesis aims to find out what gives rise to popularity in GitHub projects, by answering several questions: How can popularity be quantised in GitHub? What features influence repository popularity positively? What are the possible repository growth patterns and to what extent releases impact weekly star growth? Can we accurately predict the number of stars of a GitHub repository in a certain week? What are the features that impacted the predictions the most? It was decided that popularity should be measured by the number of stars a repository has. The correlation analysis chapter unfolded by proving that forks, watchers and other features influence the popularity of a repository whereas the number of commits and the ratio between closed and all issues do not. Multiple repository growth patterns have been found, among which fast initial growing repositories which then dimmed down, constant growing, and late growing projects. Releases impact the number of stars in a week by either being on a spike in the week of the release, a spike follows several weeks after a release, constant growth and even a decrease as no project can maintain a perennial increase in stars. Popularity prediction tried 3 machine learning algorithms: Multiple Linear Regression, Neural Networks, Random Forests and 2 inputs to the models (one with releases and one without) which proved to be inaccurate in predicting stars per week due to insufficient data, and due to the unpredictability of GitHub repository growth. A neural network with the input without releases performed best overall at individual predictions, but the neural networks with releases in the input had better overall predictions. The most important features in the predictions were the number of stars in the week before the prediction, the programming language, the number of stars 3 weeks before the prediction, the type of release in the week before the prediction and the type of license.

Contents

Chapter 1

Introduction

Open Source Software [13] has been gaining a lot of momentum over time (40 million users in 2019, 92 million users in 2024, set to become 100 million by 2025) with advanced applications that allow developers to share their code on remote repositories, have version control systems and issue trackers. GitHub [3] is a web-based platform that provides hosting for version control and collaboration, allowing users to work together on projects from anywhere. It utilizes Git [3][20], a distributed version control system, to enable multiple users to track changes in source code during software development. The amount of features and the versatility of GitHub makes it the most popular remote repository hosting platform, and with the advent of AI (GitHub Copilot) [23] and CI/CD pipelines automation and productivity have never been greater. As GitHub is so widely used in the programming community, a user feedback system had to be put in place [6].

Starring a repository functions more or less like a bookmark, after which you can view the repository on the "starred repositories" list in GitHub. After starring a repository, other projects on similar topics will be recommended to you. It has been studied that the majority of developers in GitHub star projects so they can look at them later and a smaller percentage that are currently or have used the starred repositories [5]. Thus, by looking at the number of stars in relation to other repositories, the project can be viewed as more or less popular, with the majority of developers using a project with a considerable amount of stars as opposed to one with fewer [5]. This is because developers want to make sure that the project is maintained, and the number of stars is an assurance to the developers working on the project that what they are working on is well received and still matters to the community. There are multiple other GitHub repository

attributes that are related to the popularity and these are:

1. the number of forks, with a fork being a repository which points to the original repository (the owner of the forked repository has read permissions of all forks), and is made with the intent of working on it locally until submitting a pull request to the repository owner (submitting the changes made in the fork).

2. current number of watchers, which correspond to all the people that are notified of all changes of a repository, i.e. commits, issues being created etc.

3. The dependency graph (How many projects use a particular project)

4. External references (other mentions of the project in social media, blogs etc.)

and many others.

Because the popularity of a repository indicates how used and how relevant it is, it is essential to find out what gives rise to it.

1.1 Problem statement

Objective repository popularity, which can be measured by the feedback received from users via stars [4][5][6][15], forks [1], watchers [6], frequency of utilisation [6] etc., is what ultimately decides whether a repository is successful or not [4][5][6]. However, the limited amount of studies regarding this topic have not included some variables that may relate to the popularity of a Github repository, have not researched all the effects releases may have on the number of stars and have not discussed and validated the impact of variables in predicting the number of stars over time. This thesis aims to find which GitHub features correlate with a repository's popularity, to what extent releases impact popularity and whether predicting the number of stars of a GitHub repository is feasible given only data mined through the GitHub API. The research question that we aim to answer in this research paper is: "What are the GitHub repository attributes that influence their popularity?". The research question is broken down into multiple subquestions which will be answered sequentially in the methodology chapter. The subquestions are:

Q1: How is the popularity of a repository quantised? (What is an appropriate metric for measuring repository's popularity?)

Q2: How do set features like programming languages used, type of license, type of owner (user/organisation) influence the metric explained after answering Q1?

Q3: Do Git and GitHub factors like the nr. of commits, nr. of forks, nr. of watchers, ratio between closed and total issues influence the popularity of a project (number of stars)?

Q4: What are the possible growth patterns and how do releases impact the number of stars?

Q5: Can we accurately predict the number of stars in a certain week? What are the features used for the input of the models? What are the features that impacted the prediction the most?

The research question and subquestions will all be answered during the Methodology/Results/Discussion sections.

1.2 Motivation

Popularity is defined by the attention and the appreciation that an object/idea/place/person receives. Finding what is popular and how that something got popular can give you insights into how to design new items that are popular, or maybe exploit that item. In Software Development, projects get the most attention by being available to the public, more explicitly by being Open Source.

Open Source Software is the best way to make code accessible to everyone, because everyone can see your code, they can then learn about the insides of it and they can even contribute if you accept, which may end up in boosting the project even more. The popularity of Open Source Software, especially in Github which is the topic of this thesis, stems from usefulness, ease of use and the users' opinion of the software. There is no doubt that people have been asking: 'What do I have to do to make my repository more popular on Github?'. Intuitive responses to this question are: clean code, usefulness, uses of trendy programming languages and topics. While

of course, these are all true, an in depth analysis of what are the features of a Github repository that lead to its success is needed to reveal true insights.

Finding a definitive answer to what brings success to a GitHub repository seems unlikely as there may be multiple ways to approach this problem. However, uncovering relationships between different GitHub features and a repository's popularity will provide an appropriate guide or framework for developers as to what their GitHub behaviour should be like if they would like to attract more followers. This could range between changing the type of license for a more permissive use, the frequency and magnitude of the releases and so on. Another reason for making this project is to build on top of the Mining Software Repository (MSR) literature and for this work to be used in subsequent works in the field.

Chapter 2

Background

2.1 Git & Github

Git is an open source version control system, with which one can make checkpoints of projects, revert back to different versions of a project and look at differences between files of different versions [20]. It has grown in adoption with the appearance of GitHub, which is a website which allows repositories (projects) to be hosted on the platform, and also has a UI with which the user can perform tasks that could be produced within the CLI with Git on the local machine.

In order for Git to be used in the version control of a project, the user needs to type in 'git init', which essentially 'allows' the directory in which they are working to be tracked by Git. There are 3 'areas' in which a project can be with Git: modified, staged and commited. The modified area is represented by files that have been changed but have not told git to put any of these files in a new version, the staged area represents files which Git will put into the new version, and the commited area which relates to the files from the staged area that have been already assigned a new version. The project enters the staging area when the user runs 'git add file1 file2 ... fileX', which tells git to put all of the files mentioned after 'add' to the staging area. Commiting is done by running 'git commit -m "any commit message"'. Commits are all identifiable by hashes which are codified representations of the contents of the commits. The powerful idea behind Git is that because all of the commits are stored locally, one can simply go back to a previous commit if something goes horribly wrong with 'git checkout commitHash', where commitHash is the 20 character long SHA-1 hash of the commit. Another very important feature of Git is branching, which allows

the user to 'branch out' of the main commit tree, and work on a completely separate versions with probably other features. This is best shown in Figure 1, where branch 2 is a branch made starting from the first commit in the main branch, and branch 3 is a branch made from the 3rd commit in the main branch.

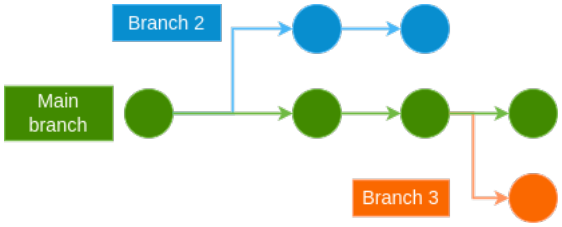

Figure 1. Git branching

Creating a branch is done by running 'git branch branchName', and moving to that branch is done with 'git checkout branchName'. This tells git that from now on the commits will happen on the branch 'branchName'. What was discussed so far are the basics of Git. The true power of Git comes from its synergy with GitHub. Because GitHub has the projects stored somewhere, that means we can get them locally. Git clone repoUrl essentially gets the entire repository with the url repoUrl from GitHub onto the local machine. Linking a remote GitHub repository to the local machine is also very handy as that is how you can get access to the whole project, including different branches that other people have made to the project on GitHub. Also essential are 'git fetch' and 'git pull', which given a branch that the user is on it fetches and gets all of the newest changes from a branch in GitHub to the local machine, and git push which pushes the contents of a branch you are on in the linked remote repository or local machine onto GitHub (if the pull request is accepted, i.e. the code you just pushed, then if the branch existed only the code is changed, and if the branch didn't exist, GitHub will also create the branch alongside all of the code.

GitHub presents numerous metrics and features which allows the developers to understand developer satisfaction with the project, make tasks, view tasks and assign developer tasks, see pull requests etc. The main page of a repository looks like Figure 2.

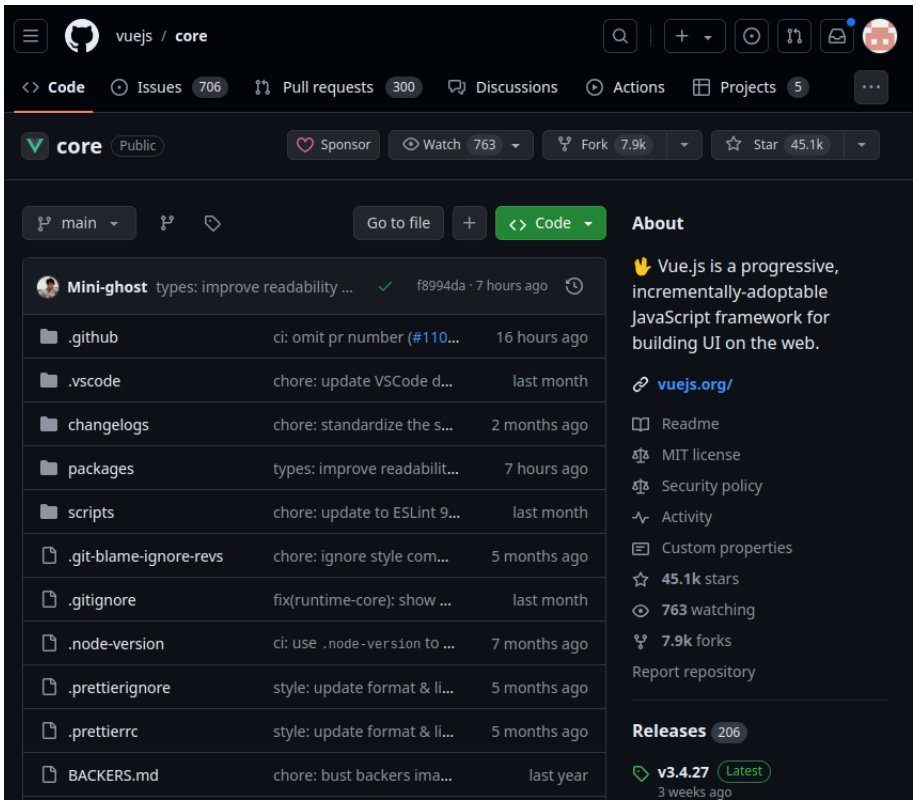

Figure 2. Example repository UI - VueJS repository

The main GitHub repository UI has multiple features, but we will discuss only the basics. The file system of the project is visible in the middle. Tasks can be visualised by clicking issues and pull requests as well. Branches can be visualised by clicking on the button on the left above the code that has 'main' written on it. After you choose a branch the file system of the branch is then shown on the UI. One can get the url of the project by clicking on the Code button and taking the url that comes from the https version. With that url you can clone the repository as discussed previously. Important features that will come up in the research will be the stars, watchers and forks, which are all done by viewers of the repository appear on the right middle, alongside with the license of the repository and below the current release of the repository. Previously mentioned in the introduction, stars act as bookmarks of projects, with users being able to visualise the projects later in a list of all their starred repositories. Forks are repositories linked to the original repositories, to which the forker can later submit a pull request to change the code with their code. The owner of the original repository

can also see all the forked repositories. Watchers in GitHub are people that are notified of every change that happens to a repository. This includes commits, pull requests, branches made etc. The release system in GitHub is largely ubiquitous, with almost all repositories following the convention of x.y.z, with x - major release, y - minor release, z - bug fix. Licenses in GitHub tell the users what they are allowed to do with the code. There are permissive licenses and more limiting ones. Permissive licenses such as the MIT License allow the developer to use the code, and can make money from software distributed which uses that particular repository as a library. Limiting licenses such as GNU General Public which allows users to use the code, but the project must be open sourced.

2.2 Correlation analysis

Correlation analysis denotes finding the strength of the relationship between 2 variables. More specifically, one variable is correlated to another if a change in the first variable prompts a change in the second variable. In statistics, there are 3 types of correlation: positive correlation, negative correlation and no correlation. Positive correlation signifies the degree of which an increase in a variable determines an increase in the other variable. On the other hand, negative correlation relates to the degree to which a decrease in a variable amounts to an increase in the other. Zero correlation indicates that there is no relationship between 2 variables. This is also depicted in the plots from figure 3.

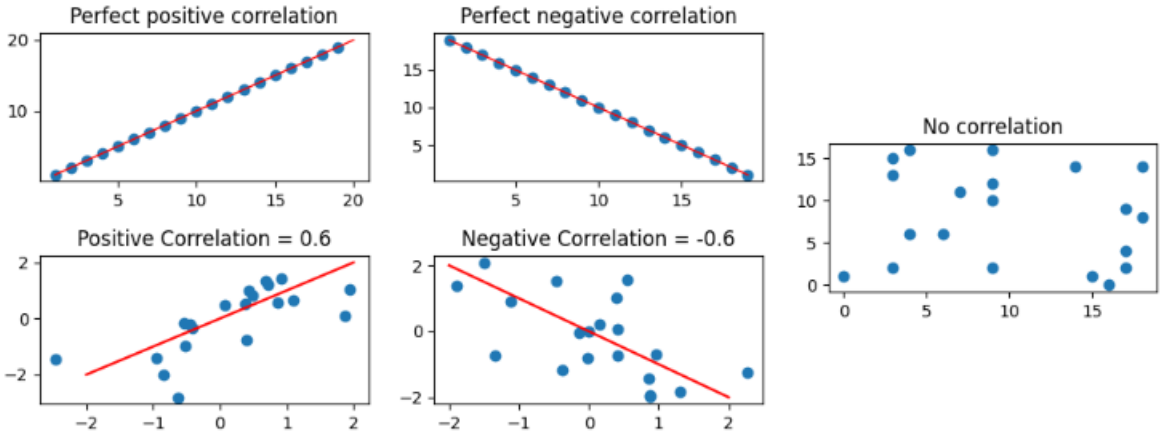

Figure 3. Types of correlation

Figure 3 shows different types of correlations. The upper-left plot shows the perfect positive correlation where every time variable x increases, variable y also increases. The upper-right plot shows the perfect negative correlation where an increase in a variable signals a decrease in the other. The bottom 2 plots represent a positive and negative correlation of 0.6 respectively where there is a margin of error of whether the change in a variable will signal an increase or a decrease. The rightmost plot shows data which has no correlation. The metric with which correlation will be calculated is Pearson's correlation coefficient.

$$\rho_{x,y} = \frac{cov(X,Y)}{\sigma_x \cdot \sigma_y}$$

Equation 1. Pearson's correlation coefficient

$$\mathrm{Cov}(x,y) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N}$$

Equation 2. Covariance formula

$$\sigma_x = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$$

Equation 3. Standard deviation formula

The covariance from Equation 2. indicates the degree to which the 2 variables grow together. A high covariance means that the variables change a lot relative to the mean and a low covariance shows that the variables do not change that much. The standard deviation shows how far values tend to be from the mean. The main metric used in the correlation analysis will be Pearson's coefficient, so we will solely refer to it from now on. Another important tool needed in correlation analysis is statistical significance.

2.3 Statistical significance

Statistical significance is another test that must be applied to the data. Significance refers to the claim that a result from data generated by testing or experimentation is likely to be attributable to a specific cause. Due to the data in this research having outliers, being left skewed (there are more repositories with less stars), and having multiple groups of data being analysed, the Kruskal-Wallis test is chosen. Kruskal-Wallis is a rank based

approach that checks whether the medians are different across all groups. It follows the formula:

$$H = (N-1)\frac{\Sigma_{i=1}^{g} n_i(\bar{r_i}-\bar{r})^2}{\Sigma_{i=1}\Sigma_{j=1}^{n_i}(r_{ij}-\bar{r})^2}$$

Equation 4. Kruskal-Wallis' H formula

- N is the total number of observations across all groups
- g is the total number of groups
- n_i is the number of observations in group i
- n_{ij} is the rank of observation j from group i
- $\bar{r_i}$ is the mean rank of group i
- $\bar{r_{ij}}$ is the mean rank of all groups

The degrees of freedom (df) in this case are the number of groups that we are comparing - 1:

$$df = |G| - 1, \text{ where:}$$

- $|G|$ is the number of groups

.

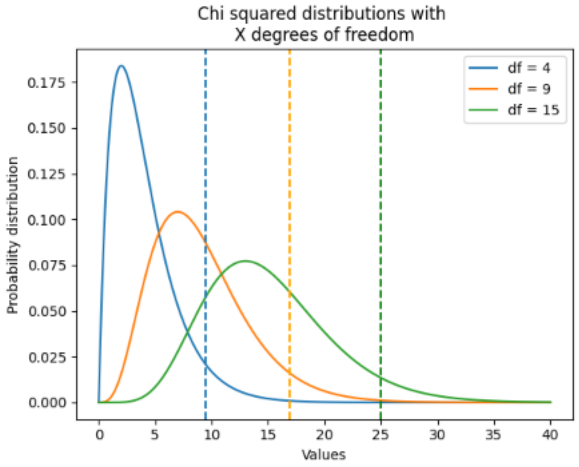

Figure 4. Chi squared distribution with X degrees of freedom

Because H follows the same distribution as χ^2 (Figure 4.), we check whether H is larger than the critical value, and if it is we reject the null hypothesis and thus the medians are different. The critical value at the significance level 0.05, signifies the x value is associated with a y value (the x value matched with the dotted lines start in Figure 4.) where the area under the graph on the right of the critical value is 0.05. The larger the H value compared to the critical value, the more statistically significant the result. Conversely, if H is smaller than the critical value, then it is known that the groups come from the same sample, and thus the results are statistically insignificant.

Another way to calculate the statistical significance is by using H to derive a p-value, which is the probability of H to be larger or equal to itself. This is done by calculating the area under the curve from H rightward. The lower the p-value, the more statistically significant the result.

In this research, both Kruskal-Wallis and Mann-Whitney (the 2 group alternative to Kruskal-Wallis) are used to calculate significance levels.

2.4 Regression algorithms

This section will elaborate on how the techniques used in the prediction of stars for Github repositories (Chapter 4.6) generally work. These encompass:

1. Multiple linear regression
2. Neural Network regression
3. Random Forest regression

2.4.1 Regression

Regression is a supervised learning task where given multiple dependant variables, a continuous variable is predicted. Supervised learning means that given data for which the answer is known, the machine learning model tweaks its parameters to minimise the errors between predicted and actual values. Before model prediction, subdividing the dataset between training and test set to train and test the model is done. There are multiple ways one can subdivide the dataset but the most common approach is 80%/20% or 75%/25%. Ensuring that the model has enough data to train is crucial as the quality of the prediction is influenced by it, along with other factors such as distribution and skewness of the data.

Time-series regression, represents predicting a continuous variable at time t, given variables at times $t-z,..,t-2,t-1$. In this research, the dependant variables are GitHub repository features such as programming language, license type, number of contributors (to name a few) and variables $X_{t-z}, X_{t-z+1}, .., X_{t-2}, X_{t-1}$, with $X_{t-z}, X_{t-z+1}, ..X_{t-2}, X_{t-1}$ being the values of variables at times $t-z,..,t-2,t-1$. These in conjunction are all used to predict the value X_t.

2.4.2 (Multiple) linear regression

Linear regression is a supervised learning algorithm, which predicts a continuous variable given another variable. This is done by approximating the line that minimises the errors between the predicted values and actual values. The line that results from this algorithm is also more commonly known as "the line of best fit", as it best fits the data. The formula for the line is the function:

$$y = ax + b$$

Equation 5. Linear regression

- x: dependent variable
- y: predicted variable

.

Multiple linear regression, as its name suggests, involves multiple dependant variables that predict one variable. The function for this is:

$$y = a_1X_1 + a_2X_2 + a_3X_3 + ... + a_{n-1}X_{n-1} + a_nX_n + b$$

Equation 6. Multiple linear regression equation

- $X_1, X_2, ..X_n$ = dependant variables
- y = predicted variable
- b = constant factor
- $a_1, a_2..a_n$ = constant factors

Similar to its 1 variable counterpart, multiple linear regression tweaks the constant factors multiplied with the variables, to find the line that best fits the data.

2.4.3 Neural Network regression

Neural networks have proven to be an invaluable machine learning algorithm in multiple areas. Their ability to understand non-linear patterns and fit the data accordingly is the main reason they are used in complex areas such as image recognition, regression and classification. A neural network (Figure 5.), has as components:

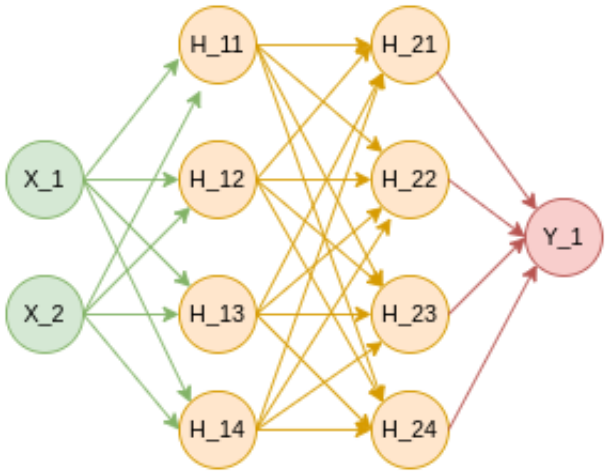

Figure 5. An artificial neural network

1. **Input layer**, comprised of X neurons (depicted in green in Figure 6). This is the layer in which the dependant variables are encoded as values (neurons). The output of this layer is connected to the input of the next layer.

2. **Output layer**, made of Y neurons (depicted in red in Figure 6). This is the layer which is represented by the predicted variable(s) (neurons). The output of every neuron in the previous layer is connected with the input of every neuron in this layer.

3. **Hidden layer(s)**, which can have an arbitrary number of neurons (The neural network in Figure 6 has 2 hidden layers depicted in yellow). The input of every neuron in the hidden layer is connected to every output of the preceding layer and the output of every neuron in the hidden layer is connected to every input of the next layer.

The value of every neuron from the subsequent layer in the network follows the linear formula:

$$X_k^{(L)} = \sigma(\Sigma_{i=0}^{n} w_{ik} X_i^{(L-1)} + b_k^{(L)})$$

Equation 7. Value of a neuron

The components of Equation 7 are:

- $X_k^{(L)}$ - value of a neuron k on layer L
- $w_{0k}, w_{1k}, .., w_{nk}$ - weights attributed to links between neurons from the previous layer and current neuron signifying the importance of a previous neuron to the current one
- $X_0^{(L-1)}, X_1^{(L-1)}, .., X_n^{(L-1)}$ - values of neurons from the previous layer
- b_k^L - bias term of neuron k of layer L
- σ - activation function (Can be a linear or non-linear function)

The most important aspect in the above function is the activation function, which can introduce non-linearity in the values of the neurons. This allows the neural network to represent much more complicated relationships than the linear regression seen before.

Neural networks learn by using **backpropagation**, a technique with which the weights of every connection of neurons and biases are changed to minimize the cost function. The cost function is usually Mean Squared Error, seen in Equation 8, but it can be set to something different such as Mean Absolute Error, Root Mean Squared Error etc.

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Equation 8. Mean squared error

The process of neural network training is the following:

For n batches perform:

1. The forward pass. Given an input the value of every neuron is computed by following Equation 7.

2. For every input in a batch perform (1)

3. After every batch perform **backpropagation**, a process in which weights and biases are changed to minimise the cost function (in this case MSE).

The **Neural Networks** used in the prediction of stars in Github repositories in Chapter 4.6 have a number X of dependant variables (i.e. input neurons), a variable number of hidden layers, number of neurons in the hidden layers and only one output neuron, the predicted number of stars.

2.4.4 Decision Trees

A decision tree is a machine learning model which makes a prediction after following a series of decisions based on the values of features in the dataset. An example of a decision tree would be:

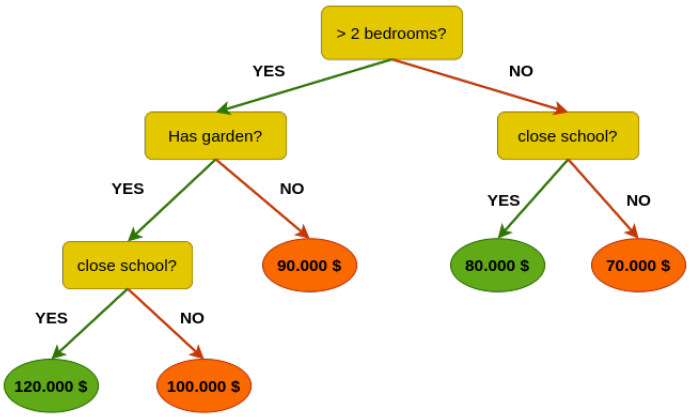

Figure 6. Example Decision Tree

Figure 6 shows an example of a decision tree of predicting house prices based on several features. The features of the dataset are used to split the data so that the split gives the lowest variance, or in other words the splits give the closest valued subsets. When a node split results in a pure split on one end (in our case if the house doesn't have a garden), in a regression decision

tree the prediction is the mean of all training data points that had the same decision outcomes. In our case, the decision outcomes are $>$ 2 bedrooms and has garden = no. If our training data consisted of 3 houses that have gone through that decision trajectory, the mean of their price would be the prediction of the next house to have the same decision splits. Thus, if those houses were 70.000$, 80.000$ and 120.000$, the prediction would be the mean of these prices, which is 90.000$. The same applies to all other nodes.

The tool with which a decision tree splits the node in the "purest" subsets in a regression task is called variance.

Variance

Variance is a measure of how much the values in a subset vary. In a regression tree, the goal is to minimize the variance within each node, leading to more accurate predictions. The variance σ of a set S can be calculated as:

$$\sigma^2 = \frac{1}{|S|}\Sigma_{i=1}^{n}(x_i - \bar{x})^2$$

Equation 9. Variance of a leaf node

where:

- σ^2 = variance
- $|S|$ = size of the subset
- x_i = value of data point i
- $\bar{x}$ = mean of all points from subset S

Decision trees choose the split with the lowest variance, i.e. the weighted sum of the variances of the child nodes.

$$\sigma_N^2 = \Sigma_{i=1}^{m}\frac{a}{n}\sigma_i^2$$

Equation 10. Variance of a node split

- σ_N^2 = variance of a decision node n
- σ_i^2 = variance of a child node
- a = number of elements in child node
- n = number of elements in parent node
- m = number of classes (in regression only 2 with < or >=)

To understand how variance is used to best split a decision tree node we will consider a small dataset of house prices and we will consider 2 splits, one that minimises the variance of the split and one that does not.

m^2	**House Price**
200	300.000 $
250	350.000 $
300	400.000 $
400	550.000 $
450	600.000 $

Table 1. House prices dataset

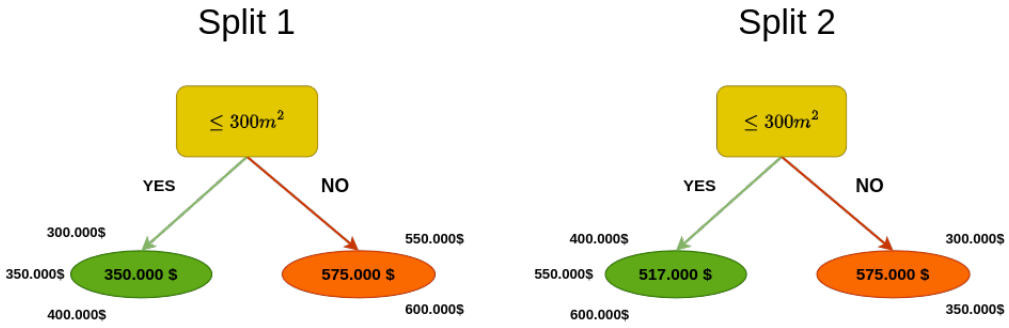

Figure 7. Decision tree splits: Split 1 minimises variance and split 2 is a split that does not minimise variance.

By looking at figure 7, we see that the values in both parts of split 1 are closer together/have less variance than the values in split 2. This is

preferred in regression as when predicting a value we want to be as close as possible to the actual value. The calculations of the variance of split 1 and split 2 are shown in appendix A.

2.4.5 Random Forest regression

Random Forest is an ensemble learning algorithm, that is used for multiple tasks such as regression and classification. It combines the results of multiple decision trees to arrive at a final result. In the case of regression, the final outcome of the random forest for a prediction is the mean of all the results of the decision trees that have had a similar decision path. Random forest functions by taking a sample of a dataset by replacement and training that subset on a tree, to reduce over-fitting. Sampling by replacement refers to putting a data point in the dataset more than once.

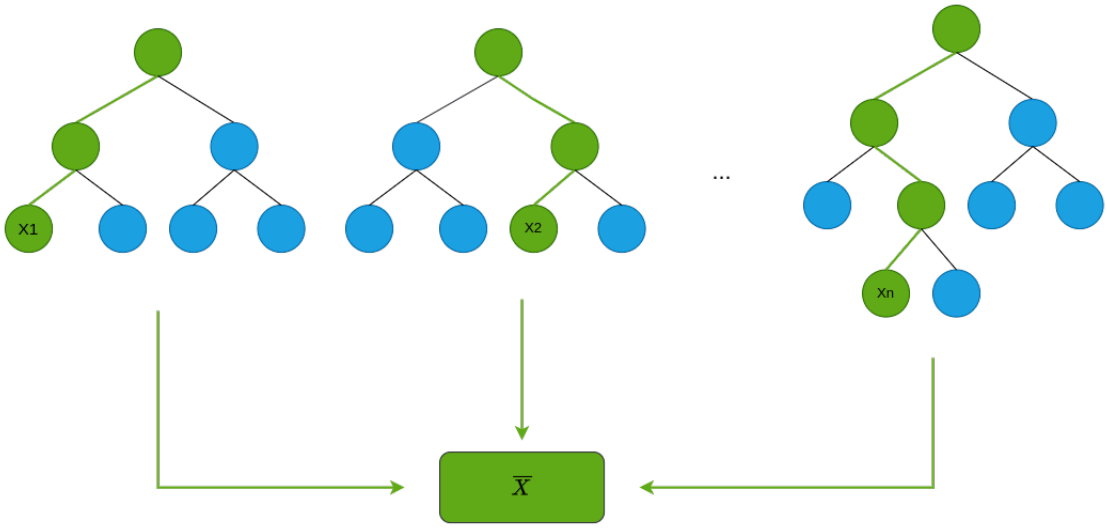

Figure 8. Generic Random Forest Regression

Figure 8. shows the structure of a random forest, with the result of the random forest being the mean of all predictions of the decision trees in the forest.

It is essential to tweak the parameters of the random forest by providing parameters like:

- maxDepth = the maximum depth of a decision tree in a Random Forest,
- nrEstimators = the total number of decision trees in a Random Forest,

- maxFeatures = the maximum number of features that a decision tree can use to split nodes,

which can help in fitting the data just right.

Chapter 3

Related Work

The topic discussed in this paper, evaluating a repository's popularity has been explored numerous times and from different perspectives. This chapter is dedicated to explaining what others have done in the field and some of the methods applied in the works discussed below are used in this paper.

Borges et al., 2016 [4], conducted similar research to this paper and they elaborate on the following factors as being proxies to popularity in Github.

1. programming language/application domain/repository owner
2. repository characterstics:
   (a) age
   (b) number of commits
   (c) number of contributors
   (d) number of forks
3. How early do repositories become popular?
4. impact of new features

They first conducted a correlation analysis of the repositories between programming language, application domain and repository owner and stars and they arrived at the conclusion that all of the aforementioned are indicators of popularity. This is because popular programming languages make the projects built in them more desirable to be viewed, application domain

also depend on the current trend and it has been discovered that repositories of organisations have more stars than individuals. However, the mean number of stars is not significantly higher for repositories built by organisations than for the ones built by users (37% for orgs and 33% for users). The second step in their analysis was to uncover whether commits, forks, age and contributors correlate to the number of stars. Apparently, age is not related to the number of stars, commits and the number of contributors are weakly related and the number of forks are greatly related. Furthermore, for most repositories the biggest surge in popularity happens after the first release and then gradually becomes constant (51% age - 50% stars, 91% age - 90% stars). They also examined the impact of new features on the number of stars. Only projects with versions x.y.z, with x = major release, y = minor release, z = bug fix were considered and it follows that the highest spike in popularity comes with major releases and medium growth is attributed to minor releases. The last step in their paper involved a time series analysis using the KSC algorithm on releases and how they influence popularity. This experiment concluded in finding 4 types of projects in terms of pace of growth. It appears that 65.7% of projects have slow growth (27.3% of stars in one year), followed by 26.9% that have medium growth (96% stars in one year), 5.7% have fast growth (469.2% stars/year) 1.9% have rampant growth (2673% stars/year). They concluded that some programming languages have a higher growth spurt than others, organisations do not have a significantly higher number of viral projects than individuals, and that older projects tend to get smaller growth than newer projects.

Borges et al. conducted 2 more studies, [5] which uses Multiple Linear Regression as an algorithm to predict the number of stars of the top 5000 repositories with at most 40K stars and by predicting the rank (where the repository falls in the dataset in terms of number of stars) of repositories that have been clustered based on a similar growth pattern using KSC. and [6], which goes in-depth into what a GitHub star actually signifies, by surveying developers that starred some repositories (1500 repositories), and they concluded that a star means 3 things: starring is appreciation towards a project, it is used as a bookmark, and some developers are starring because they are using that repository.

A. Al-Rubaye et al. [3] discuss another measure of assessing popularity on GitHub (WTPS), Weighted total popularity score, which involves a weighted sum of the number of stars and the number of forks. With this metric, the stars have a correlation of 0.925 and the forks have a correlation of 0.726, which indicates that stars are better suited for assessing popularity. They also experimented with making a graph of repositories and their

followers, with the size of a node being the popularity measure (stars, forks, WTPS, watchers) and seeing how fast the clustering coefficient is reduced based on the deletion of the nodes in decreasing popularity order. Because WTPS is more influenced by stars, WTPS and stars had lower clustering coefficient after a number X of deletions than the other 2 metrics, which indicates that they are more suited for assessing popularity.

Ren et al. [4] hypothesize that the influence of a stargazer based on their number of followers also affects the number of stars of a repository. This is intuitively correct because when someone you follow stars a repository, you will most likely be recommended that repository. They calculate a user's influence using HFN (the number of followers a user has), the NDF (Network Dynamic Factor) and the NSF (Network Static Factor). The user's influence at a certain point in time is calculated by a weighted sum of NDF and NSF, where NDF is calculated by taking into account all nodes and links in the network (developers are expected to have an effect on the followers of their followers) and the NSF (only followers or a small area of followers of followers is taken into account) is calculated by only taking into account an area of the network in the vicinity of the target node. The researchers evaluated StarIn with respect to the programming language and application type and found out that regardless of the system and language, stargazer influence does affect the overall popularity of a repository.

The study of Bidoki et al. [7] introduce the use of LSTM in predicting GitHub stars over time. Because LSTMs are very good at remembering relationships over time, they are favorable in time-series analyses. Also using LSTM's, Sahin et al. [6] attempted to predict stars, by first trying out stars as input to predict stars x days later, where x $\in$ {1,7,14,28} days, then trying to predict whether repository data such as commits, nr of resolved issues, forks, releases at different time intervals can predict stars x days later, where x $\in$ {1,7,14,28} days, and after that attempting to predict whether contributor activity such as commits, forks, stars, issues closed in an interval of time can predict stars. The models that used the repository attributes as input performed better than the ones that used developer attributes. Sajedi et al. [5] emphasized that a large following does not necessarily equate to popularity; instead, the frequency of content redistribution or repurposing is more telling. Yan et al. [6] expanded on this idea by suggesting that a user's influence is not only determined by their involvement in highly starred, forked, or watched projects but also by their activity levels on GitHub, measured through commits, issues, and discussions. Börstler et al. [7] investigates developers' perceptions of code quality and their experiences discussing it. The study includes insights from developers using GitHub for code review

and collaboration, highlighting how discussions around code quality impact software development practices. Lastly, Ziegler et al. [14] conducted a case study into how GitHub Copilot (the AI code auto-completion tool) has impacted the productivity of developers. The findings indicate that using GitHub Copilot enhances coding efficiency, reduces frustration, and allows developers to focus on more enjoyable tasks.

3.1 Comparison with related work

This thesis presents a structure similar to the study of Borges et al. [4], in that it provides a correlation analysis on some same variables (programming language, repository owner, forks, watchers), however, this study also used additional features in that regard: license type and the ratio between closed issues and all issues. Another similarity between this project and the aforementioned paper lies in the analysis of growth over time. As differences, this study zoomed in on the effect of releases on the number of stars by looking at separate repositories and identifying possible patterns that could arise after and during releases, it provides a comparison between counting the stars during the week with a release and 1 week after the release and it uses different models when predicting the number of stars and different inputs are tried on the models. In the popularity prediction section, it is similar to the study of Sahin et al. [10], which also tries multiple inputs for the machine learning model, however the inputs of this research's models differ only by either incorporating the type of releases for every week in the last 3 weeks or not, which aimed to discover whether GitHub releases matter in the number of stars gained by a repository per week. It also differs to [10]'s study by the machine learning models used. All of the studies presented tried only one machine learning algorithm in predicting the number of stars, while this research presented 3 different machine learning models. Interestingly, none of the studies presented incorporate feature importance in their research to validate the results of their machine learning models.

Chapter 4

Research

This chapter delves into assessing the factors of the most popular repositories, correlating popularity with different factors, understanding repository growth patterns, assessing the impact of releases on the number of stars and popularity prediction. It provides a framework to how such an analysis is conducted by completing the following steps:

1. After gathering the data, an outline of the main features of the dataset is presented, i.e. the distribution of programming languages, number of stars, forks, watchers, contributors and age.
2. Defining the popularity of a repository
3. Correlating repository attributes and popularity
4. Understand repository growth patterns
5. assess the impact of releases on the number of stars
6. Predicting the popularity of top GitHub projects with < 40K stars

4.1 Dataset

There are multiple frameworks for mining GitHub data, however, I discovered that the best suited framework in this scenario is PyGithub, as it is the most complete GitHub mining framework with Python support. PyGithub contains all the necessary information about repositories, users and others, but when gathering events such as star timestamps and fork timestamps, regular requests to the Github API are used. To begin with, the

top 2690 most starred GitHub repositories have been gathered. The data has been gathered in the span of 2 months, from March 2024 to May 2024. The GitHub repositories' overall data has first been mined which included the number of stars at that point using PyGithub, after which all the stars events were mined using the GitHub Events API based on the number of stars the repositories had when they were first collected. The releases of the projects are also gathered (also via the Events API) up until the day of the last star (23 March 2024). Every repository in the dataset has all the attributes from Table 1. In the growth, releases and time-series prediction parts of the thesis, (Chapters 4.5, 4.6) repositories with $>$ 40k stars are not considered because of GitHub API limits (in the time-series analysis part of the project, timestamps of star events are needed, which are provided by GitHub API. However the API does not allow the user to get more than 400 pages of data, with every page containing at most 100 star events. This then limits us to 2390 repositories with under 40000 stars). The dataset gathered contains the following features:

Dataset Features	Explanation
Path	The owner and repository name concatenated as 'OWNER/NAME'
Stars	The number of stars the repository received at the time of retrieval
Forks	The number of forks the repository has at the time of retrieval
Watchers	The number of people that subscribed to the repository and will be notified by any change to the repository
Owner Type	User or Organisation (Linus Torvalds / Facebook)
programming language	most used programming language in the repository
license type	the type of license of the repository
number of contributors	how many developers actively worked on the repository
age	the number of months since the repository's inception
issues solved	the number of tasks solved
total issues	the total number of tasks created
releases	A list of all releases of the project of the form: release name: x.y.z, timestamp: YYYY:MM:DD HH:MM:SS
Stars events	The stars events of the repository received at the time of retrieval, of the form: timestamp: YYYY-MM-DD HH-MM-SS

Table 2. Dataset features

This section will go over surface aspects of the dataset, i.e. attributes which make us get an idea about what we should expect from the repositories, which include:

1. the distribution of programming language in the dataset
2. the distribution of age, forks, stars and watchers
3. number of contributors working on projects from the dataset

Programming language is intuitively a big influencer of popularity, as one would assume that the most popular languages would have the most popular projects. When looking at the proportion of programming languages (Figure 9) , the top 3 are clear winners: JavaScript, Python and TypeScript. This is attributed to the popularity of the languages and also of frontend

development.

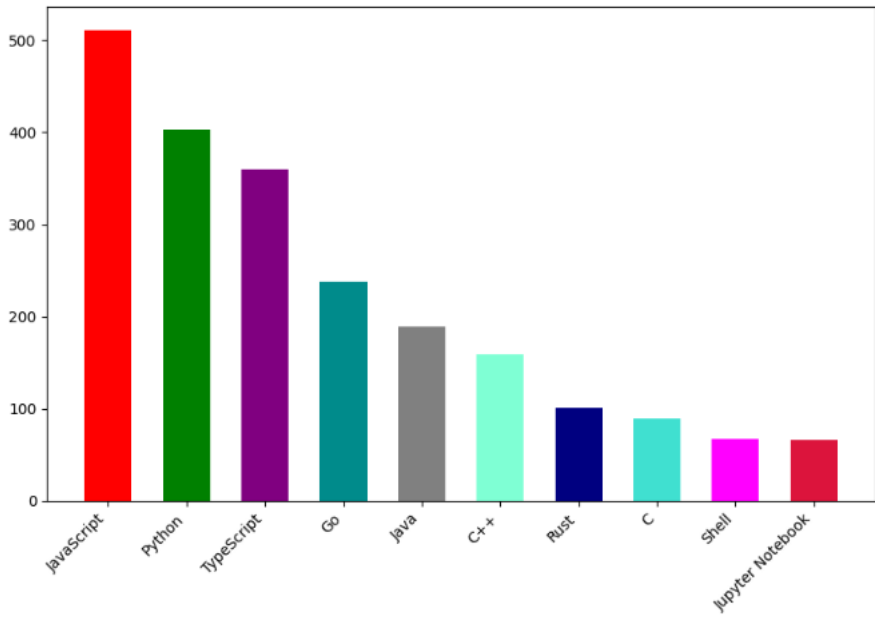

Figure 9. Programming language frequency in the top 2690 most popular repositories

The mid-field is represented by systems languages: with Go having 238 projects, Java having 180, C++ 159, Rust 101 and C at 89 projects. Go has been gaining traction lately as one of the most popular systems language with projects like Go repository, awesome-go - a list of Go frameworks and libraries and Kubernetes the most used container orchestration tool. Heavyweights like Java which is perfect for cloud computing and systems programming, C++ which is used from image and audio processing to game development and machine learning, Rust which has been voted the developer's most loved language for the 8th year in a row because of its great documentation, smart syntax and error handling and C should not be overlooked as they are used in many more fields than the "frontend" languages. The bottom 2 languages used in the dataset are Shell and Jupyter notebook. Shell is a popular scripting language and Jupyter notebook is mostly used in Machine Learning projects, which is a very hot topic these days with the surge of AI.

Other aspects to consider when examining the dataset are the age of a repository (in number of months), nr. of contributors, nr. of watchers, nr. of forks, nr. of stars. Older repositories have many more releases than more recent ones, which counts as possible stages where developers could have heard about them and thus are more likely to be popular. The number of contributors can vary as there are very popular projects like '**freeCodeCamp/freeCodeCamp**' with over 5000 contributors and '**facebook/react**' with 1600 contributors, but they can also have few contributors: '**cloudflare/pingora**' with 8 contributors. Intuitively, successful repositories have the most watchers, forks and stars.

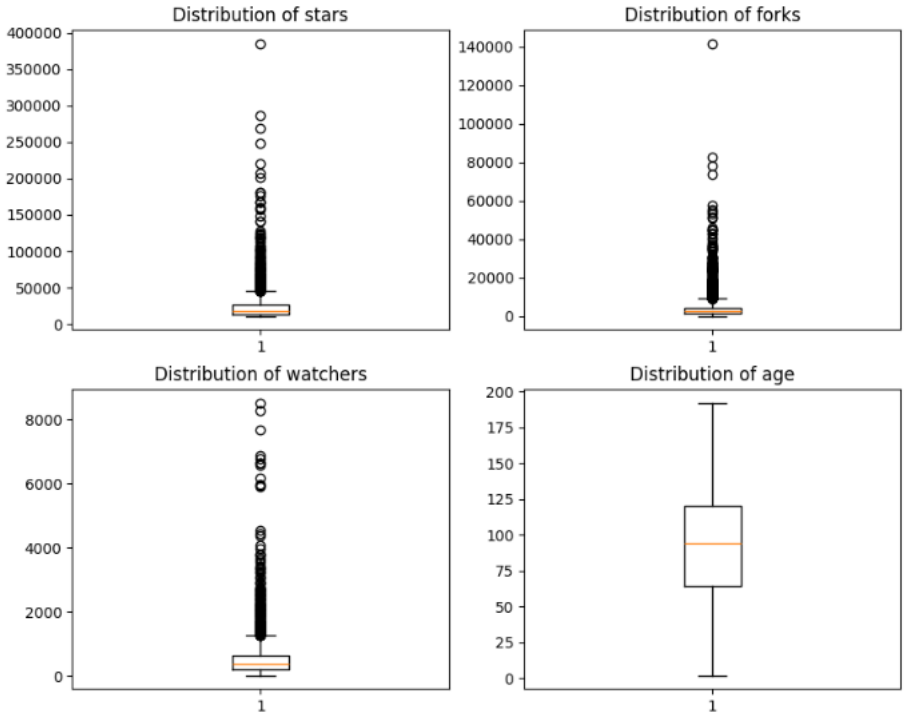

Figure 10. Distribution of stars, forks, watchers and age of repositories of the dataset.

As seen in Figure 10, the dataset contains repositories with stars ranging from 11045 to 384890, forks ranging from 82 to 141563, watchers ranging from 21 to 8499 and ages of repositories from 2 months to 192. The overall statistics can also be seen in Table 3.

	Minimum	1st quartile	Median	3rd quartile	95th percentile	Maximum
Stars	11045	13527.5	17437	26128.5	59078.5	384890
Forks	82	1330	2410	4477.0	12438	141563
Watchers	21	200	357	627	1572.5	8499
Age (months)	2	64	94	120	159	192
Contributors	1	50	137	325	1167	20722

Table 3. Overall statistics of the repository

From Table 3 it is clear that the majority of projects contain 12500-27500 stars, 1000-5000 forks and 175 - 700 watchers, and that outliers (5% of the dataset) happen above 59000 stars, 12438 forks, 1572 watchers. This indicates the rareness of highly influential repositories. The number of stars is the biggest indicator of popularity since that is how repositories get recommended to other developers. However, forks are also closely tied to popularity as they are repositories that are cloned from the original one and point to the source project. If someone views the forked repository the chance they look at the original are high and if he stars the fork he will also star the original. Watchers on the other hand represent developers that are notified of every change in the repository, but they are not contributors. Objectively, one can think of watchers as developers thinking of becoming contributors, which is pertinent to the success of the repository as the more influential the repository the more people are willing to contribute. The median age of a repository is 94, the 1st quartile at 64 and 3rd quartile at 120 indicate that the probability of a repository being well known is much higher if it has a longer lifetime. Nevertheless, there are exceptions in the dataset, with the minimum repository age in this dataset is 2 (months), and there are 53 repositories that are less than 12 months old. This is because some projects experienced viral growth since their inception. The number of contributors can also be a sign of popularity of a repository, as the more known and liked a repository is, the more chances a developer will want to help with it. The box plot below, and the metrics in Table 3., show that the majority of popular projects lie between 40 and 400 contributors.

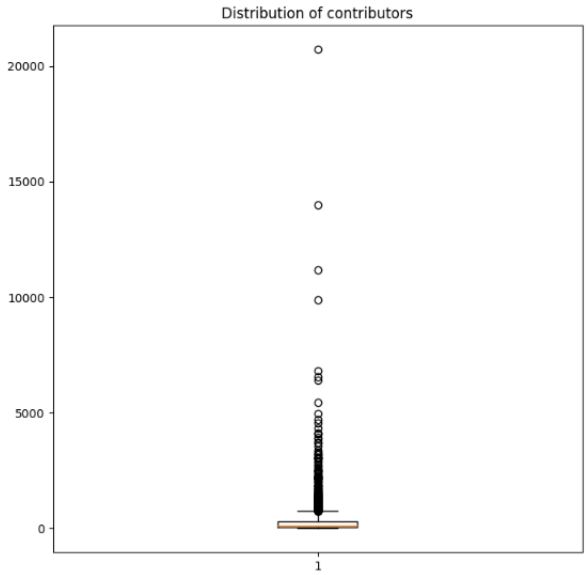

Figure 11. Distribution contributors per programming language

The number of contributors found via the API is 10% larger than the metrics seen on GitHub as GitHub counts as contributors only people with necessary permissions: i.e. developers with direct push access, commiters and issue openers. GitHub API has a more broad horizon and it also counts developers that have attempted a pull request.

The above plots and data tell us the following about the dataset:

1. the mass majority of projects are mature (median lifetime is 94 months (7.8 years), 1st quartile indicates 64 months (5.3 years))

2. most highly popular repositories have ≥ 50 contributors (75% of projects). This shows that in order for a project to be highly successful it is easier to have a lot of developers to work on it.

3. all of the projects (in the dataset) have at least 82 forks and 21 watchers and 75% of all repositories have > 1330 forks and > 200 watchers. This indicates the usefulness of the projects in the dataset.

4. minimum number of stars is 11045 and maximum of 384890, which means that the projects in the dataset have a massive fan base.

5. the most popular languages by a landslide are the front-end JavaScript, TypeScript and the most popular language Python, which is used in

many cases but most notably nowadays for machine learning. The next are the back-end languages Go, Java, C++, Rust and C, languages known for their performance. The last are Shell and Jupyter notebook, with Jupyter notebook being synonymous to Python and Machine Learning.

4.2 Defining popularity

As discussed in the introduction, popularity is a construct that can be measured in various ways. In GitHub, one can measure popularity by looking at the number of stars, watchers, forks, the dependency graph of a repository's followers etc. However, for a quantitative analytical approach, it is necessary to devise a metric for measuring popularity in GitHub. The related work provides sensible reasoning when devising popularity measures, and they also avoid GitHub's limitations by doing so. Most studies chose the number of stars as their main quantifier [4][5][15], however, there were also works that considered the weighted sum of stars and forks [1]. Watchers are mostly ignored because GitHub API does not provide timestamps for watchers and thus a time-series analysis is impossible. After careful consideration, it has been decided that stars will be the main popularity indicator, since they reflect it the most [6]. Even though forks and watchers also depict the success of a repository, they are highly correlated with the number of stars and thus it was decided to avoid them.

4.3 Correlating repository features and popularity

This section aims to answer RQ2: "How do features like programming languages used, type of license, repository owner influence the number of stars?". The question will be answered by conducting a correlation analysis.

We have seen in the dataset section that there are groupings in the frequency of projects in the dataset with JavaScript, Python and TypeScript at the top, followed by back-end languages Go, Java, C++, Rust and C as midfielders and Shell and Jupyter notebook as last. To find out whether there is a relationship between programming languages and popularity, a statistical significance test is conducted. Before testing, it is necessary to define the null hypothesis and the alternative hypothesis. In this instance, the null hypothesis refers to programming languages not being related to popularity, as there is no difference in the data, and the alternative hypothesis suggests that there is a link between programming languages and repository success. Since the data involves multiple samples (multiple programming languages and the number of stars of the projects), the Kruskal-Wallis test is chosen. Kruskal-Wallis calculates the statistical significance by comparing the means of the samples. It assumes a similar distribution, equal sample size and independent observations. Naturally, since the dataset contains more projects for some languages than others, the samples compared in the test only considers projects at regular intervals for each language so a similar distribution (Fig. 12) is achieved and the outliers are also removed. Conducting the test results in a significance level of $0.0007 < 0.001$, which is considered to be highly significant. Looking at Figure 13. there are programming langauages with a much higher mean than others, for example Rust, which has a mean of more than 3000 than the language with the 2nd largest mean, which indeed proves that programming language matters in popularity.

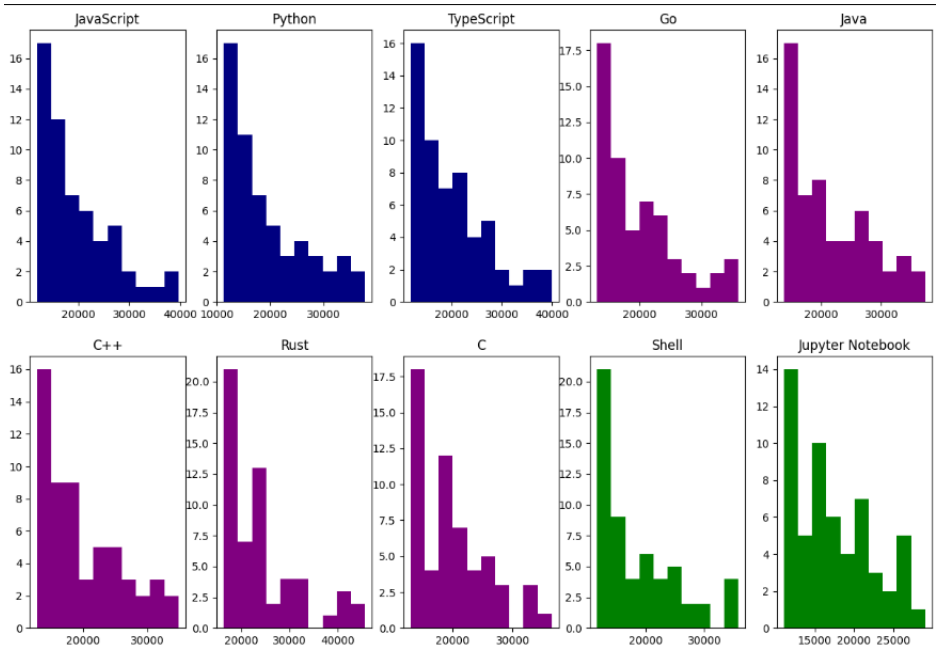

Figure 12. Distribution of stars and programming language

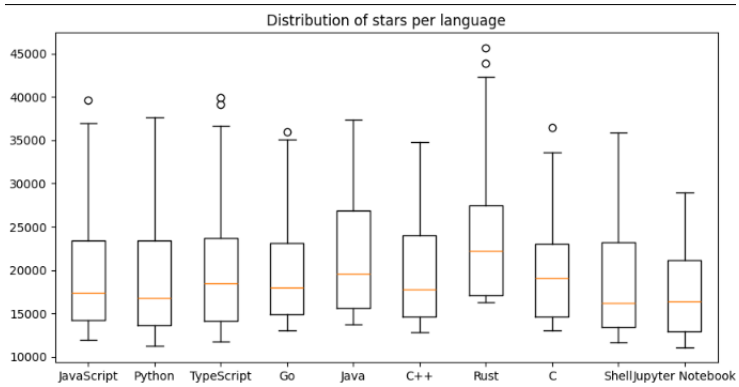

Figure 13. Box plots of stars and programming language

The second aspect to be examined is license. The license specifies the permissions that the viewer of the repository has. Most notable are:

1. MIT license, which allows the user to use the code and can also distribute it for commercial purposes, but the developer must include the original license in their project.

2. Apache License 2.0, that lets the viewer use source code and distribute it for commercial purposes, however, the developer must include the

original license and all the modifications that have been done to the original product.

3. GNU General Public License v3.0, is a copyleft license which states that the source code can be copied and modified, but it demands that user also open-sources the code.

Because of the permissions alloted to the licenses, some are more used than others. As the MIT license is the most permissive, it is also the most utilized with 1205 projects, 2nd most used license is Apache at 538 and the 3rd is GNU General Public License with 156. Other licenses include BSD 3-Clause "New" or "Revised" License (79), GNU Affero General Public License v3.0 (77), GNU General Public License v2.0 (35).

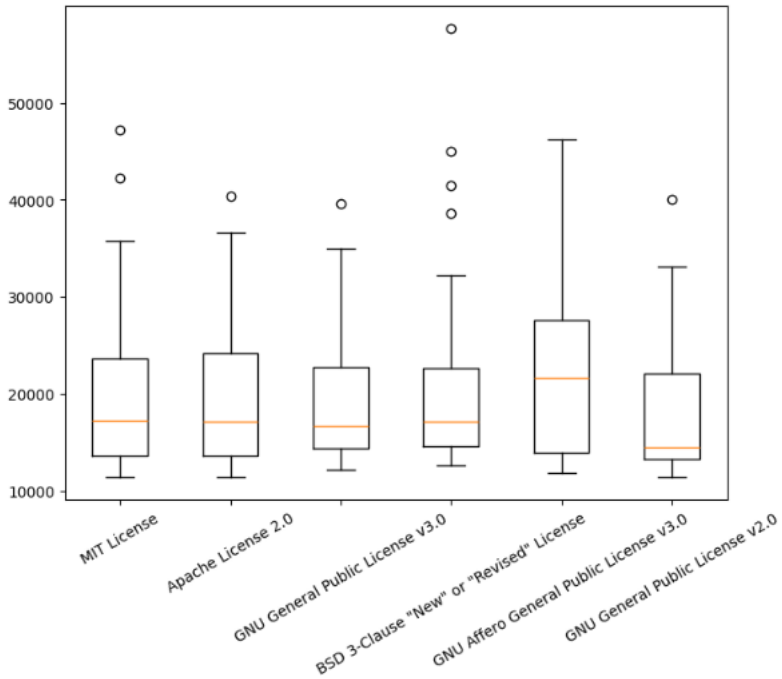

Figure 14. Licenses boxplots and stars

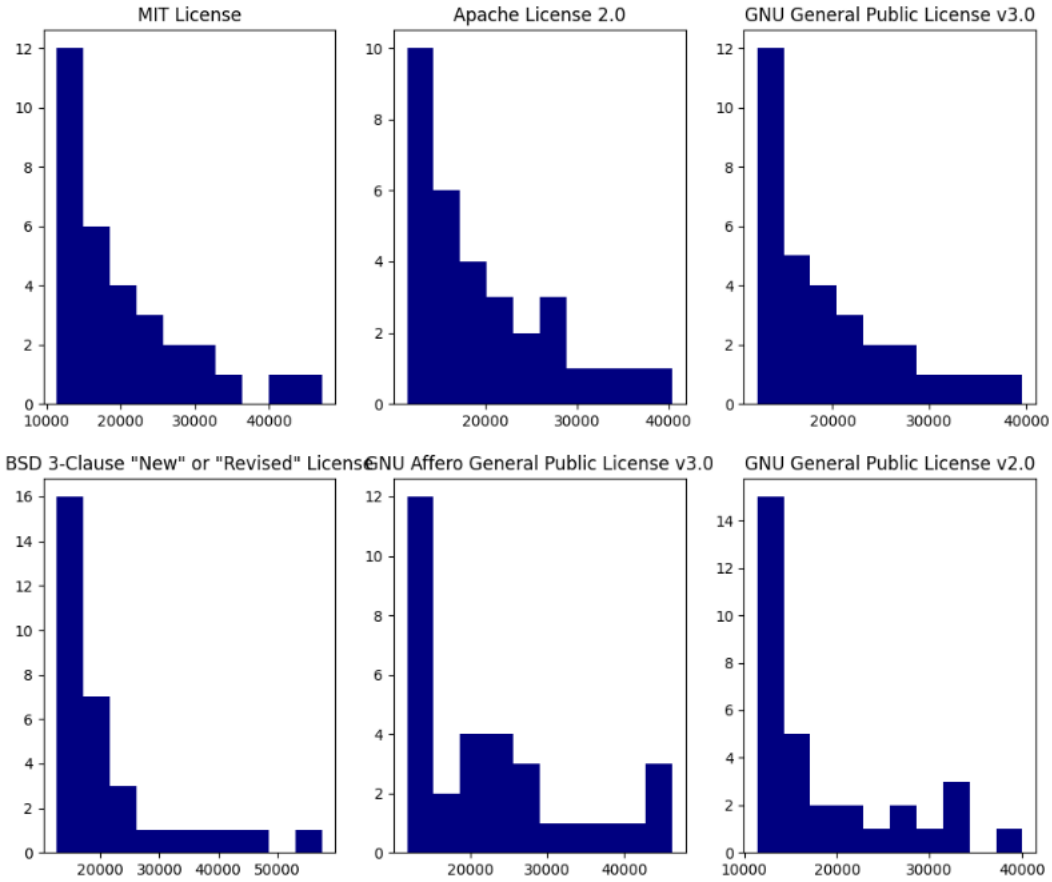

Figure 15. Stars per license distribution

After performing the same test as for programming language, the p-value for licenses is $0.3 > 0.05$, which is considered statistically insignificant. Thus, the type of license does not play a huge role in project popularity. However, as mentioned previously, there are more projects with more permissive licenses in the dataset than restrictive ones, which may hint that using a more permissive license is best when posting your project on GitHub and you want more followers.

The last important repository feature left to analyse in this chapter is owner type. By owner type it is meant either user or organisation. A pre-analysis assumption would be that repositories of organisations have more stars than ones made by users. This stems from factors like: higher budget, larger teams, marketing which all lead to more well-known projects. The

box plot in Fig. 8 shows us the distribution of stars for projects under Users and Organisations.

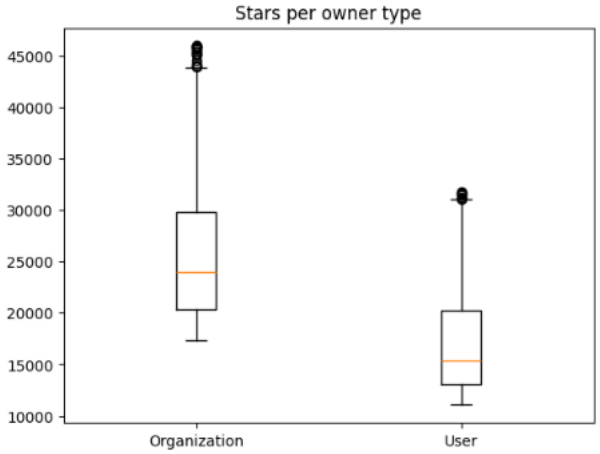

Figure 16. Stars per owner type

The plot depicts organisations having a much higher mean of stars than users, and the lowest nr of stars of an organisation's project is larger than the user's mean. This clearly indicates that owner type plays a role in popularity. To test this statistically, the Mann-Whitney test will again be conducted. The result of the test is ≈ 0, which is very satisfactory. Thus, owner type also plays a role in a repository's success.

To conclude this section, of the 3 features discussed: owner type, license and programming language, only license does not contribute to a repository's popularity. Programming language does play a role as choosing a popular language may give you a boost in followers. Organisations are also seen to have many more followers than users in their repositories due to more resources.

4.4 Repository actions and their influence on success

The goal of this section is to discover whether social coding aspects such as the number of commits, pull requests, issues solved result in influencing popularity. To prove whether the aforementioned aspects impact repository success, a correlation analysis will be conducted.

Commits in git are essential as this is how new checkpoints are added to the repository. Through commits developers advance to different stages in their development and will utimately help them arrive at the final product. However, it is not perfectly clear whether the number of commits will ultimately influence the popularity of a repository. This stems from the following:

1. Commits may simply represent code refactoring as in rewriting
2. Some commits may be redundant as they come from branches that haven't been merged to the final tree.
3. Intuitively, the number of commits does not reflect popularity as a very popular project can have few commits.

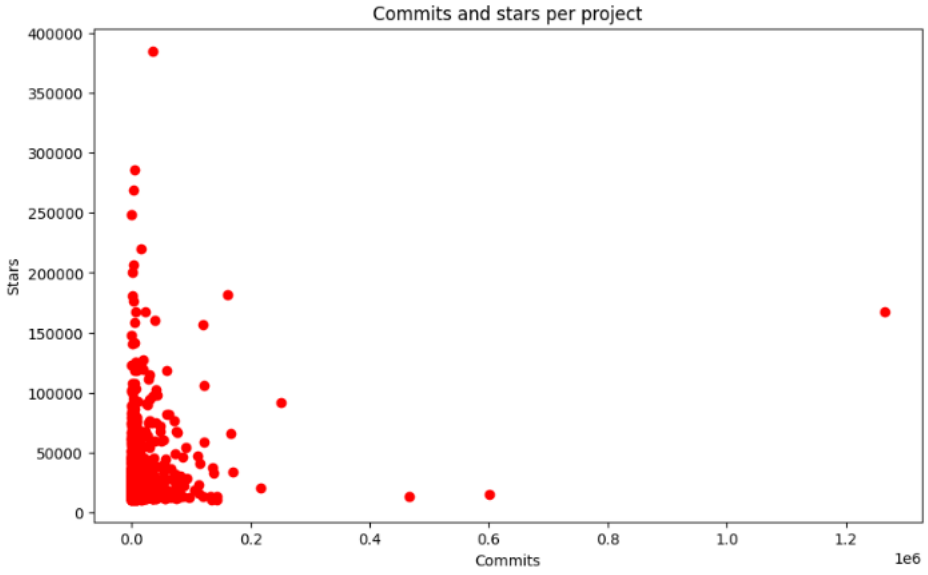

Figure 17. Commits and stars of projects

The plot in Figure 17. proves that there is a small correlation between commits and number of stars. Pearson's correlation coefficient calculated between commits and stars also indicates a correlation of ≈ 0.2. Therefore, a larger number of commits does not contribute to the overall number of stars.

Forks on GitHub represent a more sophisticated form of cloning, as they enable the forker to propose changes to the original repository through pull requests. Furthermore, forks maintain a connection to the source repository, potentially enhancing the visibility and popularity of the original project. This is because developers exploring a fork are likely to examine the original repository as well. Furthermore, forking a repository essentially means using the repository, which is a direct translation to popularity by adoption. Consequently, it is hypothesized that there exists a significant correlation between the number of forks and the number of stars a repository receives.

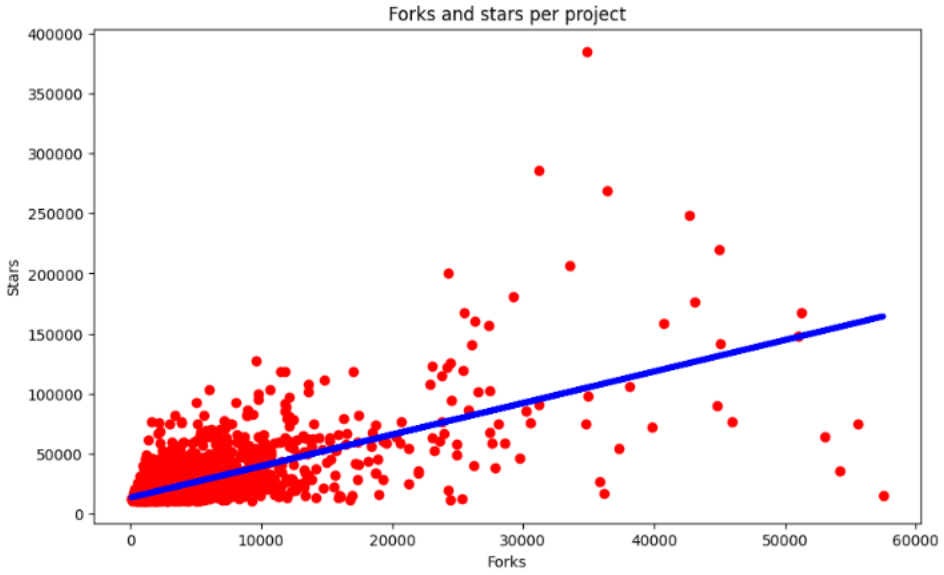

Figure 18. Forks and stars

Figure 18. shows the plot of stars and forks of projects. The plot suggests a positive correlation of 0.67 which proves that a higher number of forks does indeed most likely mean the repository is popular.

The next attribute of a repository to be examined in this chapter is watchers. Watchers represent developers that want to be notified of changes

in a repository [18]. This interest may also lead to becoming a collaborator, which can indicate the impact a repository has on its users. Because of this exact reason, the number of watchers may mean that a repository is impactful.

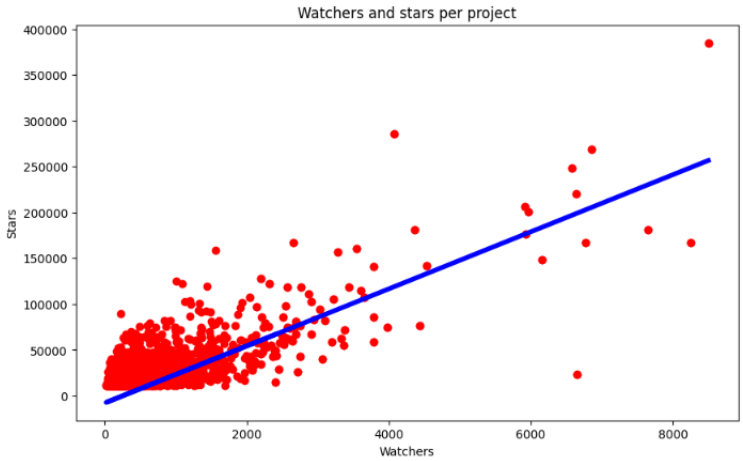

Figure 19. Watchers and stars

Figure 19. shows the plot of stars and forks of projects. The plot suggests a strong positive correlation of 0.76 which proves that a higher number of watchers most likely means the repository is popular.

Issues in Github are essentially to do tasks. They allow developers to keep track of what to work on next and they are also assigned priorities so that developers know the urgency of the task. Opening an issue means creating a task and closing one means that that task has been completed or has been canceled. To evaluate whether issues contribute to the number of stars, the ratio between closed issues and all issues for a project will be plotted against the number of stars.

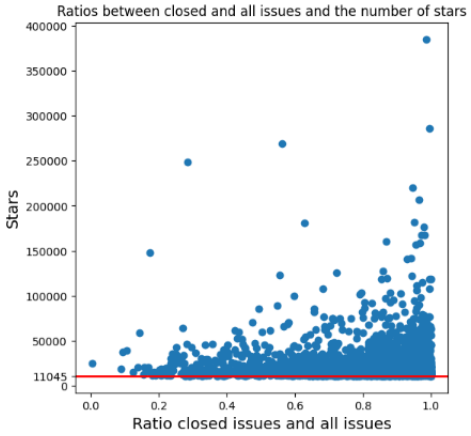

Figure 20. Ratios closed/all issues and number of stars

From figure 20. it is visible that there is no correlation whatsoever between the ratio of closed and all issues and number of stars. The Pearson correlation coefficient on this data is equal to 0.06, which confirms the insignificant correlation depicted by the plot.

4.5 How do repositories grow over time?

The questions regarding when repositories become popular and what influences repository growth are crucial in understanding GitHub projects. Firstly, looking at the proportion of projects that reach a certain threshold of stars after different periods of time will help us gain insights in the types of growth of the projects from the dataset. Moreover, looking at how releases shape weekly repository growth is also necessary to understand possible patterns.

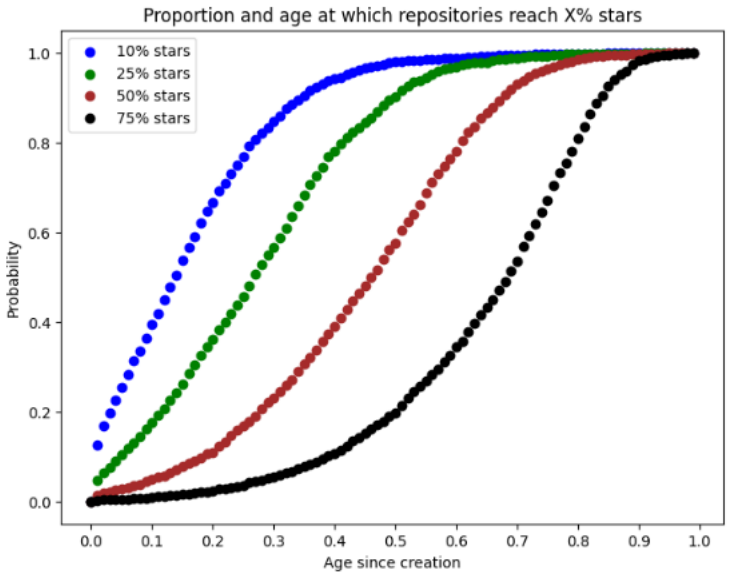

Figure 21. Age and probability when repositories reach 10/25/50/75% stars

The plot above (Figure 21.) presents a view of when and how likely a project in the dataset is to have 10%/25%/50%/75% of the total stars. In the dataset:

1. 40% of all projects reach 10% stars in the first $\frac{1}{10}$ of their lifetime and 90% of projects have 10% of their stars by $\frac{4}{10}$ of their lifetime. The latter indicates repositories that grew late. It is also interesting to note that there are 12% of projects that reach 10% of stars within $\frac{1}{100}$ of their lifetime and 22% of projects that reach 10% of stars within $\frac{4}{100}$

of their lifetime. This proves that there are early growing repositories in the dataset.

2. 50% of projects reach 25% stars by $\frac{1}{4}$ of their age, which highlights an average pattern and 90% of projects have 25% of their stars by $\frac{6}{10}$ of their lifetime, which shows that there are 10% repositories that gain 75% of stars in the 2nd half of their lifespan. Moreover, 17% percent of projects have 25% stars by $\frac{1}{10}$ of their age and 10% of all projects got 25% star by $\frac{1}{20}$ of age. This also denotes a large proportion of initially viral projects.

3. 5% of repositories gain 50% of stars in $\frac{1}{10}$ of their lifetime, 57% of repositories get 50% of their stars by $\frac{1}{2}$ age and 99% of repositories reach 50% of their stars at 90% of their age. The first finding indicates that there are repositories that have a massive boost at the start, but their growth is slow afterwards. There are more repositories that have more stars in the first half of their lifespan than the second. This is of course attributed to the hype over the project/technology.

4. 1% of all projects reach 75% stars in $\frac{1}{10}$ of their lifetime, which points to viral repositories in their inception, and this marks repositories that have lost momentum afterwards. 20% of projects reach 75% stars by 50% of their lifetime, 66% reach 75% stars by $\frac{3}{4}$ of their age.

The reasons for a repository's popularity are diverse: mention of a repository by a popular developer/youtube developer/blog post, new release, uniqueness of the project etc and multiple other aspects discussed even in this paper. However, due to this project being about analysing GitHub repositories only information that can be mined through Github API is used. Thus, the next subchapter will focus on the factors that are believed to influence the trajectory of a GitHub project the most, and that is: releases.

4.5.1 Releases and their effect on popularity

Releases represent a crucial aspect in the lifecycle of any application, and in GitHub it is no different [11][19]. They can bring new followers and/or older followers that forgot about the project and this directly relates to the number of stars. As new releases bring in new features, developers are curious to know what these are. There are several assumptions of how different versions affect the project:

1. Major releases and few subsequent minor releases impact popularity the most as this is the period when users try out the improved software. Thus, major releases or close minor changes typically reside on peaks of popularity.

2. Major releases may happen in periods of low popularity but after a period of time and maybe some small changes the repository will find itself on a spike.

3. When there are periods of inactivity in the project (i.e. no more changes at version level) and a new release is made, this will no doubt bring more people to the repository and create spikes of popularity.

4. After a spike of growth either because of a major or minor release, the trend will stabilise and can also decrease.

5. In most cases, more releases bring more stars, but there may be an increase in stars even when no changes are made.

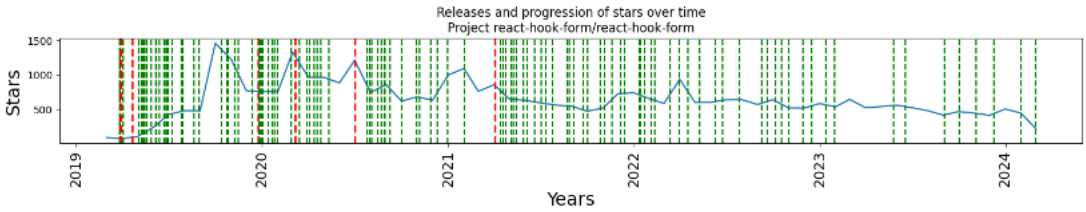

Figure 22. React-hook-form project

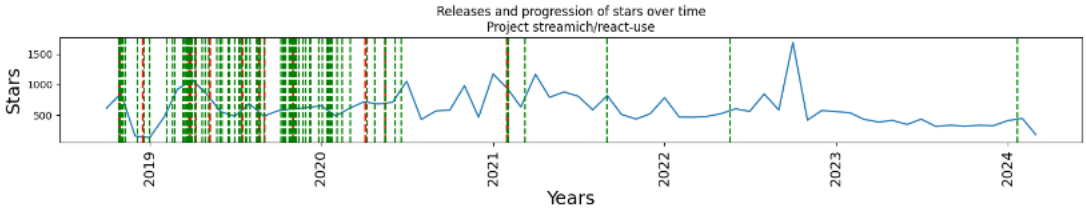

Figure 23. React-use project

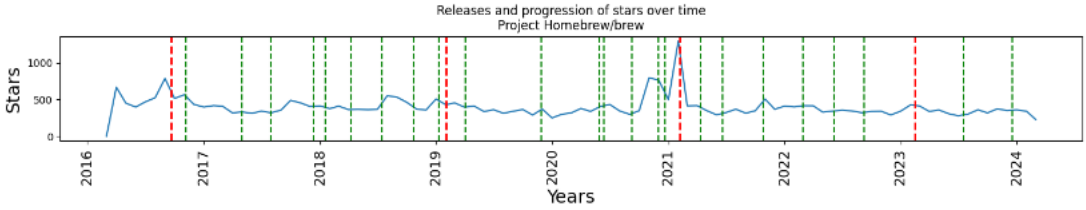

Figure 24. Homebrew project

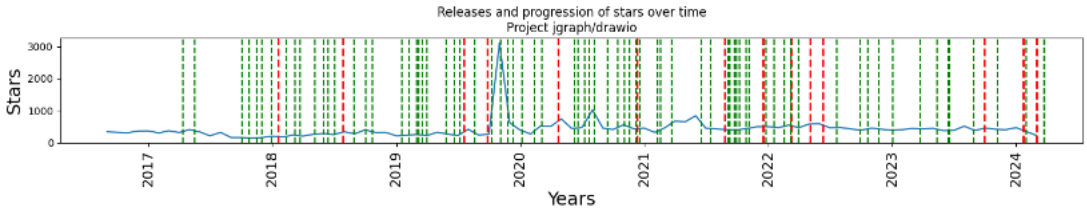

Figure 25. Drawio project

Figures 22,23,24,25 represent repositories under 40K stars and their growth over time (every week). The red dotted lines represent major releases and green lines represent minor releases. As seen in the plots, releases have various behaviours to the stars. It is the case in all of the above repositories that assumption 4 is true. No repository can maintain the same state or experience continuous growth for extended periods of time. Assumption 5 is also true, with repository 'React-use' (proj 2 - Fig. 23) showing this behaviour. This behaviour is also attributed to the fact that some repositories simply gain followers by what frameworks are also popular and React is a very popular one. This is also proven by the study of [4], that suggests that web frameworks and libraries are the 2nd most popular application domains after systems.

Projects 1,2,3 (fig.22,23,24) are best described by assumptions 1 and 2, with peaks happening at major changes, but sometimes popularity also hits a low and a major release or subsequent releases increase the trend again. They also present pauses in development after which when a release is made a boost can be seen, thus exhibiting the behaviour of assumption 3. Project 4 (fig 25. "Drawio") presents the same behaviour as the other 3, however it has a massive spike of growth after a major release, which again indicates their importance.

4.5.2 Statistical evaluation of the impact of releases on the number of stars

After gathering all the star events for more than half of the repositories (1277 repositories), it is now time to discover the degree to which releases count in weekly star growth. First, the repositories that presented a clear versioning system were selected, more specifically of the form x.y.z, where x is the major release, y is a minor release and z represents a bug fix. Only

releases of the form 'x.0.0' and 'x.y.0' are chosen and labeled with 'M' and 'm' respectively. Because the interest is in the degree to which releases account for weekly star growth, it was decided that:

1. if there are multiple releases in the same week and there were more major releases and minor releases (happened for few projects), then the last major release is selected as "label" for that week;
2. if there are only minor releases in the same week then pick the last one as "label" for that week;
3. if there is no release in a week it is left blank.

After matching every week with a release (if the week has one), and the weekly stars for every repository are also calculated, merging the 2 collections will give an overview of how many stars a release (more releases) accounts for in the week. Figure 26 is an example of a release stars table for a repository (using the pandas DataFrame).

	Week	Release Type	Stars
313	2016-09-25	M	906
352	2017-06-25	m	67
358	2017-08-06	m	59
381	2018-01-14	m	147
431	2018-12-30	m	42
449	2019-05-05	m	54
477	2019-11-17	m	72
603	2022-04-17	m	619
625	2022-09-18	m	61

```
i : 4, Project Leaflet/Leaflet, mean stars releases=225.22222222222223, mean stars=53.96306818181818
```

Figure 26. Releases table for the Leaflet project

Estimating the impact of releases on popularity is done by first calculating the ratio of the means of stars during release weeks and all weeks. Another ratio is calculated with the means of stars one week after the release and all weeks as done by [3]. This can be seen in Figure 27.

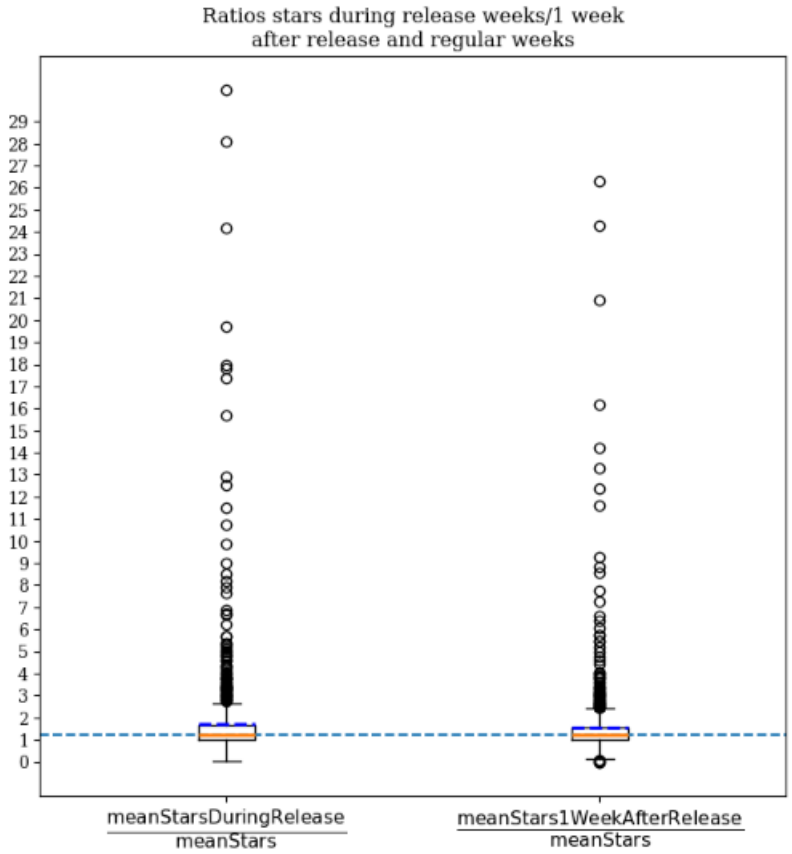

Figure 27. Ratios stars during release weeks/1 week after release and regular weeks

	1st quartile	**median**	**3rd quartile**	**90th percentile**
Ratio stars during release weeks vs. all weeks	0.99	1.24	1.65	2.55
Ratio stars week 1 week after release vs. all weeks	0.98	1.21	1.56	2.19

Table 4. Ratio stars mean during release week/1 week after release and mean all stars

From Figure 27. and table 4, it is clear that more stars happen during release weeks than 1 week after. [4] evaluate the impact of releases with ratios 1 week after releases which as seen above is less than counting stars

during releases. It is important to note that natural growth or external factors cannot be taken into account in this analysis so some repositories may present a peak in growth even though there is no release in Github. For a lot of repositories, there are no releases at the beginning even though the stars are at a peak and these are also not taken into consideration as releases. Thus, the overall ratios should be higher due to unversioned releases.

In the dataset, out of 1277 repositories, only 944 have releases. This indicates that most projects with releases become popular. It is also interesting that of the 944 projects with releases, 384 do not have a major release. The reason for this is maybe that the project does not rely that much on the version system on Github and thus they tend to not update it when major versions are released. Another reason for why the overall discrepancy between major releases and minor releases in relation to stars cannot be found is that out of the 480 projects that have major releases from the 1250 repositories, 264 of them have the maximum number of stars in a week with minor releases. Thus, even though the chances of stars in a week with a major release being larger than stars in a week with minor release are greater, it is still very uncertain whether a major release will have a higher or lower impact than expected.

4.6 Predicting the popularity of a repository

Prediction is a prevalent aspect in multiple industries and areas. Most notable ones are stock market prediction, sales predictions, weather forecasting to name a few [8][21]. Prediction is also an important asset for software as developers can align their workflows with the predictions and meet the demands of the customers. In the instance of this research, the predicted variable will be the number of stars gained by a Github repository in a certain month/week. To do this, several models were picked which are going to be assessed based on the performance of predicting different intervals of stars. Two types of inputs are given to the models, which is also another factor that can influence the outcome of the predictions. First, the chapter will start with the baseline model, which is expected to have the worst performance. Afterwards, more powerful models (i.e. models that are capable of learning non-linear relationships in the data) will be tested and the chapter will conclude with comparisons between all models and rankings.

For all models, at the beginning the inputs have the following form:

PL	L	OT	S_{t-3}	S_{t-2}	S_{t-1}

Table 5. Original input 1

PL	L	OT	R^X_{t-3}	R^X_{t-2}	R^X_{t-1}	S_{t-3}	S_{t-2}	S_{t-1}

Table 6. Original input 2

where:

- PL - programming language = Python, JavaScript etc.
- L - license type = MIT License, GNU General Public License etc.
- OT - owner type = User or Organisation
- R^X_t - type of release in week t = major/minor/no release
- S_t = stars at month t

However, machine learning algorithms do not understand labels such as 'Python', 'Java', 'MIT License'. They need inputs of numerical form. If

every programming language is thus labeled 'Python': 0, 'JavaScript':1, 'C' : 5 and so on .., how will the machine learning algorithm adjust the weights when for instance 'Python' repositories have more stars than 'C' repositories? A working approach is one that makes every instance of the categorical variable into a new variable, with 1 representing that the variable is present, and 0 that it is not. Therefore, the input for the machine learning algorithm is transformed into the final form:

PL_1	PL_2	..	PL_n	L_1	L_2	..	L_m	OT_1	OT_2	S_{t-3}	S_{t-2}	S_{t-1}
0	1	..	0	1	0	..	0	1	0	102	136	112

Table 7. Input 1 specific for ML algorithm

PL_1	..	PL_n	L_1	..	L_m	OT_1	OT_2	R^m_{t-3}	..	R^M_{t-1}	S_{t-3}	S_{t-2}	S_{t-1}
0	..	1	0	..	1	0	0	1	0	1	102	136	112

Table 8. Input 2 specific for ML algorithm

As seen in Table 7, PL2 has value 1 and all the other PL variables have value 0, meaning only PL2 is present and the others are not. This translates to PL2 is maybe Java, therefore the repository has the main programming language Java), and the other PLs are omitted because they are not the main programming language of the project. The same applies for L and OT.

In table 8, each release type are each turned into a variable representing whether it is present or not. If for example week t - 3 has a minor release but no major release, then $R^m_{t-3} = 1$, if week t - 3 also has a major release, then only $R^M_{t-3} = 1$. Same applies for R^X_{t-2} and R^X_{t-1}.

The idea of the 2 different inputs is to see whether GitHub releases can help predict star counts better with releases than without. The data used in the model's predictions is comprised of 1277 of the most popular projects, because these had their stars timestamps and releases mined. For all repositories, every 4 weeks is considered a data point as we need the first 3 weeks in order to predict the 4th. This amounts to 505132 data points. The data is split into 2 groups, of which 80% is training data and 20% is testing.

4.6.1 Baseline model: Multiple linear regression

Multiple linear regression [12] is an algorithm that finds a line that minimizes the error between the predicted and actual values, with multiple variables. In this case, the equation of the line found by multiple linear regression follows an equation with the variables from Tables 7 and 8:

$$S1_t = a_1 \cdot PL_1 + a_2 PL_2 + .. + a_n PL_n + b_1 L_1 + b_2 L_2 + .. + b_m L_m + d_1 OT_1 + d_2 OT_2 + d_3 S_{t-3} + d_4 S_{t-2} + d_5 S_{t-1} + c$$

Eq 11. Linear regression equation for input 1.

$$S2_t = a_1 \cdot PL_1 + a_2 PL_2 + .. + a_n PL_n + b_1 L_1 + b_2 L_2 + .. + b_m L_m + c_1 R^M_{t-3} + c_2 R^m_{t-3} + c_3 R^e_{t-3} + .. + c_7 R^M_{t-1} + c_8 R^m_{t-1} + c_9 R^e_{t-1} + d_1 OT_1 + d_2 OT_2 + d_3 S_{t-3} + d_4 S_{t-2} + d_5 S_{t-1} + e$$

Eq 12. Linear regression equation for input 2.

where:

- $S_t \in \mathbb{N}$ - number of stars at month t
- $a_1..a_n, b_1..b_m, c_1..d_9, d_1..d_5, e \in \mathbb{R}_+$, constants
- $PL_1, PL_2, ..PL_n$ - programming languages 1..n, where $PL_x \in \{0, 1\}$
- $L_1, L_2, ..L_m$ - licenses 1..m, where $L_x \in \{0, 1\}$
- OT_1, OT_2 - organisation type, where $OT_x \in \{0, 1\}$
- R^X_t - type of release, with X $\in \{M, m, e\}$ and $R^X_t \in \{0, 1\}$

The table below shows the RMSE score of multiple linear regression in prediction:

Model	**RMSE** all repos	**RMSE** repos w last month < 1000 stars	**RMSE** repos w last month < 500 stars	**RMSE** repos w last month < 300 stars	**RMSE** repos w last month < 1000 and > 500 stars	**RMSE** repos w last month < 500 and > 300 stars	**RMSE** repos w last month < 300 and > 200 stars	**RMSE** repos w last month < 200 and > 100 stars
Model 1	94.13	41.17	32.24	27.81	541.46	269.27	165.38	74.59
Model 2	92.12	41.16	30.03	27.87	540.38	268.68	164.71	74.32

Table 9. RMSE of multiple linear regression testing different intervals of stars

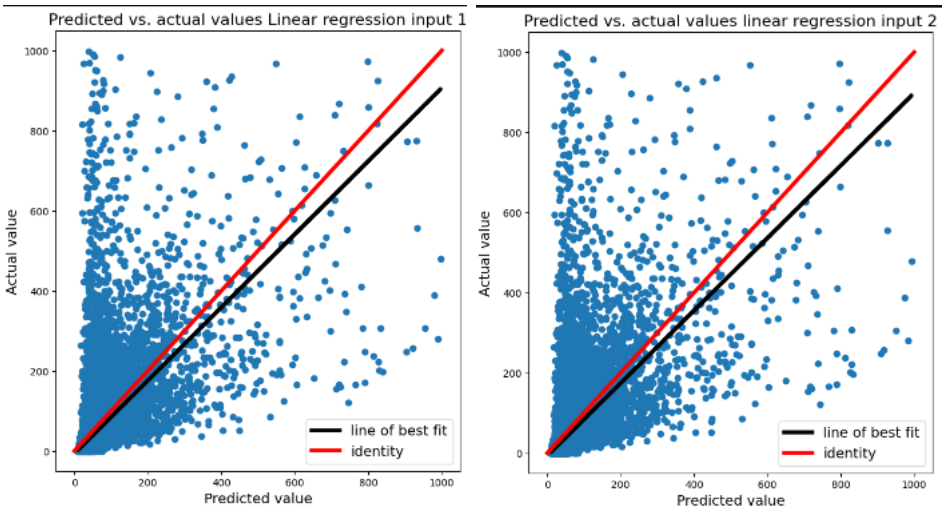

Figure 28. Linear regression predicted and actual values

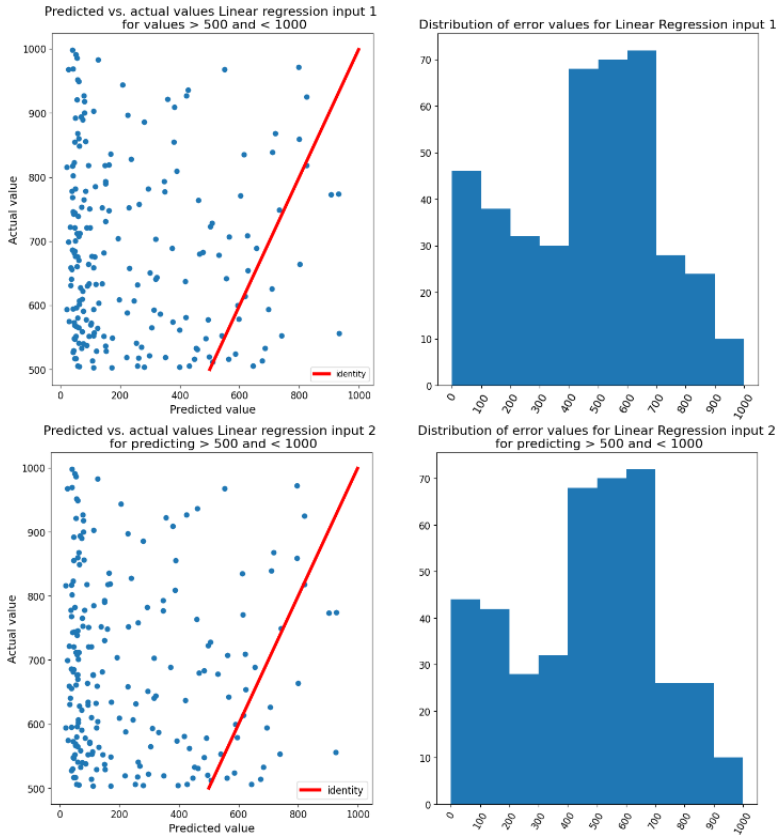

Fig 29. Linear regression prediction and error with inputs 1 and 2 with

predicted values > 500 and < 1000

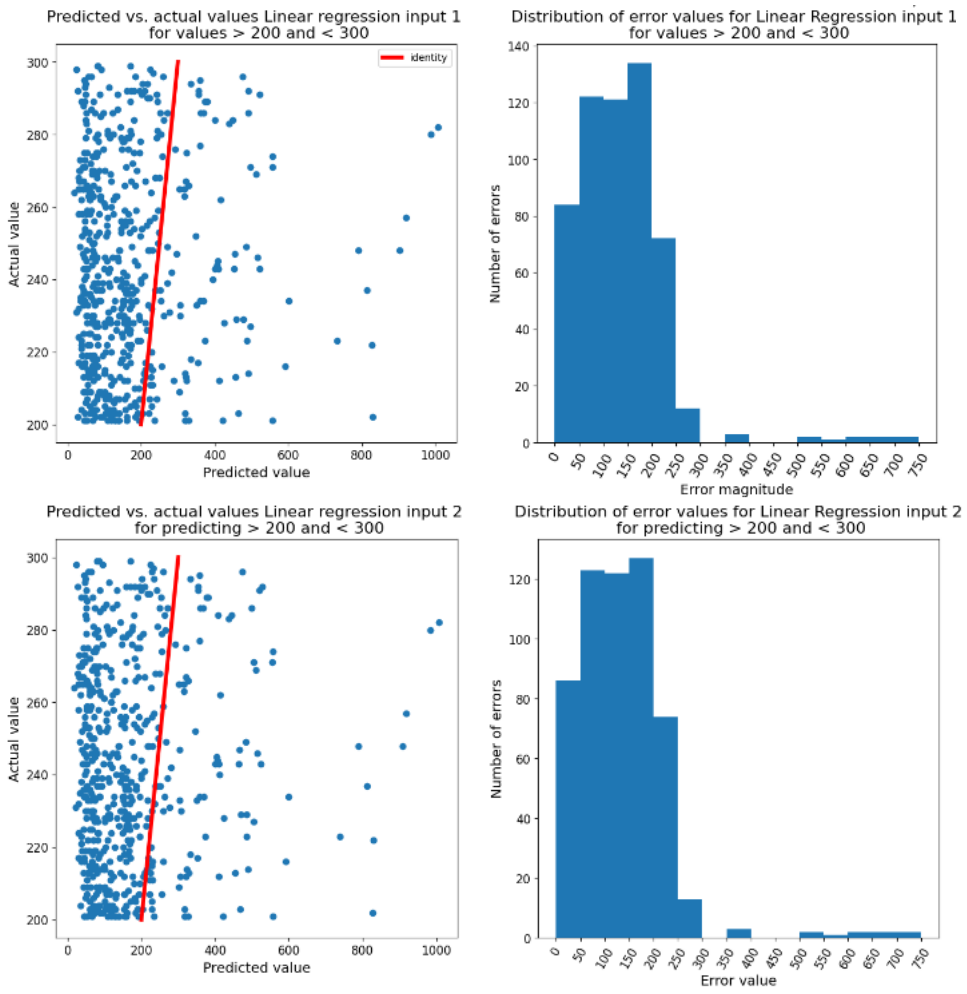

Fig 30. Linear regression prediction and error with inputs 1 and 2 with predicted values > 200 and < 300

Table 9 and Figures 28, 29, 30 provide us with enough insights of the limitations of Multiple Linear regression. Table y shows the average error of prediction within different intervals, and the fact that the average error is quite close to the minimum value to be predicted, this reflects the bad quality of MLR in this instance. The plots in Figure 28. show that both models have predicted small values when the actual value was quite high. Even though predicting a small number of stars is feasible, predicting stars > 300 becomes a daunting task for MLR, as seen in Fig 29, 30. Using Input 2

(incorporating releases in the data) gives overall very slightly better results, however, it is not a good method for prediction in this circumstance, as it cannot understand non-linear data.

4.6.2 Neural network prediction

Neural networks are complex architectures that can learn any output given a certain input [10]. Different model hyperparameters can change the performance of the model, either making it overfit, underfit or fit well with the data. The hyperparameters that will be tested in this research will be the number of hidden layers and the number of neurons in those hidden layers.

Model	**RMSE** all repos	RMSE repos w last month $<$ 1000 stars	RMSE repos w last month $<$ 500 stars	RMSE repos w last month $<$ 300 stars	RMSE repos w last month $<$ 1000 and $>$ 500 stars	RMSE repos w last month $<$ 500 and $>$ 300 stars	RMSE repos w last month $<$ 300 and $>$ 200 stars	RMSE repos w last month $<$ 200 and $>$ 100 stars
Model 1 1HL: 64	95.44	44.03	35.03	30.67	564.85	278.88	183.46	81.89
Model 2 1HL: 64	92.41	36.5	28.25	21.02	537.27	257.78	148.57	63.23
Model 1 1HL: 128	**92.05**	**35.32**	25.52	20.55	516.31	249.77	142.21	59.57
Model 2 1HL: 128	92.96	38.69	28.87	23.79	544.8	269.44	166.69	72.38
Model 1 1HL: 256	92.72	38.16	29.09	24.51	522.12	258.75	160.97	73.69
Model 2 1HL: 256	92.22	36.49	26.35	21.24	533.29	257.49	153.44	64.95
Model 1 2HLs: 128FL, 64 SL	92.23	36.72	27.2	24.51	521.39	252.80	149.42	62.81
Model 2 2HLs: 128FL, 64 SL	**92.05**	36.52	27.11	22.51	517.26	249.43	148.42	62.87
Model 1 2HLs: 256FL, 64 SL	92.28	35.32	25.24	**20.03**	521.85	253.65	143.37	59.54
Model 2 2HLs: 256FL, 64 SL	92.51	36.79	27.55	22.88	515.39	253.60	153.4	68.96
Model 1 2HLs: 128FL, 128 SL	93.27	35.87	25.68	20.4	529.48	257.50	142.66	60.77
Model 2 2HLs: 128FL, 128 SL	92.98	38.54	28.79	23.96	531.46	255.88	158.91	67.31
Model 1 2HLs: 256FL, 128 SL	93.66	39.97	30.37	25.6	549.86	269.33	170.81	72.95
Model 2 2HLs: 256FL, 128 SL	92.18	35.95	25.87	20.34	527.68	253.93	145.45	60.74
Model 1 3HL: 64FL, 64SL, 64TL	93	38.72	29.08	24.15	540.72	266.88	162.83	68.78
Model 2 3HL: 64FL, 64SL, 64TL	97.76	41.47	30.12	24.08	602.58	298.24	168.52	76.41
Model 1 3HL: 64FL, 128SL, 64TL	95.44	42.35	31.93	24.15	594.35	296.05	193.73	85.6
Model 2 3HL: 64FL, 128SL, 64TL	92.43	36.79	26.65	21.48	535.96	260.24	158.71	68.19

Table 10. RMSE of Neural Networks with different hyperparameters testing different intervals of stars

Table 10 shows different Neural Network architectures that have been explored for predicting stars at month t given the input as defined in Tables 7,8. Both the models with input 1 and 2 perform modestly because of the unpredictability of the numbers of stars even following weeks with releases. (If for example the release happens in the first day of the week then the number of stars of that week will be higher, however, if the release happens

on the last day of the week then the number of stars will be higher next week), and so models with input 2 sometimes perform worse than the models with input 1. It is important to note that the overall best prediction is made by a model with a simple architecture, with only one hidden layer and 128 neurons on that layer. Both the predictions done within a certain interval and under a threshold of stars for this model are the best overall.
Figure 31. shows the best performing model's (1HL 128) accuracy with a scatter plot and the distribution of errors.

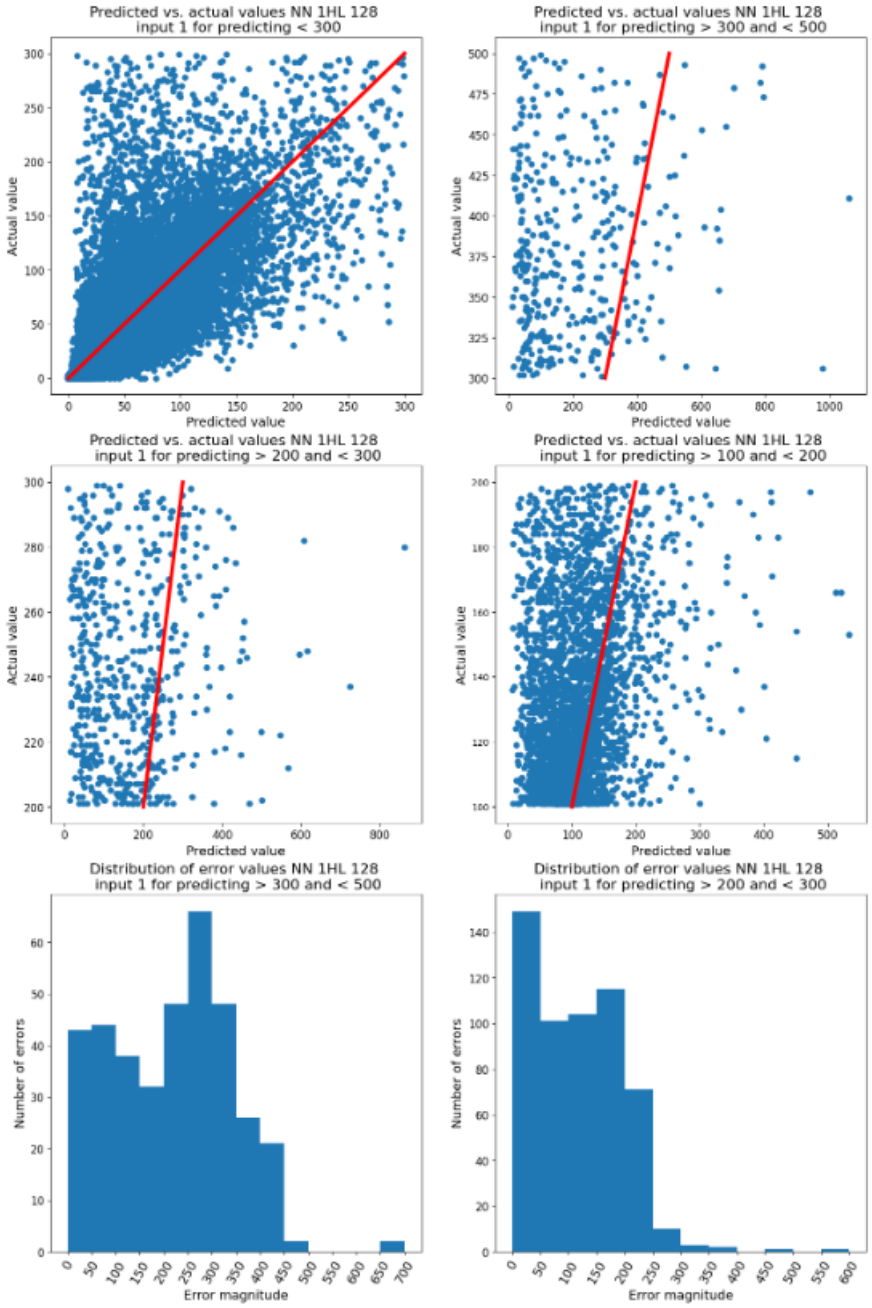

Figure 31. NN with 1HL and 128 neurons predicted and actual value plot for different intervals of stars.

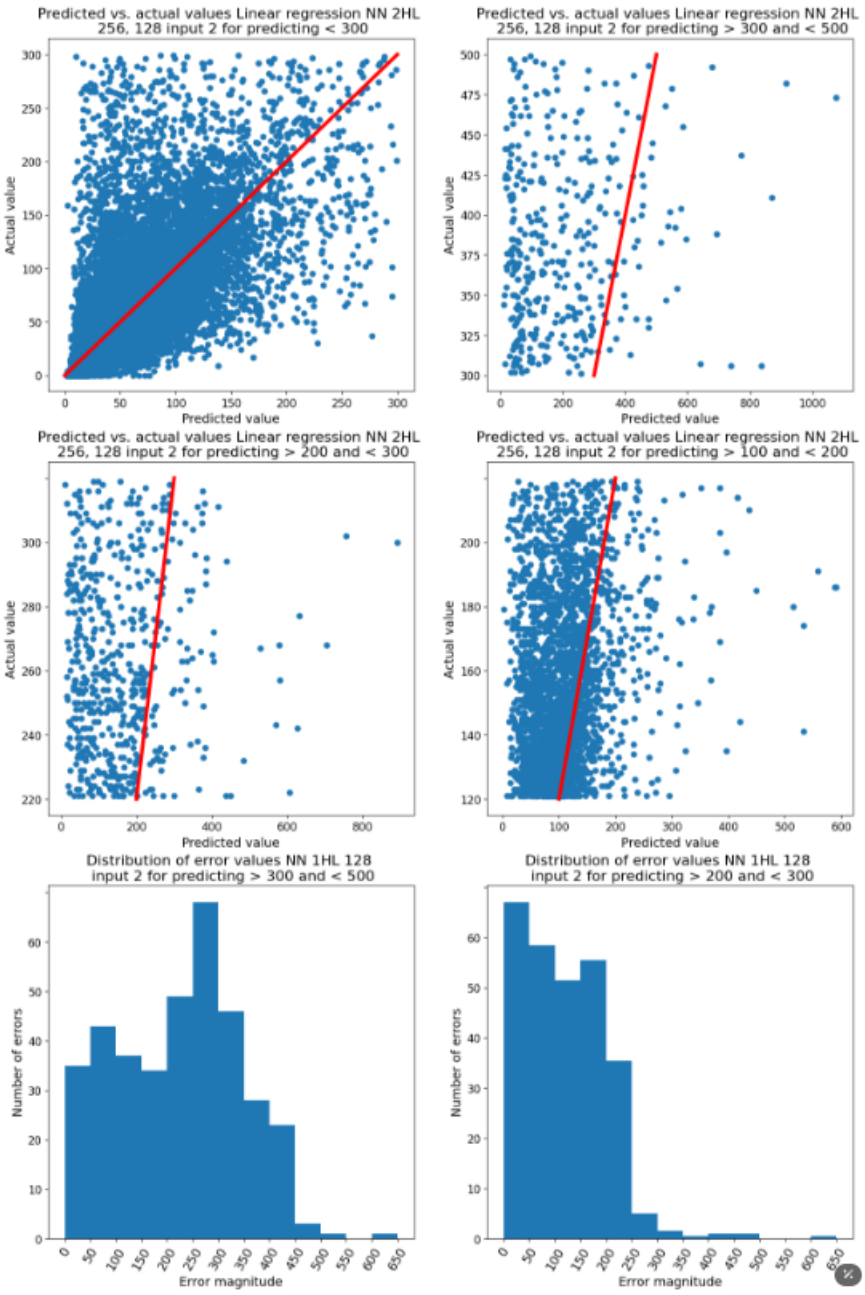

Figure 32. NN with 2HLs (1L - 256 neurons, 2L - 128 neurons) predicted and actual values plots for multiple intervals of stars

Figure 31 depicts the best model for input 1 and figure 32 depicts the best model for input 2. The model for input 2 has larger errors as can be seen in the histograms. The model for input 1 is better overall than the model for input 2, with all the metrics from table 10 being larger for the model of input 1. It appears that for this data, releases do not drastically help the model in giving a more accurate prediction.

It is crucial to note that the score achieved predicting the number of stars > 100 in the Neural Networks tested is not reliable. Even though Neural Networks can learn any output from any input, the input does not have enough information to properly predict a reliable output.

4.6.3 Random forest prediction

Random Forest is an ensemble learning method that can be used in classification and regression. Ensemble learning refers to the aggregation of results of multiple decision trees. For regression, the mean of all resulting values of the decision trees is taken as the prediction of the random forest. In our case, multiple decision trees are fit on the data, i.e. either using input 1 and input 2 and they split the data points such that the difference of the variance in the split is lowest.

Model	**RMSE** all repos	RMSE repos w last month $<$ 1000 stars	RMSE repos w last month $<$ 500 stars	RMSE repos w last month $<$ 300 stars	RMSE repos w last month $<$ 1000 and $>$ 500 stars	RMSE repos w last month $<$ 500 and $>$ 300 stars	RMSE repos w last month $<$ 300 and $>$ 200 stars	RMSE repos w last month $<$ 200 and $>$ 100 stars
nr. estimators: 40 depth: 20, input 1	110.59	72.02	67.7	65.73	523.11	273.12	205.98	96.54
nr. estimators: 40 depth: 20, input 2	105.85	64.77	59.95	57.65	521.08	274.67	227.17	96.55
nr. estimators: 40 depth: 40, input 1	109.75	71.31	66.94	65.04	523.21	265.65	211.42	95.7
nr. estimators: 40 depth: 40, input 2	107.01	66.71	62.03	59.69	522.04	280.51	231.77	101.27
nr. estimators: 50 depth: 50, input 1	110.53	72.02	67.92	65.93	510.5	274.92	212.83	96.89
nr. estimators: 50 depth: 50, input 2	105.76	64.87	60.12	57.80	518.07	275.04	225.29	97.47
nr. estimators: 60 depth: 60, input 1	112.24	74.6	70.54	68.7	517.6	269.55	210.49	97.64
nr. estimators: 60 depth: 60, input 2	107.18	66.80	62.22	60.01	516.97	273.99	214.42	94.47
nr. estimators: 80 depth: 80, input 1	110.12	71.4	67.11	65.07	518.94	276.58	207.81	94.19
nr. estimators: 80 depth: 80, input 2	**105.01**	**63.47**	58.49	56.14	523.84	273.63	229.81	97.63
nr. estimators: 100 depth: 100, input 1	110.82	72.47	68.16	66.27	524.13	268.91	221.85	94.4
nr. estimators: 100 depth: 100, input 2	93.66	39.97	30.37	25.6	549.86	269.33	170.81	72.95

Table 11. RMSE of Random Forests with different hyperparameters testing different intervals of stars

From table 11 we can see that the random forest model is not fit for predicting the data, with the minimum RMSE for repositories $<$ 300 stars at 56.14. It is not good at predicting small intervals of stars either, with the smallest RMSE of predicting stars between 100 and 200 being 94.4 which is an extremely poor performance. This happens as the model predicts a very large number of stars e.g. 950 for a repository that has 0,2,5 as last stars, because it has seen some examples with the pattern with the previous 3 weeks having few stars and the next with many stars, and so it started to overfit.

4.6.4 Performance comparison

Multiple linear regression (MLR) performed well, with accurate predictions on weeks with < 200 stars. The Neural network outperformed both MLR and RF scoring decent results in predicting values < 300 stars. Random Forest (RF) had the worst scores overall, as it overfit on training samples that had a high number of stars in the predicted week, thus predicting very large numbers for some weeks that had a small amount of stars.

The best model for predicting stars is the Neural Network with 1HL, 128 neurons on the HL and input 1. It has an average error of only 20 for values < 300, an average of 59.57 for repositoriess < 200 and > 100, 142.21 for repositories < 300 and > 200, which is decent for an environment as unpredictable as Github. Not very far off is the model with input 2, 2HLs, 256 and 128 neurons on the first and second layer respectively.

Most interestingly, the models without releases performed better than the models with releases. This shows that even though releases help in detecting a pattern in the stars to be predicted, it will make the model have a bias towards giving more/less stars than usual.

4.6.5 Feature importance

Understanding what variables influence the prediction of the model is one of the most important aspects in machine learning. SHAPley Additive explanations are a way to describe what features had the most impact on the predictions of the model.

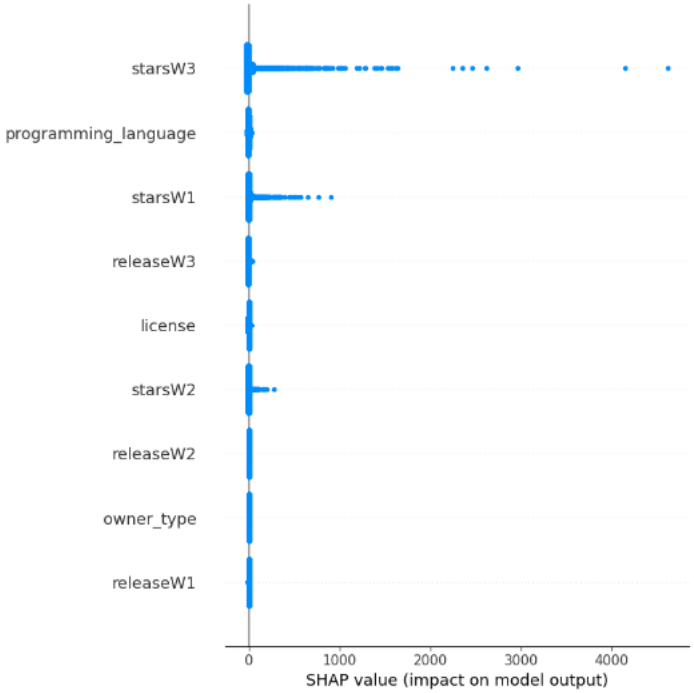

Figure 33. Feature importance in predicting the number of stars

The plot above (Figure 33) shows the importance of the variables in the prediction. Evidently, the stars in the week before the predicted week helped the prediction most. Programming language is the 2nd most important feature which is somewhat surprising. Thus, programming language does influence the stars repositories gain even on average every week. Also surprising is the fact that the number of stars in week 1 influences the prediction more than the number of stars in week 2. Releases in the week before prediction (releaseW3) also play a role in predicting the number of stars, as was expected. Owner type is not regarded as an important feature for this particular problem, and neither are releaseW1 and releaseW2.

Chapter 5

Discussion

This paper studies the effect of different features on the number of stars, GitHub star growth, the effects of releases on star growth, and popularity prediction.

The correlation part of the paper reaches the same conclusions as [4], from which this paper gets its inspiration from, the exception being the ratio between closed issues and all issues and licenses. It was expected that the high number of commits will not translate into a large number of stars, which proved to be correct. However, programming language, licenses, owner type, forks and watchers all are related to the number of stars.

The GitHub growth part is similar to [4]'s, but different thresholds of percentages of stars were analysed than the ones covered by [4], as it was necessary to see whether the percentage of stars was gained in the same percentage of time (which depicts a rather average growth trend) which turned out to be true. Also, as opposed to [4]'s thresholds at 10%,50%,90%, this thesis put the thresholds of stars at 10%, 25%, 50%, 75% to gain information about how many repositories gain that percentage in a short span, i.e. 1%-10% of their age, which reflects viral projects in their inception, and how many projects gain those stars later.

The different effects of releases on the number of stars was discussed in detail, with this thesis being the sole study which visualises and explains all possible patterns that releases have on the number of stars. It arrives at the conclusion that there are 4 possible patterns that releases have on the number of stars:

1. major releases and subsequent minor releases impact the popularity the most as they most often reside on peaks of popularity

2. releases may happen in periods of low popularity, but after a small

period of time, the project will reach a new peak.

3. When there are periods of inactivity and a release happens, the repository will most likely find itself on a spike of popularity.

4. After a spike of growth either because of a major or minor release, the trend will stabilise and can also decrease.

This work tried to predict the number of stars like other studies [2][4][5], and tried 3 different machine learning models, in order to see which one of them performs best. The machine learning models in question are Multiple Linear Regression used by [4][5], Neural Networks and Random Forest. Though there are not studies that use Neural Networks and Random Forest in predicting the number of stars of GitHub, there are some works that used different machine learning models such as LSTMs [2] and RNNs [15], which are more advanced Neural Networks.

There are several factors that impeded a truly accurate prediction of the number of stars of a GitHub project. Rapid growth of a project due to external factors could not have been taken into account given data that has been mined only via Github API. Future improvements on the prediction could be done by crawling blog posts and websites which promote the repositories from the dataset and analyse the text using NLP (Natural Language Processing) to find out whether there will be new updates coming soon and what the impact it will have based on the number of likes that post has. Another limitation of this prediction was the shortage of releases of GitHub repositories, with some repositories not stating releases even though through the lifespan of the projects several large changes happened. Time was yet another factor which impacted the result of the prediction, with more data gathered possibly resulting in more accurate predictions.

Thus, due to incomplete data and limitations of the repository data in Github involving releases, the predictions of the number of stars made in this research are not considered accurate enough.

Chapter 6

Conclusion

In the correlation part of the research, we have proved that programming language, owner type, license, forks and watchers all contribute to the popularity of a repository. Commits and the ratio of $\frac{\text{issues solved}}{\text{total issues}}$ do not influence the number of stars as there are projects with few developers (thus few commits) that are highly popular and there are repositories that still have a lot of tasks pending (features they want to release) and still have many stars.

In the growth chapter of the research, the effect of releases on stars and the time it takes for a project to reach a threshold of stars is discussed. There are 90% of projects that reach 10% of stars in $\frac{4}{10}$ of their lifetime, which means that there are 10% of repositories that gain almost all their stars following the 2nd half of their lifespan. As expected 50% of the projects attain 50% of their current amount of stars within half their age. 1% of repositories gain 75% of stars by $\frac{1}{10}$th of their lifespan, which points to repositories that grew exponentially at the beginning and then dimming down. Releases have shown to have various behaviours such as residing on peaks of popularity, starting a period of rapid growth and then a decline, or simply not affecting the popularity at all and the project even declining after a release. The statistical analysis proves that for 75% of projects, the mean of stars in weeks with releases surpasses the mean of all weeks. This was to be expected, if not even more so as in almost all repositories the beginning releases are not even documented.

The last chapter focuses on predicting the number of stars in a week t, with 3 different models: Multiple Linear Regression, Neural Networks and Random Forests, given 2 inputs, an input with releases and one without releases. Multiple Linear Regression performed surprisingly well at predicting

small numbers of stars, but gradually became worse when predicting values > 100. Neural networks performed best, however it fell into the same trap as Multiple Linear Regression, predicting modestly for stars > 200. The worst model was Random Forest, which overfit on samples that had few stars in the input and a lot of stars in the output, thus predicting a high number of stars even when the stars in the input were very small. Surprisingly, a Neural Network model with input 1 (without releases) had the best results, thus showing the unpredictability of Github project releases on the number of stars. Feature importance suggests that the number of stars in the weeks before the prediction, whether a release happened in the week before the prediction, programming language and license had the most impact in predicting the number of stars.

To conclude, we have discovered the repository features that impact a repository's popularity, looked at possible growth patterns, monitored the effect of releases on a repository's growth and predicted the number of stars of the most popular Github repositories.

Bibliography

[1] A. Al-Rubaye and G. Sukthankar. Scoring popularity in github. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 217–223, 2020.

[2] Neda Bidoki, Gita Sukthankar, Heather Keathley-Herring, and Ivan Garibay. A cross-repository model for predicting popularity in github, 02 2019.

[3] John D. Blischak, Emily R. Davenport, and Guy Wilson. A quick introduction to version control with git and github. *PLoS Computational Biology*, 12(1):e1004668, January 2016.

[4] H. Borges, A. Hora, and M. T. Valente. Understanding the factors that impact the popularity of GitHub repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344, Raleigh, NC, USA, 2016.

[5] Hudson Borges, Andre Hora, and Marco Tulio Valente. Predicting the popularity of github repositories. In *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE 2016, New York, NY, USA, 2016. Association for Computing Machinery.

[6] Hudson Borges and Marco Tulio Valente. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129, 2018.

[7] J. Börstler, Kwabena Ebo Bennin, Sara Hooshangi, Johan Jeuring, Hieke Keuning, Carsten Kleiner, Bonnie MacKellar, Rodrigo Duran, Harald Störrle, Daniel Toll, and Jelle van Assema. Developers talking about code quality. *Empirical Software Engineering*, 28:1–31, 2023.

[8] A H M Jakaria, Md Mosharaf Hossain, and Mohammad Ashiqur Rahman. Smart weather forecasting using machine learning: A case study in tennessee. In *Proceedings of ACM Mid-Southeast conference (Mid-Southeast'18)*, page 4, New York, NY, USA, 2018. ACM.

[9] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel German, and Daniela Damian. The promises and perils of mining github (extended version). *Empirical Software Engineering*, 2015.

[10] Usha A. Kumar. Comparison of neural networks and regression analysis: A new insight. *Expert Systems with Applications*, 29(2):424–430, 2005.

[11] Gillian Lemke. The software development life cycle and its application, 2018.

[12] Mark Lunt. Introduction to statistical modelling: linear regression. *Rheumatology*, 54(7):1137–1140, 04 2013.

[13] Matthew Nelson, Ravi Sen, and Chandrasekar Subramaniam. Understanding open source software: A research classification framework. *Communications of the Association for Information Systems*, 17:pp–pp, 2006.

[14] L. Ren, S. Shan, X. Xu, and Y. Liu. Starin: An approach to predict the popularity of github repository. In P. Qin, H. Wang, G. Sun, and Z. Lu, editors, *Data Science. ICPCSEE 2020. Communications in Computer and Information Science*, volume 1258, Singapore, 2020. Springer.

[15] Sefa Eren Sahin, Kubilay Karpat, and Ayse Tosun. Predicting popularity of open source projects using recurrent neural networks. In *Proceedings of the 15th IFIP International Conference on Open Source Systems*, pages 80–90, Montreal, QC, Canada, May 2019. Springer. `https://hal.archives-ouvertes.fr/hal-02305699`.

[16] Ali Sajedi and Eleni Stroulia. Measuring user influence in github: the million follower fallacy. pages 15–21, 05 2016.

[17] Ravi Sen, Siddhartha S. Singh, and Sharad Borle. Open source software success: Measures and analysis. *Decision Support Systems*, 52(2):364–372, 2012.

[18] Jyoti Sheoran, Kelly Blincoe, Eirini Kalliamvakou, Daniela Damian, and Jordan Ell. Understanding "watchers" on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 336–339, New York, NY, USA, 2014. Association for Computing Machinery.

[19] Yasmine Ska and Research Publications. A study and analysis of continuous delivery, continuous integration in software development environment. *SSRN Electronic Journal*, 6:96–107, 09 2019.

[20] Diomidis Spinellis. Git. *IEEE Software*, 29(3):100–101, 2012.

[21] Abdulhamit Subasi, Faria Amir, Kholoud Bagedo, Asmaa Shams, and Akila Sarirete. Stock market prediction using machine learning. *Procedia Computer Science*, 194:173–179, 2021. 18th International Learning Technology Conference 2021.

[22] Shu Yan, Wang Shanshan, Ren Yizhi, and Kim-Kwang Raymond Choo. User influence analysis for github developer social networks. *Expert Systems with Applications*, 108:108–118, 2018.

[23] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. Measuring github copilot's impact on productivity. *Commun. ACM*, 67(3):54–63, feb 2024.

[4] [23] [3] [1] [17] [9] [14] [16] [22] [2] [18] [15] [5] [6] [20] [13] [11] [19] [21] [8] [12] [10] [7]

Appendix A

Appendix

Decision Tree splitting example

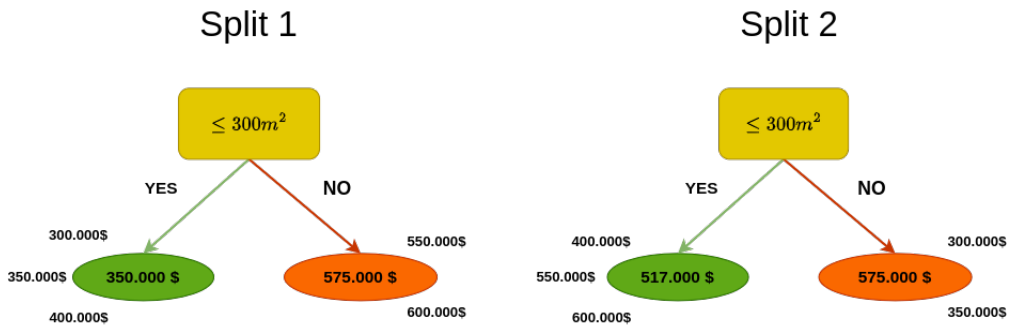

Figure 7. Decision tree splits: Split 1 minimises variance and split 2 is a split that does not minimise variance.

By looking at figure 7, we see that the values in both parts of split 1 are closer together/have less variance than the values in split 2. This is preferred in regression as when predicting a value we want to be as close as possible to the actual value.

Split 1:

$\sigma^2_{\leq 300m^2=yes} = \frac{(300.000-350.000)^2+(350.000-350.000)^2+(400.000-350.000)^2}{3} = \frac{25\cdot 10^8 \cdot 2}{3} = 16.(3) \cdot 10^8$.

$\sigma^2_{\leq 300m^2=no} = \frac{(550.000-575.000)^2+(600.000-575.000)^2}{3} = \frac{12.5\cdot 10^8}{2} = 6.25 \cdot 10^8.$

$\sigma^2_1 = \frac{3}{5} \cdot \sigma^2_{\leq 300m^2=yes} + \frac{2}{5}\sigma^2_{\leq 300m^2=no} = 14.78 \cdot 10^8$

Split 2:

$\sigma^2_{\geq 300m^2=yes} = \frac{(400.000-517.000)^2+(550.000-517.000)^2+(600.000-517.000)^2}{3} = \frac{21.539*10^9}{3} = 71.966 \cdot 10^8$

$\sigma^2_{\geq 300m^2=no} = \frac{(300.000-325.000)^2+(350.000-325.000)^2}{2} = \frac{12.5\cdot 10^8}{2} = 6.25 \cdot 10^8.$

$\sigma^2_2 = \frac{3}{5} \cdot \sigma^2_{\geq 300m^2=yes} + \frac{2}{5}\sigma^2_{\geq 300m^2=no} = 45.676 \cdot 10^8.$

From the calculations above, it arises that split 1 has a much lower variance than split 2, thus it is preferred.