INTERNATIONAL
HELLENIC
UNIVERSITY

Data Mining on Source Code

Charisiadis Christos

SID: 3301150002

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

DECEMBER 2017

THESSALONIKI – GREECE

Data Mining on Source Code

Charisiadis Christos

SID: 3301150002

Supervisor:	Prof. Christos Tjortjis
Supervising Committee Members:	Assoc. Prof. Name Surname
	Assist. Prof. Name Surname

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

DECEMBER 2017

THESSALONIKI – GREECE

Abstract

This dissertation was written as a part of the MSc in ICT Systems at the International Hellenic University.

Software companies and software engineers have always tried to find ways to improve the quality of their projects. However, assessing the quality of a piece of software is not something trivial. There are many parameters that can affect that, and most of the time it is very hard to find objective ways to measure the quality of software.

In this dissertation, I will provide a methodology that aims to assess software using automatically extracted software metrics and community based metrics, which will be GitHub Stars and Forks. This way we can use both static metrics and dynamic ones, in order to try and predict the reusability of a new piece of software.

I would like to thank my supervisor, Christos Tjortjis, for guiding me through this Dissertation.

Charisiadis Christos

18 December 2017

Contents

1 Introduction

Software applications are an increasingly important part in our everyday lives. We use many applications throughout our day, even if we do not notice it. However, designing a piece of software is still a very complex and lengthy process. Software engineers are always looking for ways to simplify and reduce the required time of designing and implementing an application. One of the most important aspects of software, is their reusability. Designing a software with focus towards its reusability, is definitely more difficult and adds an extra overhead, but pays off in the long run. We can see this from the evolution of programming through the years, with the heavy shift toward object-oriented programming which tends to be more reusable when designed properly.

Most of the research throughout the years has focused on trying to assess software quality through static metrics. Using these metrics, software engineers can monitor the quality of their projects throughout their development and try to keep within some specific thresholds. However, these thresholds are usually set in a somewhat arbitrary fashion, that usually comes from previous knowledge from other projects.

With the addition of open source software, we suddenly have a new and highly diverse pool of software projects. These can vary in size and quality. It also gives us an invaluable tool in assessing these, which is community perception. Instead of focusing on static metrics we can try to measure the quality in terms of how it is perceived by the users of the online community and how often it is actually reused by them.

1.1 Dissertation Structure

This Dissertation is split in the following sections:

1.1.1 Background

This chapter will contain a review of the relative literature of the recent years. This will give an overview of the current state of the art in order to allow the reader to reflect how the work of this Dissertation compares to similar work by other researchers.

1.1.2 Software Metrics

This chapter will give all the details about the various metrics used. This will contain both static metrics, which contain various of the commonly used metrics like cyclomatic complexity and lines of code. It will also describe the dynamic metrics that we used, which are the start and forks of a project that were extracted from GitHub.

1.1.3 Methodology

This chapter will describe the methodology that was used for gathering the data and performing the tests. It will also describe the tools that were used for this process. Finally, it will include the process that was selected to create the dataset and separate the samples into different classes.

1.1.4 Results

This chapter will contain all the experiments with the different algorithms that were performed along with their results.

1.1.5 Evaluation

This chapter contains an evaluation of the results presented in the previous chapter. It will provide a comparison between the different algorithms used and an interpretation of the overall results and present their high and low points.

1.1.6 Conclusion and Future Work

This concluding chapter contains a final assessment of the methodology and the produced results. It also contains the next steps that can be taken in order to further research this idea and the possible outcomes that can come out of this.

2 Background

Data mining on source code has seen a lot of research over the years. A large number of software metrics have been proposed and used in order to try to predict quality characteristics in software. Also, various methods have also been proposed in order to try to quantify quality in various ways.

Along with the extensive research, a number of software quality assurance tools have also been created. ComPARE is one such tool and was proposed by Cai et al. [1]. ComPARE collects various metrics from software components, and also incorporates various different models that predict software quality and reliability. The collected metrics are both static metrics extracted from the source code, but also dynamic metrics that are extracted during the programs execution. Besides assessment tools there are also many tools that can be used just for extracting the software metrics. Rudiger et al. [2] did an extensive comparison of many such tools and compared the possible different values returned per metric for each tool.

Besides assessment tools a lot of research has also been focused in finding new software metrics or evaluating the existing ones. Singla and Singh [3] created a classification of various software metrics for the different development phases of software. They separated various software metrics according to the project phase they were more relevant. The metrics we separated to Requirement Metrics, Design Metrics and Testing Metrics. For these three metric classes, they even further clustered the various metrics into sub-categories. Rosenberg and Hyatt [4] proposed and analyzed a list of 9 software metrics for object-oriented environments. They investigated if the traditional software metrics are also useful for object-oriented programs, and whether they behave differently there and also checked specific object-oriented metrics. The metrics they used were 3 traditional metrics: Cyclomatic Complexity, Size and Comment Percentage and 6 object-oriented metrics: Weighted Methods per Class, Response for a Class, Lack of Cohesion of Methods, Coupling Between Object Classes, Depth of Inheritance Tree and Number of Children. A similar work was also done by Harisson et al. [5] where they evaluated a list of object-oriented metrics called the MOOD metrics. These metrics are the: Method Hiding Factor, Attribute Hiding Factor, Method Inheritance Factor, Attribute Inheritance Factor, Coupling Factor and Polymorphism Factor. Through their work they verified the validity of the information contained in these metrics. Fenton and Neil [6] did an analysis of the existing software metrics and their success in relation to industry adoption. Through their research they show that there is a lot of academic work related to software metrics, but there is a big gap to the industry adoption, and much of the academic work is not relevant to the industry. More recently Arvanitou et al. [7] introduced the Software Metric Fluctuation (SMF). This is a property that can be used to quantify the changes of a software metric between different software versions. With this they proved that different metrics can have very different SMF values even if they describe the same software property and that source code metrics are much more sensitive to changes that design level metrics. Demyanova et al. [8] proposed an empirical list of software metrics. These make use of variable usage patterns, loop patterns and indicators of control flow complexity that they extract from the source code and proved that their metrics can be used for software verification.

Using software metrics to estimate reusability has always been a very popular and important research area as well. Sharma [9] used Support Vector Machines to predict the reusability for Function based Software systems. They used a few software metrics in great effect in order to predict the reusability of a pre-labeled dataset. The metrics they used are: Cyclometric Complexity, Halstead Software Science Indicator, Regularity Metric, Reuse-Frequency Metric and Coupling metric. Sethi and Tandon [10] explored the differences in the reusability and extracted metrics of having the same application but developed with two different ways, one with inheritance and one with interfaces. Their work concluded that interfaces show greater reusability than inheritance. Bhambri and Chhabra [11] used a slightly less common approach, since they used clustering on software metrics in order to estimate reusability. Using K-Means they clustered their dataset into 8 different reusability classes, and to evaluate their method, they used a test dataset that matched their expected results.

In most mentioned cases so far, the reusability of software was apriori knowledge, verified by experts that had manually checked the code. However there have been proposed ways to try to quantify it and create a reusability metric. Huda et al. [12] created such a metric that quantifies the reusability of object-oriented design. This metric takes into account the coupling, inheritance and encapsulation of the program. They evaluated their metric against popular windows application frameworks. A similar work was also performed by Sadana et al. [13], where they used object-oriented metrics for cohesion and coupling, in order to calculate their proposed reusability metric. Through their work they found a correlation between high cohesion and high reusability, and also correlation between low coupling and high reusability.

So far, we have mainly seen research that tries to quantify the reusability through metrics or expert opinion. There is also another approach that has been emerging the last few years where this evaluation is done through the community's opinion. Papamichail et al. [14] proposed a methodology for computing a quality score for software components, based of software metrics and the popularity of software to other developers. For this they used the most popular projects found in GitHub and generated a formula that assigns a score to each software component based on the number of GitHub Stars, the number files in the GitHub repository and the dependents of each file. Their proposed system employs a Support Vector Machine one-class classifier that filters out low quality files and then an artificial neural network to perform the reusability predictions.

3 Software Metrics

The aim of this dissertation is to use the Stars and Forks of GitHub as a metric for software reusability. In order to do this, we will gather these data from GitHub. We will also use various other software metrics in order to train classifiers and try to predict software reusability. In this chapter we will review the metrics that will be used for this process.

3.1 GitHub

GitHub [15] is a collaborative code hosting site that has been built on top of one of the most popular version control systems, git. It allows developers to host their own public and private repositories, and also includes various social features that allow the developers to collaborate with each other. It is currently the largest and most popular code hosting site [16]. This means that its users range from all levels of experience, starting from novice users and going all the way up to professional developers with many years of experience. Similarly, the projects there highly vary in quality. Due to the social and collaboration features of GitHub though, it allows the popular and better project to stand out above the rest. This is done via the Stars and Forks features.

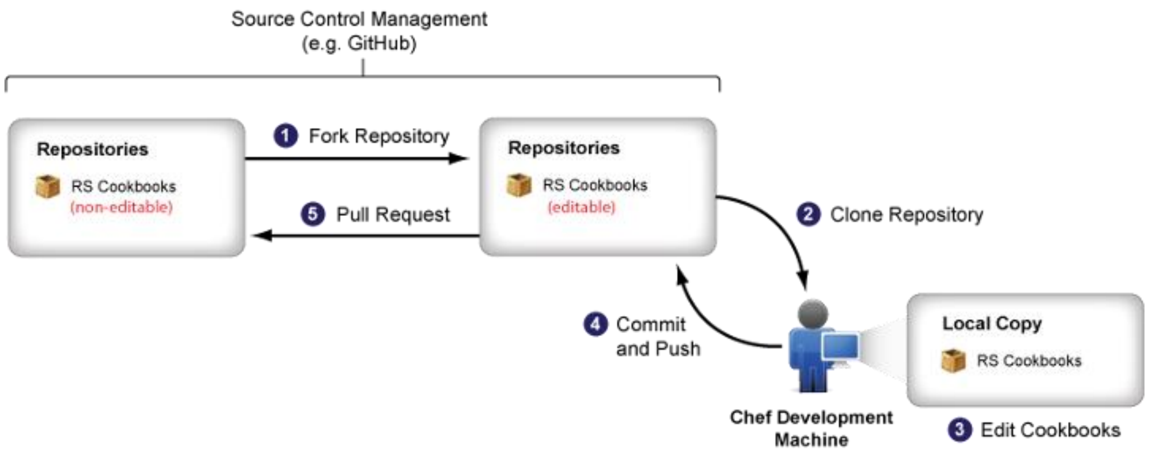
3.1.1 Stars

Stars in GitHub are a way for users to keep track of projects they find interesting and want to follow. It also allows them to find similar projects to the ones they have already starred through recommendations. Stars are generally used as a means of showing the popularity and importance of a project. GitHub allows users to search projects sorted by their number of Stars in order to find the most popular projects depending on the search criteria.

3.1.2 Forks

Forks allow users to get a copy of another user's repository, so that they can manage it themselves. They can still get the updates of the original repository, but they can also freely commit their own changes on their own forked repository without affecting the original. From that point on, they can continue developing the project on their own by adding new features or fixing possible defects, but they can also issue a pull request to the original repository, which is a proposal to merge their changes with the original repository. In Image 1 we can see how forking a repository works.

Image 1: Forking A Repository



3.2 Static Analysis Metrics

3.2.1 Cyclomatic Complexity

Cyclomatic Complexity, or sometimes McCabe Complexity, is a metric that was proposed by McCabe [17] in order to try to quantify program complexity. This metric assumes that each control statement increases the complexity of a program.

Cyclomatic Complexity is calculated using the control flow graph [18] of a program. The metric can be calculated in two different but equivalent ways:

1. The number of control statements in a program +1
2. For a given graph G when n vertices, e edges and p connected components the Cyclomatic Complexity is:

$$V\ (G) = e - n + 2p$$

Cyclomatic Complexity is very popular since it is easy to be calculated and is very intuitive to understand, since the idea that adding extra conditions and loops to programs makes them more complex is fairly common.

3.2.2 Lines of Code

This is the total number of lines of code in a program. It is used as a metric of counting the size of the program. It contains all the lines in a program including the comments.

3.2.3 Logical Lines of Code

This metric counts all the executable statements in a program. This means that it counts all the lines in a program, excluding the comments. But it also expands the lines that contain multiple statements, into one statement per line. For example, the following statement contains 2 Logical Lines of Code (one for the "for" statement, and one for the "print" statement) instead of just 1:

```
for (i = 0; i < 10; i++) print("Hello World")
```

3.2.4 Source Lines of Code

This metric contains all the source code lines of a program. This means all the lines, excluding the comment lines.

3.2.5 Comment Lines

This metric contains the number of comment lines in the program.

3.2.6 Multi-line Strings

This metric contains the number of lines that are part of multi-line strings.

3.2.7 Blank Lines

This metric contains the number of blank lines.

3.2.8 Halstead Metrics

Halstead Metrics are complexity metrics that were proposed by Halstead in 1977 [19]. They allow for estimations on the testing time, vocabulary, mistakes and effort. For their calculations they rely on the numbers of operators and operands that are used by the source code. As Halstead said, "A PC program is an execution of a calculation thought to be an accumulation of tokens which can be named either operators or operands" [20].

In order to calculate these, we first have to define the following:

n_1 = number of unique or distinct operators

n_2 = number of unique or distinct operands

N_1 = total number of occurrences of operators

N_2 = total number of occurrences of operands

The Halstead measures that were used can be calculated as follows [20]:

Program Vocabulary

This is the sum of the number of unique operators and operands throughout the program.

$$\mathrm{n} = \mathrm{n}_1 + \mathrm{n}_2$$

Calculated Program Length

This is the sum of the total operators and operands throughout the program.

$$\mathrm{N} = \mathrm{N}_1 + \mathrm{N}_2$$

Volume

This is the size of the implementation of an algorithm.

$$V = (N_1 + N_2)\ \log_2(n_1 + n_2)$$

OR

$$\mathrm{V} = \mathrm{N}\ \log_2 \mathrm{n}$$

Difficulty

This is proportional to the number of unique operators and the total usage of operands. This difficulty estimates how difficult the code is to write, or to understand when reviewing it.

$$\mathrm{D} = \frac{2}{\mathrm{n}_1}\ \frac{\mathrm{n}_2}{\mathrm{N}_2}$$

Effort

This measures the effort required for implementing or understanding the program. This is proportional to the difficulty and volume.

$$\mathrm{E} = \mathrm{D}\ \mathrm{V}$$

Time

This is the estimated time to write the program in seconds, and is based on the effort.

$$\mathrm{T} = \frac{\mathrm{E}}{\mathrm{S}}$$

$$S=18$$

Bugs

This is the expected number of bugs or errors for the program.

$$B = \frac{E^{0.667}}{3000}$$

4 Methodology

This chapter will describe the methodology that was used for gathering the data used to perform the experiments.

4.1 Data Collection

As mentioned GitHub Projects were used to extract the source code from. A list of Python 3 projects were selected in order to build a dataset. These projects range from some of the most popular projects on GitHub (based on their Stars), and some projects with upcoming popularity that were trending in the weeks and days of November 2017.

The projects that were selected are 66 and amount to 3023108 total lines of code. In Table 1: GitHub Projects you can find the list of the 66 selected projects, along with a brief description. The descriptions were taken from each project's main page.

Table 1: GitHub Projects

Project Name	Description
Algorithms	Minimal examples of data structures and algorithms in Python.
AngelSword	CMS3 prepared by the Python vulnerability detection framework.
Ansible	Ansible is a radically simple IT automation platform that makes your applications and systems easier to deploy.

Awesome-python	A curated list of awesome Python frameworks, libraries, software and resources.
Big-list-of-naughty-strings	The Big List of Naughty Strings is a list of strings which have a high probability of causing issues when used as user-input data.
Bitcoinbook	Mastering Bitcoin 2nd Edition - Programming the Open Blockchain.
CapsNet-Keras	A Keras implementation of CapsNet in NIPS2017 paper "Dynamic Routing Between Capsules".
CapsNet-Tensorflow	A Tensorflow implementation of CapsNet(Capsules Net) in Hinton's paper Dynamic Routing Between Capsules.
Capsule-networks	A PyTorch implementation of the NIPS 2017 paper "Dy-namic Routing Between Capsules".
Certbot	Certbot is EFF's tool to obtain certs from Let's Encrypt and (optionally) auto-enable HTTPS on your server. It can also act as a client for any other CA that uses the ACME protocol.
Chinese-Text-Classification	Chinese-Text-Classification, Tensorflow CNN (Convo-lutional Neural Network) to achieve the Chinese text classification.
Compose	Define and run multi-container applications with Docker.
CppCoreGuidelines	The C++ Core Guidelines are a set of tried-and-true guidelines, rules, and best practices about coding in C++.
Cr3dOv3r	Know the dangers of credential reuse attacks.
Data-science-ipython-note-books	Data science Python notebooks: Deep learning (Tensor-Flow, Theano, Caffe, Keras), scikit-learn, Kaggle, big data (Spark, Hadoop MapReduce, HDFS), matplotlib, pandas, NumPy, SciPy, Python essentials, AWS, and various command lines.
DeepAA	Make Ascii Art by Deep Learning.

Deep-Learning-Papers-Reading-Roadmap	Deep Learning papers reading roadmap for anyone who are eager to learn this amazing tech.
Diracnets	Training Very Deep Neural Networks Without Skip-Connections.
Django-rest-framework	Web APIs for Django.
Django	The Web framework for perfectionists with deadlines.
Dramatiq	Simple distributed task processing for Python 3.
Fabric	Simple, Pythonic remote execution and deployment.
face_recognition	The world's simplest facial recognition api for Python and the command line.
Flashtext	Extract Keywords from sentence or Replace keywords in sentences.
Flask	A microframework based on Werkzeug, Jinja2 and good intentions.
Glances	Glances an Eye on your system. A top/htop alternative.
Home-assistant	Open-source home automation platform running on Python 3.
Httpie	Modern command line HTTP client – user-friendly curl alternative with intuitive UI, JSON support, syntax highlighting, wget-like downloads, extensions, etc.
Incubator-mxnet	Lightweight, Portable, Flexible Distributed/Mobile Deep Learning with Dynamic, Mutation-aware Dataflow Dep Scheduler; for Python, R, Julia, Scala, Go, Javascript and more.
Incubator-superset	Apache Superset (incubating) is a modern, enterprise-ready business intelligence web application.
Interactive-coding-challenges	Interactive Python coding interview challenges (algorithms and data structures). Includes Anki flashcards.
Keras	Deep Learning library for Python. Runs on TensorFlow, Theano, or CNTK.

Localstack	A fully functional local AWS cloud stack. Develop and test your cloud apps offline.
Material-theme	Material Theme, the most epic theme for Sublime Text 3 by Mattia Astorino.
Mentalist	Mentalist is a graphical tool for custom wordlist generation. It utilizes common human paradigms for constructing passwords and can output the full wordlist as well as rules compatible with Hashcat and John the Ripper.
Mitmproxy	An interactive TLS-capable intercepting HTTP proxy for penetration testers and software developers.
Models	Models and examples built with TensorFlow.
Pandas	Flexible and powerful data analysis / manipulation library for Python, providing labeled data structures similar to R data.frame objects, statistical functions, and much more.
phishing_catcher	Phishing catcher using Certstream.
pretrained-models.pytorch	Pretrained ConvNets for pytorch: NASNet, ResNeXt101, ResNet152, InceptionV4, InceptionResnetV2, etc.
Pyro	Deep universal probabilistic programming with Python and PyTorch.
Pyschemes	PySchemes is a library for validating data structures in python.
Pyspider	A Powerful Spider(Web Crawler) System in Python.
Python-patterns	A collection of design patterns/idioms in Python.
Pytorch	Tensors and Dynamic neural networks in Python with strong GPU acceleration.
Reddit	Historical code from reddit.com.
Reinforcement-learning-an-introduction	Python implementation of Reinforcement Learning: An Introduction
Requests	Python HTTP Requests for Humans

Scikit-learn	Scikit-learn: machine learning in Python
Scrapy	Scrapy, a fast high-level web crawling & scraping framework for Python.
Source-code-pro	Monospaced font family for user interface and coding environments.
Sqlmap	Automatic SQL injection and database takeover tool.
Sshuttle	Transparent proxy server that works as a poor man's VPN. Forwards over ssh. Doesn't require admin. Works with Linux and MacOS. Supports DNS tunneling.
SSRF-Testing	SSRF (Server Side Request Forgery) testing resources.
System-design-primer	Learn how to design large-scale systems. Prep for the system design interview. Includes Anki flashcards.
Tangent	Source-to-Source Debuggable Derivatives in Pure Python.
Tensorforce	TensorForce: A TensorFlow library for applied reinforcement learning.
Thefuck	Magnificent app which corrects your previous console command.
Tornado	Tornado is a Python web framework and asynchronous networking library, originally developed at FriendFeed.
TuSimple-DUC	Understanding Convolution for Semantic Segmentation.
Micropython-upyphone	A gsm phone using pyboard and sim800l.
Utensor	AI inference library based on mbed and TensorFlo.
YouCompleteMe	A code-completion engine for Vim.
You-get	Dumb downloader that scrapes the web.
Youtube-dl	Command-line program to download videos from YouTube.com and other video sites.
ZeroNet	ZeroNet - Decentralized websites using Bitcoin crypto and BitTorrent network

All these projects were selected, in order to extract the software metrics mentioned in the previous section. They were first cloned from their GitHub repositories and then the software metrics were calculated for each project. This process was done via Python scripting and the software metrics were calculated using Radon [21]. Radon is an open source Python tool that compute various metrics from source code. Radon operates under the MIT License [22].

4.2 Predicting Reusability

As mentioned before the aim of the dissertation is to try and predict reusability based on the software metrics. And as a metric for reusability we are going to use the Stars and Forks of the project in GitHub.

As we discussed before Forks can be used as a metric for reusability. However, Forks by themselves are not very useful, since larger and more popular projects are bound to have a very high number of Forks regardless of how reusable they actually are. This is why both Stars and Forks were used as a metric. The metric that was used is the number of Forks over the number of Stars.

$$Reusability = \frac{Number\ of\ Forks}{Number\ of\ Stars}$$

Using this approach, we take into account both the reusability of the project and the popularity of it, so that popular projects don’t appear more reusable than they should.

After calculating the number Forks/Stars for each project the dataset had to be split into classes. It was then separated dataset into two reusability classes, ‘high’ and ‘medium’. And in order to set the class for each project the median number of the Forks/Stars of each project was set as a boundary. The median was 0.170198560205573, so every project above that is classified as high, and the rest as medium. This essentially splits the dataset into two equally numbered classes, so we have 33 samples of the ‘medium’ class and 33 of the ‘high’ class. In Table 2: Dataset Classes we can see the selected classes for each project, along with the numbers of Forks, Stars and their Forks/Stars score.

Table 2: Dataset Classes

Name	Forks	Stars	Forks/Stars	Class
algorithms	9621	1373	0.1427086581	medium

AngelSword	402	217	0.539800995	high
ansible	26762	9551	0.3568866303	high
awesome-python	41302	7890	0.1910319113	high
big-list-of-naughty-strings	22210	883	0.0397568663	medium
bitcoinbook	5359	1406	0.262362381	high
CapsNet-Keras	800	155	0.19375	high
CapsNet-Tensorflow	1405	345	0.2455516014	high
capsule-networks	421	56	0.1330166271	medium
certbot	20124	1846	0.0917312661	medium
Chinese-Text-Classification	101	19	0.1881188119	high
compose	11029	1769	0.1603953214	medium
CppCoreGuidelines	15669	1938	0.1236837067	medium
Cr3dOv3r	470	66	0.1404255319	medium
data-science-ipython-notebooks	10784	2878	0.2668768546	high
DeepAA	768	48	0.0625	medium
Deep-Learning-Papers-Reading-Roadmap	14629	2936	0.2006972452	high
diracnets	397	48	0.120906801	medium
django-rest-framework	9100	3011	0.3308791209	high
django	29708	12541	0.4221421839	high
dramatiq	327	12	0.0366972477	medium
fabric	9269	1512	0.1631243931	medium
face_recognition	7481	1400	0.1871407566	high
flashtext	621	70	0.1127214171	medium
flask	31106	9824	0.3158233138	high
glances	8670	632	0.0728950404	medium
home-assistant	10338	2944	0.2847746179	high
httpie	32633	2208	0.0676615696	medium
incubator-mxnet	12078	4450	0.3684384832	high

incubator-superset	16808	2624	0.1561161352	medium
interactive-coding-challenges	11376	1441	0.1266701828	medium
keras	21869	7969	0.3643970918	high
localstack	9380	460	0.0490405117	medium
material-theme	10031	703	0.0700827435	medium
mentalist	309	32	0.1035598706	medium
mitmproxy	8682	1255	0.1445519466	medium
models	23867	11516	0.4825072276	high
pandas	11749	4598	0.3913524555	high
phishing_catcher	420	90	0.2142857143	high
pretrained-models.pytorch	521	50	0.0959692898	medium
pyro	1993	180	0.0903161064	medium
pyschemes	346	11	0.0317919075	medium
pyspider	10145	2638	0.2600295712	high
python-patterns	13212	3084	0.2334241599	high
pytorch	9288	1966	0.2116709733	high
reddit	14163	2592	0.1830120737	high
reinforcement-learning-an-introduction	2146	859	0.4002795899	high
requests	28678	5266	0.1836250785	high
scikit-learn	23048	12144	0.5269003818	high
scrapy	23905	6123	0.2561388831	high
source-code-pro	11795	1038	0.0880033913	medium
sqlmap	9947	2314	0.2326329547	high
sshuttle	8740	733	0.0838672769	medium
SSRF-Testing	389	77	0.1979434447	high
system-design-primer	20997	2469	0.1175882269	medium
tangent	1206	63	0.052238806	medium
tensorforce	964	142	0.1473029046	medium

thefuck	32188	1578	0.0490244812	medium
tornado	14626	4321	0.2954327909	high
TuSimple-DUC	220	39	0.1772727273	high
Micropython-upyphone	334	19	0.0568862275	medium
uTensor	405	35	0.0864197531	medium
YouCompleteMe	15094	1726	0.1143500729	medium
you-get	31145	3191	0.201363034	high
youtube-dl	31145	5824	0.1869963076	high
ZeroNet	10290	1300	0.1263362488	medium

After having set the classes for each project Weka [23] was used in order to try to predict and evaluate those classes. For all the experiments that will follow in the next chapter, 10-fold cross validation was used. The classifiers that were used are some of the most popular that perform well in a wide range of problems. The different classifiers will be compared with each other in regards to Accuracy, Precision, Recall and F-Measure and the results will be interpreted in each case. Each classifier was also tested with various parameters in order to try and achieve the best and most balanced results.

5 Results

For the actual testing, a range of different classifier was used, in order to see how the dataset behaves under each different classifier. The classifiers that were tested are:

- Naive Bayes
- J48
- Random Forest
- Multilayer Perceptron
- Support Vector Machines

5.1 Naive Bayes

Naive Bayes Classifier [24] is one of the simplest yet highly efficient existing classifier. It is very easily understood and also has great performance. Despite these Naive Bayes has proven to be able to complete with much more complex classifiers in terms of results. In Table 3: Naïve Bayes we can see the results from Naive Bayes Classifier on our dataset.

Table 3: Naïve Bayes

Class	Accuracy	Precision	Recall	F-Measure
medium	-	0.600	0.909	0.723
high	-	0.813	0.394	0.531
	65.15115	0.706	0.652	0.627

5.2 J48

J48 is the open source implementation of the C4.5 algorithm for decision tree generation developed by Quinlam [25]. In Table 4: J48 we can see the results for the J48 algorithm.

Table 4: J48

Class	Accuracy	Precision	Recall	F-Measure
medium	-	0.585	0.939	0.721
high	-	0.846	0.333	0.475
	63.6364	0.716	0.636	0.600

5.3 Random Forest

Random Forests [26] are an ensemble learning method of many decision tree classifiers. The max depth for the decision trees was set to 1, since the results greatly decreased for higher values. The results can be seen in Table 5: Random Forest.

Table 5: Random Forest

Class	Accuracy	Precision	Recall	F-Measure

medium	-	0.600	0.909	0.723
high	-	0.813	0.394	0.531
	65.1515	0.706	0.652	0.627

5.4 Multilayer Perceptron

MLP or multilayer perceptron is a feedforward artificial neural network that is trained with the backpropagation algorithm [27]. MLPs consist of at least three layers, an input and output layer, and one or more hidden layers. During the tests the best results were found when using a single hidden layer of 30 nodes. The learning rate was 0.3, the momentum 0.2 and the network was trained for 50 epochs. The results can be found in Table 6: Multilayer Perceptron.

Table 6: Multilayer Perceptron

Class	Accuracy	Precision	Recall	F-Measure
medium	-	0.627	0.970	0.762
high	-	0.933	0.424	0.583
	69.697	0.780	0.697	0.673

5.5 Support Vector Machines

SVMs or Support Vector Machines [28] are a supervised learning model that tries to distinguish two classes mapped in an N-dimensional space with as clear and wide gap as possible. For the tests that were performed the SVMs were trained with the Stochastic Gradient Descent (SGD) [29] algorithm. The best results were found when training for 15 epochs with a learning rate of 0.25, and they can be found in Table 7: Support Vector Machines.

Table 7: Support Vector Machines

Class	Accuracy	Precision	Recall	F-Measure
medium	-	0.652	0.909	0.759
high	-	0.850	0.515	0.642

	71.2121	0.751	0.712	0.701

6 Evaluation

This chapter will briefly review the results from Chapter 5 and compare the different classifiers.

We can see that he accuracy from the different classifiers ranges from about 63% up to about 71%. The difference is not huge, but we can see that some models behave better than others. When looking more closely to the models we can see that MLPs and SVMs have the better results. These two are the more complex models, from those that were tested, and also those that cannot be easily interpreted by humans.

However, what is even more interesting is when we take a look at the precision and recall scores of each class. We can see that in all classifiers the two classes had a big difference in their precision and recall scores. In the ‘medium’ class the precision was as low as 0.585 for some classifiers, while the recall as high as 0.97. This means that there were too many samples classified as ‘medium’ even though they were not, leading to the low precision. Also, due to the high recall we can conclude that most samples of the ‘medium’ class were correctly classified as ‘medium’. In retrospect, for the ‘high’ class we can see that the precision was as high as 0.933 but the recall as low as 0.333. This means that many ‘high’ samples weren’t correctly classified as ‘high’, leading to the low recall, but when a sample was classified as ‘high’ it was correctly classified most of the time.

Taking into consideration what we have seen so far from the precision and recall of the two classes, we can conclude that many samples of the ‘high’ class are classified as ‘medium’, but not the other way around. The cause of this is most likely due to the way the ground truth of the classes was established. The classes were separated by their Forks/Stars score based on the median value. However, it seems that this split is not the best and there are some clear issues with it. This is a very clear area that need to be improved, and the next chapter will present some next steps that can be taken to improve this.

7 Conclusion and Future Work

In this Dissertation I have proposed a methodology for trying to assess software reusability. This methodology tries to move away from some more common approaches of manual code assessment or setting metrics thresholds, but aims to provide a way to assess the reusability based on community opinion. It tries to use the Stars and Forks of GitHub in order to assess this. The idea behind this is fairly straight forward: “If something is actually being reused by other people, then it can be considered as having good reusability”.

I used various different classifiers in order to try to predict the reusability of a list of GitHub projects that I separated into two different classes: ‘medium’ and ‘high’ reusability. From the results we can see that to some extent we can predict the classes of the dataset. The accuracy isn’t extremely high, but it shows some promising results, and that there is some merit to the whole process.

Of course, through the process we can see that there are also some areas that require further research and improvement. The separation of the classes is not optimal and wasn’t performed in a very efficient way. This became obvious when looking at the high different between the precision and recall of the two different classes.

Taking these into account, they allow us to see potential paths for future work on improving and researching new things. Improving the way, the classes are separated is a very clear thing that should be explored more. They should be separated in a more automatic way. This could be either through a statistical analysis of the dataset, or even better through clustering of the data in order to try to find the potential reusability classes.

Besides the class separation, there are many other areas that contain potential for future work. More projects could be utilized in order to create a larger dataset, containing more diverse projects in term of both reusability and popularity. Also, even more software metrics could be extracted from the said projects. Currently in this Dissertation a very important group of software metrics is missing, which is object-oriented metrics. This was because some of the python projects that were selected were not object-oriented projects, and thus those metrics could not be used. However, with different project selection, or even by selecting a different programming language like Java, those metrics can also be utilized in order to see how they affect the results.

In conclusion, I think this dissertation shows the importance of software reusability and the need to try to find new ways to assess it. In the current era, that we have so much data available, we should find new ways to utilize them. And even more importantly since we have so much information about the actual users and how they react, this should be the area to focus, because after all, how the users behave is the best evaluation we could possibly have.

Bibliography

[1] Xia Cai, Michael R. Lye, Kam-Fai Wong, Mabel Wong, "ComPARE: A Generic Quality Assessment Environment for Component-Based Software Systems", 2001

[2] Rudiger Lincke, Jonas Lundberg, Welf Lowe, "Comparing Software Metrics Tools" in Proceedings of the 2008 international symposium on Software testing and analysis (ISSTA '08), 2008

[3] Shefali Singla, Dheerendra Singh, "Classification of Software Metrics and Open Source Tools for Software Development Phase", 2014

[4] Linda H. Rosenberg, Lawrence E. Hyatt, "Software Quality Metrics for Object-Oriented Environments" in Crosstalk Journal, 1997

[5] R. Harrison, S.J. Counsell, R.V. Nithi. "An Evaluation of the MOOD Set of Object-Oriented Software Metrics" in IEEE Transactions of Software Engineering, 1998

[6] Norman E Fenton, Martin Neil, "Software Metrics: Successes, Failures and New Directions" in Journal of Systems and Software, 1998

[7] Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Paris Avgeriou, "Software metrics flunctuation: a property for assisting the metric selection process" in Information and Software Technology, 2016

[8] Yulia Demyanova, Thomas Pani, Helmut Veith, Florian Zuleger, "Empirical software metrics for benchmarking of verification tools" in Formal Methods in System Design, 2017

[9] Upasana Sharma, Parminder Singh, Parvinder S. Sandhu, "Investigating Performance of SVM Based Classifier for Software Reusability Prediction"

[10] Annushri Sethi, Ritu Tandon, "A Novel Approach to Find Reusability using Coupling and Cohesion Metrics", 2017

[11] Shweta Bhambri, Sheetal Chhabra, "Estimation of Software Reusability Based on Clustering" in International Journal of Engineering Science, 2016

[12] Mahfuzul Huda, Yagya Dutt Sharma Arya, Mahmoodul Hasan Khan, "Quantifying Reusability of Object Oriented Design: A Testability Perspective" in Journal of Software Engineering and Applications, 2015

[13] Neha Sadana, Surender Dhaiya, Manjot Singh Ahuja, "A Metric for Assessing Reusability of Software Components" in International Journal of Computer Application, 2014

[14] Michail Papamichail, Themistoklis Diamantopoulos, Andreas Symeonidis, "User-Perceived Source Code Quality Estimation based on Static Analysis Metrics" in 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2016

[15] https://github.com/

[16] F. Thung, T.F. Bissyande, D. Lo and L. Jiang, "Network structure of social coding in GitHug" in CSMR. IEEE, 2013

[17] Thosmas J. McCabe, "A Complexity Measure" in IEEE Transaction on Software Engineering, 1976

[18] Frances E. Allen, "Control Flow Analysis" in SIGPLAN Notices, 1970

[19] Maurice H. Halstead, "Elements of Software Science", 1977

[20] Chandra Segar Thirumalai, Hariprasad Thirunavukkarasu, G. Vidhyagaran, K. Seenu, "Software Complexity Analysis Using Halstead Metrics" in International Conference on Trends in Electronics and Informatics, 2017

[21] https://pypi.python.org/pypi/radon

[22] https://opensource.org/licenses/MIT

[23] https://www.cs.waikato.ac.nz/ml/weka/

[24] I. Rish, "An empirical study of the naive Bayes classifier", 2001

[25] Ross J. Quinlan, "C4.5: Programs for Machine Learning", Morgan Kaufmann Publishers, 1993

[26] Leo Breiman, "Random Forests. Machine Learning", 2001

[27] David E. Rumelhard, Geoffrey E. Hinton, R. J. Williams, "Learning Internal Representations by Error Propagation", MIT Press, 1986

[28] C. Cortes, V. Vapnik, "Support-vector networks", 1995

[29] Leon Bottou, “Large-Scale Machine Learning with Stochastic Gradient Descent”, 2010

Appendix

In this appendix you can find the dataset that was created and used for this Dissertation. You will notice that the dataset contains some extra attributes that were not used during the testing, but were added to the dataset for completeness purposes. These are the name of the GitHub Project, the Stars, the Forks and the Forks/Stars score. They were excluded from the testing because the name cannot really be used at all, and the numbers of Stars, Forks and Forks/Stars score were used to create the class attribute, and thus shouldn't be used at all for the testing. However, they can be useful for other tests, like clustering the dataset.

```
@RELATION github



@ATTRIBUTE name STRING

@ATTRIBUTE cc NUMERIC

@ATTRIBUTE loc NUMERIC

@ATTRIBUTE lloc NUMERIC
@ATTRIBUTE sloc NUMERIC
@ATTRIBUTE comments NUMERIC
@ATTRIBUTE single NUMERIC
@ATTRIBUTE multi NUMERIC
@ATTRIBUTE blank NUMERIC
@ATTRIBUTE vocabulary NUMERIC
@ATTRIBUTE length NUMERIC
@ATTRIBUTE calculated_length NUMERIC
@ATTRIBUTE volume NUMERIC
@ATTRIBUTE difficulty NUMERIC
@ATTRIBUTE effort NUMERIC
```

```
@ATTRIBUTE time NUMERIC
@ATTRIBUTE bugs NUMERIC
@ATTRIBUTE stars NUMERIC
@ATTRIBUTE forks NUMERIC
@ATTRIBUTE stars_forks NUMERIC
@ATTRIBUTE class {medium,high}


@DATA
algo-
rithms,3.4282560706401766,7423,3908,3921,806,743,1579,1180,2979,4939,1
1179.001446928209,22842.228602745636,561.2721951809662,121388.14428153
76,6743.785793418764,7.6140762009152185,9621,1373,0.1427086581,medium
An-
gelSword,2.986344537815126,13942,8544,9668,800,800,1950,1524,6437,8596
,25237.311981696865,39983.70954557437,582.9249711168869,105597.5529983
1102,5866.530722128393,13.32790318185815,402,217,0.539800995,high
ansi-
ble,5.789847484634646,711063,291005,541514,47384,48913,16424,104212,11
4903,187379,711127.5988636708,1292045.9413218452,7830.294244028591,916
3274.86014063,509070.82556336746,430.6819804406146,26762,9551,0.356886
6303,high
awesome-py-
thon,7.5,80,43,42,13,13,11,14,28,35,115.65156546374811,168.25742227201
613,2.5,420.6435556800403,23.369086426668908,0.056085807424005374,4130
2,7890,0.1910319113,high
big-list-of-naughty-
strings,6.0,63,18,19,14,15,6,23,10,10,13.509775004326938,23.2192809488
7362,2.0,23.21928094887362,1.289960052715201,0.007739760316291207,2221
0,883,0.0397568663,medium
bitcoin-
book,3.0,103,56,56,18,18,0,29,14,21,30.75488750216347,61.8289214233104
4,1.9166666666666665,70.28615177913805,3.904786209952114,0.02060964047
4436815,5359,1406,0.262362381,high
CapsNet-
Keras,2.3,472,221,236,56,40,113,83,147,228,722.1461238084346,1285.3801
970067645,19.087071718931476,8246.860920439078,458.15894002439325,0.42
84600656689216,800,155,0.19375,high
```

```
CapsNet-Tensor-
flow,5.391304347826087,813,413,449,101,98,90,176,268,427,1311.38182213
08604,2347.430502177923,25.520277688528708,9902.076842897837,550.11538
01609909,0.7824768340593076,1405,345,0.2455516014,high
capsule-net-
works,1.894736842105263,263,169,177,7,7,5,74,90,171,536.0524269110014,
1110.1068794723744,7.5886075949367084,8424.16549675555,468.00919426419
72,0.37003562649079147,421,56,0.1330166271,medium
certbot,2.3913934426229506,56049,25981,33261,4902,5020,6023,11745,5344
,7790,27352.76072229173,45996.12453772269,496.4066112075863,276383.094
5690647,15354.616364948037,15.33204151257423,20124,1846,0.0917312661,m
edium
Chinese-Text-Classifica-
tion,3.9166666666666665,532,306,340,72,72,42,78,66,94,216.892628700004
65,389.80379526207435,11.382417582417583,1200.2835195287334,66.6824177
515963,0.12993459842069144,101,19,0.1881188119,high
com-
pose,3.5673239436619717,24855,12799,19556,278,278,698,4323,4040,6216,2
6664.845083475368,44243.50155169116,206.8809235719723,273017.421688156
8,15167.63453823094,14.747833850563724,11029,1769,0.1603953214,medium
CppCoreGuide-
lines,6.769633507853404,6719,2516,3446,1345,1329,978,966,1696,3333,173
32.693928589713,34804.936052636396,23.965218793678098,471947.234899542
47,26219.290827752357,11.6016453508788,15669,1938,0.1236837067,medium
Cr3dOv3r,2.8333333333333335,310,158,217,45,39,21,33,107,183,631.033341
37421,1123.0905797411885,2.381818181818182,1329.295439634382,73.849746
64635456,0.3743635265803961,470,66,0.1404255319,medium
data-science-ipython-note-
books,2.4121951219512194,6687,3117,3398,242,275,1458,1556,1445,2386,89
02.973652495555,16090.69684662866,105.30646285479915,132187.7850211759
2,7343.765834509771,5.363565615542886,10784,2878,0.2668768546,high
DeepAA,2.1,347,251,268,25,23,0,56,164,312,949.879998509391,1985.007926
1178062,15.866228070175438,15766.5226756918,875.9179264273223,0.661669
3087059354,768,48,0.0625,medium
Deep-Learning-Papers-Reading-
Roadmap,2.6666666666666665,132,112,110,1,1,0,21,47,68,230.130686535627
7,377.71204191407935,4.512820512820513,1704.5466506891785,94.697036149
3988,0.12590401397135978,14629,2936,0.2006972452,high
```

```
dirac-
nets,2.327272727272727,763,476,535,6,8,68,152,208,394,1033.04023998152
96,2253.8555478402986,24.596288998357966,15725.691711163223,873.649539
5090678,0.7512851826134328,397,48,0.120906801,medium
django-rest-frame-
work,3.2466414054426456,33332,18098,22587,1186,1130,3464,6151,5156,832
4,30234.620225937764,53664.91707286757,313.71507786109845,299787.32326
90032,16654.851292722404,17.88830569095585,9100,3011,0.3308791209,high
django,2.4837589605734767,318455,172348,222212,20489,20823,24814,50606
,31385,48895,175838.98531993537,308318.80917913246,2544.1612358875254,
2040194.939875996,113344.16332644377,102.77293639304324,29708,12541,0.
4221421839,high
drama-
tiq,2.5481481481481483,5883,3009,3289,566,489,784,1321,753,1004,2779.2
030389826928,4582.219370478585,112.19739373633,18612.86880578363,1034.
048266987979,1.5274064568261951,327,12,0.0366972477,medium
fab-
ric,2.7633711507293355,8765,4344,5023,659,646,1658,1438,1156,1726,6109
.813843252351,10245.815452333063,89.46221480620058,55984.012134556906,
3110.222896364272,3.415271817444354,9269,1512,0.1631243931,medium
face_recogni-
tion,1.9166666666666667,1638,649,830,346,340,85,383,242,355,1040.79317
61683576,1802.4373419998574,25.925595238095237,5540.131020551554,307.7
850566973085,0.6008124473332858,7481,1400,0.1871407566,high
flashtext,5.0,949,440,503,120,108,179,159,107,257,498.5305460230142,15
00.2185492925153,14.161764705882353,13768.070328097627,764.89279600542
36,0.5000728497641717,621,70,0.1127214171,medium
flask,3.591187270501836,15706,7820,8891,951,999,2373,3443,2732,4348,17
249.356167331083,29834.387527457588,150.62898664359838,158860.64519009
54,8825.591399449746,9.944795842485863,31106,9824,0.3158233138,high
glances,4.801418439716312,19138,9109,10122,4150,4842,1127,3047,3853,58
58,21291.849087373004,37104.70450645701,346.5619815079076,291429.61104
821897,16190.533947123276,12.368234835485675,8670,632,0.0728950404,me-
dium
home-assis-
tant,2.8410857572718156,250812,132911,173556,5530,21294,7738,48224,317
20,47787,150643.84651282776,261233.85789792633,3049.589943813649,12120
89.1575952151,67338.28653306731,87.0779526326419,10338,2944,0.28477461
79,high
```

```
httpie,3.7818574514038876,6497,3155,4416,382,378,396,1307,1753,2685,97
43.641080772755,16436.40095729674,111.05220639986848,73471.39140759644
,4081.743967088692,5.478800319098911,32633,2208,0.0676615696,medium

incubator-
mxnet,3.8516816377689578,114497,57649,66400,13520,13615,19134,15348,26
609,48221,163970.01366514707,331363.0041014389,2161.506196044087,30574
88.2376696137,169860.45764831212,110.45433470047955,12078,4450,0.36843
84832,high

incubator-super-
set,3.501605136436597,24981,11586,19581,971,1069,927,3404,2690,4484,17
205.67155578989,31419.58072079948,164.93497515538468,268399.9439629197
6,14911.10799793999,10.473193573599826,16808,2624,0.1561161352,medium

interactive-coding-chal-
lenges,1.6223628691983123,4446,3197,3291,46,42,0,1113,738,961,1708.629
1137599258,3247.884165940303,111.33660989548083,7926.2240958386465,440
.3457831021451,1.0826280553134342,11376,1441,0.1266701828,medium

keras,4.427700348432055,58260,28891,35683,2522,2385,10838,9354,12666,2
3063,91753.39690120035,177341.24942191428,691.9652587670097,1668314.16
78594071,92684.12043663376,59.11374980730481,21869,7969,0.3643970918,h
igh

lo-
calstack,3.6402535657686212,10038,6446,7525,638,634,209,1670,2577,3947
,14899.220916655968,25160.449778018665,177.28194851131593,153806.65748
97356,8544.814304985313,8.386816592672886,9380,460,0.0490405117,medium

material-
theme,2.7142857142857144,872,369,647,8,20,12,193,119,163,462.214166885
2933,776.6925931930143,15.220454545454547,3190.472260881502,177.248458
93786122,0.2588975310643381,10031,703,0.0700827435,medium

mental-
ist,4.041284403669724,3315,2263,2356,126,84,308,567,588,1047,3494.4819
550419966,7004.470711533335,52.46746835909434,64055.01083864263,3558.6
117132579243,2.3348235705111113,309,32,0.1035598706,medium

mitmproxy,3.92435987784825,53891,32237,39999,1968,1460,3221,9211,10736
,16811,55527.39896960162,98321.74859488639,906.3571874791853,469329.40
749024093,26073.85597168007,32.773916198295495,8682,1255,0.1445519466,
medium

mod-
els,3.6103177436629776,156203,66898,92558,16327,16929,22379,24337,2352
7,41692,126785.7018723069,258160.2090285164,2444.5329605924576,2050200
```

```
.9389762299,113900.05216534589,86.05340300950535,23867,11516,0.4825072
276,high
pan-
das,4.823361988847584,328833,183394,218644,20910,20344,26426,63419,545
92,107927,411160.21470631,857614.5952806079,2699.6779460834055,9673668
.689728705,537426.0383182615,285.8715317602024,11749,4598,0.3913524555
,high
phish-
ing_catcher,15.0,300,67,235,33,23,7,35,48,84,239.31111664374467,469.13
685006057716,4.780487804878049,2242.7029905334907,124.59461058519393,0
.1563789500201924,420,90,0.2142857143,high
pretrained-mod-
els.pytorch,1.559782608695652,4391,1954,3660,492,69,114,548,276,643,11
74.84145525732,2531.0715483043496,68.95337967998066,15947.458576002893
,885.9699208890496,0.8436905161014497,521,50,0.0959692898,medium
pyro,3.228601252609603З,15503,8382,9489,1063,942,2284,2788,3814,6319,2
0919.71980741966,39510.6701977952,362.422007056042,276199.8675498222,1
5344.437086101236,13.17022339926507,1993,180,0.0903161064,medium
pyschemes,2.7142857142857144,425,283,319,3,14,0,92,51,74,208.307370823
92017,356.0507275842214,5.974137931034483,1301.0174732459996,72.278748
51366664,0.11868357586140715,346,11,0.0317919075,medium
pyspi-
der,3.490353697749196,16500,10556,12511,879,916,524,2549,2976,4885,159
75.983991906243,29671.057889394688,276.2011334747333,213053.2946046672
6,11836.294144703734,9.89035262979823,10145,2638,0.2600295712,high
python-pat-
terns,1.3380726698262244,4758,2409,2493,549,579,546,1140,441,571,1438.
9076473306388,2328.284480952383,50.3906185300207,5056.986520173977,280
.94369556522076,0.7760948269841277,13212,3084,0.2334241599,high
pytorch,3.0329995448338645,73299,38113,50257,2734,2587,6777,13678,1540
1,26566,93862.23925732012,180473.26784822086,1254.9954367340922,157352
1.7219445019,87417.8734413611,60.157755949406955,9288,1966,0.211670973
3,high
red-
dit,3.8410268487988697,64155,33080,42855,7155,7432,3022,10846,10353,17
718,70534.58305672788,134036.02304023068,674.0370422630574,1382428.715
0407312,76801.59528004064,44.67867434674358,14163,2592,0.1830120737,hi
gh
```

```
reinforcement-learning-an-introduc-
tion,3.34375,5222,3320,3320,1101,1108,44,750,1941,3543,11873.444136788
254,23376.124395410952,155.02777624156252,174517.16725220322,9695.3981
8067796,7.792041465136986,2146,859,0.4002795899,high
re-
quests,4.211221122112211,9368,4580,5632,859,843,1013,1880,1708,2720,12
071.357579016258,20672.670617282343,85.09237917280956,170393.320285093
74,9466.295571394096,6.890890205760781,28678,5266,0.1836250785,high
scikit-
learn,3.153128950695322,202941,85559,101747,14925,14813,46711,39670,34
497,63020,209260.01789946939,423145.4072274017,3062.0333731235282,4046
074.810818118,224781.93393434008,141.04846907580028,23048,12144,0.5269
003818,high
scrapy,2.4809437386569875,35659,21953,25772,1304,1217,1894,6776,4505,6
127,18841.938550256546,30585.464439161795,523.1045825128801,138076.648
2194148,7670.924901078605,10.195154813053934,23905,6123,0.2561388831,h
igh
source-code-
pro,3.25,216,124,149,19,16,8,43,20,26,70.2129994085646,112.37013046707
143,2.6666666666666665,299.6536812455238,16.64742673586243,0.037456710
15569048,11795,1038,0.0880033913,medium
sqlmap,5.509081196581197,42451,21669,28402,5891,3284,3180,7585,11506,2
0285,75362.15390836878,146796.63869434022,789.9725050660625,1327234.68
4346324,73735.26024146244,48.932212898113434,9947,2314,0.2326329547,hi
gh
sshut-
tle,4.974545454545455,5433,3457,4257,379,339,18,819,1673,2805,10460.06
9967064308,18828.69820996047,96.93241751988239,125436.4809840749,6968.
693388004162,6.2762327366534905,8740,733,0.0838672769,medium
SSRF-Test-
ing,3.21875,1049,898,935,21,20,0,94,409,624,2762.256384747253,4471.854
910466374,18.630689545109348,23737.23919705749,1318.7355109476382,1.49
06183034887914,389,77,0.1979434447,high
system-design-pri-
mer,1.6285714285714286,213,112,122,4,12,20,59,24,33,36.36452797660028,
78.30628370550306,3.7666666666666666,78.21022682391191,4.3450126013284
4,0.026102094568501027,20997,2469,0.1175882269,medium
tan-
gent,2.632197414806110 5,10544,5212,5826,1121,1198,1212,2308,1718,3032,
9304.505705195772,18358.8220624155,188.60273665519014,130578.759093403
29,7254.375505189071,6.1196073541385,1206,63,0.052238806,medium
tensor-
force,3.6666666666666665,17286,6734,9606,2357,2282,2494,2904,2993,4664
,15497.015915336342,27729.344650201165,317.8203228071778,188945.049627
14488,10496.947201508048,9.243114883400388,964,142,0.1473029046,medium
thefuck,2.3383838383838382,13038,6350,9698,213,227,209,2904,2799,3370,
7725.605389248328,12423.424877039552,401.29894562102004,27343.51640666
321,1519.0842448146227,4.141141625679865,32188,1578,0.0490244812,me-
dium
tor-
nado,2.5879167854089484,43586,23490,26420,4128,3976,5946,7244,5862,903
7,36452.00129403817,62211.35482116145,432.3917015513697,540659.1408079
123,30036.618933772912,20.73711827372049,14626,4321,0.2954327909,high
TuSimple-
DUC,3.8533333333333335,1367,827,979,149,121,63,204,391,693,1939.218742
9952594,3864.6398008580204,40.011549529282085,19679.086030049653,1093.
2825572249808,1.2882132669526736,220,39,0.1772727273,high
Micropython-upy-
phone,2.467948717948718,1164,949,944,32,16,3,201,306,588,1943.83233450
38,4056.4341292905615,22.256864803747376,35016.411777749345,1945.35620
98749635,1.3521447097635204,334,19,0.0568862275,medium
uTen-
sor,1.778061224489796,24864,2247,13609,237,232,7808,3215,605,1079,4293
.549718376219,8305.488702661878,38.998112891183574,101950.25616769199,
5663.903120427332,2.7684962342206263,405,35,0.0864197531,medium
YouCompleteMe,2.4240506329113924,12876,5824,8473,1511,1481,471,2451,10
30,1593,5085.001811549347,9144.745113226545,107.12266791667105,55615.7
26912340615,3089.7626062411455,3.0482483710755144,15094,1726,0.1143500
729,medium
you-
get,4.775919732441472,12598,8780,9459,810,684,444,2011,3824,6022,21620
.970753675763,38920.69415330699,329.80070118349073,297956.261972012,16
553.125665111784,12.973564717768992,15847,3191,0.201363034,high
youtube-
dl,5.2743609212857505,128906,51521,109997,4310,2638,1042,15229,20673,3
1427,113151.45329712417,200612.8121954109,1832.6888599489837,1645742.6
509081253,91430.14727267333,66.8709373984704,31145,5824,0.1869963076,h
igh
```

```
ZeroNet,4.238142785821268,27869,16204,17564,2633,2174,2563,5568,8981,1
5722,57519.940602795694,111204.47313801305,676.1681589682894,1209874.4
481182613,67215.2471176812,37.06815771267102,10290,1300,0.1263362488,m
edium
```