Clustering Scratch projects by code complexity traits and project traits

Brent Meeusen[1]

Supervisor(s): Fenia Aivaloglou[1]**, Sole Pera**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 26, 2024

Name of the student: Brent Meeusen
Final project course: CSE3000 Research Project
Thesis committee: Fenia Aivaloglou, Sole Pera, Jorge Martinez Castaneda

Abstract

Scratch is a popular, visual programming language aimed at children, and is used by teachers and after school code clubs to teach their students about programming. Measuring whether they understand the underlying concepts, however, is a difficult task. In this research, we tried clustering Scratch projects by complexity to help students improve their programming skills. We did this by selecting an existing data set to extract features that indicate code complexity. Before, researchers attempted clustering on one metric that globalises the project's complexity. Different researchers set out to measure the growth of the students by clustering the projects the students created. With that in mind, we adopt a partition-based clustering algorithm to cluster the projects, as this method indicates outliers. We examine the quality of these clusters using the silhouette coefficient. We set up five experiments with different input vectors to make out the impact each input has on the clusters. We did not find a clear indication of the projects being clustered by the selected features. This could mean that Scratch projects are not suitable to measure a high-level understanding of programming concepts. Including the project name in the input vector had a negligible effect on the outcome of the experiments.

1 Introduction

Scratch is a platform that introduces programming to a young audience by drag-and-dropping blocks of code. Most users are 8 to 18 years old[1]. To make the platform even more accessible, it is possible to follow tutorials or use another project as a starting point. The platform is immensely popular; every month, its website is visited by tens of millions of people[2].

As written by Falkner et al. in [3], several countries have a computer science curriculum to use in schools. For example, England already has a mandatory computer science course for children between the ages of 5 and 16 in state-funded schools. They also found that visual programming languages like Scratch are often used in these curricula, regardless of the age of the children attending school.

The formal classroom context is not the only one where Scratch is popular. Aivaloglou and Hermans [1] found that 89% of after school code clubs that participated in their study use Scratch to teach their students how to program. Code club lessons do not have to follow national curricula and students apply voluntarily for the program, in contrast to some school programs.

One reason for Scratch's popularity might be its design. The goal was to create an environment that is "more tinkerable, more meaningful, and more social than other programming environments" [15, p. 63]. Moreover, the makers wanted to make their platform easier to get started with than the other attempts made to get children and teens into programming [15]. Another reason could be that the users do not identify Scratch as a programming environment [9].

Even though the students who participated in the study presented in [9] did not feel like they were programming when using Scratch, they did use the programming concepts defined by the team more frequently after one year. For instance, in the first year, just over 30% of the projects used a loop, whereas one year later this had grown to almost 50%.

It seems to be difficult, however, to objectively measure whether students actually grasp the concept they have applied in their project. In [11], Meerbaum-Salant et al. tried mixing the Bloom and SOLO taxonomy together. Seiter and Foreman introduced a new model to assess computational thinking in grades 1 to 6 [17]. Nevertheless, it remains unclear whether starting in a block-based environment helps when transitioning to text-based programming [18]. If we manage to measure the student's understanding of computer science concepts, study materials for the different levels of comprehension could be developed. This could help students improve their programming skills.

To see if students make progress when using Scratch, we want to learn whether it is possible to cluster the projects on several code and project traits. We expect to find distinct groups of "simple" projects (e.g., two characters having a conversation) and "advanced" projects (e.g., playable games). If such clusters exist, it may be possible to predict what types of projects a user makes, and see if that changes over time (e.g., the first project a student creates is in a simple cluster, the project that same student creates a year later is in an advanced cluster). This could mean that the given user has learnt CS and/or Scratch concepts.

With this research, we hope to contribute to a better understanding of how children learn how to apply programming concepts. To accomplish that, we will try to find an answer to the research question:

> *"Could Scratch projects with a Dr. Scratch mastery score of 16 or above be clustered by different code complexity traits and project traits?"*

The Dr. Scratch mastery score is further explained in section 2.1. What code and project traits were selected and why is described in section 3.3.

In this research, we will attempt to cluster Scratch projects based on features that encapsulate the project's code complexity, as well as project features. The objective is to find projects with similar complexity.

All silhouette coefficients we found are below 0.50, which indicates that the quality of the clusters is not great. Usually, there is one large cluster with most of the projects, and many projects are considered outliers. Most of the other clusters are tiny. Apart from the largest clusters, the cluster sizes range from 10 to 235 projects.

The rest of the paper is organised as follows: in section 2, we take a look at related work that has already been done. In section 3, we elaborate on the methodology. Section 4 presents the results of the experiments, followed by the responsible research in section 5. In section 6, the discussion and the limitations of the research are described. Finally, sec-

[1] Data from https://scratch.mit.edu/statistics/. Visited on 2023-11-16.

[2] See footnote 1.

tion 7 lays out the conclusion and puts ideas forward for future works.

2 Related work

Dr. Scratch is a web application[3] that can be used to analyse Scratch projects [12]. From the project, it gives an integer score from 0 (inclusive) to 3 (inclusive) on seven concepts: abstraction and problem decomposition, logical thinking, synchronization, parallelism, algorithmic notions of flow control, user interactivity and data representation. The project's Dr. Scratch mastery score is the sum of the scores of the seven concepts, and thus ranges from 0 (inclusive) to 21 (inclusive). The criteria for each concept are presented in [12, Tab. 1]. Additionally, it can detect some bad programming habits, such as unreachable code or repetition of code.

We split the rest of this section in two parts. The first section specifically elaborates on researches that cluster Scratch projects, the second looks at clustering software repositories.

2.1 Clustering Scratch projects

There have been some researches that have tried to cluster Scratch projects. For example, in [14], Moreno-Léon et al. clustered projects by their Dr. Scratch mastery score. The same researchers found in [13] that software metrics and mastery score have a strong correlation. However, they noticed a growing gap between the projects and the best fitting line as the mastery score approaches 21 (the maximum score), possibly indicating that the maximum score is not suited well enough to cluster the most complex projects. This does not seem to affect [14] much, since most of the projects analysed have a mastery score of 16 or less, as shown in [14, Fig. 2].

Moreno-Léon et al. analysed 250 projects from 5 different categories: Art, Music, Animations, Stories and Games [14]. The mean mastery score for these categories are 7, 8, 8, 10, and 15, respectively. After running a K-means algorithm on their data set, they found three clusters where the cluster centres have mastery scores of 7, 9, and 17. The first cluster contains mostly art, music and animations. In the second cluster, story projects appear the most often. Games are the largest category in the last cluster.

In [19], Yang et al. clustered projects by first calculating the IDF of the blocks used in the projects, to then run K-means++ to cluster the projects. They selected 3852 users with at least 50 original projects. They found four clusters, all of which show an upwards learning curve over the 50 projects. The speed of the learning curve, however, was different for each cluster.

This work aims to find clusters of projects based on the complexity of the projects. As discussed above, only clustering by Dr. Scratch mastery score as done in [14] works best for projects with a lower mastery score. We aim to cluster projects with a mastery score of 16 and above by using more detailed code traits to determine the project's complexity. Both [14] and [19] used a centroid-based clustering algorithm. These algorithms are sensitive to outliers [5]. We will use a density-based clustering algorithm, because it can filter out outliers [5].

[3] http://drscratch.org/

2.2 Clustering software repositories

When broadening the view to software repositories instead of only looking at Scratch repositories, new ideas and techniques are discovered. For example, Jurczko and Madeyski [7] tied clustering software projects by 19 object-oriented metrics (e.g., Lack of Cohesion Of Methods, Number of Public Methods, Cohesion Among Methods, Cyclomatic Complexity). Even though Scratch does not support object-oriented programming, reading about the different metrics that were used gave us some ideas to incorporate in this research.

Kawaguchi et al. introduced a tool called MUDABlue in [8] that automatically categorises open source repositories. They applied Latent Semantic Analysis (LSA) on identifiers to determine what categories exist, to then cluster the repositories into the found categories. McMillan et al. also used LSA (in their paper, they refer to it as Latent Semantic Indexing, or LSI for short) in CLAN to look at API calls in Java projects [10].

Both MUDABlue and CLAN only look at the source code to cluster the projects. Rokon et al. introduced Repo2Vec, a tool that looks at metadata, source code and repository directory structure [16]. According to the authors, other tools only look at project's metadata. Borges and Valente [2] also looked at metadata; they clustered GitHub projects by number of stars. However, the data set that we use does not contain much metadata; the bottom 75% of the projects have both 0 remixes and 0 favourites, with at most 4 views [6, Tab. II]. Therefore, we will not take the metadata of the projects into account when clustering the Scratch projects.

3 Methodology

The goal of the research is to see if Scratch projects can be clustered by code traits and project traits. To achieve this, we need to 1) select a data set, 2) choose a clustering algorithm, 3) choose the features, 4) cluster the projects, and 5) analyse the clusters. This last step is described in section 4.

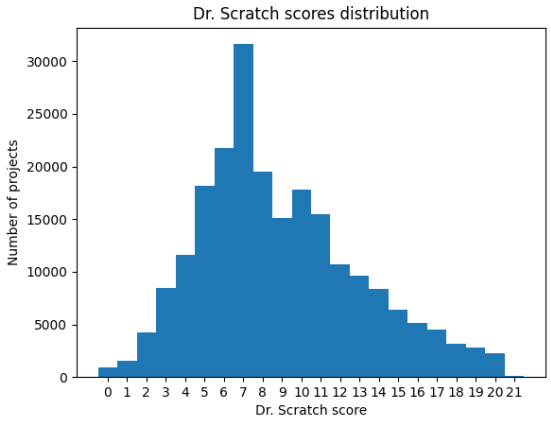
3.1 Selecting a data set

Initially, the idea was to use the data set generated by Aivaloglou and Zeevaarders [20] because it contains much information about the projects and users. We ran into multiple issues trying to initialise the database, however. Therefore, we decided to search for an alternative. We then looked into the data set generated by Hermans et al. [6]. Even though it contained fewer projects (233,491 instead of 1,019,310 non-empty projects) and less information about the projects, it was well documented and we were able to set it up without trouble.

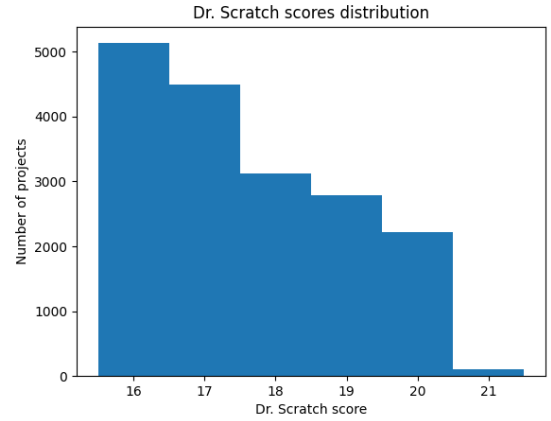
As presented in [6, Tab. III], the data set contains information on 250,163 projects, of which 233,491 are non-empty and 231,050 are analysed by Dr. Scratch. The results of the Dr. Scratch analysis can be found in the "Grades" table. The entire layout of the database can be found in [6, Tab. I].

To get an impression of the data, we generated a histogram of the Dr. Scratch mastery scores for all projects, displayed in figure 1a. As mentioned in section 2, Moreno-Léon et al.

found in [13] that the accuracy of the best fitting lines decreases significantly when the mastery score is 16 or above. Therefore, we chose to exclude the projects with a mastery score lower than 16. After this filter, the data set contains 17,868 projects. We generated a histogram of this new data set as well, seen in figure 1b.



(a) All projects



(b) Only projects with a Dr. Scratch mastery score ≥ 16

Figure 1: Histograms of the Dr. Scratch mastery scores of the projects.

3.2 Choosing a clustering algorithm

Instead of using a partitioning-based clustering method like K-means, we will implement a density-based clustering method. As reported by Han et al. in chapter 10 in [5], it can find clusters of any shape instead of only spherical clusters. They describe the DBSCAN, OPTICS and DENCLUE algorithms. We chose to work with the DBSCAN algorithm.

Checking the cluster quality

To identify the quality of the clusters, we will apply the silhouette coefficient as described in [5]. We will consider the clusters to be of good quality if the silhouette coefficient is 0.50 or above. Additionally, we will generate summaries of the clusters and analyse them manually.

3.3 Choosing the features

As seen in section 2.1, the Dr. Scratch mastery score appears to give a solid indication for the complexity of the project. However, as the mastery score approaches 21, the deviation between the best fitting line and the project in question increases. To also make clusters of these more complex projects, we want to choose features that represent some of the intricacies about the projects' complexity. Furthermore, since Scratch projects can be remixed, there might be clues in the names of the repositories. Similar names might indicate that the projects have been remixed, or have been created following the same course or tutorial.

Taking this into account, we selected the following features:

- Project name;
- Cyclomatic complexity;
- Number of blocks used;
- Number of custom procedures used;
- Number of sprites used.

Except for the project name, all features are integer values. We will vectorise the project names using a bag-of-words approach.

We extracted the values from the selected data set using SQL queries. We then put all values per project together in one file. We also computed the number of blocks per procedure and the number of blocks per sprite to normalise those features. This leaves us with three categories of inputs:

- Data retrieved from the database;
- Normalised data;
- The vectorised project name.

To get an impression of the data, we computed the minimum and maximum values for the selected features and collected the results in table 1. For each experiment, we will generate a PCA. The PCAs can be seen in figure 2, and are explored in section 4.2.

Feature	Min value	Max value
Cyclomatic Complexity	1	246
Number of blocks	0	29378
Number of custom procedures	0	372
Number of sprites	0	408

Table 1: The minimum and maximum values for the selected features.

3.4 Clustering the projects

To cluster the projects, we first need to extract the data of the selected features for all projects from the database. Then, we will implement a DBSCAN algorithm to cluster the projects. The project names will be vectorised by applying a bag-of-words algorithm. The code can be found in the GitHub repository[4].

[4]https://github.com/BrentMeeusen/CSE3000-Clustering-Scratch-Projects

To get an impression of the optimal values for the hyperparameters, we will run the algorithm on the data points only including the data retrieved from the database with $min\_samples[5, 60]$ with steps of 5, and $eps[0.5, 5.0]$ with steps of 0.5. Next, we will analyse the results of this experiment and select the optimal values for the hyperparameters which we will use in the following experiments. Finally, we will run the same algorithm with the following 5 different input vectors:

1. The data from the database;
2. The normalised data;
3. The data and the project names;
4. The data and the normalised data;
5. The data, the normalised data and the project names.

By doing multiple experiments with different input vectors, we expect to see the impact of the different inputs.

4 Results

In this section, we take a look at the results of the experiments. First, we elaborate on exploring the hyperparameters. Then, we will analyse the outcomes of the next five experiments.

As described in section 3.2, we consider the clusters to be good when the silhouette coefficient is greater than or equal to 0.50.

4.1 Exploring hyperparameters

After running the initial experiment, we found that a higher $epsilon$ and a lower $min\_samples$ give the best silhouette coefficients. We found that when the minimum number of nearby samples required increases, there are fewer but larger clusters, as well as more outliers. We saw the same effect when the epsilon increases.

The highest silhouette coefficient was $0.2300...$ with an epsilon of 9.0 and 10 as the minimum number of samples. The silhouette coefficient is often much lower when using a $min\_samples$ of 5 compared to a $min\_samples$ of 15. Therefore, we chose to run the next experiments using $epsilon$ $[8.0, 8.5, 9.0, 9.5, 10.0]$ and $min\_samples$ $[10, 15, 20]$.

4.2 The next experiments

After exploring the hyperparameters, we set up five experiments, all running with an epsilon of $[8.0, 8.5, 9.0, 9.5, 10.0]$ and a minimum number of samples of $[10, 15, 20]$. The difference between the experiments is the input vector.

For all experiments, we found that the number of outliers is the highest and the number of projects in the largest cluster is the lowest when looking at the lowest eps and highest $min\_samples$ (in this case 8.0 and 10, respectively) compared to other configurations of the hyperparameters within the experiments. Including the project names does not seem to influence the clustering process significantly. For instance, the lowest and highest silhouette coefficient for experiments 1 and 4 differ by 0.0049 and 0.0231, and for experiments 3 and 5 the difference is 0.0057 and 0.0146.

In all experiments, we found that the lowest number of outliers always corresponds with the highest number of projects in the largest cluster, and vice versa.

Experiment 1: data only

In this experiment, the clustering algorithm only looked at the data that was directly gathered from the database. The normalised data and the project names are excluded.

We found the lowest silhouette coefficient of 0.0141 at $eps = 8.5$ and $min\_samples = 10$, and the highest silhouette coefficient of 0.2300 at $eps = 9.0$ and $min\_samples = 10$. There are roughly 60, 40 and 30 clusters for 10, 15 and 20 $min\_samples$. As the highest silhouette coefficient is observed to be lower than 0.50, we consider that the clusters may not to be of great quality.

The largest cluster's Dr. Scratch mastery score ranges from 16 to 21. The lowest cyclomatic complexity is 1, the highest is 35. The minimum number of blocks used is 16, the maximum is 873. The minimum number of custom procedures used is 0, the maximum is 18. The minimum number of sprites used is 1, the maximum is 49.

This experiment has the second highest silhouette coefficient. When we take a look at figure 2a, we do not see any obvious clusters other than the largest cluster in the middle left of the graph. Smaller clusters could exist just outside the largest cluster. This is supported by looking at the summary; we see 12,716 projects in the largest cluster and 3,194 outliers. The other 52 clusters have anywhere between 10 and 208 projects.

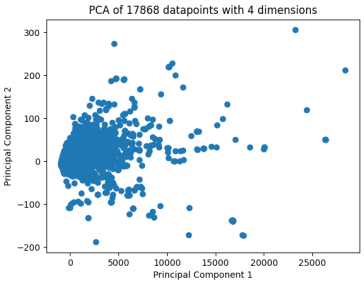
Experiment 2: normalised data only

In this experiment, the clustering algorithm only looked at the normalised data. The data gathered from the database and the project names are excluded.

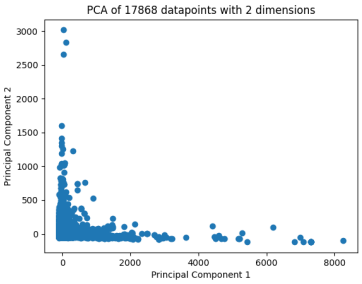
We found the lowest silhouette coefficient of 0.1874 at $eps = 10.0$ and $min\_samples = 15$, and the highest silhouette coefficient of 0.4078 at $eps = 9.5$ and $min\_samples = 20$. There are roughly 50, 35 and 20 clusters for 10, 15 and 20 $min\_samples$. As the highest silhouette coefficient is observed to be lower than 0.50, we consider that the clusters may not to be of great quality.

The largest cluster's Dr. Scratch mastery score ranges from 16 to 21. The lowest cyclomatic complexity is 1, the highest is 246. The minimum number of blocks used is 0, the maximum is 29,378. The minimum number of custom procedures used is 0, the maximum is 372. The minimum number of sprites used is 0, the maximum is 408.

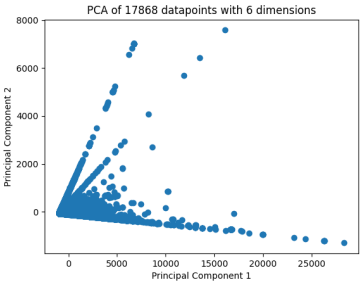
Although this experiment has the highest silhouette coefficient, we consider the outcomes to be meaningless. This is for multiple reasons. First, because the clustering is done with only two features. Second, when we compare the extreme values of the largest cluster and compare that to table 1, we see that the largest cluster contains the projects with all the highest and lowest values. It is highly implausible that if all outliers are in the same cluster, the overall cluster quality is fine. And third, when we take a look at figure 2b, we do not see any obvious clusters other than the largest cluster in the bottom left corner. Some smaller clusters could form in the top left corner and in the bottom right corner. This is supported by looking at the summary; there are 14,957 projects



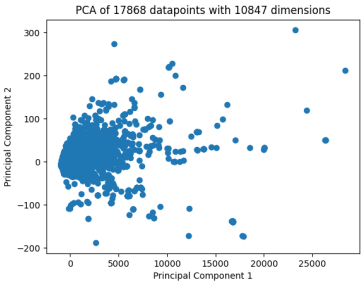
(a) Experiment 1: data only



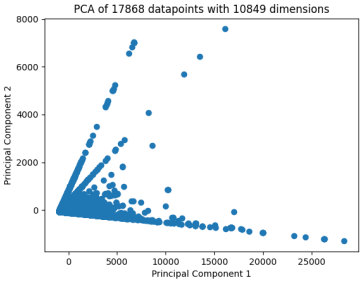
(b) Experiment 2: normalised data only



(c) Experiment 3: data and normalised data



(d) Experiment 4: data and project names



(e) Experiment 5: data, normalised data and project names

Figure 2: PCAs of the five experiments.

in the largest cluster and 1,862 outliers. The other 21 clusters only have 20 to 125 projects.

Experiment 3: data and normalised data

In this experiment, the clustering algorithm looked at the data that was directly gathered from the database, and the normalised data. The project names are excluded.

We found the lowest silhouette coefficient of -0.5690 at $eps = 10.0$ and $min\_samples = 15$, and the highest silhouette coefficient of -0.4731 at $eps = 8.0$ and $min\_samples = 15$. There are roughly 100, 70 and 60 clusters for 10, 15 and 20 $min\_samples$. As the highest silhouette coefficient is observed to be lower than 0.50, we consider that the clusters may not to be of great quality.

The largest cluster's Dr. Scratch mastery score ranges from 16 to 21. The lowest cyclomatic complexity is 1, the highest is 22. The minimum number of blocks used is 28, the maximum is 506. None of the projects in this cluster use any custom procedures. The minimum number of sprites used is 2, the maximum is 34.

This experiment has the lowest silhouette coefficient. When we take a look at figure 2c, we can see the largest cluster in the bottom left corner. The outward lines could be clustered together, too. The summary, however, shows different results; there are 6,665 projects in the largest cluster and 8,342 outliers. The other 63 clusters have anywhere between 10 and 221 projects.

Experiment 4: data and project names

In this experiment, the clustering algorithm looked at the data that was directly gathered from the database, and the project names. The normalised data is excluded.

We found the lowest silhouette coefficient of 0.0092 at $eps = 8.5$ and $min\_samples = 10$, and the highest silhouette coefficient of 0.2069 at $eps = 9.5$ and $min\_samples = 10$. There are roughly 60, 45 and 35 clusters for 10, 15 and 20 $min\_samples$. As the highest silhouette coefficient is observed to be lower than 0.50, we consider that the clusters may not to be of great quality.

The largest cluster's Dr. Scratch mastery score ranges from 16 to 21. The lowest cyclomatic complexity is 1, the highest is 35. The minimum number of blocks used is 16, the maximum is 966. The minimum number of custom procedures used is 0, the maximum is 18. The minimum number of sprites used is 1, the maximum is 50.

This experiment has the third highest silhouette coefficient. When we take a look at figure 2d and compare it with figure 2a, we cannot find any visual differences, which suggests resemblances between the outcomes of the two experiments. The summaries of the two experiments are also similar. For example, the number of projects in the largest cluster and the number of outliers only differ by a few hundred projects. We see 12,928 projects in the largest cluster and 3,168 outliers. The other 51 clusters have anywhere between 10 and 208 projects. We compared the outcomes of the two experiments in table 2.

Experiment 5: data, normalised data and project names

In this experiment, the clustering algorithm only looked at the data that was directly gathered from the database. The data is not normalised and the project names are excluded.

Feature	Experiment 1	Experiment 4	Experiment 3	Experiment 5
Projects in largest cluster	12,716	12,928	6,665	6,671
Number of outliers	3,194	3,168	8,342	8,651
Number of clusters excluding largest cluster	52	51	63	44
Smallest cluster size	10	10	10	18
Largest cluster size, excluding largest cluster	208	208	221	235
Highest silhouette coefficient	0.2300	0.2069	-0.4731	-0.4585
Lowest cyclomatic complexity	1	1	1	1
Highest cyclomatic complexity	35	35	22	22
Lowest number of blocks	16	16	28	28
Highest number of blocks	873	966	506	501
Lowest number of custom procedures	0	0	0	0
Highest number of custom procedures	18	18	0	0
Lowest number of sprites	1	1	2	2
Highest number of sprites	49	50	34	31

Table 2: A comparison of the results of experiments 1 and 4, and 3 and 5.

We found the lowest silhouette coefficient of -0.5747 at $eps = 8.0$ and $min\_samples = 20$, and the highest silhouette coefficient of -0.4585 at $eps = 9.0$ and $min\_samples = 20$. There are roughly 100, 70 and 55 clusters for 10, 15 and 20 $min\_samples$. As the highest silhouette coefficient is observed to be lower than 0.50, we consider that the clusters may not to be of great quality.

The largest cluster's Dr. Scratch mastery score ranges from 16 to 21. The lowest cyclomatic complexity is 1, the highest is 22. The minimum number of blocks used is 28, the maximum is 501. None of the projects in this cluster use any custom procedures. The minimum number of sprites used is 2, the maximum is 31.

This experiment has the second lowest silhouette coefficient. When we take a look at figure 2e and compare it with figure 2c, we cannot find any visual differences, which suggests resemblances between the outcomes of the two experiments. The summaries of the two experiments are also similar. For example, the number of projects in the largest cluster and the number of outliers only differ by a few hundred projects. We see 6,671 projects in the largest cluster and 8,651 outliers. The other 44 clusters have anywhere between 18 and 235 projects. One notable difference between experiment 3 and this experiment, however, is the number of clusters. Experiment 3 generated 64 clusters in total, whereas experiment 5 only generated 45. We compared the outcomes of the two experiments in table 2.

5 Responsible Research

The Netherlands Code of Conduct for Research Integrity [4] is based on 5 principles: honesty, scrupulousness, transparency, independence, and responsibility. In this section, we will take a look at how we implemented these principles during the research, and where we could have done better.

We wrote openly and transparently about the limitations of our research. Also, we did not fabricate or alter the data we worked with. By looking through some of the rows, we found some confidential information in some project titles. To protect these individuals, we decided not to include the data found and used in the GitHub repository.

Throughout the research, we used scientific sources and methods. For example, we read about different clustering algorithms in [5], and we used common practices, like the bag-of-words approach to vectorise strings.

Finally, we used GitHub to publish the code so that everyone is able to look at the code. We used well known libraries to save time, effort, and possible errors. These libraries could contain bugs, so this solution is not perfect. We did accidentally retrieve some personal data. The data we found was already publicly available on the Scratch website and in the data set that we used. We did not spread the data even further by publishing it again.

6 Discussion

The clusters Moreno-Léon et al. found in [14], show different types of projects. Looking at the mean Dr. Scratch mastery scores of their clusters, the projects with a high mastery score are likely to be games. In our research, we did not find clear clusters. However, we only looked at the projects with a mastery score of 16 and above, whereas Moreno-Léon et al. included all projects. Indeed, the mean mastery score of the three clusters they found are 7, 9, and 17. Moreno-Léon et al. also found that the deviation from the best fitting line increases when the project's mastery score is 16 or above [13]. Thus, one of the possible reasons we did not find such clusters could be because the projects we tried to cluster are not similar enough.

A possible reason why the silhouette coefficients are low is that there are too little dimensions, or the ranges of the dimensions are too wide or too narrow. The experiments without the name in the input vector had at most 6 dimensions.

As seen in section 4, including the vectorised project names do not have a considerable impact on the cluster quality. We think this is because the four selected features have wide ranges as can be seen in table 1, whereas the project names add many dimensions with only 0 or 1 as possible values.

6.1 Limitations

The data set we used is from 2016, so the data we worked with is quite old. Using a more recent data set could give different, more up-to-date results. Moreover, we selected features we thought would reflect the complexity of a project. However, the outcomes do not show this. Choosing different features or normalising more features could change the outcomes significantly.

Another limitation is that the current PCAs only show the result of the PCAs, but it does not show what data point belongs to what cluster. We could have used different colours to differentiate between (a number of) clusters, like the largest cluster and the outliers.

7 Conclusions

In this research, we explored part of the Scratch repository to see if we could find clusters based on some code and complexity traits. We investigated the largest clusters and its properties, and compared similar experiments with each other.

The highest silhouette coefficient we found when including more than just normalised data is 0.2300 in experiment 1. As 0.2300 is less than 0.50, we consider the clusters not to be of sufficient quality. Looking at the data in a bit more detail supports this finding, as does looking at the corresponding PCA graph, figure 2a. Hence, we conclude that we did not find a way to cluster Scratch projects by the selected code and project traits. We found that including the vectorised project names using a bag-of-words approach has a negligible effect on the outcome of the experiment.

A possible explanation for the low cluster quality could be that the Scratch environment is not suitable enough to measure a keen understanding of programming concepts. To check this hypothesis, a future work could attempt to cluster projects in other programming languages or environments.

References

[1] E. Aivaloglou and F. Hermans. How is programming taught in code clubs? exploring the experiences and gender perceptions of code club teachers. *19th Koli Calling International Conference on Computing Education Research (Koli Calling '19)*, 2019.

[2] H. Borges and M. Valente. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 9 2018.

[3] K. Falkner, S. Sentance, R. Vivian, S. Barksdale, L. Busuttil, E. Cole, C. Liebe, F. Maiorana, M. McGill, and K. Quille. An international comparison of k-12 computer science education intended and enacted curricula. *19th Koli Calling International Conference on Computing Education Research (Koli Calling '19)*, 2019.

[4] KNAW; NFU; NWO; TO2 federatie; Vereniging Hogescholen; VSNU. Netherlands code of conduct for research integrity. *DANS*, 2018.

[5] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[6] F. Hermans, E. Aivaloglou, J. Moreno-Léon, and G. Robles. A dataset of scratch programs: Scraped, shaped and scored. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017.

[7] M. Jureczko and L. Madeyski. Towards identifying software project clusters with regard to defect prediction. *PROMISE '10: Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, 9 2019.

[8] S. Kawaguchi, P. Garg, M. Matushita, and K. Inoue. Mudablue: An automatic categorization system for open source repositories. *11th Asia-Pacific Software Engineering Conference*, 1 2005.

[9] J. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk. Programming by choice: Urban youth learning programming with scratch. *SIGCSE*, 40, 3 2008.

[10] C. McMillan, M. Grechanik, and D. Poshyvanyk. Detecting similar software applications. *2012 34th International Conference on Software Engineering (ICSE)*, 6 2012.

[11] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari. Learning computer science concepts with scratch. *Computer Science Education*, 23, 2013.

[12] J. Moreno-Léon, G. Robles, and M. Román-González. Dr. scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, 2015.

[13] J. Moreno-Léon, G. Robles, and M. Román-González. Comparing computational thinking development assessment scores with software complexity metrics. *2016 IEEE Global Engineering Education Conference (EDUCON)*, 2016.

[14] J. Moreno-Léon, G. Robles, and M. Román-González. Towards data-driven learning paths to develop computational thinking with scratch. *IEEE Transactions on Emerging Topics in Computing*, 8, 2017.

[15] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: Programming for all. *Communications of the ACM*, 11 2009.

[16] M. Rokon, P. Yan, R. Islam, and M. Faloutsos. Repo2vec: A comprehensive embedding approach for determining repository similarity. *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 9 2021.

[17] L. Seiter and B. Foreman. Modeling the learning progressions of computational thinking of primary grade students. *ICER '13: Proceedings of the ninth annual international ACM conference on International computing education research*, 2013.

[18] D. Weintrop. Block-based programming in computer science education. *Communications of the ACM*, 62, 8 2019.

[19] S. Yang, C. Domeniconi, M. Revelle, M. Sweeney, B. Gelman, C. Beckley, and A. Johri. Uncovering trajectories of informal learning in large online communities of creators. *L@S '15: Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, 3 2015.

[20] A. Zeevaarders and E. Aivaloglou. Exploring the programming concepts practiced by scratch users: an analysis of project repositories. *2021 IEEE Global Engineering Education Conference (EDUCON)*, 2021.