A comparative analysis of coding approaches in machine learning among computer science students and non-computer science students

Grgur Dujmovic[1]

Supervisor(s): Gosia Migut

[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 28, 2024

Name of the student: Grgur Dujmovic
Final project course: CSE3000 Research Project
Thesis committee: Gosia Migut, Myrthe Tielman

Abstract

The increasing presence of Machine Learning in all fields of study requires an improvement in how it is taught. Previous research on this topic examined how to teach ML concepts and highlighted the importance of using technology and leveraging relevant pedagogical content knowledge. It did not compare the impact of previous programming knowledge on the students' approach to solving ML problems. This paper explores the differences in implementation of 60 Machine Learning coding assignments using metrics that were determined by previous research to be a good indicator of code quality and computational thinking. By analysing the code submissions with these metrics, the results show several interesting insights about the students' use of functions, variables and the explanations of their thought process. However, results of the metrics are mostly inconclusive. The results from this study highlight the need for additional research on this topic to ensure that people with limited Computer Science knowledge are able to learn about it and implement it in their disciplines.

1 Introduction

The field of Machine Learning is relatively new and is rapidly expanding. It is becoming everpresent in our daily lives and as such it creates a need for more people who are familiar with its concepts. Because it is inherently a Computer Science concept, students that study Computer Science become acquainted with it quite well. However, an issue arises when there is a need to teach it to students that do not study Computer Science. Machine Learning is being implemented in many disciplines to improve the processes currently in place. Physics, medicine, sports, art, architecture and marketing are just a couple of examples of areas where Machine Learning is being implemented. With the increasing demand for people that are familiar with it, more and more curriculums are starting to add courses that introduce Machine Learning to non-Computer Science students and students with varying degrees of knowledge when it comes to programming. This creates an issue when it comes to teaching these students Machine Learning concepts. How to approach teaching them these topics? Is it different than teaching Computer Science students the same topics?

This paper will compare the code of Computer Science students and students that are taking a Machine Learning course while not studying Computer Science in an attempt to explore differences and similarities between their approaches to solving the same set of assignments. The Computer Science students are taking a Machine Learning course as a mandatory part of their bachelor degree, while the non-Computer Science students are taking it as part of their minor.

1.1 Previous research

Previous research primarily focused teaching on students with no previous knowledge of these ML concepts. A comprehensive ten-year systematic literature review (Martins Gresse Von Wangenheim, 2022) reviewed techniques that are currently used to teach Machine Learning in high school. Through an extensive review of the current methods of teaching Machine Learning they concluded that high school students showed a gain in knowledge of basic Machine Learning concepts and algorithms but it was more difficult to teach them the underlying statistical concepts and certain AI techniques. The paper also explains the importance of introducing appropriate technologies for teaching Machine Learning, such as python, for implementing classical Machine Learning algorithms. This is a way for students to gain knowledge by implementing these algorithms themselves. They state that there needs to be more guidance on adjusting the content and technologies to effectively teach Machine Learning in a high school setting. This paper does not go over the difference that previous knowledge has on the performance of students, but it does show that teaching Machine Learning through technology is the way to go.

Machine Learning instructors were interviewed as a part of a study in an effort to gain more insight into the variables that affect teaching Machine Learning to students that don't study Computer Science. The first paper of the study (Sulmont et al., 2019a), explores the pedagogical content knowledge required to teach Machine Learning to non-Computer Science students. It discovers and reasons about preconceptions held by non-Computer Science students. By identifying multiple preconceptions and barriers to learning, it suggests that ML instructors need to teach ML while being aware of these preconceptions and barriers to increase the effectiveness of their teaching. The paper also suggests further research on developing more pedagogical content knowledge for teaching ML. The second paper of the study (Sulmont et al., 2019b), aims to identify difficulties that students have when learning Machine Learning concepts. They concluded that it is more difficult to teach Machine Learning to students that are not majoring in Computer Science, but not because of the algorithms, but rather the high-level design decisions and comparing models. Additionally, they concluded that misconceptions and preconceptions were common and presented difficulties for these students. The paper suggests that these challenges can be solved by changing the lecturing methods and developing targeted instruction materials. This paper implies that there are differences and unique difficulties in teaching Machine Learning to non-Computer Science students, but does not compare the two.

A study examining teaching methods used to teach Machine Learning in pre-university education (Temitayo Sanusi, 2021) identifies pedagogical approaches used in teaching Machine Learning. Through this, the paper aims to determine the impact and design of these approaches on teaching Machine Learning. It identifies suitable pedagogical frameworks suitable for teaching Machine Learning. The author emphasizes the need for future research on this topic for a comprehensive conclusion on the topic and to determine the best approach for teaching Machine Learning.

These articles examine current approaches and propose alternatives, but they do not compare the performance of students with and without prior programming experience. To bridge

this knowledge gap, this paper will provide a comprehensive comparison between the code solutions of two groups. This comparison may be used to tailor educational strategies so students with no prior Computer Science experience can get a better understanding of ML coding practices they are missing.

1.2 Research questions

How do coding approaches for solving Machine Learning problems of computer science students differ compared to students without a computer science background? Sub-questions:

- Is there a statistically significant difference in code metrics in the solutions of computer science students versus students with no computer science background?
- Can the coding practices of computer science students in machine learning be reliably distinguished by using coding metrics?
- Are there observable differences in comments and readability, between the machine learning code produced by computer science students and those without a computer science background?

2 Methodology

To answer the research question and the sub questions, a program will be developed. This program will go through code examples and provide statistics for the relevant metrics. After the metrics are gathered, an analysis and interpretation will be performed, which will look for any correlation in the data that supports a conclusion.

2.1 Code analysis

The code examples that will be used are provided and anonymized by TU Delft. These code examples come from two groups of students. The first group are Computer Science students from TU Delft. They are considered to have prior programming experience and will serve as the first group for the analysis. The code is gathered from bonus assignments they have completed during their Machine Learning course. The second group are students that participated in a minor course Machine Learning and Introduction to AI. Since students that study Computer Science cannot take this minor, they are considered to have no prior programming experience. The assignments they have completed are nearly identical to the previous group.
The gathered code examples will be collected and compiled into numbered files into separate directories for group and edition of the course. There will be 60 samples, 30 for each group, 10 for each edition. After the code examples are organized, a program will be developed to go through each file and record relevant statistics. All of these files are Jupyter notebooks, hence there will need to be a way to extract information only from code blocks. The nbformat python library will be used for these purposes.

2.1.1 Metrics

Most of the previous research for code analysis metrics is concerned with large codebases and system optimization. This will not be applicable here as we are dealing with relatively small projects. Previous studies offer insight for deciding which what metrics are most appropriate to use for evaluation. The most common metrics used in the software measurement process were compiled from a total of 226 studies and determined to be "lines of code, McCabe's cyclomatic complexity and number of methods and attributes" (Nuñez-Varela et al., 2017) (In this case attribute refers to a function attribute or a variable in the code). Furthermore, metrics of "non-comment source lines and the number of function declarations were strongly correlated with certain measures of quality". The measures of quality were "number of known errors encountered and number of modification requests made during development.. . . , time taken to attend to these and a subjective assessment of program complexity" (Harrison et al., 1996). Additionally, to evaluate computational thinking of the students, six metrics will be added with the "... aim to quantify CT skills such as problem breakdown, pattern recognition, and communication" (Kong et al., 2018). There were many more other metrics from this paper but they were not applicable to these notebooks. Knowing this, we can conclude that these metrics will be relevant for this research project. In conclusion, the metrics that will be used are:

1. Common methods
   - 1.1 Lines of code
   - 1.2 Number of functions
   - 1.3 Number of attributes
2. Measures of quality
   - 2.1 Cyclomatic complexity
   - 2.2 Comment percentage
3. Computational thinking
   - 3.1 Number of code cells
   - 3.2 Number of markdown cells
   - 3.3 Mean lines per code cell
   - 3.4 Mean words per markdown cell
   - 3.5 Number of markdown words
   - 3.6 Number of function calls

These metrics will be collected by functions that go through the code blocks of the Jupyter notebooks.

3 Experiment setup

To measure the metrics a new python project will be created. There will be three main parts to the processing of the data. The first part of processing the data is organizing the data in a way that it is anonymized and processed. The second part is systematically going through the data with custom functions to measure each metric and save the results. The third and final part is visualizing the data and performing a statistical analysis.

3.1 Organizing the data

We will be working with already anonymized data, courtesy of TU Delft. This data will be organized in two groups, computer science students and non-computer science students.

This separation is because the data of these two groups will be compared. Within each group there is a further separation in 3 years, for each year that the submissions are from. Per year, there are 10 unique student submissions. Each student submission contains 1-2 Jupyter notebooks. When parsing the data, there will be one measure that takes the average value of a metric for a whole year, and another measure that takes each individual measure for the whole year, for the purposes of statistical computations.

3.2 Parsing the data

There will be a function to measure each individual metric. These functions will take in the Jupyter notebook and return the value of the metric for the notebook. There are a total of 11 metrics, but 11 custom functions are not needed.

3.2.1 Collecting metrics

There need to be functions for specifically extracting code and markdown cells from the Jupyter notebooks for computing other metrics. Furthermore, for computing mean lines of code and mean markdown words, we can simply divide the total number of lines/words in the file with the number of the appropriate cell type. To measure complexity, number of attributes and function number and function calls, the python ast module will be used. This module is used to help process python syntax trees.

Cyclomatic complexity is defined as the number of paths through a program. (McCabe, 1976). It is a way to estimate how complex a program is. In a regular setting, it is measured by counting the number of decision points or branches within the code. For the purposes of this paper, the definition will be expanded to include loops in the program. The reason for this is that the assignments that are provided inherently do not require branching, but including loops in the solution is an indication of a better algorithmic understanding. Hence, to collect the complexity metrics, the program looks for instances of loop declarations and boolean operations by checking python's abstract syntax tree. Similarly, to collect function metrics, the program checks for instances of function declarations and function calls.

3.2.2 Aggregating metrics

The metrics are collected in two ways. The first way is used to create a graph for a visual overview of the results. This consists of student results per group per year being summed up and averaged. This gives an average value of a metric for a student group in a year and allows for simple visualization. However, doing this means that all statistical values are lost. That is why the data is simultaneously being added to an array that contains all data points. These data points are similarly separated into group and year, but we can perform statistical analysis on them, which will be required to determine the significance of results.

3.3 Statistical analysis

After gathering the results there needs to be a way to determine whether these results are significant, i.e. do they have any correlation or relevance to the research topic. For this purpose we will compute confidence intervals, t-tests and Cohen's d.

Confidence intervals are calculated with a 95% confidence level. They will provide an estimated range for a value with a 95% confidence that the true value is within that interval. To assess the reliability of differences between groups, we will say that an overlapping confidence interval is not significant. The t-tests will provide insight into the statistical significance of the difference between two groups. If the p-value of the t-test is < 0.05, it will be considered significant. Cohen's d measures the magnitude of the difference between groups. It is used to support other statistical calculations.

4 Results

Measuring all of the code metrics from the Jupyter notebooks yielded the results as seen in fig 1. The results span across three years, three iterations of the courses. The assignments that each group had to do were identical, with the only differences being in how the assignments were submitted. With an initial look at all of the plots and their confidence intervals we can notice that most of the differences are not significant. A closer look at each metric is required to get an overall understanding.

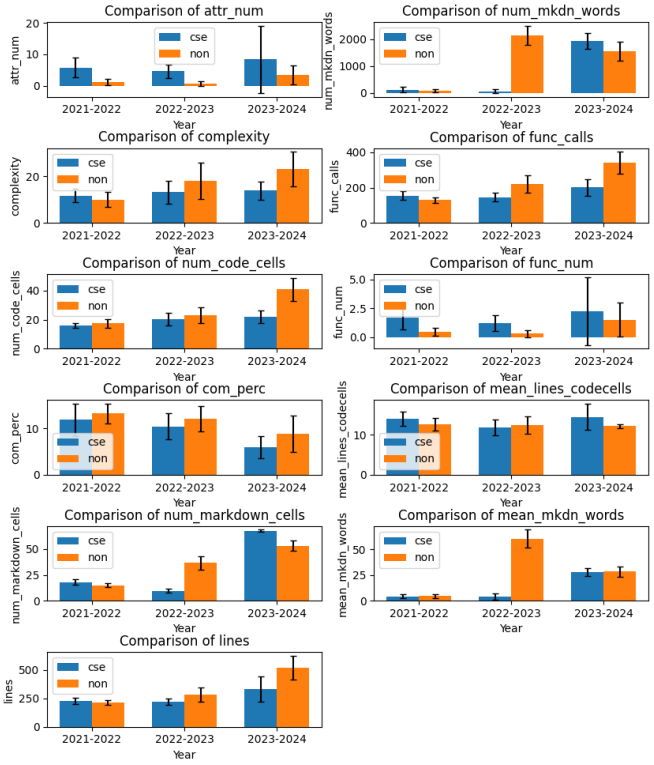

Figure 1: This figure show bar plots of all the computed metrics. The black lines are 95% confidence intervals.

4.1 Trends and confidence intervals

1. **Comment percentage**: The group of non-Computer Scientists consistently have a higher comment percentage. However, the confidence intervals are always overlapping, showing that the difference is not significant.

2. **Mean lines in code cells**: Both groups seem to consistently have the same mean lines per code cell. While there is no difference, this correlation might be interesting.
3. **Number of markdown cells**: The number of markdown cells varies significantly for both groups across the years, this might be because of different ways of teaching per generation.
4. **Number of attributes**: Computer Science students consistently have a higher number of attributes in their code. The large confidence interval for 2023-2024 may be because of notebook processing errors which will be discussed in section 5.
5. **Number of words in markdown cells**: The number of words in markdown cells has a significant difference in 2022-2023, which could point to non-Computer Science students being prone to explaining their process in detail.
6. **Number of lines of code**: There is a trend of the number of lines increasing for both groups, and it seems that it is increasing faster for non-Computer Science students, which could point to them coding in a more verbose manner.
7. **Number of code cells**: Similarly to lines of code, there is an increasing trend, with non-Computer Science students increasing faster. There appears to be a significant difference for the 2023-2024 values.
8. **Cyclomatic complexity**: The confidence intervals are consistently overlapping. Even though there seems to be an increasing difference trend, no significant conclusions can be made.
9. **Mean words in markdown cells**: There seems to be some irregularity in the data, as the 2022-2023 is substantially bigger than the rest. If we consider that an outlier the metric seems to be equal for both groups.
10. **Number of function calls**: Non-Computer Science students have a significantly higher number of function calls than Computer Science students. This may be because of the content of previous classes. This will be mentioned in section 5.
11. **Number of functions**: Almost counter-intuitively, Computer Science students have a higher number of functions. The values are low for both groups, indicating that this may not be relevant.

4.2 P-values and Cohen's d

The p-values and values of Cohen's d can be found in fig 2. Since only p-values < 0.05 are considered significant we will be analyzing those. There are 12 instances of p being less than 0.05 and there is no instance of one metric having a significant p value across all 3 years. Plotted alongside the p-values is the respective Cohen's d value. The higher this value is, the more significant the effect size is.

We will consider a result relevant if there is a significant p-value for at least two years. There are a couple of results of note. Firstly, the number of functions and number of attributes both have significant p-values for two years in a row. This could indicate a difference in algorithmic thinking, where students decided to create a function or save a value in an attribute to reuse. Cohen's d for the number of attributes is higher than for function number, implying a higher significance for the difference. Secondly, the number of markdown

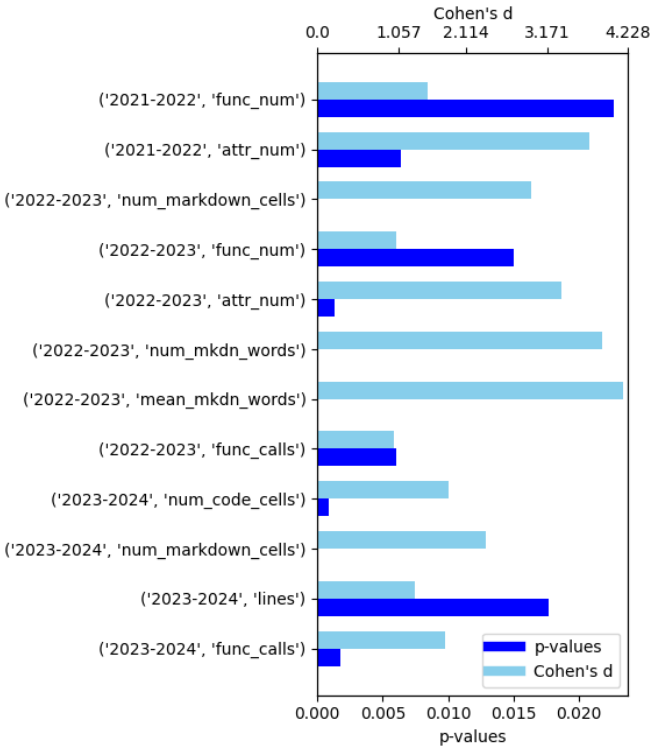

Figure 2: This plot shows the p-values alongside values for Cohen's d for code metrics. Only metrics that have a p-value < 0.05 are shown.

cells shows significant p-values. This might be because students were inclined to explain their process to demonstrate their way of thinking. Finally, the number of function calls has a significant p-value, signifying that one group calls functions more consistently. Since neither group defines many functions, these are likely library functions.

5 Discussion

This section goes over the results in more detail, providing insight into the significance of them and possible factors that affected them.

5.1 Significant differences

The results generally do not have a consistently significant difference. The most significant results are how the student groups use functions and markdown cells. The differences in number of attributes are noteworthy. It depicts that CS students think in a more modular sense, assigning values to variables for potential reuse. This could also be connected to CS students having a deeper understanding of programming concepts as a whole. Keeping this in mind, the number of function calls is greater for non-CS students than it is for CS students. Function calls represent calls even from library functions. Because the number of defined functions is so low for both groups, this means that non-CS students utilize more library functions. These assignments were written in python, and since non-CS students had a python course as part of their minor, while CS students had no python course, they were

more proficient with the language and knew which methods to invoke. However, even without the python course, CS students were capable of using programming concepts they learnt about with different languages.
Another observed difference was the use of comments and markdown cells between the groups. Even though the differences are mostly not significant, we can observe a trend of the comment percentage lowering at a similar rate for both groups. This may be accompanied by the fact that the number of markdown cells is trending upwards. For some reason, both groups are starting to comment their code less and explain their code and thinking in the markdown cells of the Jupyter notebooks. The most likely reason for this is the way the assignment is given. Instead of explaining the code in comments, students are required to explain their code and thought process in the given markdown cells. This limits the effectiveness of these metrics, because it is not a conscious choice made by the students, but rather a condition of the assignment.

5.2 Insignificant differences

The rest of the metrics show similar values or insignificant differences for both groups. The reason behind this is likely the way the assignments were given. The assignments do not give room for different approaches and are quite linear when it comes to solving. They are limited to exploring the data and implementing well documented algorithms. While this is a good way to introduce students to Machine Learning concepts, it does not help in improving or evaluating their computational thinking. Both groups have a similar number of lines of code, showing that, on average, both groups solved the assignments in a straightforward manner.

5.3 Outside factors

There are some factors that have affected the collected data. Firstly, 2021-2022 values were from the COVID era, when there were no/very limited in person classes and everything was being handled online. This is probably why most of the metrics from that period do not fit with the trend from the other two years. Secondly, many submissions had syntax errors in them that created issues when trying to parse them to gather metrics. This is particularly interesting because if they have undiscovered syntax errors, that implies they were never ran. Some of these errors were minor and would not alter the metrics, but a few could not be fixed because the values of the metrics would change. The data for this study contained a total of 60 assignments, 10 per student group per year. If there were more submissions per student group, the confidence intervals might have been smaller, resulting in more opportunities of significant differences, and if there were submissions from a longer time frame, there would have been a possibility to analyze trends and evaluate how different instances of the courses affected the metrics.

6 Responsible Research

This section presents steps taken to maintain the integrity of the research. It covers the process of data collection and how the research could be repeated/reproduced.

6.1 Data collection

The data for this study was collected by TU Delft. It is from assignments students have submitted for their Machine Learning course. For the group of Computer Science students, it was part of an optional bonus assignment, while for the non-Computer Science students it was a mandatory assignment. To use this data for the purposes of the project, I filled out a data management plan and a research ethics checklist, and wrote an informed consent form and an ethics committee proposal, as required by TU Delft. In these documents I described which precautions I will be taking to keep the data safe. The data was kept on TU Delft servers to avoid privacy concerns. Furthermore, I explained what I would be doing with the data and why it is required for my research. I sent a proposal to the TU Delft ethics committee detailing steps that I would take to anonymize the data so the students assignments cannot be traced back to them. The research I am performing does not make use of any personal data. Finally, I created an informed consent form to allow students to withdraw their data from the research. All of these steps were done to ensure there was no breach of ethics and data privacy, nor violation of student rights.
In the end, I was not personally responsible for anonymizing the submissions, although the process for it was ready. This resulted in even better data privacy, as only employees of TU Delft had access to the original data.

6.2 Reproducibility

The study in and of itself is not difficult to repeat. The code for analysing the Jupyter notebooks is available on GitHub. The only required part for repeating the study is the student data. This might not be easy to come by due to privacy concerns. If the data is acquired, it has to be organized in a specific folder hierarchy for the program to process it. The program is created to be modular, so it is trivial to add new metrics, or to compare different groups of students.
Reproducing the results of the study might prove difficult, even at TU Delft. The groups of students may greatly differ in their programming ability per generation, the lecturers might be different. Different university may have completely different assignments and ways of teaching. The variability of the results is the reason why there are three different statistical methods being used. Making sure that any conclusions made from the data have a very low probability of being flukes is pivotal to the legitimacy of the study.

7 Conclusion and Future Work

In conclusion, this paper provides an insight into coding approaches of Computer Science and non-Computer Science students by exploring the differences in coding practises. We analyzed 60 assignments by considering different code metrics from previous research, and by conducting a statistical analysis we can answer the stated research questions. The primary research questions of this paper is ”How do coding approaches for solving Machine Learning problems of computer science students differ compared to students without a computer science background?”. The results showed us that Computer Science students tend to think more modularly, by

creating more variables for cleaner and reusable code. Furthermore, the results show that there are not any other significant differences between the coding approaches. For the first and second sub-questions, "Is there a statistically significant difference in code metrics in the solutions of computer science students versus students with no computer science background?" and "Can the coding practices of computer science students in machine learning be reliably distinguished by using coding metrics?", the results show that there are rarely any statistical differences in the metrics, meaning that one could not reliably distinguish the previous knowledge of a student based on their assignment. Finally, the answer to the last sub-question, "Are there observable differences in comments and readability, between the machine learning code produced by computer science students and those without a computer science background?", is no, because the comments and markdown cells for both groups, in which they explain and reason about the task, do not have a significant statistical difference.

7.1 Improvements

To enhance the robustness of this study, there are multiple things that should be considered to improve it if it were to be repeated. Firstly, a more diverse dataset that includes submissions from different institutions and time frames would result in a more comprehensive analysis. When collecting this dataset we should keep in mind that the assignments the students have completed should be identical to not affect the integrity of the findings. An analysis of this dataset would provide a broader perspective on how students with different backgrounds approach solving ML assignments. When the data comes from multiple institutions and from students of different backgrounds, it is even possible to evaluate the students' approaches in more detail, allowing a different teaching approach depending on the students background. Secondly, a form of qualitative analysis could be introduced, by not only analysing the code but also interviewing students, resulting in a deeper insight behind their approach and reasoning while writing code. Expanding on this, a third improvement could be controlled experiments, where the students solve the problems under observation while explaining their thought process. This would provide an even more detailed look into how students of different backgrounds approach problems.

7.2 Future work

Building upon the results of this study, there are several opportunities for further research. Firstly, there could be an exploration of ML coding assignment structures. A comparative analysis of guided assignments, such as the ones in this paper, versus open-ended projects, which could highlight and improve the students computational thinking. Perhaps students would learn more if they were left to their own devices and given freedom to approach a given problem how they see fit. Secondly, an examination of prerequisite courses would provide insight into how these courses affect the students ability to solve ML problems. This would allow instructors to determine which knowledge from the courses applies to ML, and could be used to help tailor teaching methods and course content for students with different backgrounds. Finally, an exploration of collaborative learning for ML could examine the effect of students of various backgrounds working on assignments together. It would investigate whether students with varying academic backgrounds could mutually positively affect their understanding of ML concepts and coding practices.

8 References

Harrison, R., Samaraweera, L. G., Dobie, M. R., & Lewis, P. H. (1996). An evaluation of code metrics for object-oriented programs. Information and Software Technology, 38(7), 443–450. https://doi.org/10.1016/0950-5849(95)01081-5

Martins, R. M., & Gresse Von Wangenheim, C. (2022). Findings on teaching machine learning in high school: A ten - year Systematic Literature Review. Informatics in Education. https://doi.org/10.15388/infedu.2023.18

Nuñez-Varela, A. S., Pérez-Gonzalez, H. G., Martínez-Perez, F. E., & Soubervielle-Montalvo, C. (2017). Source code metrics: A systematic mapping study. Journal of Systems and Software, 128, 164–197. https://doi.org/10.1016/j.jss.2017.03.044

Sulmont, E., Patitsas, E., & Cooperstock, J. R. (2019a). Can you teach me to machine learn? Proceedings of the 50th ACM Technical Symposium on Computer Science Education. https://doi.org/10.1145/3287324.3287392

Sulmont, E., Patitsas, E., & Cooperstock, J. R. (2019b). What is hard about teaching machine learning to non-majors? insights from classifying instructors' learning goals. ACM Transactions on Computing Education, 19(4), 1–16. https://doi.org/10.1145/3336124

Temitayo Sanusi, I. (2021). Teaching machine learning in K-12 Education. Proceedings of the 17th ACM Conference on International Computing Education Research. https://doi.org/10.1145/3446871.3469769

Kong, S.C., Andone, D., Biswas, G., Crick, T., Hoppe, H.U., Hsu, T.C., Huang, R.H., Li, K.Y., Looi, C.K., Milrad, M., Sheldon, J., Shih, J.L., Sin, K.F., Tissenbaum, M., & Vahrenhold, J. (Eds.). (2018). Proceedings of the International Conference on Computational Thinking Education 2018. Hong Kong: The Education University of Hong Kong.

McCabe, T. J. (1976). A complexity measure. IEEE Transactions on Software Engineering, SE-2(4), 308–320. https://doi.org/10.1109/tse.1976.233837