A Preliminary Study on Using Text- and Image-Based Machine Learning to Predict Software Maintainability

Markus Schnappinger[1(✉)], Simon Zachau[1], Arnaud Fietzke[2], and Alexander Pretschner[1]

[1] Technical University of Munich, Munich, Germany
{markus.schnappinger,simon.zachau,alexander.pretschner}@tum.de
[2] itestra GmbH, Munich, Germany
fietzke@itestra.de

Abstract. Machine learning has emerged as a useful tool to aid software quality control. It can support identifying problematic code snippets or predicting maintenance efforts. The majority of these frameworks rely on code metrics as input.

However, evidence suggests great potential for text- and image-based approaches to predict code quality as well. Using a manually labeled dataset, this preliminary study examines the use of five text- and two image-based algorithms to predict the readability, understandability, and complexity of source code.

While the overall performance can still be improved, we find Support Vector Machines (SVM) outperform sophisticated text transformer models and image-based neural networks. Furthermore, text-based SVMs tend to perform well on predicting readability and understandability of code, while image-based SVMs can predict code complexity more accurately.

Our study both shows the potential of text- and image-based algorithms for software quality prediction and outlines their weaknesses as a starting point for further research.

Keywords: Software maintainability · Expert judgment · Maintainability prediction · Machine learning · Text classification · Image classification

1 Motivation

With the rise of software, the assessment and improvement of its quality is an increasingly vital challenge. To support software quality control, a variety of automated tools and measurements exist. Still, some quality attributes are hard to determine without manual reviews [44]. As human analysts are expensive, predicting such properties with machine learning drew attention over the past

Both Simon Zachau and Markus Schnappinger should be considered main authors.

D. Mendez et al. (Eds.): SWQD 2022, LNBIP 439, pp. 41–60, 2022.
https://doi.org/10.1007/978-3-031-04115-0_4

years. For instance, it has successfully been applied to identify code smells [10, 13,33], support fault localization [48], and predict the maintainability of source code [42] or code changes [23,27].

Most contemporary studies rely on static code metrics as proxies for the actual source code. However, as Ray et al. [37] and Hindle et al. [19] point out, source code and natural language share certain characteristics, too. Accordingly, techniques originally designed for natural language have successfully been applied to source code, e.g. to aid recovery attacks against obfuscated programs [39], predict bugs [19], or identify code smells [32].

Hence, we hypothesize such algorithms can be employed for software quality prediction, too. Since readability and understandability are characteristics of both natural language and source code, we conjecture these attributes can be predicted particularly well using text-based machine learning.

Furthermore, there exists evidence that software analysts already build a strong hypothesis based on their first impression of the code. When analyzing and labeling the code used in this study, several experts confirmed they have been able to get an accurate intuition of the quality of a code snippet by looking at a visual representation of its overall structure, without going into syntactic or semantic detail. This applies in particular to assessments about the complexity and understandability of code. In this study, we try to mimic this process by training machine learning algorithms on images of source code. An example of such a visual representation is provided in Fig. 1.

Gap: Many researchers successfully applied machine learning on software metrics to predict software quality [17,23,27,35,43]. However, despite recent advances in text and image classification, these techniques are not used so far to predict software quality attributes as perceived by human experts.

Solution: This preliminary study explores the potential of two yet unapplied machine learning techniques for quality prediction. In this study, we conduct experiments in both a multi(4)-class and a binary classification setting. We compare the performance of five text-based and two image-based machine learning approaches using a publicly available, manually labeled dataset. The code is sampled from seven software projects and contains both open-source and proprietary projects. The learned quality label corresponds to the consensus of at least three analysts.

Contribution: Using text-based input, Support Vector Machines outperform other algorithms including Naive Bayes, BERT, RoBERTa, and CodeBERT by a large margin. Considering binary classification, they reach the same accuracy as an average human analyst. They are able to predict the readability, understandability, and complexity of source code with Matthews Correlation Coefficients (MCC) above 0.61 and F-Scores above 0.81, while a ZeroRule baseline classifier yields an MCC of 0.0 and F-Scores below 0.38. Furthermore, we observe better performance for binary classification than for ordinal multiclass prediction. While the naive baseline is outperformed by far in the first case, it is only slightly exceeded in the second case. This observation holds for both text-based

Fig. 1. First impressionistic, unreadable visualization of source code. The example on the left shows code that was later on considered hard to maintain, while the example on the right is rather easy to maintain. The examples feature the classes `Cells.java` and `UniformTexture.java`, resp., from *Art Of Illusion* [41].

and image-based algorithms. Here, image-based Support Vector Machines yield the best results as well with MCCs between 0.43 and 0.67 and F-Scores between 0.71 and 0.76.

These results are promising on the one hand, but are not yet applicable in practice on the other hand. Nevertheless, this preliminary study demonstrates the potential of image- and text-based classification algorithms for quality prediction and identifies which weaknesses remain to be addressed in further research. In particular, data preprocessing poses a major challenge.

Outline. The remainder of this paper is organized as follows. First, we describe in detail the experimental design including the dataset used, machine learning algorithms implemented, and data preprocessing techniques applied. Second, the experiment results are presented. This is followed by a critical discussion of the results and the limitations we identified. Eventually, we synthesize related

research and alternative approaches to predicting software maintainability. The last chapter summarizes the study and presents our final conclusions.

2 Experimental Design

In this study, we examine the performance of text-based and image-based machine learning algorithms to predict the source code attributes readability, understandability, and complexity. Readability describes how easy it is for humans to syntactically parse written information [38]. In contrast, understandability is concerned with the ease of extracting relevant concepts and comprehending the semantics of a text or code snippet. Both attributes contribute significantly to software maintainability [1,47] and are also key characteristics of natural language texts. Hence, we investigate the use of classification algorithms from the natural language domain to predict these source code attributes.

Besides, the complexity of code has received lots of attention. The most popular approaches to capture the human intuition of code complexity are McCabe's cyclomatic complexity [30] and the cognitive complexity measure referred to by SonarQube [4]. While the effectiveness of these metrics is controversial, we observed human experts are able to build strong intuitions about code complexity even at first glance. This observation was made during the creation of the dataset described in Sect. 2.1. To recreate that first impression of an expert, we utilize images of source code. Then, we investigate the use of image classification to predict the complexity.

This section elaborates on the investigated algorithms, the used dataset, evaluation metrics, and preprocessing techniques.

2.1 Dataset

Unfortunately, there are only few software quality datasets publicly available. In 1993, Li and Henry [27] published a dataset containing the number of changed lines per code file. This attribute is often used as a proxy for software maintainability, e.g. in prediction experiments by [26,27,51]. However, that dataset does not distinguish between different maintainability aspects. Hence, there is no possibility to target specific sub-characteristics such as readability or complexity.

In contrast, we consider a manually labeled dataset that provides expert evaluations of the readability, understandability, perceived complexity, modularity, and overall maintainability of Java classes [40,41].

This dataset is a collection of code snippets from five open-source and four proprietary projects reviewed and rated by various experts. In total, 70 professionals participated in the creation of the dataset. The participants are affiliated with 17 different organizations including Airbus, Audi, BMW, Facebook, and Oracle. Eventually, the study was able to collect around 2,000 assessments, covering 519 Java classes.

Software quality consists of several sub-aspects such as maintainability or security [20]. Similarly, maintainability can also be divided into several sub-aspects. In our case, the assessment focuses on the sub-categories complexity, modularity, readability, and comprehensibility as well as the overall maintainability judgment of the expert. This decomposition guides the experts which viewpoints to consider during the assessment and mitigates the threat to construct validity, i.e. different participants might not share the same understanding of the broad term *maintainability*. Furthermore, this division enables researchers to focus on specific sub-aspects such as readability or comprehensibility.

Despite this decomposition into sub-characteristics and limited viewpoints, the subjective nature of the assessment remains problematic. Therefore, each class was evaluated by at least three experts. The Expectation-Maximization algorithm [8] finally aggregates their votes and computes the most probable 'true' label for each maintainability category. For more information about the selection of the study objects and the detailed labeling procedure please refer to [40] and [41].

In the presented machine learning experiments, we consider the aggregated consensus rating as the label. For our study, we have access to all open-source and two proprietary projects. The open-source dataset contains 304 entries, i.e. Java classes, which are extended to 374 entries by the two commercial projects. We conduct our experiments on both the open-source and extended versions to compare if the additional data makes a difference.

The experts labeled each code file on a four-part Likert scale, indicating whether they *fully agree, slightly agree, slightly disagree,* or *fully disagree* the code fulfills a certain quality attribute. This enables a fine-granular ordinal multiclass classification. In addition, we also examine a less fine-grained binary classification setting. Here, we separate the code into supposedly perfect (*strongly agree*) and not fully perfect code. Problematically, the dataset is imbalanced: Most code files are labeled as readable, understandable, and not complex, whereas very few entries are considered the opposite. This can lead to underrepresented labels getting only little attention during training and to distorted evaluation results. For the binary setting, the distribution is less imbalanced. The distributions for both settings are depicted in Table 1. The values in parentheses denote only the publicly available data.

2.2 Architectures and Algorithms

There are a plethora of machine learning architectures available. In the following, we explain the chosen algorithms in detail. Besides the text and image classification algorithms described below, we deploy Support Vector Machines (SVM) [5], which are capable of processing both texts and images.

Text-Based Learning. Naive Bayes [31] is a common classifier for text-based input. Here, a TF-IDF analysis preprocesses the text and determines how important specific terms in the analyzed text are. Furthermore, various transformer architectures have become prevalent in text-based machine learning use

Table 1. Distribution of the dataset in both the multiclass (top) and binary case (bottom). The values in parentheses denote the number of entries from open-source projects.

Multiclass label	Number of data points		
	Readability	Understandability	Complexity
Strongly agree	203 (183)	193 (157)	26 (22)
Weakly agree	111 (79)	96 (76)	51 (41)
Weakly disagree	47 (38)	60 (51)	75 (60)
Strongly disagree	13 (4)	25 (20)	222 (181)
Binary label			
Supp. perfect code	203 (183)	193 (157)	222 (181)
Other	171 (121)	181 (147)	152 (123)

cases. Hence, we employ BERT [9], CodeBERT [12], and RoBERTa [28] in this study, too.

While BERT, the Bidirectional Encoder Representations from Transformers, can suffer from unfortunate random initialization, RoBERTa (Robustly optimized BERT approach) is considered more stable [28]. CodeBERT, in contrast, was designed specifically to analyze source code and its connection to natural language [12]. Due to the small size of our dataset, we have resorted to pretrained, publicly available models[1] and then fine-tuned them to the downstream task of maintainability prediction. For more information about these models please refer to [9,12,28].

Image-Based Learning. Convolutional neural networks are known to recognize specific features within images and classify images based on these structures. Due to external limitations of this study, we could not test all available neural network and deep learning setups. AlexNet [25] is a reasonable choice here since it consists of basic layers that integrate well with most machine learning frameworks. For the configuration of the network, we follow Karpathy et al. [22]. One challenge for convolutional neural networks, in general, is the need for a large training dataset.

[1] BERT: https://huggingface.co/bert-base-uncased.
RoBERTa: https://huggingface.co/roberta-base.
CodeBERT: https://huggingface.co/microsoft/codebert-base.

2.3 Training and Evaluation

Since the dataset is quite small, we dedicate 80% of the data for training and 20% for testing. This is a trade-off to accommodate both needs – a large enough training part for the training-intensive architectures, as well as a large enough testing part for evaluating and comparing the approaches. Using stratified splits accounts for equal label distributions in both partitions, thus mitigating the effects of the imbalanced label distribution. In contrast to related work [42], we shuffled the dataset before splitting and did not consider project boundaries.

During training, we applied grid-search cross-validation to identify the best performing hyper-parameters and internal preprocessing options. For the text-based approaches, we examined e.g. the use of stemming, camelCase splitting, and the number of tokens to be respected in n-grams. For the SVM architectures, we varied their internal kernels, namely Polynomial, Sigmoid and Radial Basis kernels.

Metrics to evaluate machine learning models are based on different perspectives on the confusion matrix. For multiclass classification, there are two ways to calculate performance scores: In macro-aggregation, the respective metric is applied to each class separately and aggregated afterward. Aggregating all classes before calculating the respective metrics is called micro-aggregation. In this study, we use micro-aggregation. In this case, F-Score, precision, recall, and accuracy yield identical values when evaluating multiclass predictions. For the remainder of this paper, we will thus only use F-Score to refer to this value. Due to the imbalance of the dataset, we consider the Matthews Correlation Coefficient (MCC) [14] as well. Its use is common and suggested in the defect prediction domain, where imbalanced data distributions are commonly observed [50]. The MCC measures the alignment of two raters while considering agreement might happen by chance. In our context, we consider the learned model the first rater, and the ground truth as the output of a second rater. A value equal to zero indicates random alignment, while a value of 1 indicates perfect alignment and a value of -1 corresponds to perfect inverse alignment.

To put the performance of all learned models into context, we establish two baselines. A naive ZeroRule classifier identifies the most common label in the training set and always predicts this label. Due to its constant nature, the MCC of this classifier is 0. Comparing the ratings of the individual human experts to the eventual consensus vote, we find frequent deviations between them. In fact, the average expert is only aligned with the consensus in approx. 63–70% of the cases, depending on the considered quality attribute. This human-level performance provides an illustrative, second baseline.

Both baselines are summarized in Table 2. For readability reasons, the table is restricted to performances on the extended dataset only. Please note the performance of the ZeroRule classifier depends on the data distribution. In our case, its values are identical for the binary and multiclass settings. This is because the most common class in the multiclass analysis is identical with the supposedly perfect code in the binary setting.

Table 2. Baselines for multi-class prediction and binary prediction

Multiclass baselines	Readability		Understand.		Complexity	
	MCC	F-Score	MCC	F-Score	MCC	F-Score
Average expert	0.451	0.658	0.440	0.633	0.511	0.703
ZeroRule classifier	0.000	0.543	0.000	0.516	0.000	0.594
Binary baselines						
Average expert	0.613	0.797	0.621	0.804	0.581	0.940
ZeroRule classifier	0.000	0.543	0.000	0.516	0.000	0.594

2.4 Preprocessing for Text-Based Prediction

Before we can use the labeled code files for machine learning, we have to preprocess them. For the text-based analysis, the code files are parsed as raw text and then tokenized. For the transformer models, we use their integrated tokenizers. As such, the BERT model comes with its own tokenizer, as do the RoBERTa and CodeBERT models. Since these transformer architectures only accept inputs of a length shorter than 512 tokens, we have to split the file into multiple parts and treat each part as a distinct data point if it originally contains more tokens [45]. Thus, the dataset size increases. Notably, the labels in our dataset have been assigned to the whole Java class. After splitting the code, we assign the original label to all its parts.

In contrast, for Naive Bayes and text-based SVM we could use the complete files. The necessary features are produced by a Term Frequency – Inverse Document Frequency (TF-IDF) analysis of the code file.

2.5 Preprocessing for Image-Based Prediction

For the image-based analysis, we transform the code files into syntax-highlighted images. We decided to add syntax-highlighting to i) ease the identification of relevant structures and ii) mimic an analyst opening the file in a code editor. The same color theme is used for all images. First, we transform the Java files to PDF files using `PDFCode`[2]. Second, these files are converted into PNG files of 680×680 pixels. This size ensures the color from the syntax-highlighting is still visible although single characters might be no longer readable, depending on the length of the code. Due to resource constraints, our implementation of AlexNet downsizes the images to 224×224 pixels similar to the original AlexNet [25]. Our experiments with higher resolution images have not led to significant improvements.

The code is positioned in the top center of each image. An example is provided in Fig. 2. There, the code is unreadable by design.

[2] https://github.com/xincoder/PDFCode.

Fig. 2. Image of syntax-highlighted source code from `UniformTexture.java` from *Art Of Illusion* [41]

2.6 Experiment Execution

AlexNet is implemented on top of `Keras` [6], while BERT and its derivatives use `PyTorch` [11]. Naive Bayes and SVMs are based on `scikit-learn` [34].

We conducted every experiment using the extended dataset and using only the open-source data. This allows for analyzing the effect of additional data points and increases the reproducibility of our results for those without access to the confidential data. A replication package is publicly available on GitHub[3].

Every experiment was executed with different random seeds to mitigate the effects of random bias. We limited ourselves to two seeds as we did not find large differences between the runs. The reported values correspond to the average.

[3] https://github.com/simonzachau/SWQD-predict-software-maintainability.

3 Experiment Results

Table 3. Prediction results on the extended dataset and performance obtained on the open-source data in parentheses.

Multiclass classifier	Readability		Understandability		Complexity	
	MCC	F-Score	MCC	F-Score	MCC	F-Score
Naive Bayes	0.196 (0.038)	0.587 (0.607)	0.136 (0.112)	0.540 (0.533)	0.000 (0.000)	0.600 (0.590)
SVM (text-based)	**0.358** (0.398)	**0.640** (0.705)	**0.332** (0.301)	**0.613** (0.598)	**0.284** (0.241)	**0.633** (0.623)
BERT	0.032 (0.017)	0.306 (0.316)	0.023 (0.017)	0.276 (0.273)	0.005 (−0.038)	0.259 (0.232)
RoBERTa	0.013 (0.001)	0.290 (0.314)	0.010 (−0.001)	0.280 (0.262)	−0.013 (0.021)	0.240 (0.271)
CodeBERT	−0.009 (0.027)	0.274 (0.327)	0.004 (0.042)	0.263 (0.295)	−0.012 (0.021)	0.234 (0.269)
SVM (image-based)	**0.232** (0.470)	**0.580** (0.713)	**0.302** (0.427)	**0.580** (0.631)	**0.402** (0.337)	**0.673** (0.615)
AlexNet	0.000 (0.000)	0.293 (0.607)	0.000 (0.000)	0.113 (0.205)	0.000 (0.000)	0.600 (0.164)
Binary classifier						
Naive Bayes	0.555 (0.538)	0.780 (0.779)	0.521 (0.695)	0.760 (0.844)	0.464 (0.478)	0.747 (0.746)
SVM (text-based)	**0.609** (0.554)	**0.807** (0.787)	**0.660** (0.657)	**0.827** (0.820)	**0.637** (0.523)	**0.827** (0.771)
BERT	−0.013 (−0.042)	0.629 (0.568)	−0.029 (0.031)	0.646 (0.656)	0.027 (0.005)	0.585 (0.574)
RoBERTa	−0.001 (−0.017)	0.613 (0.600)	0.026 (0.035)	0.674 (0.697)	−0.050 (−0.015)	0.615 (0.608)
CodeBERT	0.028 (0.016)	0.627 (0.602)	0.036 (−0.002)	0.680 (0.646)	0.018 (0.032)	0.653 (0.645)
SVM (image-based)	**0.513** (0.565)	**0.760** (0.795)	**0.430** (0.530)	**0.713** (0.762)	**0.667** (0.495)	**0.840** (0.754)
AlexNet	0.000 (0.000)	0.453 (0.500)	0.006 (0.000)	0.500 (0.500)	0.000 (0.000)	0.400 (0.590)

For each classification approach, we investigate both the performance in a multiclass setting and a binary setting. The latter provides a first impression about the quality of the source code, while the multiclass prediction is more fine-grained. Table 3 lists the results concerning MCC and F-Score for each predicted quality attribute. The values in parentheses refer to the performance obtained using only the open-source data. The table shows the results for multiclass prediction in the top part, while the bottom part displays the results obtained for binary classification. Here, we combined three classes of the multiclass setting into one class as described in Sect. 2.1. To ease a comparison with the multiclass performance, we use F-Score and MCC to evaluate the binary prediction, too. Please note the micro-averaged F-Score yields the same value as the micro-averaged accuracy, precision, and recall scores.

3.1 Text-Based Classification

We find text-based SVMs outperform all other text-based approaches concerning MCC and F-Score independently of the predicted quality attribute. In the multiclass case, readability can be predicted with an MCC of 0.36 and F-Score of 0.64; understandability with an MCC of 0.33 and F-Score of 0.61; and complexity with an MCC of 0.28 and F-Score of 0.63. For binary predictions, an MCC of 0.61 and F-Score of 0.81 is reported for readability; 0.66 and 0.83 for understandability; and 0.64 and 0.83 for complexity.

The second-best classifier is Naive Bayes. It delivers the second-best results for readability and understandability. However, for the complexity label, its results are identical to the constant ZeroRule classifier. Notably, BERT, RoBERTa, and CodeBERT perform worse than the naive baseline classifier regarding the F-Score. Their MCC is close to 0 in all experiments, thus indicating only little information was learned during training.

In the binary setting, text-based SVMs outperform other text classification approaches as well. However, the difference to Naive Bayes is much smaller compared to the multiclass prediction. The obtained performance values are notably higher than in multiclass settings. The MCC is at 0.61, 0.66, and 0.64, resp.

3.2 Image-Based Classification

SVMs appear superior for image-based classification as well. AlexNet yields an MCC of 0 in all experiments, indicating the algorithm was not able to learn any relevant information and performed only as well as the constant classifier. Notably, its F-Score in the multiclass setting is even below the baseline for readability and understandability. In the binary case, AlexNet achieved F-Scores slightly above the baseline while the MCC remains at 0. In contrast, SVM obtained an MCC of 0.51 and F-Score of 0.76 for readability, 0.43 and 0.71, resp., for understandability, and 0.67 and 0.84 for complexity.

3.3 Interpretation

For an easier comparison of the seven approaches, we visualize the MCC obtained on the extended dataset in Fig. 3 (multiclass classification) and Fig. 4 (binary classification).

In our experiments, we found Naive Bayes and SVMs to perform better than convolutional neural networks and transformers. Further, we observe binary classification yields better results than multiclass prediction. On the extended dataset, the text-based approaches tend to perform better when predicting readability and understandability while the image-based approaches predict complexity more accurately. This is in line with our hypotheses.

Naive Bayes and SVMs perform better than expected, whereas AlexNet and the transformer approaches are below expectations. The SVM can play to its strengths of performing well on small datasets. At the same time, the small size of the dataset, as well as its imbalance, are likely to be the problem for convolutional neural networks and transformer architectures. Another evidence for this is that the interpretation of the dataset as binary classes almost exclusively achieved higher scores than in the multiclass scenario.

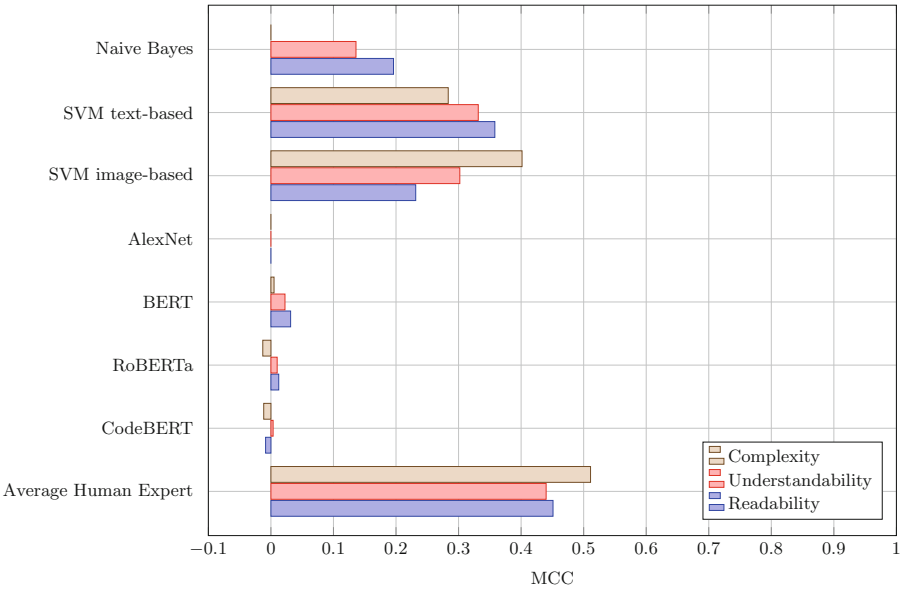

Fig. 3. Comparison of the MCC for multiclass classification

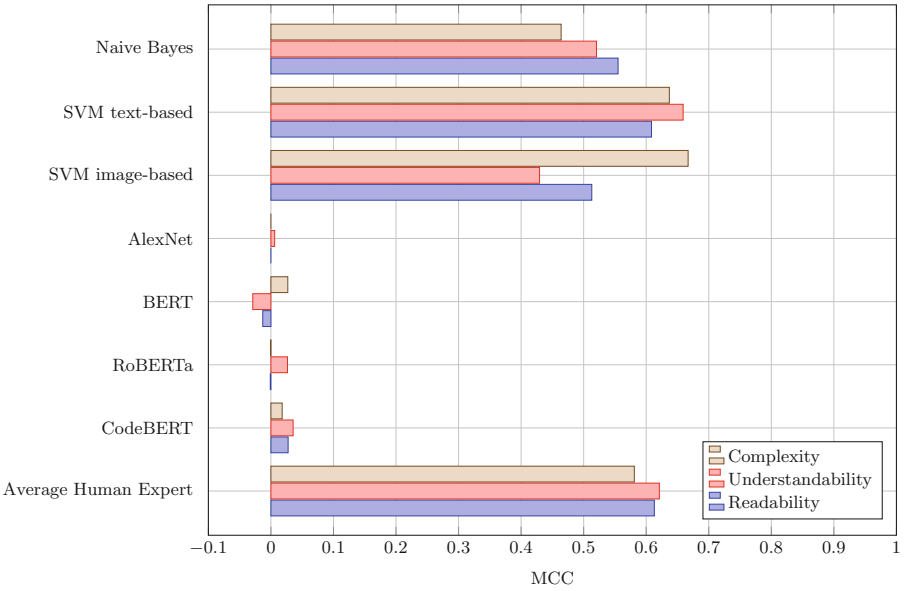

Fig. 4. Comparison of the MCC for binary classification

The transformer models BERT, RoBERTa, and CodeBERT achieve similar results in all experiments. However, the obtained MCC is close to 0. Thus, these models can be compared to randomly selecting a label. Still, all possible labels have been predicted at least once in our experiments.

Analyzing the predicted classes, we found some algorithms did not predict certain labels at all. For instance, Naive Bayes did only predict one out of four possible labels for complexity, and only two out of four labels for readability in the multiclass setting. In the binary interpretation, all labels are predicted at least once, which leads to higher scores. In our tests with AlexNet, always only one class is predicted in every experiment. This holds across both multiclass and binary prediction as well as across the extended and the open-source dataset. This renders the numbers unusable to compare to other approaches.

4 Discussion

The results are promising on the one hand, but also demonstrate room for improvement on the other hand. As of now, we observe large deviations in the ordinal multiclass prediction between the performance of the trained models and human performance. While the ZeroRule baseline is only slightly exceeded in this setting, it is outperformed by far in the binary setting. It is encouraging to see the performance of the text- and image-based SVM model even reaches the performance of an average human expert concerning the MCC. However, this binary setting probably oversimplifies a complex problem. Still, these results provide evidence on the potential of applying text and image classification algorithms to predict software quality.

The extended dataset contributes 70 additional data points. However, we cannot confirm that more available data leads to better results in general. For instance, text-based SVM yielded a higher MCC and higher F-Score predicting multiclass readability on the smaller dataset. The same observation is made using the image-based SVM to predict readability or understandability. Still, in most cases, better performance was observed using the extended dataset.

So far, we are not aware of other studies using image-based classification on source code. A summary of related work is presented in Sect. 5. Due to the recentness of the dataset we used, only few comparable experiments are available. The most comparable experiment is described in [42]. The authors focused on the *overall maintainability* judgment of the code instead of single quality attributes like readability or understandability. Their models are based on static code metrics. Besides, they apply a different validation technique respecting project boundaries, while we shuffled the dataset before splitting it. Thus, the experiment settings are too different to reasonably compare the results.

We found preprocessing to be an extensive challenge for both image-based and text-based inputs. Most image-based machine learning architectures require an input of quadratic and constant size. Source code neither has a defined length nor a quadratic layout. We chose an image size of 680×680 pixels, which is a reasonable trade-off between the high number of dimensions and the training time.

Another challenge is the layout of the image. Either the code is displayed as large as possible, or with normalized font size. We decided not to normalize due to large discrepancies in the length of the code files. The longest file in this dataset is 28 pages long (ISO A4 format). Had we chosen to scale the size according to this maximum value, the majority of the images would have appeared completely void. Since the algorithms try to recognize structural patterns, we chose to display these structures as prominently, i.e. large, as possible.

In the text-based experiments, the hyper-parameters about stemming and camelCase splitting did not lead to significant differences in the performance of the SVM and Naive Bayes classifiers.

In this experiment, BERT and its derivatives could not predict any quality attribute reliably. We attribute the poor performance of the transformer models mainly to one characteristic: the limited input length. As they accept only input smaller or equal to 512 tokens, we had to split most code files. This is a problem if the quality defects leading to a bad quality judgment are not evenly distributed across a Java class, which is a reasonable assumption. If the use of text-based machine learning for quality evaluation is to be moved forward, quality labels on the granularity of smaller code snippets are needed. However, manually labeling a sufficient amount of data points was out of scope for this study.

For image-based approaches, AlexNet always predicted one class and neglected all others. Contrary to expectations, that one class varied between experiment runs and was not always the majority class. Oftentimes, such behavior indicates a bug. To validate our setup, we replaced the images of one class with black dummy images to contrast the otherwise predominantly white images. In that experiment, the prediction achieved a perfect result ($MCC = 1$). This confirms the correctness of the implementation. We also conducted experiments using AlexNet with larger input images. However, it did not yield a noticeable difference. We hypothesize that the content of the images looks too similar for the convolutional neural network to identify useful characteristics.

4.1 Threats to Validity

The biggest threats to validity are introduced by the selected algorithms and used dataset. Though it is manually labeled and believed to contain the consensus of expert assessments, the dataset contains only a relatively small number of samples. Still, it is significantly larger than other, commonly used datasets such as the Li-Henry dataset [27]. However, the construct validity of manually labeled datasets is inherently threatened. Especially regarding software quality, which is deliberately defined vaguely [20,21], there exist several different viewpoints. We mitigated this threat by—instead of referring to the broad term software quality or maintainability—asking for evaluations with respect to more precise sub-attributes. In addition, the labeling platform offers explanations of the single attributes as tooltips. However, we recognize that participants may still interpret these terms differently.

As mentioned earlier, a large amount of machine learning algorithms exists. We made sure to include both simple (Naive Bayes and SVMs) and more

sophisticated models such as transformers and neural networks. We acknowledge using more or different approaches may have led to different results. Due to the limitations of this study and the long training periods required by most approaches, we had to limit ourselves to a subset of all applicable algorithms. To mitigate the potential effects of random seeds, we performed each experiment twice and report the average. Though we did not find significant deviations between the runs, one could repeat the experiment several times more. Another threat is the bias introduced by the chosen train-test split, which we mitigated by shuffling and stratifying the data. While we are going to further improve these weaknesses in future experiments, we see value in this preliminary study and its results. To increase the reproducibility of our results, we report the performance of our classifiers on the publicly available dataset in Sect. 3 and provide a replication package.

4.2 Future Work

There are various ways of how to further improve our results and setup. Even the extended version of the used dataset is small compared to those typically used for image- or text-based learning. A larger and more balanced dataset can likely improve results for most of our approaches. The dataset at hand also admits a numerical interpretation of the labels. Hence, modeling the prediction as a regression model is an interesting possibility. In the future, we plan to analyze other machine learning architectures and incorporate techniques from Explainable Artificial Intelligence to foster the debugging and interpretation of the results.

5 Related Work

Automated software quality evaluation and control is an increasingly important topic. Lately, machine learning has been used to evaluate characteristics that typically need to be interpreted by human experts. This includes, e.g., maintainability prediction [17,27,43] or code smell detection [13,33]. An overview of machine learning techniques for code smell detection is provided in [10].

Text-based models for code have been utilized by Palomba et al. [32] to identify code smells based on textual analysis. Salem and Banescu [39] used the TF-IDF of source code to foster metadata recovery attacks on obfuscated source code. Corazza et al. [7] performed a study where they manually analyzed code comments and predicted human ratings using TF-IDF as well. Buse and Weimer [2,3] developed a metric for code readability based on entropy within the code. Their model was later refined by Posnett et al. [36]. While they do predict the readability of code, they use static measurements as features. In contrast, we use textual or image representations of the code.

To predict software maintainability, several related studies use a dataset published by Li and Henry [27], which refers to the number of changed lines as a proxy for maintainability. The data is drawn from only two software systems,

which are programmed in Classic-ADA. Then, regression models are used to predict the number of changed lines [27]. Kaur and Kaur [23] summarize 27 experiments using this dataset. Furthermore, neuro-genetic algorithms were used by Kumar et al. [26], while van Koten and Gray applied Bayesian Networks [46].

Similar to these studies, Malhotra and Lata [29] use the observed changes in software systems as their target variable. Then, they discretize the data into binary classes corresponding to high and low maintainability. However, they do not provide the threshold used to separate them and mostly focus on the effects of data preprocessing techniques.

Another dataset for C programs was created in 1987 by Harrison and Cook [15]. This dataset is used for example by Xing et al. [49], who trained support vector machines on it, or by Khoshgoftaar et al. [24], who used regression models.

Other studies aim to predict the rating of human experts instead of code changes. Using the same dataset as in our study, the maintainability of code was predicted in [42]. Here, the authors employ a human-level baseline as well to put the performance of the evaluated machine learning classifiers into context. However, static code metrics are used as input and a different aspect of the dataset was chosen as the label. Hegedűs et al. [17] predicted the perceived changeability of methods using a three-fold label. They achieved an accuracy of 0.76, while the constant baseline classifier already yielded an accuracy of 0.67. On class-level, Schnappinger et al. [43] achieved an accuracy of 0.81, using a three-fold scale, too. Hayes and Zhao [16] used the perceived maintainability of software developed by students and developed a regression model to predict the judgment.

So far, we observe studies relying on human evaluations often do not report which maintainability sub-aspects the experts focused on [16,35,43], do not share their data publicly [43], or rely on the opinion of a single expert [16,18,35].

In this study, we target three fine-granular sub-dimensions of maintainability and evaluate classification techniques chosen specifically for these attributes. We explore the use of image and text classification algorithms to predict the readability, understandability, and complexity of source code.

6 Conclusion

Current machine learning approaches for predicting expert software quality evaluations often base their prediction on static code metrics. In related domains, image and text classification reached significant results as well, suggesting their potential use in quality prediction. In this study, we investigate how well *text-based* and *image-based* classification algorithms can predict readability, understandability, and complexity of code. We compare five text-based machine learning architectures (Naive Bayes, Support Vector Machines, BERT, RoBERTa, CodeBERT) and two image-based classifiers (Support Vector Machines, AlexNet). The labels are drawn from a publicly available, manually labeled dataset. We examine both a fine-granular ordinal multiclass classification and binary classification settings.

Using text-based input, Support Vector Machines outperform other algorithms by a large margin. In the binary classification setting, they are able to predict the readability, understandability, and complexity of source code with Matthews Correlation Coefficients above 0.61 and F-Scores above 0.81. Regarding image-based classification, Support Vector Machines yield the best results as well with F-Scores between 0.71 and 0.76. Although the employed models outperform a ZeroRule baseline classifier, the multiclass prediction does not yet reach an operational level. In contrast, in a simplified binary setting, our models reach human-level results. This demonstrates the potential of image and text classification algorithms.

However, in this preliminary study, we identified several open challenges for future research: In our view, the main challenge for the applicability of these approaches is currently posed by their need for fixed-size inputs. Indeed, state-of-the-art transformer models require text samples of fixed length. Similarly, most image-based algorithms assume a constant image size. This requires a preprocessing of source code files of unbounded length and arbitrarily complex structure into fixed-size data points, which in our experiments caused a deterioration of data quality. In particular, the partitioning of source code into fixed-length strings or fixed-size images did not match the granularity of the available labels.

This preliminary study opens an interesting line of research in quality prediction. As this was our first foray into using text- and image-based machine learning for software quality prediction, we are confident that subsequent work will improve on the identified limitations.

References

1. Banker, R.D., Datar, S.M., Kemerer, C.F., Zweig, D.: Software complexity and maintenance costs. Commun. ACM **36**(11), 81–95 (1993)
2. Buse, R., Weimer, W.: A metric for software readability. In: Proceedings of the 2008 International Symposium on Software Testing and Analysis, pp. 121–130. ACM (2008)
3. Buse, R., Weimer, W.: Learning a metric for code readability. IEEE Trans. Software Eng. **36**(4), 546–558 (2010)
4. Campbell, G.A.: Cognitive complexity: an overview and evaluation. In: Proceedings of the 2018 International Conference on Technical Debt, pp. 57–58 (2018)
5. Chang, Y.W., Hsieh, C.J., Chang, K.W., Ringgaard, M., Lin, C.J.: Training and testing low-degree polynomial data mappings via linear SVM. J. Mach. Learn. Res. **11**(48), 1471–1490 (2010)
6. Chollet, F.: Keras (2015). https://github.com/fchollet/keras
7. Corazza, A., Maggio, V., Scanniello, G.: Coherence of comments and method implementations: a dataset and an empirical investigation. Software Qual. J. **26**(2), 751–777 (2018)
8. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Stat. Soc.: Ser. B (Methodol.) **39**(1), 1–22 (1977)
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North, pp. 4171–4186 (2019)

10. Di Nucci, D., Palomba, F., Tamburri, D.A., Serebrenik, A., De Lucia, A.: Detecting code smells using machine learning techniques: are we there yet? In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 612–621. IEEE (2018)
11. Facebook: Pytorch (2020). https://pytorch.org
12. Feng, Z., et al.: CodeBERT: a pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP 2020. Association for Computational Linguistics, Online, November 2020
13. Fontana, F.A., Zanoni, M., Marino, A., Mäntylä, M.V.: Code smell detection: towards a machine learning-based approach. In: 2013 IEEE International Conference on Software Maintenance, pp. 396–399. IEEE (2013)
14. Gorodkin, J.: Comparing two k-category assignments by a k-category correlation coefficient. Comput. Biol. Chem. **28**, 367–374 (2004)
15. Harrison, W., Cook, C.: A micro/macro measure of software complexity. J. Syst. Softw. **7**(3), 213–219 (1987)
16. Hayes, J.H., Zhao, L.: Maintainability prediction: a regression analysis of measures of evolving systems. In: 21st IEEE International Conference on Software Maintenance (ICSM 2005), pp. 601–604. IEEE (2005)
17. Hegedűs, P., Bakota, T., Illés, L., Ladányi, G., Ferenc, R., Gyimóthy, T.: Source code metrics and maintainability: a case study. In: Kim, T., et al. (eds.) ASEA 2011. CCIS, vol. 257, pp. 272–284. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-27207-3_28
18. Hegedűs, P., Ladányi, G., Siket, I., Ferenc, R.: Towards building method level maintainability models based on expert evaluations. In: Kim, T., Ramos, C., Kim, H., Kiumi, A., Mohammed, S., Ślęzak, D. (eds.) ASEA 2012. CCIS, vol. 340, pp. 146–154. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35267-6_19
19. Hindle, A., Barr, E.T., Gabel, M., Su, Z., Devanbu, P.: On the naturalness of software. Commun. ACM **59**(5), 122–131 (2016)
20. ISO/IEC: ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Technical report (2010)
21. Jung, H.W., Kim, S.G., Chung, C.S.: Measuring software product quality: a survey of ISO/IEC 9126. IEEE Softw. **21**(5), 88–92 (2004)
22. Karpathy, A., Fei-Fei, L., Johnson, J.: Convolutional neural networks for visual recognition, Stanford University (2017). http://cs231n.github.io
23. Kaur, A., Kaur, K.: Statistical comparison of modelling methods for software maintainability prediction. Int. J. Software Eng. Knowl. Eng. **23**(06), 743–774 (2013)
24. Khoshgoftaar, T.M., Munson, J.C.: Predicting software development errors using software complexity metrics. IEEE J. Sel. Areas Commun. **8**(2), 253–261 (1990)
25. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Adv. Neural. Inf. Process. Syst. **25**, 1097–1105 (2012)
26. Kumar, L., Naik, D.K., Rath, S.K.: Validating the effectiveness of object-oriented metrics for predicting maintainability. Procedia Comput. Sci. **57**, 798–806 (2015)
27. Li, W., Henry, S.: Object-oriented metrics that predict maintainability. J. Syst. Softw. **23**(2), 111–122 (1993)
28. Liu, Y., et al.: Roberta: a robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019)
29. Malhotra, R., Lata, K.: An empirical study on predictability of software maintainability using imbalanced data. Software Qual. J. **28**(4), 1581–1614 (2020)

30. McCabe, T.J.: A complexity measure. IEEE Trans. Software Eng. **4**, 308–320 (1976)
31. Murphy, K.: Naive Bayes classifiers. Univ. Br. Columbia **18**(60) (2006)
32. Palomba, F., Panichella, A., De Lucia, A., Oliveto, R., Zaidman, A.: A textual-based technique for smell detection. In: 2016 IEEE 24th International Conference on Program Comprehension (ICPC), pp. 1–10. IEEE (2016)
33. Pecorelli, F., Palomba, F., Di Nucci, D., De Lucia, A.: Comparing heuristic and machine learning approaches for metric-based code smell detection. In: 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), pp. 93–104. IEEE (2019)
34. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
35. Pizzi, N.J., Summers, A.R., Pedrycz, W.: Software quality prediction using median-adjusted class labels. In: Proceedings of the 2002 International Joint Conference on Neural Networks, IJCNN 2002, vol. 3, pp. 2405–2409. IEEE (2002)
36. Posnett, D., Hindle, A., Devanbu, P.: A simpler model of software readability. In: Proceedings of the 8th Working Conference on Mining Software Repositories, pp. 73–82. ACM (2011)
37. Ray, B., Hellendoorn, V., Godhane, S., Tu, Z., Bacchelli, A., Devanbu, P.: On the 'naturalness' of buggy code. In: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 428–439 (2016)
38. Raymond, D.R.: Reading source code. In: CASCON, vol. 91, pp. 3–16 (1991)
39. Salem, A., Banescu, S.: Metadata recovery from obfuscated programs using machine learning. In: Proceedings of the 6th Workshop on Software Security, Protection, and Reverse Engineering, pp. 1–11 (2016)
40. Schnappinger, M., Fietzke, A., Pretschner, A.: Defining a software maintainability dataset: collecting, aggregating and analysing expert evaluations of software maintainability. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 278–289. IEEE (2020)
41. Schnappinger, M., Fietzke, A., Pretschner, A.: A software maintainability dataset, September 2020. https://doi.org/10.6084/m9.figshare.12801215
42. Schnappinger, M., Fietzke, A., Pretschner, A.: Human-level ordinal maintainability prediction based on static code metrics. In: Evaluation and Assessment in Software Engineering, EASE 2021, pp. 160–169 (2021)
43. Schnappinger, M., Osman, M.H., Pretschner, A., Fietzke, A.: Learning a classifier for prediction of maintainability based on static analysis tools. In: Proceedings of the 27th International Conference on Program Comprehension, pp. 243–248. IEEE (2019)
44. Schnappinger, M., Osman, M.H., Pretschner, A., Pizka, M., Fietzke, A.: Software quality assessment in practice: a hypothesis-driven framework. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, p. 40. ACM (2018)
45. Sun, C., Qiu, X., Xu, Y., Huang, X.: How to fine-tune BERT for text classification? In: Sun, M., Huang, X., Ji, H., Liu, Z., Liu, Y. (eds.) CCL 2019. LNCS (LNAI), vol. 11856, pp. 194–206. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32381-3_16
46. Van Koten, C., Gray, A.: An application of Bayesian network for predicting object-oriented software maintainability. Inf. Softw. Technol. **48**(1), 59–67 (2006)
47. Von Mayrhauser, A., Vans, A.M.: Program comprehension during software maintenance and evolution. Computer **28**(8), 44–55 (1995)

48. Wong, W.E., Gao, R., Li, Y., Abreu, R., Wotawa, F.: A survey on software fault localization. IEEE Trans. Software Eng. **42**(8), 707–740 (2016)
49. Xing, F., Guo, P., Lyu, M.R.: A novel method for early software quality prediction based on support vector machine. In: 16th IEEE International Symposium on Software Reliability Engineering (ISSRE 2005), pp. 10-pp. IEEE (2005)
50. Yao, J., Shepperd, M.: Assessing software defection prediction performance: why using the matthews correlation coefficient matters. In: Proceedings of the Evaluation and Assessment in Software Engineering, pp. 120–129 (2020)
51. Zhou, Y., Leung, H.: Predicting object-oriented software maintainability using multivariate adaptive regression splines. J. Syst. Softw. **80**(8), 1349–1361 (2007)