Feature Analysis of Supervised Machine Learning Models in IDE-Based Learning Analytics

Exploring the use of correlation coefficients and p-values as feature utility measures through estimating student performance in an introductory programming course

Boye Borg Nygård

Master of Science in Computer Science
Submission date: June 2018
Supervisor: Hallvard Trætteberg, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Due to the recent proliferation of large datasets collected on human behavior in digital environments, IDE-based learning analytics using supervised learning has emerged as a scientific field. However, due to its novelty, research methods tailored to the needs of IDE-based learning analytics are yet to be developed. In this paper, a methodology for evaluating features used in supervised learning models in relation to their effect on the model's performance is presented. We show that correlation coefficients in combination with p-values can be used as a measure of a feature's usefulness. The goal of the method is to enable researchers to understand and compare different features, allowing a higher degree of utilization of previous research, and increasing the overall research value of supervised learning in IDE-based learning analytics.

Sammendrag

(Norwegian translation of the Abstract)

Stor tilgang på data om menneskelig adferd på digitale plattformer, har gitt grobunn for IDE-basert læringsanalyse ved bruk av veiledet-læring. Grunnet forskingsfeltets korte fartstid finnes det få utarbeidede metodikker tilpasset IDE-basert læringsanalyse. I denne masteroppgaven presenterer vi en metodikk for å evaluere inndata i veiledet-læring sin påvirkning på modellens prestasjon. Vi viser at korrelasjonskoeffisienter i kombinasjon med sannsynlighetsverdier kan bli brukt som mål på nytteverdi av inndata. Målet med metoden er å øke forståelse av inndata sin nytteverdi og forenkle prosessen å sammenlikne ulike inndata. Dette for å øke gjenbrukbarheten av tidligere forskning, og på den måte forsterke forskningsverdien av veiledet-læring innen IDE-basert læringsanalyse.

Preface

This paper is a master thesis written at the Department of Computer Science (IDI), Norwegian University of Science and Technology (NTNU). The writing of the thesis, and all work related to it has been conducted between January 22nd and June 15th, 2018.

I would like to thank my supervisor, Hallvard Trætteberg, the support and feedback provided throughout the writing of this paper. In addition, I would like to thank Siddise Hirpa, Emil Henry Flakk, and Pernille Wangsholm for fruitful discussions and constructive feedback.

Table of Contents

List of Figures and Tables

1 Introduction

1.1 Context

With the recent proliferation of large datasets collected from human behavior in digital environments, scientific disciplines concerned with analyzing and understanding human behavior has burgeoned. One such discipline is *learning analytics*, which analyzes data from students and their environment to understand and improve the learning process [4]. In 2012 the U.S. Department of Education published an issue brief on the use of learning analytics, which concerns itself with analyzing learning processes, in the U.S. system of higher education [5]. This issue brief highlighted the usefulness of predictive models forecasting student performance and encouraged researches to continue conducting research that would help instructors become more efficient. Since then, the field of learning analytics has sky rocketed, and the 2016 Horizon Report on higher education said they expected "*learning analytics to be increasingly adopted by higher education institutions*" [6].

Yacef [7] showed the trends of using data gathered from learning processes in machine learning system as early as 2009, and in 2011 the first International Conference on Learning Analytics and Knowledge was held. Aided by the rapid development of machine learning, learning analytics has grown from a niche to a well-established research area. In 2010, only 118 research articles containing the phrase "learning analytics" were published [8]. This number has increased every year since then, and in 2017 over 5500 articles were published (see Figure 1), an increase of over 4500%.

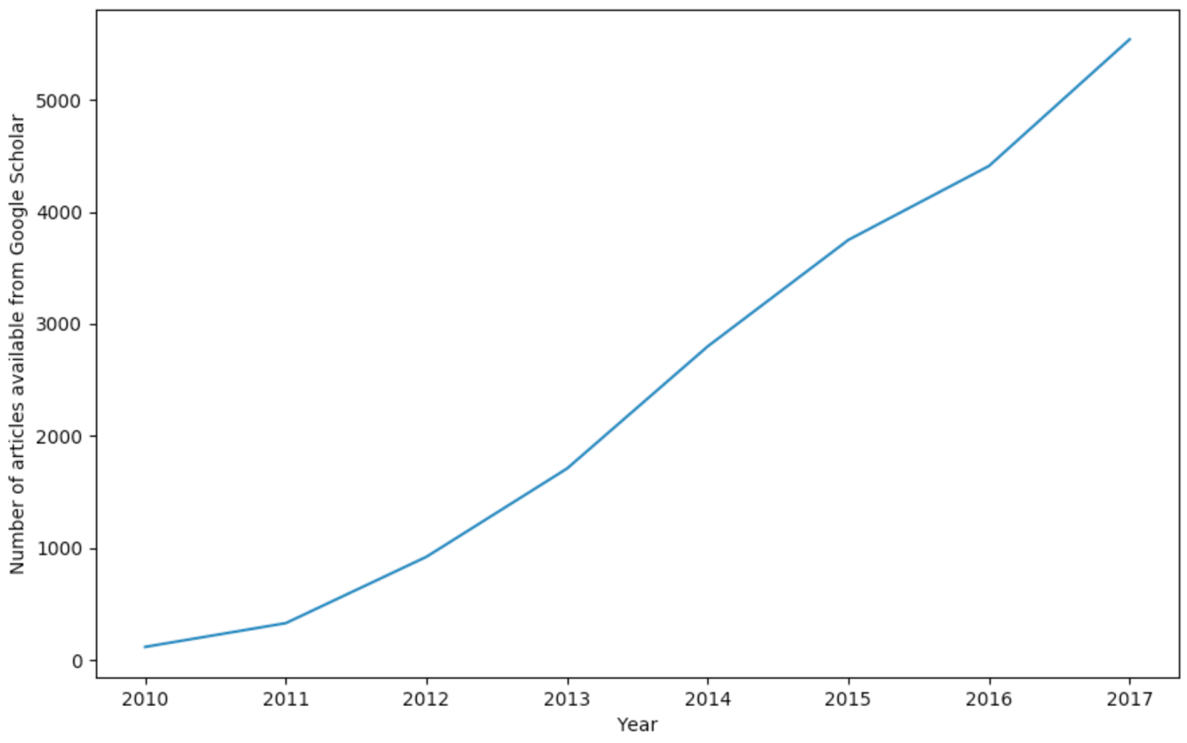


Figure 1: Number of published articles each year containing the phrase "learning analytics" available from Google Scholar.

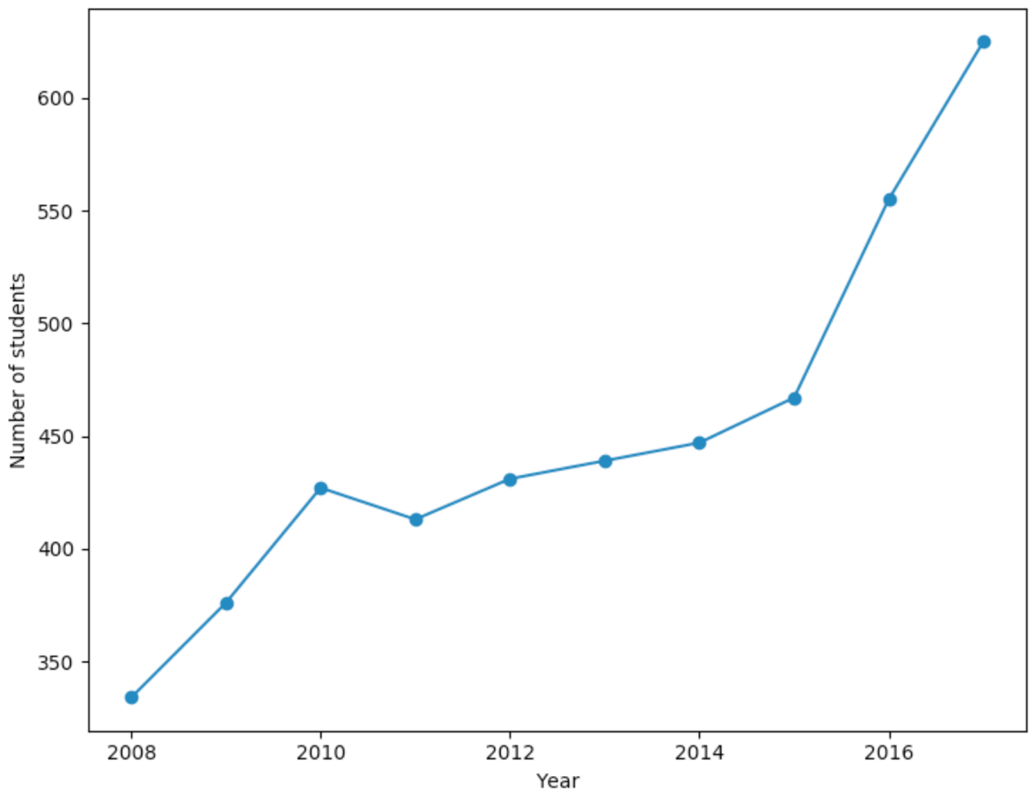


Figure 2: Number of students participating in TDT4100 at NTNU.

In addition to the raise of learning analytics, the demand for programming in education is increasing [9]. At NTNU, the number of students enrolling in the introductory course to object oriented programming (TDT4100) has increased with over 87% since 2008 (see Figure 2). In order to best meet this high demand for programming knowledge, teachers need insight on the learning process of their students. With the aid of learning analytics, a teacher could better understand how their students learn, and more easily identify students in need of assistance.

Capturing data from a learning process is not trivial, as the learning environment usually consists of several sub-environments that are not all easily monitored. For programming courses, these sub-environments might include online quizzes, lectures and Q&A sessions with a teaching assistant. However, studies has shown that most programming courses offered at universities are using an integrated development environment (IDE) [10, 11]. This makes the IDE a common sub-environment across many programming courses at different universities, thus making IDE-based learning analytics an important area of research.

1.2 Problem

When comparing the search results from Google Scholar[1], we find that only 2% of articles published on learning analytics explores IDE-based learning analytics using classification [12, 13] (about 4% explores IDE-based learning analytics without the mention of classification [14]). Within this small niche of learning analytics, most published research is not reproduceable nor follows any common research method [15-20]. When conducting learning analytics using classification, one of the main

[1] https://scholar.google.no

challenges is to find what features that will provide useful information to the classification model. Regardless, very few research articles published on IDE-based learning analytics discuss or analyze the usefulness of the features. Some articles do not even contain a complete list of the used features. This makes it difficult to compare, validate, reproduce and build upon the existing research, yielding low research value.

A model for conducting IDE-based learning analytics was proposed by Hundhausen, et al. [3]. This method includes several features that might be used in a classification model. However, no approach on evaluating the usefulness of these features in relation to the model's target, nor any methods for evaluating the learning analytics system as a whole, is described. To improve on this, and encourage research on IDE-based learning analytics, we want to extend the model by proposing a way to measure the features usefulness and statistical significance. In this paper, we seek to answer the following research question:

> *Is an analysis evaluating the usefulness of features valuable to IDE-based learning analytics research?*

1.3 Limitations

This paper is written as the master thesis required to receive a master's degree in computer science (M.Sc.) at NTNU and is completed within a time limit of 20 weeks. Due to the time limitation, we were not able to collect new data for the experiment and had to use existing data previously collected.

2 Background

2.1 Learning Analytics

Most papers on **learning analytics** (LA) refer to the definition provided by the 1st International Conference on Learning Analytics and Knowledge 2011 when defining the field of LA [4]:

> *“Learning analytics is the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimising learning and the environments in which it occurs”*[2]

The collection of educational data is often referred to as **educational data mining** (EDM) [7]. There are several ways of using EDM in learning analytics. An issue brief published by the U.S. Department of Education [5] propose five different learning analytics applications where EDM can be used:

* Modeling knowledge, behavior, and experience
* Profiling
* Modeling domain knowledge components
* Trend analysis
* Adapting learning experience

All learning analytics systems consists of three main steps [2]:

[2] https://tekri.athabascau.ca/analytics/

1. Collect data from a learning environment
2. Analyze the data
3. Use the analysis to improve the learning environment

When learning analytics are applied to improve the learning environment, that environment changes. This new, changed environment might contain information that is not currently collected nor analyzed, and therefore not used by the system. Imagine learning analytics is used to predict the final grade of students, and that this prediction is used by the teaching staff to decide which students to assist. The use of this system introduces new information to the learning environment that is currently not utilized by the system. For example, observing how students react to receive assistance might yield useful information that could improve the prediction. To include this new information, the three steps above must be executed on the new learning environment. However, this might result in the environment changing yet again. Thus, learning analytics is by many considered an iterative process [2, 3].

2.2 Machine Learning

Machine learning is a field of artificial intelligence employing statistical techniques to learn patterns in empirical data [21]. Machine learning systems are usually divided into three main categories; *supervised learning*, *reinforcement learning,* and *unsupervised learning*, depending on the feedback provided to the system while learning [22]:

* **Unsupervised learning:** The system is not provided with feedback, and the system has to find structure in the inputs by itself. The system is often used to find unknown patterns

and relations within data (clustering) or discover outliers and abnormal data points.

* **Supervised learning:** Each input is accompanied with a correct or desired output (feedback). The system has to find patterns and relations between the input and output, then use this to approximate a function that maps the inputs to the outputs, called the hypothesis.

* **Reinforcement learning:** The system performs a set of actions and is infrequently provided with feedback in the form of reward or punishment in relation to its performance.

In this paper, we will focus on supervised learning used in conjunction with learning analytics. Hence, unsupervised and reinforcement learning will not be discussed.

2.2.1 Supervised Learning

The goal of supervised learning is to approximate a target function, f, that maps some set A to another set B (Equation 1). Both f, A and B are unknown.

$$f: A \rightarrow B \tag{1}$$

To be able to approximate the target function, the system is given a set of observations, D (Equation 2). The observations consist of n input-output pairs of the unknown function f.

$$D = \{(x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_n, f(x_n))\} \tag{2}$$

The task of the supervised learning system is to create a new function, $\hat{f}$, that approximates f based on the observations provided [22]. To do this, we need a function that measure how different the approximation is from the target function, called a **loss function**. This uses the input–output pairs from the observation to evaluate how closely $\hat{f}$ resembles f. By minimizing the loss, the approximation becomes as close as possible to the target function.

Several different loss functions exists [23], as different machine learning models requires different loss functions. One deciding factor of which loss function to use is what kind of learning problem the model solves. Supervised learning is usually divided into two different problem categories: **classification** and **regression**. When the output is a number, such as a stock price, the problem is called regression; when the output consists of a finite set of classes, such as dog breeds, the problem is called classification [22].

When using supervised learning on certain problems, the data is not easily divided into inputs. For example, imagine predicting the sentiment of a movie review (classification) based on the content of the review. One could think it would be possible to use every word in the review as an input. However, this turns out to be difficult, since every review probably is of different lengths, resulting in each observation having a different number of inputs. In addition, it is natural to assume that not all words are relevant when trying to predict the sentiment, such as “a”, “to” and so on. A principle called Ockham’s razor states that there exist an optimal subset of the data, that is as small as possible while still providing the machine learning model with enough information to create the best possible model [24]. However, finding this subset is not trivial.

Extracting inputs from the data is often called *feature engineering* [25], and each extracted input is called a **feature.** An example of a feature from the above example might be the number of exclamation marks or the number of times the word “good” is used. Selecting features is often done by assumptions, meaning features that are assumed to be relevant are included. The **bias** is a measure of how dependent our model is on our prior beliefs and assumptions. A small feature set will often result in a high bias. When creating a supervised learning model, a low bias is desired, which can be achieved by adding more features. However, by adding more features, irrelevant information might be introduced into the system, which will increase the **variance** of the model. This is known as the bias–variance tradeoff [1]. The more features, the more complex the model becomes, and the variance goes up. However, too few features yield a high bias that might be equally as bad (see Figure 3).

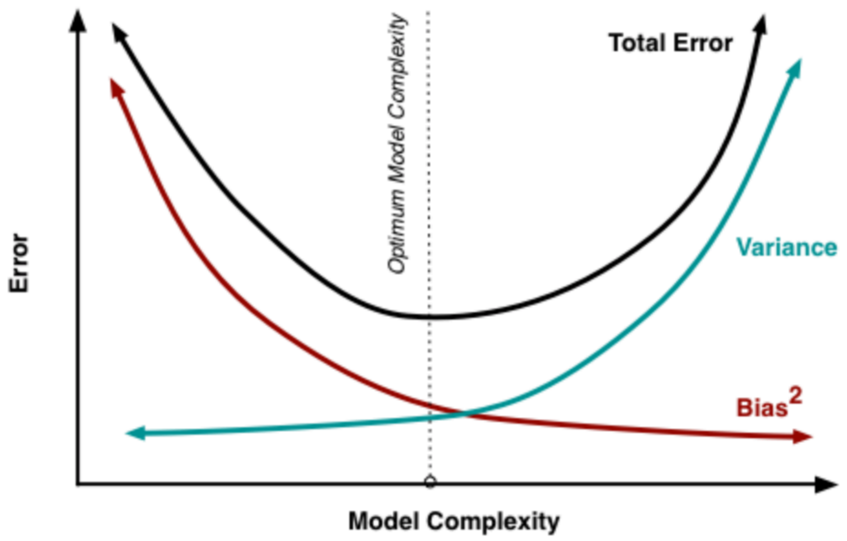


Figure 3: Visualization of bias and variance [1]

2.2.2 Evaluation

The process of optimizing the hypothesis function by minimizing the loss is called **training.** After training the model, we want to know how well it performs on new, unseen, inputs. This is called **testing.** To achieve this, the data is divided into two parts, D_1 and D_2, such that $D_1 \cup D_2 = D$ and $D_1 \cap D_2 = \emptyset$. D_1 can be used to train the model and D_2 to test the model afterward. However, different splits of D_1 and D_2 can yield different results. A common way of reducing this variance is called **k-fold cross-validation.** Cross-validation is a technique where the data is split into k equal sections. Then the model is trained k times on $k-1$ sections of the data, each time tested and evaluated on the section not used for training (see Figure 4). Each of the k sections are used exactly once for testing. The metric used for evaluating the performance of the model (see Section 2.3) is averaged over all tests completed.

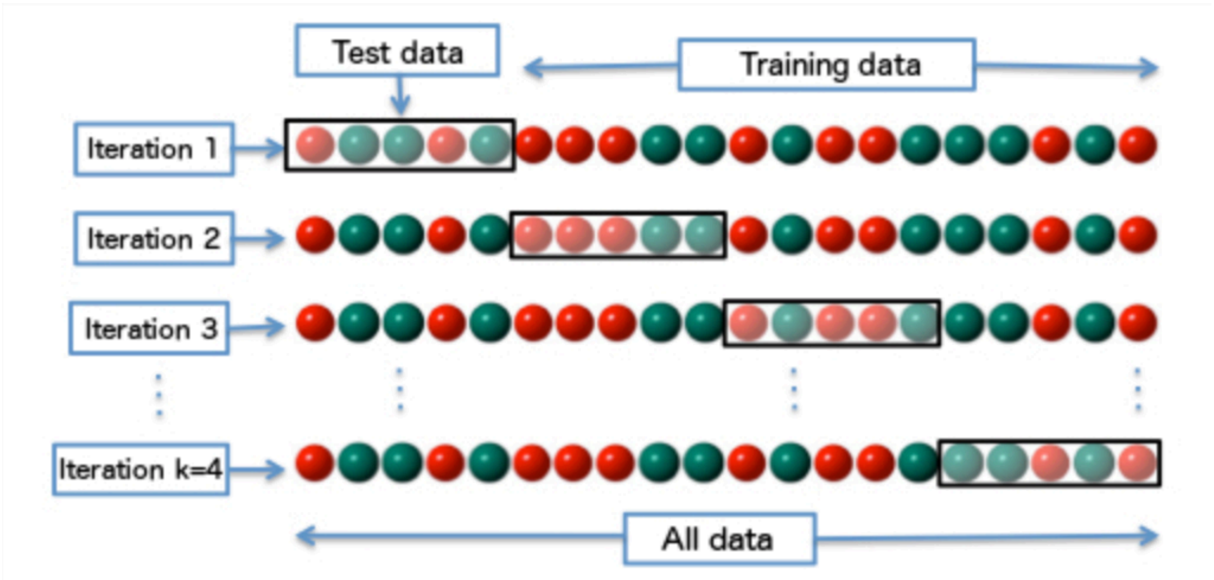


Figure 4: Example of 4-fold cross-validation (Fabian Flöck CC BY-SA 3.0).

2.3 Quality Assessment

2.3.1 Mean Square Error

A common metric for computing the error of a regression model is the **mean square error** (MSE). This measures the mean of the square errors of the system (see Equation 3), where y is the output of the training data, $\hat{y}$ is the output of the regression model, and n is the number of training samples.

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3}$$

When used in conjunction with k-fold cross-validation, the average of the mean square errors computed at each fold can also be used.

2.3.2 Precision and Recall

Two common metrics to test the quality of a classification model is **precision** and **recall.** To calculate precision and recall we need to categorize each classification of a given class into four categories: true positive, false positive, true negative and false negative (see Table 1).

	Is of class A	Is not of class A
Classified as A	True positive	False positive
Not classified as A	False negative	True negative

Table 1: Definition of true positives, false positives, true negatives and false negatives.

Precision is the number of samples correctly classified as a given class (true positive), divided by the total number of samples classified as that class (Equation 4). Precision is a measure of how precise the system is when classifying a certain class, i.e. how much you should trust it when it says something is of a certain class.

$$precision = \frac{true\,positives}{true\,positives + false\,posetives} \tag{4}$$

Recall is the number of samples correctly classified as a given class (true positive), divided by the total number of samples of that class (Equation 5). Recall is the ratio of correctly classified samples within a certain class.

$$recall = \frac{true\,positives}{true\,positives + false\,negatives} \tag{5}$$

2.3.3 F1-score

F1-score is a way to combine precision and recall into one metric. This is done by computing the harmonic mean of the precision and recall (Equation 6).

$$F1 = 2 \times \frac{precision\ \times recall}{precision + recall} \tag{6}$$

2.3.4 Correlation Coefficient

A correlation coefficient measures how correlated to variables, X and Y, are. One such coefficient is called **Pearson correlation coefficient** (PCC), and was introduced by Karl Person in the 1880s by ideas drawn from Francis Galton [26]. It ranges from negative one to positive one and are calculated by comparing the covariance of the two variables to the product of their standard deviations (see Equation 7). If the two variables have a positive correlation, meaning X increases when Y increases and vice versa, the correlation coefficient will be positive. When there is a negative correlation, X decreases when Y increases and vice versa, the coefficient is negative. If there is no correlation between the two variables, the coefficient is zero. The farther away from zero, the stronger the correlation.

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{7}$$

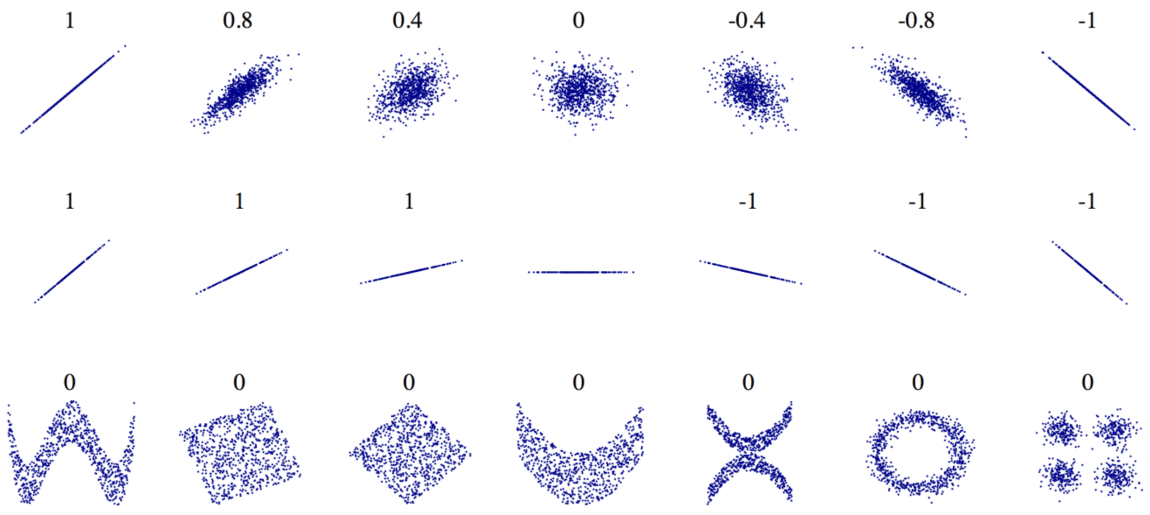


Figure 5: Different sets of points (x, y) and the Pearson correlation coefficient of x and y for each set (Denis Boigelot, CC0, public domain).

The equation for calculating PCC (Equation 7) divides the covariance of the two variables by the standard deviation of both variables multiplied together. PCC can be used to test if a feature (input) provide useful information in relation to the output in a supervised learning problem. When computing PCC for one feature and the output from the observations in a supervised learning problem, the covariance and standard deviations are usually unknown and must be estimated. If the feature, X, consists of n values, $[x_1, \ldots, x_n]$, and the output, Y, also contains n values, $[y_1, \ldots, y_n]$, then PCC can then be approximated through Equation 8.

$$\hat{\rho}_{X,Y} = \frac{\widehat{cov}(X,Y)}{\hat{\sigma}_X \hat{\sigma}_Y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \times \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{8}$$

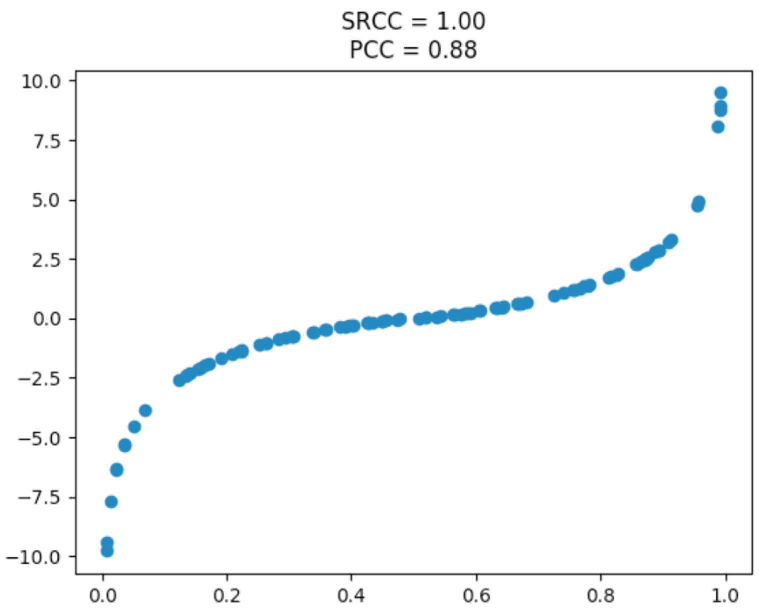


Figure 6: Comparison of Pearson correlation coefficient and Spearman's rank correlation coefficient.

The PCC is only able to detect linear correlation (see Figure 5). To counter this, a variation of PCC, called **Spearman's rank correlation coefficient** (SRCC) might be used instead. SRCC is similar to PCC, but instead of using the values from the two variables (X and Y) directly, their *ranks*, rg_X and rg_Y, are used (see Equation 9). Ranks are computed by individually sorting the values within each variable in ascending order. The rank of each value is its relative position within the sorted value list. The use of ranks makes SRCC more suited to find non-linear correlations (see Figure 6). It is however not able to detect non-monotonic correlation (see Figure 7).

$$\hat{\rho}_{rg_X, rg_Y} = \frac{\widehat{cov}(rg_X, rg_Y)}{\hat{\sigma}_{rg_X}\hat{\sigma}_{rg_Y}} \tag{9}$$

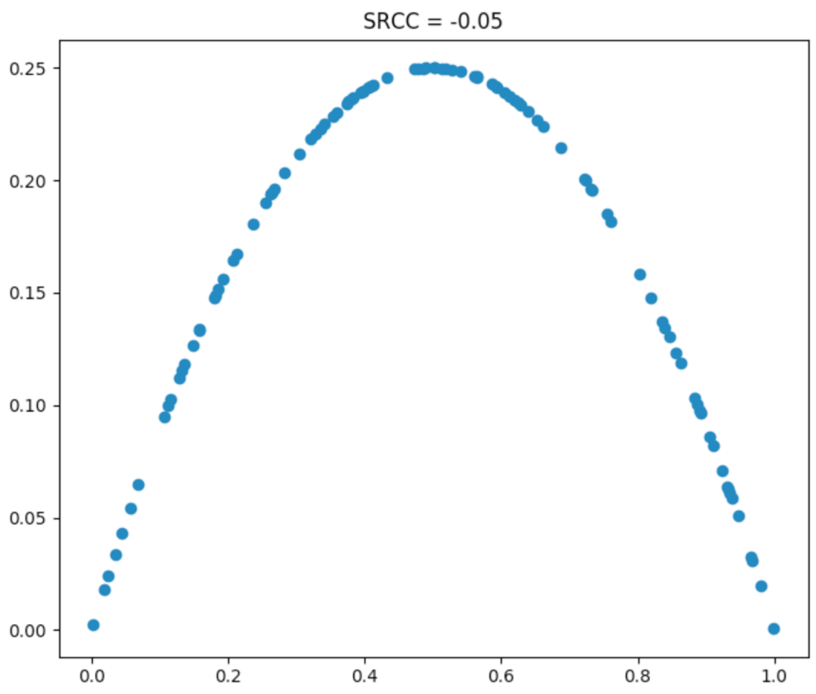


Figure 7: SRCC for non-monotonic correlation

2.3.5 P-value

In addition to the correlation coefficient, a metric indicating how statistically significant the correlation is can be useful. This can be estimated through a **p-value**, estimating the probability of the results happening by chance.

When evaluating the significance of a correlations in quantitative data, we start by assuming that there is no relationship. This is called the *null hypothesis* [27]. Then the probability of detecting the observed data if the null hypothesis is true, the p-value, is computed. Commonly, the null hypothesis is considered to stand if the p-value is greater than 0.05, or 1 in 20 [27]. The correlating is said to be significant if the p-value is less than 0.05, meaning there is less than 5% chance of the observations happening by chance.

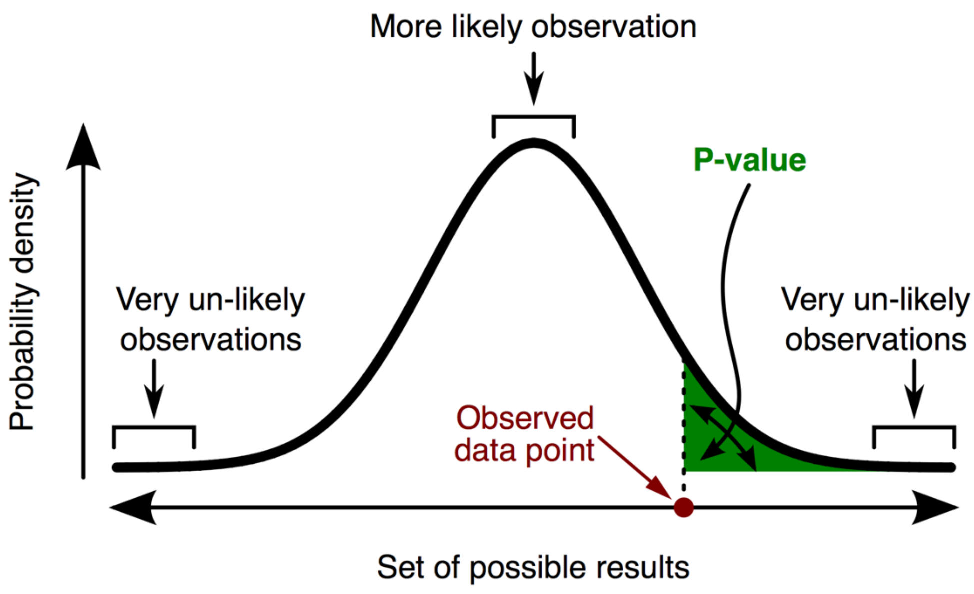


Figure 8: The p-value (shaded green) of an observation assumed to be drawn from a gaussian distribution when the null hypothesis is true (Chen-Pan Liao, CC BY-SA 3.0).

To calculate the p-value, one must assume the results of the statistical model is drawn from a specific distribution. A common assumption is that the statistical model generates a gaussian distribution (see Figure 8). One should always keep in mind that if this assumption does not hold, the p-value is not statistically valid. Hence, p-values should only be considered an indication of significance, and not an absolute fact, unless the underlying distribution is known.

3 Related Work

Over the past years, several papers have been published proposing different models for conducting learning analytics. In 2012, Chatti, et al. [2] presented a general process model for learning analytics. This is one of the most cited articles on learning analytics. Later, in 2017, Hundhausen, et al. [3] presented a similar model, tailored to the needs of IDE–based learning analytics. In this chapter, both models will be presented.

3.1 A Reference Model for Learning Analytics

In 2012, Chatti, et al. [2] presented a general reference model for learning analytics. They described the learning analytics process using an iterative cycle containing three steps: *data collection and pre–processing*, *analytics and action*, and *post–processing* (see Figure 9).

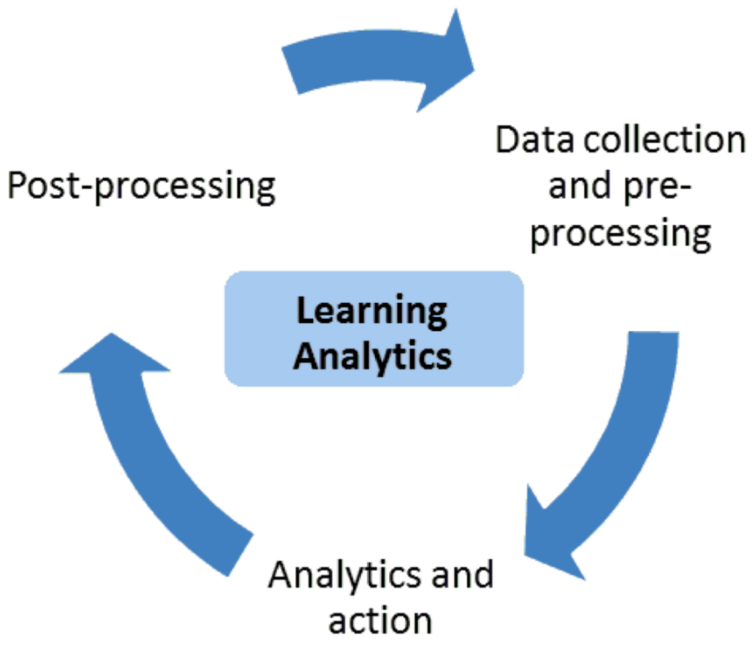


Figure 9: Learning analytics process by Chatti, et al. [2]

Data collection and pre-processing includes the collection of the data, and the preparation needed to use it in learning analytics methods. The pre-processing might include tasks such as data cleaning, data transformation and data reduction, but the paper does not elaborate on how to conduct such tasks.

In the analytics and action step, the data collected and pre-processed in the previous step is analyzed, and based on those results, actions are taken. Different method of analyzing is briefly mentioned, for instance clustering and classification, but evaluation of the results is not discussed. The actions can be viewed as the outcome of the learning analytics. Depending on the analysis conducted, an action might be making a teacher aware of students in need of assistance or providing feedback to students on how they should study towards the exam.

The last step, post-processing, involves improving the data and system. This could be to adjust or collect a new dataset, finding new variables to extract from the data to be used in the following iteration, altering the method used in the analytics step, or similar tasks.

3.2 IDE-Based Learning Analytics

In 2017, Hundhausen, et al. [3] published an article on using learning analytics on data collected through an *integrated development environment* (IDE). The article presents a similar iterative process model to Chatti, et al. [2] (see Figure 10).

The main difference from the model proposed by Chatti, et al. [2] (see Figure 9) is that the *analytics and action* step is split into multiple steps. The analytics task is here its own step (step two), and the action task is split into two parts, *design intervention*, and *deliver intervention*. The model does not explicitly contain any post-processing step, though this might be included within the first step for every iteration after the first.

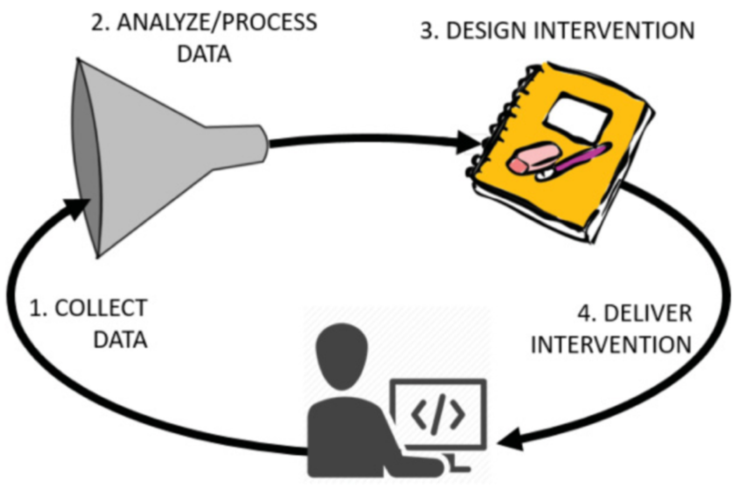


Figure 10: IDE-based learning analytics process model by Hundhausen, et al. [3]

In addition to the process model, the article presents different variables that could be extracted from the collected data and used in the analysis. These variables, often called features, are specific to IDE-based learning environments, such as BluJ, Eclipse and NetBeans. However, the article does not discuss how to evaluate the usefulness of each feature, nor the system as a whole.

4 Methodology

We are looking at the use of supervised learning in IDE-based learning analytics. One of the main challenges when using supervised learning is feature selection, hence quantifying the usefulness of different features in research conducted on IDE-based learning analytics using supervised learning might be valuable. This could make it easier for others to understand how different features affect the results, which features work well and which do not. This makes it easier for others to identify and focus on promising features, allowing researchers to build and improve upon one another.

However, the process model presented by Hundhausen, et al. [3] (see Section 3.1) does not address analysis of the extracted features. To improve this, we want to propose a method that can be used when conducting feature analysis on features used by supervised learning models in IDE-based learning analytics. The proposed methodology consists of three parts. First an initial analysis of the data is conducted. The extracted features are then analyzed. Last, we assess the performance of the resulting supervised learning model.

4.1 Target Analysis

When analyzing quantitative data, it is important to understand the structures and distributions within the data being analyzed, as this might affect the interpretation of the analysis [27]. Imagine trying to predict if a student is going to fail a course using supervised learning on data collected during the previous semester. If 10% of the students failed the course the previous semester, a supervised learning system would be able to achieve an accuracy of 90% by always predicting that students would pass. Although an overall accuracy of 90% might seem impressive by itself, the system has only learned that most students pass. Only by knowing the underlying distribution of the target data can the system be evaluated. Hence, an initial analysis of the target data is imperative, and should always be included. This will both deepen the readers understanding of the problem and enable the reader to evaluate the system in context of its target data.

4.2 Feature Analysis

The initial analysis provides the required knowledge of the data to start analyzing features (see Section 2.2.1). The goal of a feature is to provide useful information, improving the supervised learning model. A feature will only provide useful information if there exist some correlation between it and the output we want to predict. To test if this correlation exists, we can compute the Spearman's rank correlation coefficient (SRCC) (see Section 2.3.4).

In addition to SRCC, we should also test if the two variables (the feature and the output) are correlated to a significant level [27]. This can be achieved by calculating the p-value (see Section 2.3.5) using a t-test [27]. This p-value will indicate how likely it is that the correlation indicated by SRCC happened by chance. If the likelihood of this is less that 5% ($p < 0.05$), the correlation is said to be significant [27].

Keep in mind that this only provides an indication of how useful a feature is. SRCC are for example unable to detect non-monotonic correlation (see Section 2.3.4).

4.3 Model Analysis

After creating the desired features, they can be used to train a supervised learning model. How to measure the model’s performance depends on whether it is a classification or regression problem. Classification models can use precision, recall (Section 2.3.2) and f1-score (Section 2.3.3). In addition, a confusion matrix could be computed to provide a more in-depth understanding of how the model performs. Regression models can be evaluated by its mean square error (see Section 2.3.1). Both classification and regression models should be evaluated as a whole and for different outputs. Are the models equally good at predicting the different outputs, or is it better at some? These are important questions to answer in order to understand the performance of the model and should be viewed in context of the analysis of the target data.

When choosing what data to use for training and what to use for evaluating the model, randomness is introduced. To reduce this randomness, 10-fold cross-validation (see Section 2.2.1) should be used [22]. More folds are better, but also more computational expensive, and 10-fold cross-validation has proved to be sufficient in most cases [28].

Most supervised learning models have a lot of hyperparameters (parameters used by the model, in addition to the training data, while learning). A common approach is to use a grid- or random-search to discover good parameter settings [29]. However, choosing the best parameters is a difficult task, and an area of active research [29, 30] that will not be explored in this paper.

5 Experimental Setup

NTNU offers an introductory course to object-oriented programming, TDT4100. To test the proposed method, we want to conduct a learning analytics experiment on this course where we perform analysis of the data, feature and classification.

Throughout the course, students are to solve mandatory programming assignments using the Eclipse IDE and the Java programming language. Within each assignment, the students can choose between multiple programming problems of varying difficulty. Each programming problem has a corresponding set of unit tests, allowing the students to test their solution against the requirements of the problem. We want to use data collected through the used IDE to predict student performance on the final exam.

5.1 Data Description

During the spring of 2017, 610 students at NTNU enrolled in TDT4100. During the semester, data was collected for each programming problem within four different assignments (5, 6, 8 and 9). The students used the Eclipse IDE to write, run and test their code, and an Eclipse plug-in was used to capture the students programming process[3]. This plugin was made to trigger on events within the editor, and store data related to the event. In this experiment, we focus on four different events:

* **Edit** – Triggers when files are saved. Stores a snapshot of the current file contents.

[3] https://github.com/hallvard/jexercise/tree/master/no.hal.learning

* **Debug** – Triggers at certain points during a debugging session, e.g. when breakpoints are hit. Stores breakpoints and other debugging data.
* **Test** – Triggers when the tests are executed. Stores the number of succeeded and failed tests.
* **Compiler warning and errors** – Triggers on builds. Stores the position of the warning and the warning category and type.

The plug-in stores the data as XML files containing all events captured by the plug-in, one file for each programming problem and student. This produced a total of 2012 XML data files. In addition, the number of points (0–100) and the grade (A–F) achieved by each student on the final exam was recorded and stored as a CSV file. Both the XML files and each exam result are associated with a randomized anonymous id, unique to each student (this cannot be traced back to identifiers used in the learning management system used at NTNU). This makes it possible to map each programming problem to a specific exam result.

The dataset is fairly small, only 610 students. Predicting a student's performance based on all programming problems solved by that student throughout the course would result in a dataset of 610 samples. To increase this, we want to instead predict the student performance based on a single programming problem. This would allow us to use each programming problem individually, yielding a larger dataset. This means we want to create a supervised learning model using features computed based on a single programming problem as input. The output of the model should be the performance of the student that solved the problem. Limitations to this decision are discussed later in Section 6.5.

5.2 Pre-Processing

The result achieved by each student are solely based on their score on the final exam. This result might vary from day to day based on how the student feel the day of the exam. To reduce this variance, and get a more performance oriented measure, we group students into three different groups, based on their final grade: low, medium and high. Students that got an E or F on the final exam are placed in the low performance group, C and D students are in the medium performance group, and students with A or B are in the high performance group.

6 Experimental Results

6.1 Analysis of Distribution

Most students, around 47.1%, are in the medium performance group, while around 30% of the students reside in the high performance group, and 23% within the low performance group (see Figure 11). This distribution shows that a classifier optimized for accuracy within this distribution should prefer to predict the medium performance group.

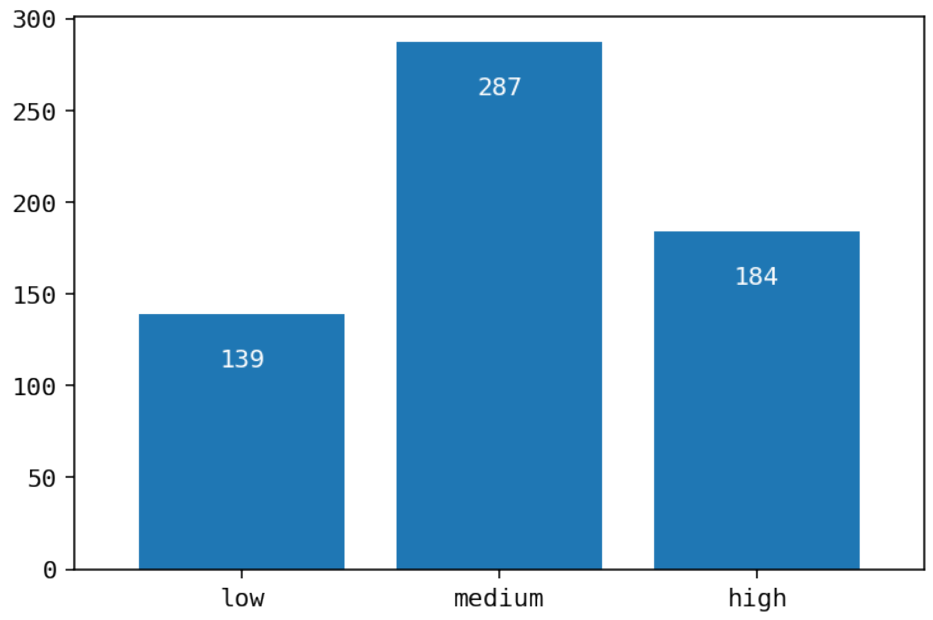


Figure 11: Number of students within each performance group.

However, since each student completes several assignments, containing multiple programming problems, the distribution shown in Figure 11 will not necessarily show the correct distribution for our task of predicting performance using a single programming problem. In fact, the distributions will only be equal if each performance group has the same distribution of solved programming problems per student. The total number of programming problems solved within the assignments is 1905, which yields an average of 3.12 programming problems per student. From Figure 12 we can see that this is not equally distributed within the different performance groups. The high performance group has an average of approximately 4.28 problems per student, the medium performance group has an average of approximately 2.85, and the low performance group has an average of approximately 2.14.

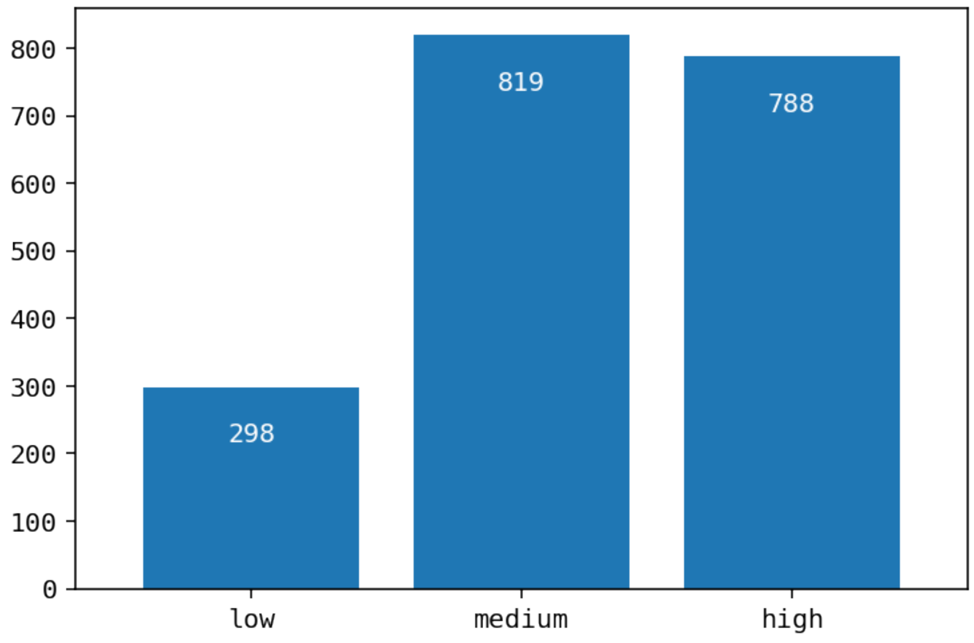


Figure 12: Number of programming problems within the different performance groups.

The students can choose between several different problems of varying difficulty in each assignment, meaning some students might start on one programming problem, but then change their mind and do another. In addition, the students do not need to complete all assignments to satisfy the course requirements, hence some might start on a programming problem, but then decide not to complete it. These discarded programming problems might introduce a lot of noise. If we regard programming problems where only one third or less of the tests has passed as discarded, we can try to remove these to reduce the noise (see Figure 13).

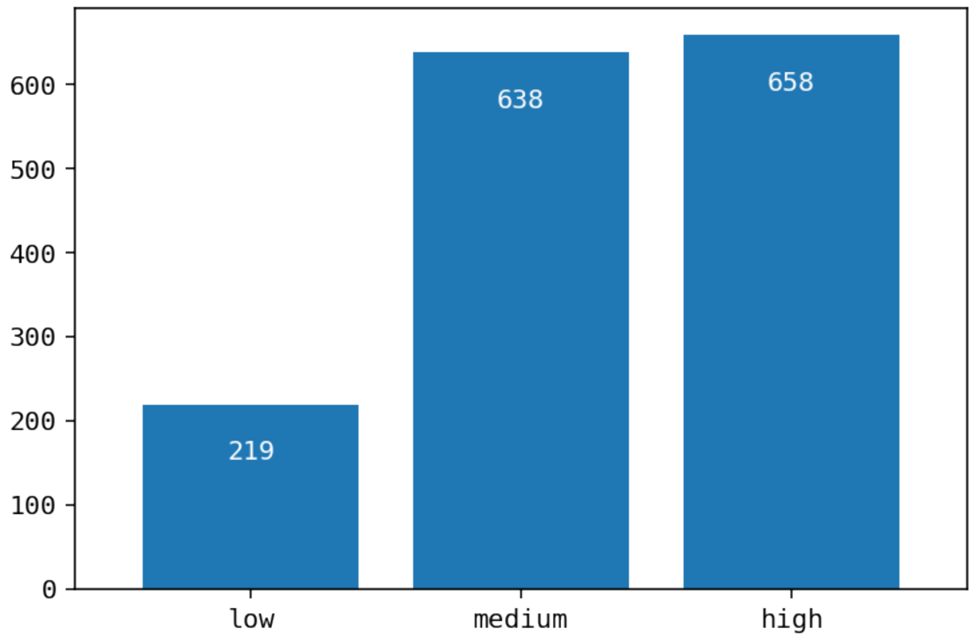


Figure 13: Number of programming problems within each performance group after removing discarded problems.

When removing the discarded programming problems, the total number of programming problems are reduced to 1515, a reduction of around 20%. The number of programming problems within the high performance group is reduced by 16.5%, the medium performance group is reduced by 22.1% and the low performance group by 26.5%. This shows that the high performance group on average completes twice as many programming problems as the low performance group. In addition, a lower performance level yields a higher probability of the student discarding the programming problem.

6.2 Feature Description

Twelve features were extracted from each of the 1515 programming problems. These fall into two categories. Half of the features are **naïve features**, mainly computed by counting, and without the use of domain knowledge. The other half are **complex features**, where both domain knowledge of the course and general programming knowledge are utilized. To extract features, a general feature extraction system was developed[4]. This was used to extract all features mentioned, and the code is available on GitHub[5].

6.2.1 Naïve Features

Debugger Used

This feature checks if the student used the debugger within the IDE while solving the programming problem. The feature value is one if the debugger was used, zero if not.

Completion

The ratio of passed tests. Keep in mind that programming problems with completion less than or equal to one third are removed, thus the value of this feature falls between one third and one.

Debug Runs

The number of times the debugger was used by the student while solving the programming problem. Zero if the debugger was not used, otherwise the number of times it was used.

Compiler Warnings and Errors

The total number of warnings and errors, accumulated on each build. If a warning is persistent between two builds, it will be counted twice. This is intentional, as the value should be higher for students that do not bother resolving the warnings and errors prompted by the IDE.

[4] https://github.com/openwhale/spritz
[5] https://github.com/boyeborg/tdt4100-exercise-feature-extraction

Average Edit Size

The average size of an edit within a programming problem. This measures how much, on average, the student modifies the code between saves.

Final Size

The total size of the student’s code after the last edit.

6.2.2 Complex Features

Total Time

An estimation of the total time spent on solving the programming problem. The students had approximately 14 days to complete each assignment, therefore we have to remove pauses from the programming processes. We estimate this by removing the time between two consecutive edits exceeding 10 minutes. This is not accurate but gives an estimation of the time spent on solving the problem.

Work After Completion

The size of all edits made after all tests has passed. Here we attempt to measure the amount of work done after the problem is completed. If the student does not complete the problem, the value is set to zero.

Edit Center of Mass

Computes the center of mass for all edits within a programming problem weighted by their size, relative to the first and last day of the assignment. This is done by plotting a graph of all edits relative to their timestamp. Each edit is then weighted by their size, and the center of mass of the graph is computed. The result is scaled between zero and one, where a value of 0 indicates the start of the assignment, 1 indicates the end of the assignment deadline and 0.5 indicates half way through the assignment deadline. A feature value of 0.5 indicated that the student has distributed the work evenly over the duration of the assignment, while a number larger than 0.5 indicates the student has done more work after the half way point of the deadline.

Non-Profitable Work Sessions

This measures the number of times the student has worked on the programming problem without it leading to an increase of passed tests. This is measured comparing the number of passing testes of two consecutive test runs where the student has made edits to the codebase in-between.

Complexity

This compares the McCabe cyclomatic complexity [31] of the student’s final solution and the proposed solution by the teaching staff.

Indentation Errors

This counts the number of indentation errors found when passing the student’s source code to the java style checker *checkstyle*[6].

6.3 Feature Analysis

To better understand the relationship between features and the performance groups, Spearman’s ranked correlation coefficient (SRCC) (see Section 2.3.4) and p-values (see Section 2.3.5) are computed for all features (see Table 2). Only two of the naïve features are significant, while four of the complex features are

[6] https://github.com/checkstyle/checkstyle

significant. This results in a total of six features with a significant correlation to the performance groups.

Feature	SRCC	p-value
Debugger Used	-0.02765	0.28220
Completion	**0.07923**	**0.00203**
Debug Runs	-0.02647	0.30326
Warnings	-0.01443	0.57461
Average Edit Size	**-0.05405**	**0.03541**
Final Size	0.00459	0.85817
Total Time	**-0.14543**	**0.00000**
Work After Completion	-0.01923	0.45447
Edit Center of Mass	**-0.26000**	**0.00000**
Non-Profitable Work Sessions	**-0.11573**	**0.00001**
Complexity	-0.02622	0.30774
Indentation Errors	**-0.10174**	**0.00007**

Table 2: Spearman’s ranked correlation coefficient with p-values for all features. Significant features are bold.

Of the significant features, the direction of the correlation is not very surprising. For example, a higher completion indicates higher performance, and more indentation errors indicates lower performance. The two naïve features have both the lowest SRCC values and the highest p-values. This indicates that the naïve features are perhaps too simple to contain valuable information.

One should keep in mind that even though a feature might get a high probability value, it does not necessarily mean that the concept the feature is meant to represent is void. It might simply mean that the way the concept is estimated and measured does not correlate with the performance levels. For example, just because the feature measuring complexity has a high p-value does not mean the concept of code complexity is irrelevant. It just means that the way the complexity is estimated in this paper does not provide a significant correlation.

6.4 Classification

A support vector machine [22] is used as the supervised learning model, and all features are scaled to have a mean of 1, and unit variance. When training the model on all extracted features using 10-fold cross validation (see Section 2.2.2), an accuracy of **53.5%** is achieved (see Table 3).

		Predicted Performance		
		Low	Medium	High
True Performance	Low	25	**124**	70
	Medium	37	**376**	225
	High	12	236	**410**

Table 3: Confusion matrix of the prediction model using all features

The model struggles to identify low performing students, with a recall of only 11.9% and precision of 34.6% for the low performance group (see Table 4). In the original data, less than 15% of the programming problems were solved by students belonging to the low performance group. This might explain why the model struggles with this group.

	Low	Medium	High
Precision	0.346	0.509	**0.581**
Recall	0.119	0.587	**0.621**
F1	0.171	0.543	**0.599**

Table 4: Precision, recall and F1-score of the different performance groups using all features

		Predicted Performance		
		Low	Medium	High
True Performance	Low	10	**143**	66
	Medium	14	**388**	236
	High	8	264	**386**

Table 5: Confusion matrix of the prediction model using only significant features

After removing the insignificant features ($p \geq 0.05$), the model performs slightly worse, with an accuracy of **51.8%** (see Table 5). This shows that insignificant features can provide useful information to a prediction model and should not be discarded just based on correlation coefficient and significant values. It seems feature reduction affected the model's performance on the low performance group the most, resulting in a 56.1% reduction of the F1-score. The F1-score of the medium and high performance groups only decreased by 0.04% and 4.7% respectively (see Table 6).

	Low	Medium	High
Precision	0.273	0.489	**0.560**
Recall	0.045	**0.609**	0.586
F1	0.075	0.541	**0.571**

Table 6: Precision, recall and F1-score of the different performance groups using only significant features

However, when we remove all the significant features, only keeping the insignificant ones, the model has an accuracy of **43.7%.** In comparison, our target data consists of 658 high performance samples, hence a model always predicting the high performance group, regardless of its input, would achieve an accuracy of 43.4%.

To test which features contains the most information, the model can be trained using each feature individually (see Table 7). As shown, the significant features tend to achieve a higher accuracy.

Feature	Accuracy
Edit Center of Mass	0.495
Total Time	0.485
Non-Profitable Work Sessions	0.471
Debugger Used	0.452
Average Edit Size	0.449
Indentation Errors	0.442
Complexity	0.442
Work After Completion	0.438
Debug Runs	0.436
Completion	0.434
Warnings	0.434
Final Size	0.426

Table 7: Model accuracy when only trained on one feature. Significant features are bold.

6.5 Evaluation of Results

There could be several reasons for the achieved accuracy of 53.5%. The task of predicting student performance based on a single programming problem is probably very hard. Capturing student progress and development by only using a single problem is impossible. This could rather be achieved by computing features for each student as opposed to for each programming problem individually. Doing this would however reduce the dataset further. Another approach would be to create features for each assignment. An assignment as a whole can contain more information than each individual programming problem. This could also be extended to create one classifier for each assignment. Each assignment can contain different information, as the students are tested in different parts of the curriculum in the different assignments. In addition, the students' programming skills increase throughout the course, which would be better captured by individual classifiers for each assignment. This would however require a larger dataset.

In addition, the number of collected features are low in comparison to similar research. For example, Vihavainen [19] is able to achieve an accuracy of 78%, but uses almost 200 features. We believe that the accuracy would increase by computing more features and by using a larger dataset.

7 Discussion

The purpose of the experiment is to test if correlation coefficients and p-values can be used to evaluate the utility of a feature. A low p-value and high correlation coefficient should indicate high utility. Similarly, a high p-value and low correlation coefficient should indicate low utility.

When training the model using only the features with high p-values ($p \geq 0.05$), an accuracy of 43.7% was achieved, only 0.3 percentage points higher than a model always predicting the most common performance group (high). On the other hand, when training the model using only significant features ($p < 0.05$), an accuracy of 51.8% was achieved. This suggests significant features contain more information than the insignificant ones, hence their utility is greater.

By training the model on each feature individually (see Table 7), an estimate of each feature's usefulness is derived. The three best performing feature corresponds to the significant features with the highest correlation coefficient, appearing ordered by their correlation coefficients. This indicates that not only does a significant feature provide more information than an insignificant one, but a higher correlation coefficient implies higher utility.

The best classification results are achieved using all the features, both significant and insignificant. This may be because some features might not provide useful information by themselves but might prove useful in combination with others [32]. The p-values and correlation coefficients should therefore only be considered an estimation of a feature's utility, and not an absolute fact. However, as shown, the significant features affect the accuracy of the model more than the insignificant features. Even though the p-values and SRCC are not useful when selecting features in the conducted experiment, others might use these values to decide what features they want to use and compare feature utilities. If all research on IDE-based learning analytics contained this kind of analysis, researchers would be able to compare different ways of measuring concepts and build upon each other's ideas.

The dataset used in the experiment only consists of 1515 programming problems. In context of machine learning, this can be considered a fairly small dataset [22]. To ensure the results presented here are valid, a similar experiment should be conducted on a larger dataset to validate the results. We encourage further research to conduct such an experiment.

In the experiment, only one supervised learning model is used, namely a support vector machine. The experiment should preferably have been conducted using several different models, but due to time limitations we were unable to do this. We will however encourage others to conduct similar experiments using different machine learning models to validate the results presented. For replicability, all code used to produce the presented results are included in the appendix. In addition, the code used to extract each feature is available on GitHub[7]. The dataset used is however not publicly available due to privacy concerns.

[7] https://github.com/boyeborg/tdt4100-exercise-feature-extraction

8 Conclusion

Due to the large variation of learning environments, no single strategy for collecting learning data exists. Because of this, collecting the data might be a difficult task and pose a hindrance for many to apply learning analytics on their learning environment. Hence, good insight into which features might provide useful information is valuable, as this will enable researchers to focus on collecting the data needed to compute useful features. As shown, the proposed analysis provides the information needed to identify and compare the usefulness of different features, enabling researchers to more easily build upon previous work.

With this, we conclude that the proposed feature analysis is valuable and encourage future research to include this kind of analysis.

References

[1] S. Fortmann-Roe, "Understanding the bias-variance tradeoff," 2012.

[2] M. A. Chatti, A. L. Dyckhoff, U. Schroeder, H. Th, and #252, "A reference model for learning analytics," *Int. J. Technol. Enhanc. Learn.,* vol. 4, no. 5/6, pp. 318-331, 2012.

[3] C. D. Hundhausen, D. M. Olivares, and A. S. Carter, "IDE-Based Learning Analytics for Computing Education: A Process Model, Critical Review, and Research Agenda," *ACM Trans. Comput. Educ.,* vol. 17, no. 3, pp. 1-26, 2017.

[4] G. Siemens, "Learning Analytics:The Emergence of a Discipline," *American Behavioral Scientist,* vol. 57, no. 10, pp. 1380-1400, 2013.

[5] T. E. D. Mining, "Enhancing teaching and learning through educational data mining and learning analytics: An issue brief," in *Proceedings of conference on advanced technology for education*, 2012.

[6] L. Johnson, S. Adams Becker, M. Cummins, V. Estrada, A. Freeman, and C. Hall, "NMC Horizon Report: 2016 Higher Education Edition," The New Media Consortium, Austin, Texas2016.

[7] R. B. a. K. Yacef, "The State of Educational Data Mining in 2009: A Review and Future Visions," *JEDM,* vol. 1, no. 1, pp. 3-17, Oct. 2009 2009.

[8] (2018-05-27). *Articles available on Google Sholar from 2010 containing "learning analytics*". Available: https://scholar.google.no/scholar?q=%22learning+analytics%22&as_sdt=0%2C5&as_ylo=2010&as_yhi=2010

[9] T. N. Government. (2016, 2018-04-24). *Koding blir valgfag på 146 skoler*. Available: https://www.regjeringen.no/no/aktuelt/koding-blir-valgfag-pa-146-skoler/id2481962/

[10] R. Mason, G. Cooper, and M. Raadt, *Trends in Introductory Programming Courses in Australian Universities – Languages, Environments and Pedagogy*. 2012.

[11] E. Murphy, T. Crick, and J. H. Davenport, "An Analysis of Introductory University Programming Courses in the UK," *arXiv preprint arXiv:1609.06622,* 2016.

[12] (2018-05-29). *Search results for "learing analytics" on Google Scholar*. Available: https://scholar.google.no/scholar?as_q=&as_epq=learning+analytics

[13] (2018-05-29). *Search results for "learning analytics", "IDE" and "classification" on Google Scholar*. Available: https://scholar.google.no/scholar?as_q=IDE+classification&as_epq=learning+analytics

[14] (2018-08-06). *Search results for "IDE" and "learning analytics" on Google Scholar*. Available: https://scholar.google.no/scholar?as_q=IDE&as_epq=learning+analytics

[15] A. Ahadi, R. Lister, H. Haapala, and A. Vihavainen, "Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance," presented at the Proceedings of the eleventh annual International Conference on International Computing Education Research, Omaha, Nebraska, USA, 2015.

[16] P. Blikstein, M. Worsley, C. Piech, M. Sahami, S. Cooper, and D. Koller, "Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming," *Journal of the Learning Sciences,* vol. 23, no. 4, pp. 561-599, 2014/10/02 2014.

[17] A. S. Carter, C. D. Hundhausen, and O. Adesope, "The Normalized Programming State Model: Predicting Student Performance in Computing Courses Based on Programming Behavior," presented at the Proceedings of the eleventh annual International Conference on International Computing Education Research, Omaha, Nebraska, USA, 2015.

[18] A. S. Carter, C. D. Hundhausen, and O. Adesope, "Blending Measures of Programming and Social Behavior into Predictive Models of Student Achievement in Early Computing Courses," *ACM Trans. Comput. Educ.,* vol. 17, no. 3, pp. 1-20, 2017.

[19] A. Vihavainen, "Predicting Students' Performance in an Introductory Programming Course Using Data from Students' Own Programming Process," in *2013 IEEE 13th International Conference on Advanced Learning Technologies*, 2013, pp. 498-499.

[20] C. Watson, F. W. B. Li, and J. L. Godwin, "Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior," in

2013 IEEE 13th International Conference on Advanced Learning Technologies, 2013, pp. 319–323.

[21] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM,* vol. 55, no. 10, pp. 78–87, 2012.

[22] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[23] L. Rosasco, E. D. Vito, A. Caponnetto, M. Piana, and A. Verri, "Are loss functions all the same?," *Neural Comput.,* vol. 16, no. 5, pp. 1063–1076, 2004.

[24] P. Domingos, "The Role of Occam's Razor in Knowledge Discovery," *Data Mining and Knowledge Discovery,* journal article vol. 3, no. 4, pp. 409–425, December 01 1999.

[25] S. Scott and S. Matwin, "Feature engineering for text classification," in *ICML*, 1999, vol. 99, pp. 379–388.

[26] S. M. Stigler, "Francis Galton's Account of the Invention of Correlation," *Statistical Science,* vol. 4, no. 2, pp. 73–79, 1989.

[27] B. J. Oates, *Researching information systems and computing*. Sage, 2005.

[28] T. Fushiki, "Estimation of prediction error by using K-fold cross-validation," *Statistics and Computing,* journal article vol. 21, no. 2, pp. 137–146, April 01 2011.

[29] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research,* vol. 13, no. Feb, pp. 281–305, 2012.

[30] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," 2013.

[31] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering,* vol. SE-2, no. 4, pp. 308–320, 1976.

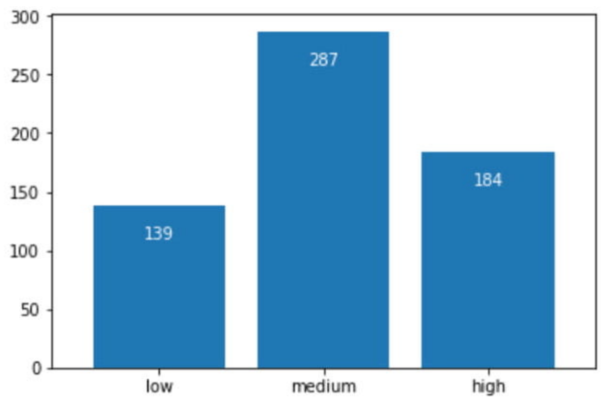
[32] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research,* vol. 3, no. Mar, pp. 1157–1182, 2003.

Appendix

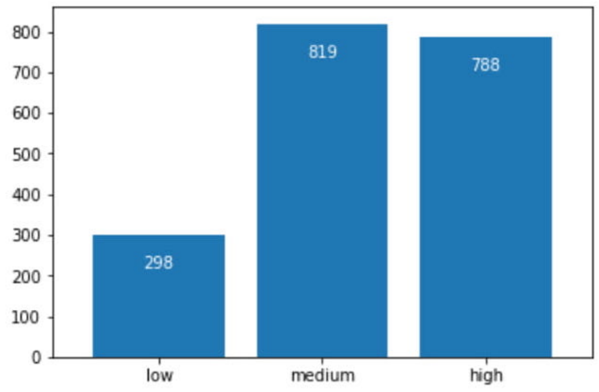
Data Analysis

In [1]:

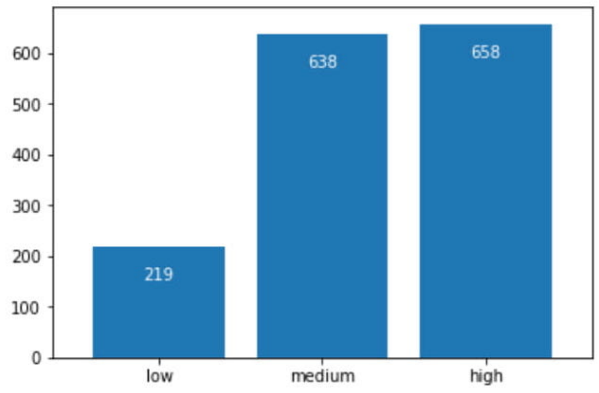
```
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

np.random.seed(42)

grades = pd.read_csv('data/grades.csv')

labels = ['low', 'medium', 'high']

grades['performance'] = pd.cut(grades.grade, 3, labels=labels)

def plot_distribution(data):
    count = data.value_counts(sort=False)
    count.plot(kind='bar', color='C0', width=0.8)
    plt.xticks(rotation=0)

    for i, v in enumerate(count):
        pos = v - max(count) * 0.1
        plt.text(i, pos, '{}'.format(v), color='white', ha='center')

plot_distribution(grades.performance)
plt.show()
```



```
In [2]: assignments = dict()
        available_assignments = [5, 6, 8, 9]

        for ass_num in available_assignments:
            assignment = pd.read_csv('data/data_{}.csv'.format(ass_num))
            assignments[ass_num] = pd.merge(assignment, grades, on='studentId')

        programming_problems = pd.concat(assignments.values())

        plot_distribution(programming_problems.performance)
        plt.show()
```



```
In [3]: discard_filter = programming_problems.completion > 1/3
        plot_distribution(programming_problems[discard_filter].performance)
        plt.show()
```



Feature and Classification Analysis

In [1]:
```
import numpy as np
import pandas as pd

# Set seed for reproducability
np.random.seed(42)

# Mute warnings
np.warnings.filterwarnings("ignore")

# Paths to the data
assignment_paths = [
    'data/data_5.csv',
    'data/data_6.csv',
    'data/data_8.csv',
    'data/data_9.csv'
]
grade_path = 'data/grades.csv'

# Read each assignment
assignments = [pd.read_csv(path) for path in assignment_paths]

# Normalize each assignemnt
from sklearn.preprocessing import scale

for i in range(len(assignments)):
    assignments[i] = assignments[i][assignments[i].completion > 1/3]
    norm_cols = assignments[i].columns.drop('studentId')
    assignments[i][norm_cols] = scale(assignments[i][norm_cols])

# Read the data
assignments = pd.concat(assignments)
grades = pd.read_csv(grade_path)

# Join the data on studentId, and remove studentId
assignments = pd.merge(assignments, grades, on='studentId')
assignments = assignments.set_index('studentId')

# Group the grades in low (0), medium (1) and high (2)
grades = assignments['grade']
assignments['gradeGroup'] = np.ceil(grades / 2.1).astype(int) - 1
```

```
In [2]: from scipy.stats import spearmanr

        spearman_corr = spearmanr(assignments)

        feature_names = assignments.columns[:-3]
        correlation = spearman_corr[0][-1][:-3]
        p_values = spearman_corr[1][-1][:-3]

        print("{:30} {:12} {:7}".format("feature", "correlation", "p-value"))
        print("-"*51)
        for values in zip(feature_names, correlation, p_values):
            print("{:30} {:8.5f} {:11.5f}".format(*values))
            print("-"*51)
```

```
feature                        correlation  p-value
---------------------------------------------------
debuggerUsed                   -0.02765     0.28220
---------------------------------------------------
workAfterCompletion            -0.01923     0.45447
---------------------------------------------------
completion                      0.07923     0.00203
---------------------------------------------------
numDebugRuns                   -0.02647     0.30326
---------------------------------------------------
editCenterOfMass               -0.26000     0.00000
---------------------------------------------------
numWarnings                    -0.01443     0.57461
---------------------------------------------------
totalTime                      -0.14543     0.00000
---------------------------------------------------
numNonprofitableWorkSessions   -0.11573     0.00001
---------------------------------------------------
averageEditSize                -0.05405     0.03541
---------------------------------------------------
complexity                     -0.02622     0.30774
---------------------------------------------------
finalSize                       0.00459     0.85817
---------------------------------------------------
indentationErrors              -0.10174     0.00007
---------------------------------------------------
```

```
In [3]: from sklearn.svm import SVC
        from sklearn.model_selection import KFold
        from sklearn.metrics import accuracy_score, recall_score
        from sklearn.metrics import precision_score, f1_score, confusion_matrix

        def cross_validation(features, classes, clf, num_folds=10):

            y = np.array(classes)
            X = np.array(features)

            num_classes = len(np.unique(classes))

            accuracy = 0
            precision = np.zeros(num_classes)
            recall = np.zeros(num_classes)
            f1 = np.zeros(num_classes)
            confusion = np.zeros((num_classes, num_classes))

            num_folds = 10
            kf = KFold(n_splits=num_folds, shuffle=True)

            for train_index, test_index in kf.split(X):
                X_train, X_test = X[train_index], X[test_index]
                y_train, y_test = y[train_index], y[test_index]

                clf.fit(X_train, y_train)
                predictions = clf.predict(X_test)

                accuracy += accuracy_score(y_test, predictions)
                precision += precision_score(y_test, predictions, average=None)
                recall += recall_score(y_test, predictions, average=None)
                f1 += f1_score(y_test, predictions, average=None)
                confusion += confusion_matrix(y_test, predictions)

            print("Average accuracy: {:.3f}".format(accuracy/num_folds))
            print()
            format_str = "{:10} {:<10.3f} {:<10.3f} {:<10.3f}"
            print("{:10} {:10} {:10} {:10}".format("", "low", "medium", "high"))
            print(format_str.format("precision", *precision/num_folds))
            print(format_str.format("recall", *recall/num_folds))
            print(format_str.format("f1", *f1/num_folds))
            print()
            format_str = "{:10} {:<10.0f} {:<10.0f} {:<10.0f}"
            print("{:10} {:10} {:10} {:10}".format("", "low", "meium", "high"))
            print(format_str.format("low", *confusion[0]))
            print(format_str.format("medium", *confusion[1]))
            print(format_str.format("high", *confusion[2]))
```

```
In [4]: features = assignments.drop([
            'grade',
            'gradePercent',
            'gradeGroup'
        ], axis=1)

        classes = assignments['gradeGroup']

        cross_validation(features, classes, SVC(C=15, gamma=0.05))
```

```
Average accuracy: 0.535

           low        medium     high
precision  0.346      0.509      0.581
recall     0.119      0.587      0.621
f1         0.171      0.543      0.599

           low        meium      high
low        25         124        70
medium     37         376        225
high       12         236        410
```

```
In [5]: features = assignments.drop([
            'grade',
            'gradePercent',
            'gradeGroup',
            'debuggerUsed',
            'numDebugRuns',
            'numWarnings',
            'finalSize',
            'workAfterCompletion',
            'complexity'
        ], axis=1)

        cross_validation(features, classes, SVC(C=15, gamma=0.05))
```

```
Average accuracy: 0.518

           low        medium     high
precision  0.273      0.489      0.560
recall     0.045      0.609      0.586
f1         0.075      0.541      0.571

           low        meium      high
low        10         143        66
medium     14         388        236
high       8          264        386
```

```
In [6]: features = assignments.drop(assignments.columns.drop([
            'debuggerUsed',
            'numDebugRuns',
            'numWarnings',
            'finalSize',
            'workAfterCompletion',
            'complexity'
        ]), axis=1)

        cross_validation(features, classes, SVC(C=15, gamma=0.05))
```

```
Average accuracy: 0.437

           low        medium     high
precision  0.000      0.447      0.439
recall     0.000      0.303      0.718
f1         0.000      0.349      0.540

           low        meium      high
low        0          51         168
medium     4          192        442
high       1          187        470
```

```
In [7]: import operator
        from sklearn.model_selection import cross_val_score

        def mean_accuracy(clf, features, classes, cv=10):
            features = assignments.drop(
                assignments.columns.drop(features),
                axis=1
            )
            return cross_val_score(
                clf, features, classes, cv=cv
            ).mean()

        feature_accuracy = dict()

        for feature in assignments.columns.drop(
            ['grade', 'gradePercent', 'gradeGroup']
        ):
            clf = SVC(C=15, gamma=0.05)
            accuracy = mean_accuracy(clf, [feature], classes)
            feature_accuracy[feature] = accuracy

        sorted_accurcies = sorted(feature_accuracy.items(),
                                  key=operator.itemgetter(1),
                                  reverse=True)

        for acc in sorted_accurcies:
            print("{}: {:.3f}".format(*acc))
```

```
editCenterOfMass: 0.495
totalTime: 0.485
numNonprofitableWorkSessions: 0.471
debuggerUsed: 0.452
averageEditSize: 0.449
indentationErrors: 0.442
complexity: 0.442
workAfterCompletion: 0.438
numDebugRuns: 0.436
completion: 0.434
numWarnings: 0.434
finalSize: 0.426
```