

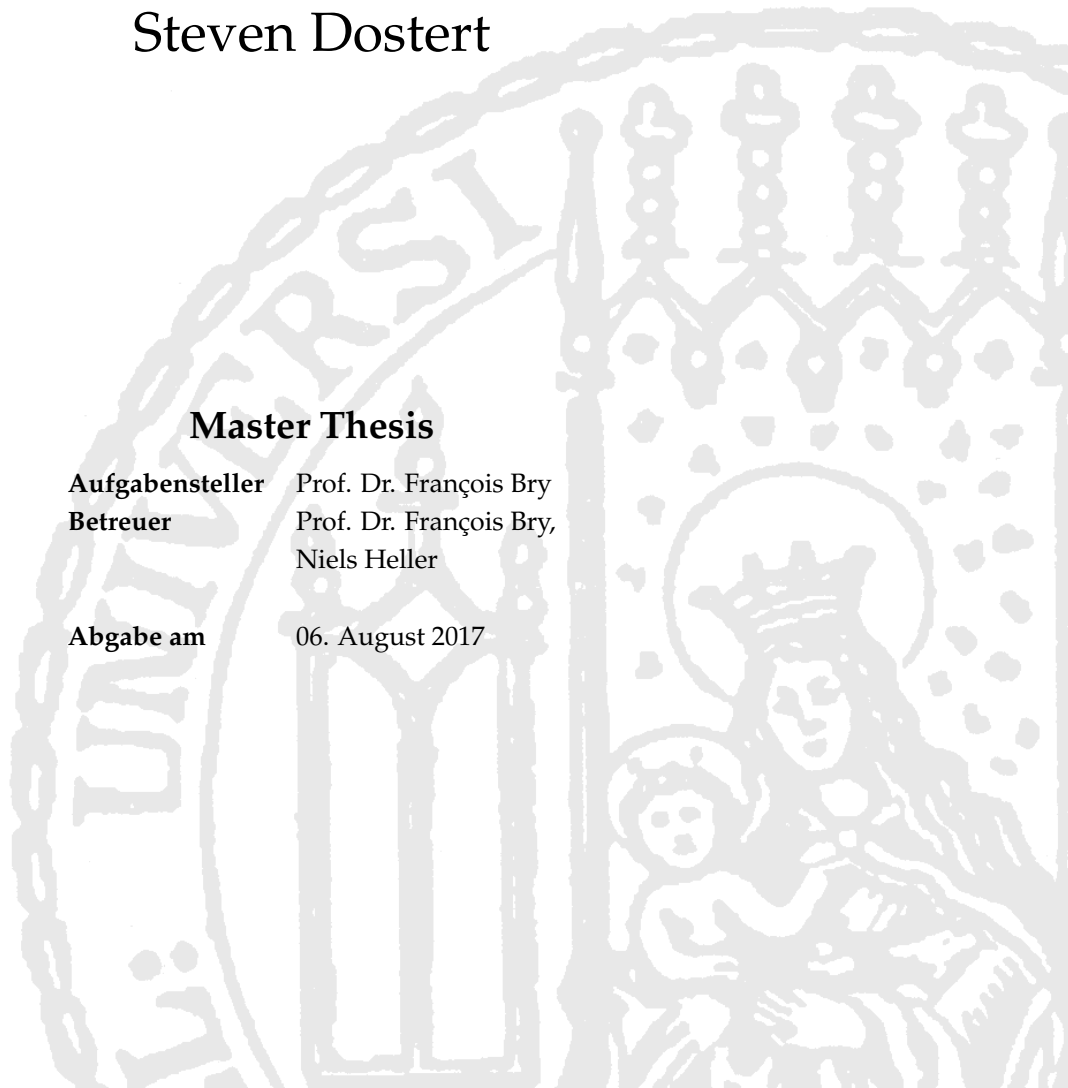
INSTITUT FÜR INFORMATIK
der Ludwig-Maximilians-Universität München

PREDICTING THE LEARNING
BEHAVIOUR OF STUDENTS
FROM THEIR WEEKLY WORK
ASSIGNMENTS

Steven Dostert

Master Thesis

Aufgabensteller	Prof. Dr. François Bry
Betreuer	Prof. Dr. François Bry, Niels Heller
Abgabe am	06. August 2017



Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, den 06. August 2017

Steven Dostert

Abstract

Although weekly assignments provide a good opportunity for Students to test their abilities and to deepen their knowledge, students tend to skip assignments within a course. This harms the learning of students and makes it difficult for professors and tutors to teach students in the best possible way and thus to prepare them optimally for their future professional life.

There are multiple reasons why students skip assignments. Two major groups of such reasons are discussed in this thesis: On the one hand students' personal characteristics, on the other hand the students' behaviour (work out or skip an assignment) with previous assignments. In a past course prior to this work, students' submissions have been classified based on the the students' behaviour or on their performances. In this thesis, the classification is described and a statistical model is presented that fits to it. This model is used to predict students' future behaviour (work out or skip an assignment) based on parts of the classified data. Then, these predictions are validated with other data (not used in building the predictor). The predictions could be used to make the students more conscious of the risks associated with skipping assignments and thus to incite them not to skip assignments.

The statistical model used for building the afore-mentioned predictor is a Hidden Markov Model. It fits well the two previously mentioned major groups of behaviours since it considers the students' recent behaviours when predicting and it includes personal characteristics (expressed by the model's hidden states). This thesis provides an evaluation of the predictor which shows that the Hidden Markov Model provides good results for prediction of the skipping behaviour.

Moreover, a large part of this thesis is concerned with the implementation of a modular framework making it possible to cover similar use cases. To this aim, the implementation has been designed to be completely independent from the type of data gathered (like assignments' classifications), the statistical model applied and the predicted behaviour. By implementing the described use case using this framework and by integrating it into an existing system, the usage and its modular expandability is demonstrated.

Zusammenfassung

Obwohl wöchentliche Übungsaufgaben Studenten eine gute Möglichkeit bieten, ihre Fähigkeiten zu testen und ihr Wissen zu vertiefen, neigen sie häufig dazu die Aufgaben eines Kurses nicht mehr abzugeben. Dies schadet den Studenten beim Lernen und erschwert den Professoren und Tutoren die Studenten bestmöglich auszubilden und sie somit optimal auf ihr späteres Berufsleben vorzubereiten.

Es gibt mehrere Gründe, weshalb Studenten Übungsaufgaben nicht mehr abgeben. Zwei Hauptgruppen für solche Gründe werden in dieser Thesis betrachtet: Zum einen persönliche Eigenschaften der Studenten zum anderen das Verhalten (bearbeitet Übungsaufgabe, oder gibt diese nicht ab) bei vorherigen Aufgaben. In einem vergangenen Kurs im Vorfeld dieser Arbeit wurden die Abgaben von Studenten basierend auf ihrem Verhalten beziehungsweise ihrer Leistungen klassifiziert. In dieser Arbeit wird diese Klassifizierung erläutert und ein statistisches Modell vorgestellt, welches zu dieser Klassifizierung passt. Das Modell wird dazu genutzt das Verhalten (bearbeitet Übungsaufgabe, oder gibt diese nicht ab) der Studenten basierend auf Teilen der klassifizierten Daten vorherzusagen. Diese Vorhersagen werden dann anhand der übrigen (nicht zum Erstellen des Prädiktor genutzten) Daten validiert. Die Vorhersagen könnten dazu genutzt werden, dem Studenten das Risiko des "nicht Abgebens" bewusst zu machen und ihn somit dazu anzutreiben, die Aufgaben zukünftig abzugeben.

Das statistische Modell, welches gewählt wurde, um den zuvor erwähnten Prädiktor zu erstellen, ist das Hidden Markov Model. Es passt gut zu den zwei zuvor genannten Verhaltens-Hauptgruppen, da es das vorherige Verhalten der Studenten berücksichtigt und deren persönliche Eigenschaften abbildet (ausgedrückt durch die versteckten Zustände des Modells). Diese Arbeit liefert eine Evaluation des Prädiktors, bei der gezeigt wird, dass das Hidden Markov Model gute Resultate für die Vorhersage des "nicht Abgebens" liefert.

Außerdem beschäftigt sich ein großer Teil dieser Arbeit mit der Implementierung eines modularen Frameworks, welches es ermöglicht, ähnliche Anwendungsfälle abzubilden. Aus diesem Grund ist das Framework so konstruiert, dass es komplett unabhängig von der Art der gesammelten Daten (z.B. Aufgabenklassifizierung), des angewandten statistischen Modells und dem vorherzusagenden Verhalten ist. Die Nutzung sowie die modulare Erweiterbarkeit des Frameworks wird durch die Umsetzung des beschriebenen An-

wendungsfalls und durch die Integration in ein bestehendes System demonstriert.

Acknowledgments

First and foremost, I would like to thank Prof. Dr. François Bry for the opportunity to write this master thesis at the Teaching and Research Unit Programming and Modelling Languages. Furthermore for the remarks, suggestions and the feedback he provided on the way to this thesis.

I want to express my deep gratitude to my advisor Niels Heller for suggesting and introducing me to this topic and for providing the data set used in this thesis. I am thankful for his constant support, his encouragement and his guidance during the development of my work. He helped me whenever I needed support and contributed a lot of useful ideas.

Also, I would like to thank Melina Kellner for her moral support and for proof-reading the thesis. Last, but not least, special thanks to my family for providing general support throughout my years of study.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Approach	3
2	Related Work	7
3	Classification	11
3.1	Data Collection	11
3.2	Training Data Selection	13
4	Statistical Models	17
4.1	Relative Frequency	17
4.2	Markov Models	18
4.2.1	Markov Chain	18
4.2.2	Hidden Markov Model	20
5	Technical Implementation	25
5.1	Prediction Framework	26
5.1.1	Observation and Prediction Tags	26
5.1.2	Prediction Unit	28
5.1.3	Behaviour Log	33
5.1.4	Prediction Unit Factory	33
5.2	Integration	35
5.2.1	Server Start Up	36
5.2.2	History and Behaviour Log Integration	36
5.2.3	Client-Server Communication	38
5.3	Framework Extension	40
5.4	Summary	41

6	User Interface	43
6.1	Students' Behaviour Table	43
6.2	View Control	44
6.3	Behaviour Log Control	45
6.4	Behaviour Log Selection	45
7	Evaluation	47
7.1	Method	47
7.1.1	Metrics for Measuring Prediction Performance	48
7.1.2	Cross-Validation	50
7.1.3	Evaluation Aspects	51
7.2	Results	51
7.2.1	Tag Independent Evaluation	51
7.2.2	Tag Dependent Evaluation	52
7.2.3	Classification Evaluation	53
7.3	Discussion	54
8	Conclusion	57
	Bibliography	61

1.1 Motivation

A university course often consists of weekly lectures and weekly tutorials in which the lectures' contents are clarified as well as techniques and methods learned in the lectures are applied. To be prepared for the next tutorial the students often receive homework assignments which should be completed. These assignments should be submitted by the students before the expiration of a deadline and may be corrected by tutors.

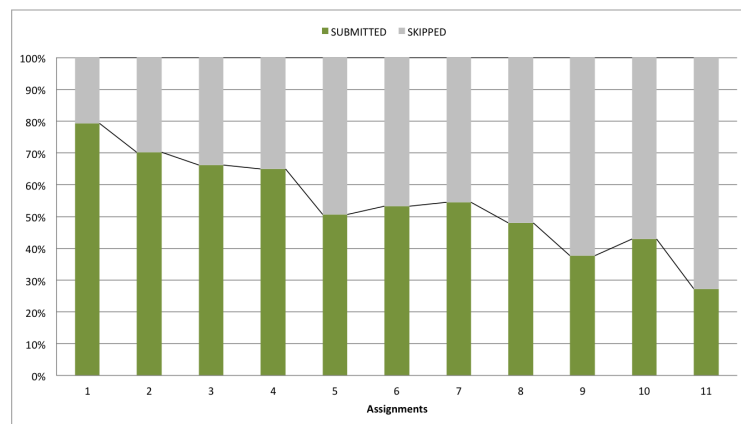


Figure 1.1: Skipping/Submitting behaviour distribution of 82 randomly selected students in an introductory course in Theoretical Computer Science (TCS). The data was provided by research assistant Niels Heller from the Ludwig-Maximilian University of Munich.

During a course many students tend to skip or unduly delay their homework assignments. Figure 1.1 shows the relative frequency distribution of the students' submission behaviour of an finished introductory course in Theoretical Computer Science. The data was collected during the summer semester in 2016 at the Ludwig-Maximilian University of Munich. In the 12 week long course 11 optional assignments were handed out. By providing a near-faultless submission the students were rewarded with so-called "bonus points" boosting their final examination's marks. Additionally they received comments on their submissions pointing to the errors they made. While the amount of students who submit an assignment is relatively high (about 80%) at the beginning of the course, it decreased over time almost linearly to 25%.

Skipping an assignment doesn't necessarily mean that a student stops attending the course. It is also possible that he just stopped submitting his solutions when he realized that the feedback of the tutors does not help him in his learning. This on the other hand is very unlikely if the submissions yield bonus points. However the skipping could also imply that the student stopped learning for this course during the current semester. Which in turn leads to last minute learning in the last days before an exam. This so called "cramming" might be thought by many students to be a possible way to pass an examination. Selyukh (2007) and McIntyre and Munson (2008) have observed that, at some universities, students can pass examinations through cramming and "still achieve good grades". However, these authors also note that cramming has a negative effect on future retention: "Generally, cramming puts information in the short-term storage area of the brain, whereas long-term studying places it in the long-term memory." To avoid students' cramming it is important to keep them engaged, since universities (or professors) have the goal to teach students in the best possible way for their later professional life.

Besides the fact that it is also a waste of time for the students and the tutors to start with a course just to stop learning on it after a few weeks, there is another reason why it is important to prevent the students from skipping the assignments. An assignment is likely to a test for the student even if it is optional and not graded. As described by Roediger and Karpicke (2006, p. 249 ff.), "If students are tested on material and successfully recall or recognize it, they will remember it better in the future than if they had not been tested". This means that we can archive a "powerful positive effect on future retention" by testing students. It bears mentioning that Roediger and Karpicke have shown that this positive effect applies "even without feedback". Anyway if the submissions are corrected by tutors, a convenient feedback is given to the students on their submissions' performances which they can use to assess themselves.

But what could be potential explanations for the students' skipping behaviour? Personal characteristics like loss of motivation or time constraints are possible reasons. It is easily conceivable that the students' motivation shrinks with bad results in earlier assignments. Additionally the chance to gain the bonus points become smaller which reinforces this negative characteristic. A bad result in an earlier assignment may also indicates a gap in knowledge. This knowledge gap can only be closed by relearning some course material (or even specific fundamentals) which is very time-consuming and leads to procrastination. The relearning is especially necessary for courses where understanding later chapters

depends on having understood earlier chapters. Through this additional effort and delay the student may tend to skip.

All these explanations indicate a dependency between the behaviour on earlier assignments and the future behaviour. Therefore it is necessary to get the reason for the previous behaviour. In practice it is almost impossible to detect all influences that cause students to skip their assignments as it would require the teachers (e.g. tutors) to gather personal information from the students. But for detecting growing knowledge gaps gathering the important information is possible for tutors without much extra work by analysing the students' submission. Since each submission will be assessed, either by a tutor or automatically, the assignments can be tagged based on the mistakes the student made. So the mentioned recent behaviour of a student can, in part, be represented by a series of tags where each tag represents the behaviour in a submission. A possible behaviour classification per submission could be: "no error", "error", "insufficient knowledge" and "skipped". This categorisation is focusing the skipping behaviour of a student. All these states are reasonable, because they are well-known in the teaching routine. "No error" and "error" is an acceptable behaviour since they imply that the student has dealt with the subject, whereas "insufficient knowledge" as well as "skipped" indicate a bad behaviour which should be prevented. "Insufficient knowledge" can also be seen as gap in knowledge. It has to be said that a categorisation which is given by human beings is always a subjective assessment. A more detailed look at this states is shown in chapter 3.

Based on this it is possible to assume that the behaviour on previous assignments can be used to predict the students' future behaviour. To be more precise the tagged data of the past can be used to predict the next tag the student will receive. This prediction can be delivered to students as part of a weekly feedback that would either keep their motivation high or incite them to provide the additional weekly work necessarily for getting better success predictions in the future. If the student stays focused, the decrease of submits (figure 1.1) could be prevented.

	Assignment 1	Assignment 2	...	Assignment x
Student 1	error	skipped	...	skipped
Student 2	no error	error	...	insKnowledge
...
Student x	no error	no error	...	error

Figure 1.2: Example representation of tag histories.

1.2 Approach

The tagged submission of a student is called the *tag sequence*. In the further work all tag sequences of one closed course are named *tag history*. It can be represented as a table where

the rows are the different students and the columns are the assignments of the course. An entry is the behaviour of a student in an assignment as visualized in figure 1.2. Such a tag history can be represented by a statistical model. To predict the students' future behaviour this statistical model is used to make predictions based on the students' recent behaviour. Therefore submissions of all students of already closed courses need to be tagged, so that they act as training data for this model.

While rows in the tag history are completed behaviour sequences of students, the *personal recent behaviour sequence* describes the tagged submissions of one student during a running course. Figure 1.3 shows such a sequence as a graphical representation. Based on a trained statistical model and the students' *personal recent behaviour sequence* the probabilities for the different states in the next step can be calculated.

Running Course	Assignment 1 06.11.2016	Assignment 2 03.12.2016	Assignment 3 08.01.2017	Assignment 4 10.02.2017	Assignment 5 06.03.2017
Student	insufficient Knowledge	error	skipped	error	?

Figure 1.3: Example representation of a *personal recent behaviour sequence* until assignment 4. Assignment 5 is not available yet and should be predicted.

Besides predicting the skipping behaviour of a student, other use cases can be imagined, where a prediction should be made. So could, for example, the mark the student will receive in the exam be predicted based on the submission behaviour. Or, the day of the students' submission might be interesting for the tutors, so that they schedule their assessment. In these use cases the classifications are different. Therefore a *prediction framework* has been implemented which is extendable for further use cases. Within this framework *prediction units* take care of the statistical model that makes the predictions. To be independent of the statistical model and the type of tags the prediction unit is designed modular. To predict the future behaviour during a running course the framework has been integrated into an existing application. This application is a current research project of the Teaching and Research Unit Programming and Modelling Languages at the Ludwig-Maximilian University of Munich — called *Backstage 2*. It is a redevelopment and extension of the previous platform *Backstage*¹. Additionally an user interface has been realised that can be used by tutors.

Once a prediction for a student is made, it has to be interpreted by tutors, who should decide how this information is used. It may then be used to inform the student about his possible behaviour to affect him on an early stage. This process is called *feedforward*. It is similar to the more familiar process *feedback* in which the past behaviour is analysed to improve the future (Basso and Olivetti Belardinelli, 2006, p. 73 ff.). Feedforward instead tries to have an influence on the receiver by showing him the possible future behaviour ("learning from the future" Dowrick (2012, p. 215)). This works through the strictly interconnection between feedforward and a kind of self-regulation — the so-called *homeostasis*. Homeostasis is "the ability of a system to regulate its internal environment in order to main-

¹<http://www.en.pms.ifi.lmu.de/research/backchannels/index.html> and <http://backstage.pms.ifi.lmu.de/>

tain a steady state. When a deviation from a stable state occurs, an automatic adjustment process starts with the aim of restoring the initial equilibrium.” (Basso and Olivetti Belardinelli, 2006, p. 75). Basso and Olivetti Belardinelli say feedforward “is a process adjusting behaviour in a continuative way”. For example, feedforward could incentivise students to overcome a tendency to skip assignments. Here the predicted skipping behaviour acts as “perturbation” which “must be eliminated in order to achieve a desired goal” (Basso and Olivetti Belardinelli, 2006). Also a positive stimulus could be sent to the student to keep him motivated if the prediction turns out positive. This method may be even more important, since most of the students are motivated at the beginning of a course as described in section 1.1. Using this start motivation it could solve the problem of decreasing submits. For this purpose a prediction should be made once an assignment has been tagged.

At first other papers, which bother related topics, are treated in this work (chapter 2). How the classification was done and how the different tags are assessed is shown in chapter 3. Furthermore this chapter describes why the training data selection is important for predicting the students’ behaviour. In chapter 4 different statistical models are discussed and described which are or might be used to predict the students behaviour. The main focus of this chapter relies on the Hidden Markov Model that was used for the predictions. chapter 5 explains the core of this work, the technical implementation of an prediction framework. It describes on the one hand the design of this framework and on the other hand the integration in an existing application. How tutors or teachers can enter the assessed students’ current behaviour in this application to predict the students’ future behaviour is shown in chapter 6 by explaining the implemented user interface. The predictions of two statistical models are evaluated based on different evaluation aspects in chapter 7. The work is summarised and a conclusion is given in chapter 8. Furthermore this chapter gives an prospect how the predictions could be used in the future to affect the students’ behaviour.

CHAPTER 2

Related Work

Predicting students' behaviour is an up-to-date topic in the educational environment which involves different aspects and approaches. There are a lot of studies focussing on a multitude of aspects regarding this topic. Since this work considers skipping, which is a kind of dropout, studies with such a focus are analysed.

Predicting the students' dropout does not only concern traditional universities, also the e-learning environment is affected since these so called Massive Open Online Courses (MOOCs) became more and more popular. MOOCs have one advantage over university courses regarding collecting data, but also the disadvantage to keep the students motivated as said by Halawa et al. (2014, p. 7): "While MOOCs offer educational data on a new scale, many educators have been alarmed by their high dropout rates". Also Balakrishnan (2013, chapter 1) observes a "significant problem, since virtually anyone can register for the course and the consequences for failing a course are minimal. This results in a large number of students enrolling in the course without ever participating once it begins, as well as students continuing to drop out at virtually every point during the course". The benefit of MOOCs is that the operators can easily gain data since each activity is done online by which they can log the students behaviour at any time. For example they can track activities about when and how often students watch a lecture or work on an assignment.

Due to this benefit — and also as consequence of the disadvantage — there are two groups of people which have a special interest in the topic e-learning. On the one hand the operators of these platforms since they want to keep their users active, on the other hand researchers since they benefit from the amount of data. Therefore studies in this field often rest on MOOCs instead of traditional universities (e.g Balakrishnan (2013), Halawa et al. (2014) and Qiu et al. (2016)).

There are two sorts of dropout predictions which base on different time periods. On one side dropout predictions focusing the complete duration of study, on the other predictions of a specific course which means that only one semester is involved. Aulck et al. (2016) and

Obsivac et al. (2012) try to predict if a student will finish one's degree. Ameri et al. (2016) try the same and additionally predict when the dropout will happen. In contrast Ahadi et al. (2015) and Yadav et al. (2012) are concerning with students' exam performance in a running semester.

Depending on the desired predictions that should be made and the time frame that is analysed, the data which is collected for classification is different. The predictions can be based on multiple features. A feature describes an observable activity or characteristic of students. Three main data classification groups can be extracted:

Students' behaviour and performance This group includes observable activities of a student (like attendance or the required time for a class test), or an assessed version of these activities (exam marks, submission performance, etc.). Yadav et al. (2012) use "students' past performance data [...] to predict the students' performance". The past performance data they use includes among others previous semester marks, class test grade and attendance. Ahadi et al. (2015) analysed an "introductory programming course organized at the University of Helsinki" (Ahadi et al., 2015, section 3.2). To assess the students' programming performance they automatically gathered information using a special programming environment: "For each student that consented to having their programming process recorded, every key-press and related information such as time and assignment details was stored" (Ahadi et al., 2015, section 3.2).

Personal characteristics The group personal characteristics contain information given to a student by nature (e.g. gender, race, age) or influences which may affect the students in the given situation (like number of courses attending during a semester or foreknowledge from previous education). They can either be gathered by personal interviews or are already given by an existing system. Besides the programming performance Ahadi et al. (2015) have also gathered personal characteristics of students like gender, age and their study programme. Personal characteristics are used for predicting the students' behaviour by further studies. Aulck et al. (2016) for example collect additional data like race or previous schooling.

Social characteristics and activities A further group is social characteristics and activities like circle of friends or social (media) activities. These activities have been collected by Obsivac et al. (2012). Besides the two previously described groups they have analysed the students' social behaviour including "explicitly expressed friendship", "mutual e-mail conversation" and "visited personal pages".

The treated studies often compare different machine learning algorithms to receive the best possible predictions regarding the considered use case. For that reason they often use existing machine learning software (e.g. *Weka Knowledge Explorer*¹) which offers multiple algorithms.

¹Weka: Data Mining Software in Java. Available at <http://www.cs.waikato.ac.nz/~ml/weka/>

For example Yadav et al. (2012) compare different decision tree algorithms like ID3, C4.5 and CART using Weka. By doing this they received an accuracy of about 50 – 60% for the three algorithms when predicting the students' end semester mark based on the past performance.

Ahadi et al. (2015) also examined multiple decision tree algorithms (e.g. J48, Random Forest) as well as rule learners (e.g. PART) and Bayesian classifiers (Naive Bayes, Bayesian Network). As mentioned before, they used different features out of the classification groups students' performance and personal characteristics to predict the final grade. Ahadi et al. received the best accuracy using the Random Forest algorithm with more than 85% on a given test set. By evaluating a separate semester they only received an accuracy of 71% – 80%.

Obsivac et al. (2012) tested similar machine learning algorithm types including but not limited to decision tree (J48), lazy learner (IB1), rule learner (PART) and Bayesian classifier (Naive Bayes). They have evaluated the influence of their social behaviour classification. By doing this Obsivac et al. have shown that the social behaviour classification can increase their previous dropout predictions accuracy for all algorithms. The best results have been gained using PART (without social behaviour features: 82%; including: 93%).

Compared to this work the treated studies focusing on selecting the best performing features as well as the best algorithm which match their given use case. In this work the classification scheme has already been defined and a suitable statistical model is searched. The focus rather relies on implementing an adaptable and extendable framework which can be applied on different use cases.

There are different kinds of tagging to predict the future behaviour of a student. The tagged data is separated into two groups. On the one hand the *observations* which are the tags of the observed behaviour of a student and on the other hand the *predictions* which describe a future behaviour. *Predictions* can either be in the same set of tags as the *observations* (*consistent set*) or a completely *different set*. To predict the students' skipping behaviour with respect to work assignments a consistent set of error tags is used in this thesis. A possible example for different sets is the prediction of the examination mark based on the behaviour in work assignments. In section 3.1 the classification of data to predict the behaviour regarding work assignments is explained more precisely. Reasons why the selection of the correct training data is important and depends on different criteria is explained in section 3.2. The data used in this chapter were provided by research assistant Niels Heller from the Ludwig-Maximilian University of Munich. Yet there is no published version where the data is presented but there will be one in the future.

3.1 Data Collection

An introductory course in Theoretical Computer Science at the Ludwig-Maximilian University of Munich was used to collect data of the students' behaviour. In this course there were eleven optional assignments. Each of these assignments had a different number of exercises. While the course started with 345 participants, 82 were randomly selected. For the submissions of these 82 students a special record was made. Each exercise was tagged with comments and assigned to the student who received these comments. Comments which were assigned repeatedly were for example "good" and "missing" for the obvious reasons. Other typical comments were common misconceptions of an exercise, with which missing knowledge could be connected. All comments were summarized per assignment and linked to the student. Thereby each submission of the 82 students could be tagged by

one of the following tags:

Skipped This tag is assigned if the student had not even submitted the assignment or if a bulk of exercises received the comment “missing”.

No error Obviously a submission is tagged as “no error” if all exercises were handed in without any mistakes.

Insufficient knowledge If a students’ mistake of an exercise is judged as “insufficient knowledge” by the correcting person, the assignment is marked with this tag. It is related to the previously mentioned knowledge gaps.

Error By process of elimination a submission is said to exhibit an “error” if none of the other tags can be applied.

To detect “insufficient knowledge” the exercises should be prepared carefully in the first place. This means that typical mistakes should be gathered (e.g. from earlier courses) for each exercise. By doing this, the corrector knows what he has to look for.

For example a programming assignment could come with a *unit test* where some results automatically reveal insufficient knowledge. To be more precise assuming the following simple programming assignment: “Write a function which calculates the sum of 6 and 4, divided by 2!”. Evidently the correct answer would be 5. Whereas the result 8 acts as an indicator for insufficient knowledge, since it perhaps implies a conceptional misunderstanding of parentheses (listing 3.1). It has to be said that it is only an assumption that the reason for such a result is insufficient knowledge and not only a simple mistake. So could a forgotten *return* either mean a careless mistake or insufficient knowledge about functions. Anyway if the unit test detects one of these predefined wrong answers, the exercise could be marked as “insufficient knowledge” instantly. Of course the correct answer could be verified by a unit test and automatically tag the exercise as “no error” as well. In this way the assessment of the submissions can be automated or at least accelerated.

```

1 | # correct
2 | def divide():
3 |     return (6 + 4) / 2 # output: 5
4 |
5 | # incorrect
6 | def divide():
7 |     return 6 + 4 / 2    # output: 8
8 |
9 | # incorrect
10 | def divide():
11 |     6 + 4 / 2    # return missing

```

Listing 3.1: Correct and incorrect divide function implementations in python

While “skipped” is of special interest since we want to prevent the student from skipping the assignments, “insufficient knowledge” is very important as well, because the regarding learner would have to go over the course material once more while the course itself

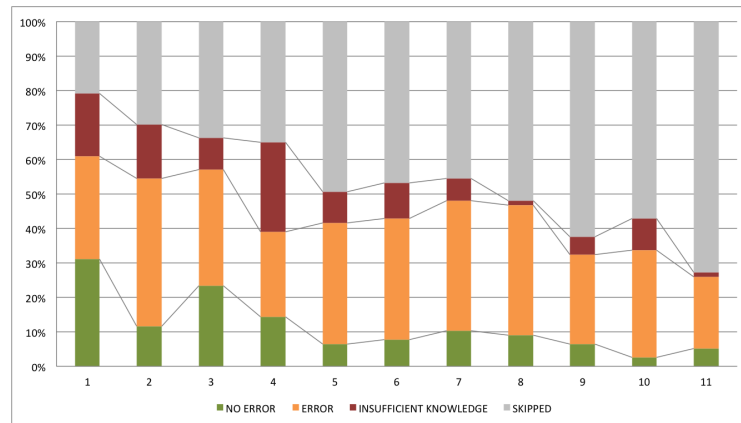


Figure 3.1: Tagged data distribution of an introductory course in Theoretical Computer Science

continues. The category “insufficient knowledge” may be related to Radatz (1979) “Errors Due to Deficient Mastery of Prerequisite Skills, Facts, and Concepts”. Which in turn refers to the earlier mentioned knowledge gaps.

In the further work the 82 tagged submissions of the course in Theoretical Computer Science act as training- and also as test-set. How this is done is shown in chapter 7. The relative frequency distribution of the courses’ tagged data is shown in figure 3.1.

3.2 Training Data Selection

As seen in figure 1.1, more students tend to skip assignments as time goes on. Statistics of other courses confirm this trend. Figure 3.2, 3.3 and 3.4 show the submission behaviour of three additional courses in different semesters. The growth of skipping is visible in all of these statistics.

Compared to figure 1.1 there are fewer submissions right from the beginning of the course. This could have different reasons. On one side there are different kinds of how the data was collected. While the data of the introductory course in Theoretical Computer Science of chapter 1 was collected by randomly selecting some students, the data of the three statistics in this chapter include all students of the course. This does not necessarily have to be a reason for the differences between the statistics, but it could be one. Furthermore could a delayed registration of some students to the course - e.g. a few days before the exam begins - be an explanation for these differences. Such a behaviour leads to an equalisation of data.

The small number of submissions in the course “Logik und Diskrete Strukturen” (eng.: Logic And Discrete Structures) in summer semester 2015 may be expounded through the fact that there was no bonus point system for the exam, while “Formale Sprachen und Komplexität” (eng.: Formal Languages And Complexity) in summer semester 2015 and 2016 had this. A combination of all three courses is shown in figure 3.5. It clarifies the decrease of submissions over the semester period.

All these statistics point out that it is important to select training data which fits the course. This means, if one attempts to make predictions based on training data, the training data should be taken from courses with similar properties and educational approaches. Possible selection criteria are number of assignments, course (e.g. earlier semesters), professor of the course and bonus point system. For these reasons it is important that the software implementation provides a way to select different training data. Moreover the data should be exchangeable in the running system to evaluate various types of training data for different courses.

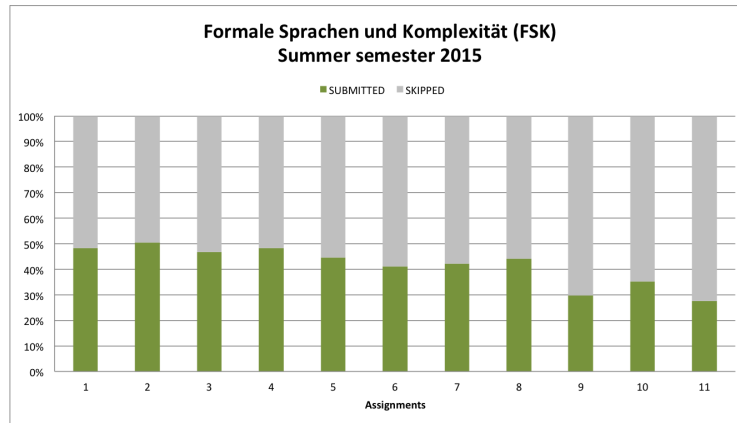


Figure 3.2: Skipping/Submitting behaviour distribution of 272 students in “Formale Sprachen und Komplexität” (FSK) in summer semester 2015 (SS15) at the Ludwig-Maximilian University of Munich. By submitting a near-faultless solution the students could gain bonus points for the exam.

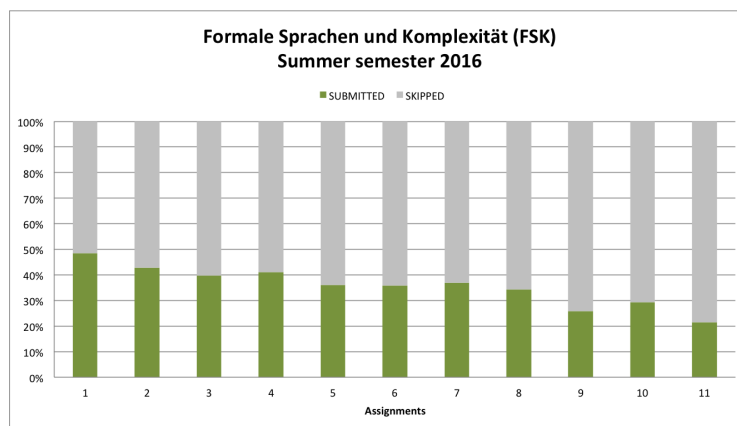


Figure 3.3: Skipping/Submitting behaviour distribution of 616 students in “Formale Sprachen und Komplexität” (FSK) in summer semester 2016 (SS16) at the Ludwig-Maximilian University of Munich. By submitting a near-faultless solution the students could gain bonus points for the exam.

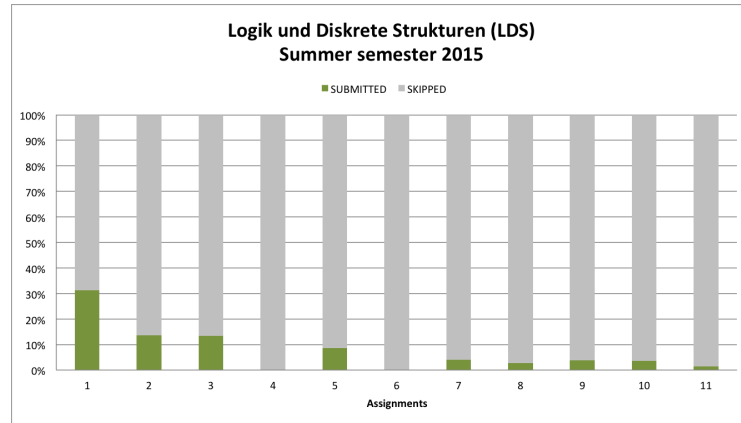


Figure 3.4: Skipping/Submitting behaviour distribution of 290 students in “Logik und Diskrete Strukturen” (LDS) in summer semester 2015 (SS15) at the Ludwig-Maximilian University of Munich. The assignments were optional and without the possibility of bonus points for the exam.

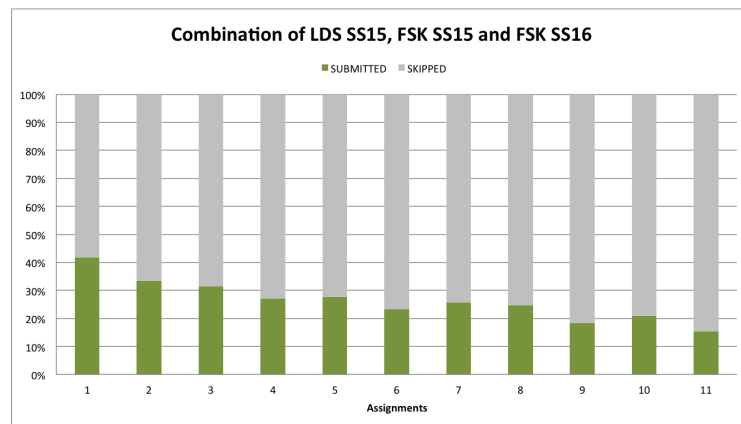


Figure 3.5: Combined statistic of “Logik und Diskrete Strukturen” in summer semester 2015 and “Formale Sprachen und Komplexität” in summer semester 2015 and 2016.

CHAPTER 4

Statistical Models

There are different statistical models which can be used to predict the behaviour of students based on the tagged data. An appropriate prediction model should provide answers to the following questions:

- (1) How to train a model given a tag history?
- (2) How to determine the probability of the next possible state (with respect to the students' personal recent behaviour sequence)?
- (3) How to represent students' personal characteristics, like motivation or time constraints?

First a deeper look at two models, which were implemented in the context of this thesis, is given at first. In section 4.1 a relative frequency estimation is discussed and in section 4.2 is shown why a Hidden Markov Model better fits to the addressed problem. All following applications of the considered models are based on the tag history of the introductory course in Theoretical Computer Science described in chapter 3.

4.1 Relative Frequency

The relative frequency estimation offers a plain statistical model where the occurrence of each tag in the tag histories is counted based on the assignment. To get a distribution this count is divided by the quantity of students. So for each assignment a distribution of given tags is computed. If more than one tag history is used as training data, the tables are simply concatenated. Figure 4.1 shows an example with two tag histories.

Even if this model with its simple structure addresses two of our three main problems ((1) and (2)), it is very unspecific. The students' personal recent behaviour sequence has no impact of the predicted next tag which means each student receives the same prediction for

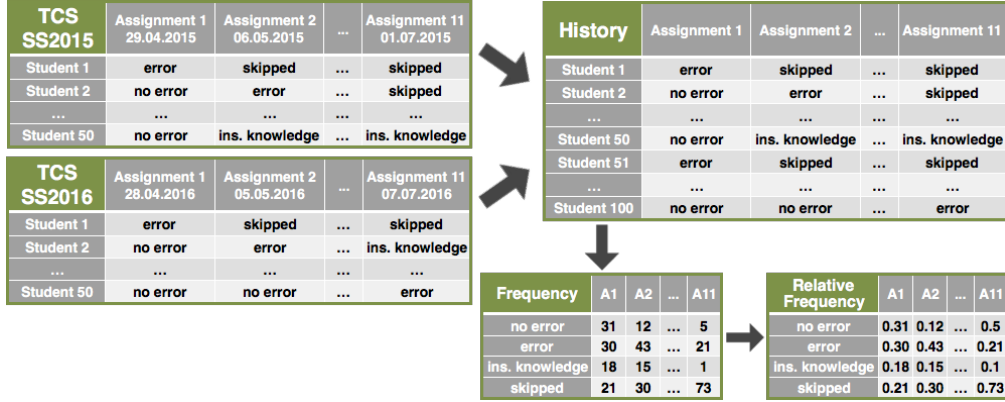


Figure 4.1: Conversion of two exemplary histories into a relative frequency distribution. The two tag histories are concatenated.

a specific assignment. Moreover personal characteristics are not represented. To take these points into account another statistical model should be used.

4.2 Markov Models

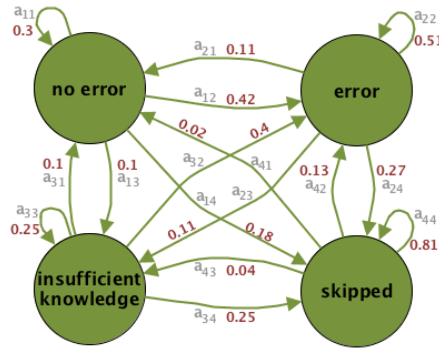


Figure 4.2: Visualisation of the trained Markov Chain based on the Theoretical Computer Science course introduced in chapter 3.

4.2.1 Markov Chain

To express the personal recent behaviour sequence a *Markov Chain* can be used as described by Rabiner (1989, Section II.). The Markov Model describes a set of states. In our particular use case the states are the observed tags “no error” (S_1), “error” (S_2), “insufficient knowledge” (S_3) and “skipped” (S_4). A set of probabilities is associated with each state to represent the transitions between them as seen in figure 4.2. To a specific time, in our case when a new submission is assessed, “the system undergoes a change of state”. The transition probabilities of such a model are defined by the Matrix A :

$$A = \{a_{ij}\} = \begin{bmatrix} 0.30 & 0.42 & 0.10 & 0.18 \\ 0.11 & 0.51 & 0.11 & 0.27 \\ 0.10 & 0.40 & 0.25 & 0.25 \\ 0.02 & 0.13 & 0.04 & 0.81 \end{bmatrix} \quad (4.1)$$

The Markov Chain is trained by counting the transitions between a state and its following state. This count is normalized by the outgoing transitions per state. Table 4.1 visualises this procedure based on the “error” state. So for example the normalized value for the transition “error” to “skipping” is calculated by $\frac{73}{274} \approx 0.27$.

Transition: “ error ” to ...	“error”	“no error”	“insufficient knowledge”	“skipping”
Transition count	31	139	31	73
Transitions from “error” to any other state	31 + 139 + 31 + 73 = 274			
Transition distribution	0.11	0.51	0.11	0.27

Table 4.1: Example of how the Markov Chain matrix is trained.
Representation of the “error” state transitions.

Given a sequence of states “insufficient knowledge”, “error”, “skipped”, “error” and the exemplary model, the probability can be calculated:

$$\begin{aligned}
P(O|Model) &= P[S_3, S_2, S_4, S_2|Model] \\
&= P[S_3] \cdot P[S_2|S_3] \cdot P[S_4|S_2] \cdot P[S_2|S_4] \\
&= \pi_3 \cdot a_{32} \cdot a_{24} \cdot a_{42} \\
&= (0.25)(0.4)(0.27)(0.04) \\
&= 0,00108
\end{aligned} \quad (4.2)$$

Where O and π_i are defined as

$$\begin{aligned}
O &= S_3, S_2, S_4, S_2 \\
\pi_i &= \frac{1}{N}, \quad 1 \leq i \leq N
\end{aligned}$$

Since a way to express tagged sequences is found, the origin of the trained model (1) and the personal characteristics (3) are still unaddressed. Anyway personal characteristics are affecting the student and can not be observed for each one in the given situation, e.g. the loss of motivation. Like Rabiner says “there is often some physical significance attached to the states or to sets of states of the model” (Rabiner, II. B.). In our special use case there are two personal characteristic states. Either the student is active or inactive, where inactive — for example — means different things like a lack of motivation to solve the assignments or time-dependent problems which lead to an unduly delay.

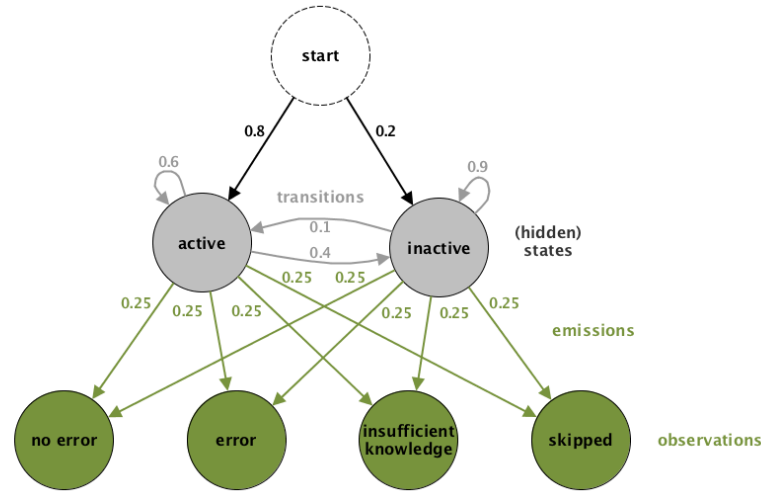


Figure 4.3: Initial Hidden Markov Model with two different hidden states and four observations.

4.2.2 Hidden Markov Model

For this reason we need an additional “underlying stochastic process that is not observable” but can be specified by the observable states (Rabiner, II. A.). These kinds of models are called *Hidden Markov Model* (HMM). Unlike the Markov Chain, the respective observable state is determined by a non-observable (hidden) state as shown in figure 4.3. This means that the probability to end up in a specific observable state depends on the current hidden state. Like it is demonstrated in Figure 4.4 there are two transitions once the state changes. First the models hidden state changes depending on the *transition probabilities* and second the observation is defined by the *emission probabilities*. This implies that the probability of an observation only depends on the last hidden state rather than on all previous hidden states, since this calculation of all permutations of states “is computationally unfeasible” (Rabiner, p. 262).

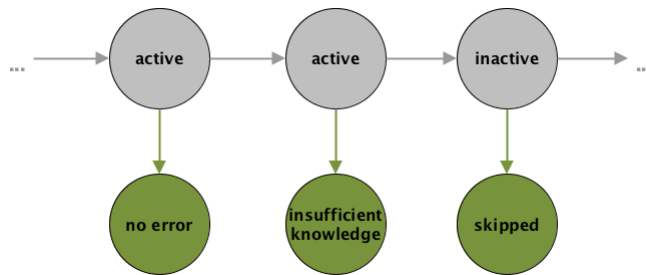


Figure 4.4: Transitions of an Hidden Markov Model. The current (hidden) state determines the next observation.

That HMM seems to be a fitting model for predicting students’ dropout is also described by Balakrishnan (2013): “HMMs prove a suitable choice since the hidden state can model

latent characteristics of the student that influence their will to persevere, and we can then infer these from their observable interactions [...]. Furthermore, an HMM [...] allows us to infer a student's behavior in the next time step based on their previous state and their currently observable actions."

The first problem of interest is how to train a HMM depending on the tag histories. In general the HMM is characterised by two lists of states and three probability distribution sets as seen in figure 4.3. To predict the behaviour of students with respect to work assignments, these characteristics are defined as follows:

(Hidden) states S Like mentioned before the hidden states describe the personal characteristics of a student. As it is impossible to know each circumstance which affects the student, the personal characteristics are summarised into two groups. On the one hand the positive characteristics (*active* S_{active}) and on the other the negative characteristics (*inactive* $S_{inactive}$).

$$S = \{S_{active}, S_{inactive}\} \quad (4.3)$$

Observation symbols O The results of the assessment are the observation symbols. As described in chapter 3, there are four tags: "no error" (O_{no_err}), "error" (O_{err}), "insufficient knowledge" (O_{ins_know}) and "skipped" ($O_{skipped}$).

$$O = \{O_{no_err}, O_{err}, O_{ins_know}, O_{skipped}\} \quad (4.4)$$

Transitions A The transitions are the state probability distributions. For example, if the system is in state *active* it describes the probabilities to switch to the state *inactive* or to stay in the same state. Assuming that keeping a personal characteristic is more likely than switching, the transitions are chosen.

$$A = \begin{bmatrix} 0.6 & 0.4 \\ 0.1 & 0.9 \end{bmatrix} \quad (4.5)$$

Emissions B The emissions are the observation symbol probability distributions. They define the probability of each tag depending on the current state. To have a minimum of influence to the training process, the emissions are kept equally distributed.

$$B = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix} \quad (4.6)$$

Initial state distribution π If the system is not in any state yet, the initial state distribution decides which state is used to start with. As initial parameters the submission distribution of assignment one from 1.1 is chosen.

$$\pi = \begin{bmatrix} 0.79 & 0.21 \end{bmatrix} \quad (4.7)$$

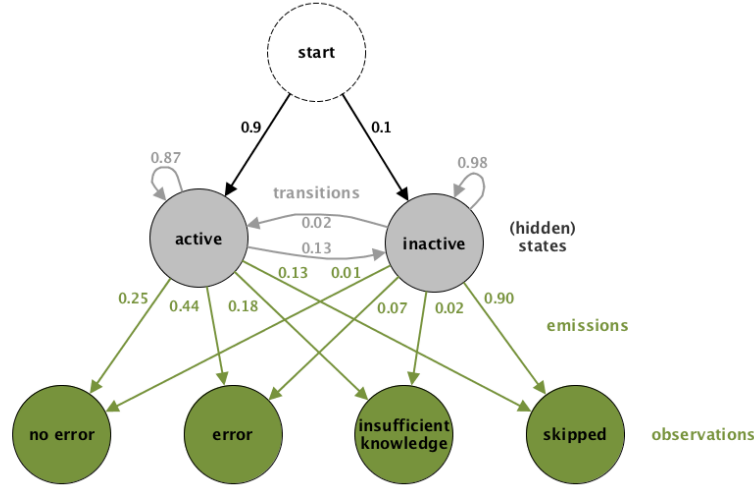


Figure 4.5: Hidden Markov Model with trained probabilities. The Theoretical Computer Science course introduced in chapter 3 is used as training data.

Before the model can be trained, the model needs to be initialised by specifying the transition, emission and initial probability distribution sets as well as the hidden states and observation symbols. The compact notation for the parameter set of the model looks like:

$$\lambda = (A, B, \pi) \quad (4.8)$$

As defined by Rabiner as *problem 3*, the initial model can be optimized by tag histories using the *Baum-Welch algorithm* (Rabiner, III. C.). Once the model is trained, the probability distribution sets are adjusted like exemplary shown in figure 4.5.

Running Course	Assignment 1 06.11.2016	Assignment 2 03.12.2016	Assignment 3 08.01.2017	Assignment 4 10.02.2017	Assignment 5 06.03.2017
Student	insufficient Knowledge	error	skipped	error	?

Figure 4.6: Exemplary personal recent behaviour sequence of a student up to assignment 4. Assignment 5 should be predicted.

Based on this trained model and the students' personal recent behaviour sequence the prediction should be made. As Rabiner describes *problem 1*, a trained model and an observable sequence can be used to calculate the probability of that sequence using the *Forward-Backward algorithm* (Rabiner, III. A.). Assuming a personal recent behaviour sequence until assignment four during a running course.

$$Seq_{recent} = O_{ins_know} O_{err} O_{skipped} O_{err} \quad (4.9)$$

The fifth assignment outcome of the student should be predicted (exemplified by figure 4.6). To predict the future behaviour the personal recent behaviour sequence must be extended

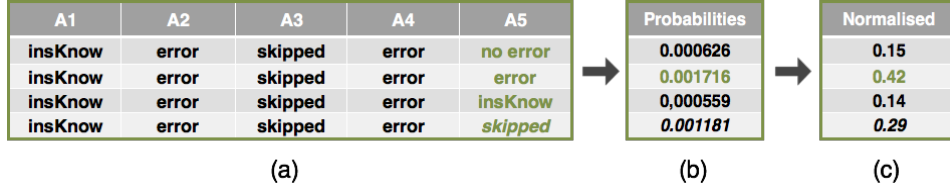


Figure 4.7: Process to calculate a distribution of the HMM predicted probabilities based on a user sequence. (a) Brute force approach: Extension of the user sequence with each possible step. (b) Predicted probabilities of each step provided by the Hidden Markov Model. (c) Normalised probability distribution.

with each observation symbols (figure 4.7 (a)).

$$\begin{aligned}
 Seq_{next} &= Seq_{recent} O_{next} \\
 &= O_{ins_know} O_{err} O_{skipped} O_{err} O_{next}
 \end{aligned}
 \tag{4.10}$$

where

$$O_{next} \in \{O_{no_err}, O_{err}, O_{ins_know}, O_{skipped}\}$$

Now it is possible to calculate the probability for each of these sequences (figure 4.7 (b)). Comparing these probabilities respecting the maximum likelihood the best guess for the next behaviour is received. As the other observable states might be of interest as well, a probability distribution should be created. For this purpose the probabilities have to be normalised. The normalisation is done by dividing each of these probabilities by the one of the personal recent behaviour sequence. This is reasonable since the personal recent behaviour sequence is already concluded thus it has a probability of 1. In general, it can be said:

$$P(O_{next}|\lambda) = \frac{P(Seq_{next}|\lambda)}{P(Seq_{recent}|\lambda)} \tag{4.11}$$

Figure 4.7 (c) exemplified such a normalised probability distribution of the previously defined sequence.

Additionally it is possible to predict the underlying hidden states (*Rabiner problem 2*). The *Viterbi algorithm* reveals a hidden state sequence given an observation sequence (Rabiner, III. B.). The prediction of the underlying hidden states is not further pursued in this work.

Summarised the Hidden Markov Model solves our initial main problems to predict the students' behaviour regarding work assignments as follows:

- (1) How to train a model given a tag history?

Based on an initial model and a tag history, the model can be trained using the *Baum-Welch algorithm*.

- (2) How to determine the probability of the next possible state (with respect to the students' personal recent behaviour sequence)?

The *Forward-Backward algorithm* calculates the probability of the extended personal recent behaviour sequence concerning a trained model. Afterwards the probabilities can be normalised to receive a probability distribution.

- (3) How to represent students' personal characteristics, like motivation or time constraints?

The personal characteristics are expressed by the hidden states of a HMM.

Technical Implementation

To combine different classification types (categories) and statistical models an extendable prediction framework has been implemented. The main components of this framework are the prediction units. They are responsible for training the statistical models and for predicting the possible next behaviour of the students. How the prediction framework is structured so as to be modular is addressed in section 5.1. Section 5.2 describes the integration in an existing system. What has to be done so as to implement further prediction units and tags to extend the framework is discussed in section 5.3. The server-side of this system is implemented in *Scala*¹ using the *Play Framework*². The prediction framework is implemented in Scala as well. The Hidden Markov Model, used to predict the students behaviour, is part of the Smile library³.

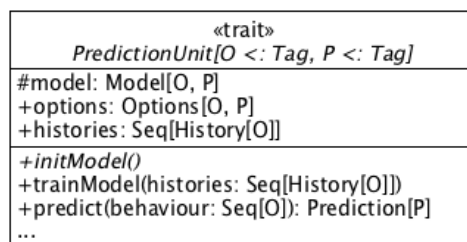


Figure 5.1: Simplified UML diagram of the `PredictionUnit` trait. It shows the attributes as well as the functions which are needed to predict the students' behaviour.

¹The Scala Programming Language, version 2.11. Available at <https://www.scala-lang.org>

²Play Framework, version 2.5. Available at <https://www.playframework.com>

³Smile - Statistical Machine Intelligence and Learning Engine, version 1.2. Available at <http://haifengl.github.io/smile>

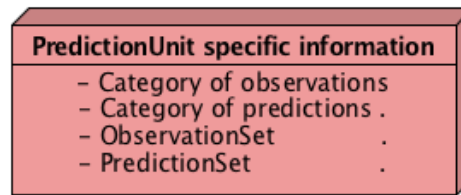


Figure 5.2: Simplified visualisation to show which tag category is necessary for which interaction of the `PredictionUnit`. (a) Initial options. (b) Training data (c) Transformation of observations into predictions

5.1 Prediction Framework

As seen in chapter 3, there are different kinds of classifications. Therefore the prediction unit needs to be independent of the tagged data. Furthermore the statistical model needs to be exchangeable as described in chapter 4. To reach this the `PredictionUnit` is implemented as a `trait` without any dependency on the statistical model and the category of tagged data. Only the concrete implementation of this trait determines the model as well as the observation and prediction categories. Figure 5.1 shows the design of the `PredictionUnit` trait. It shows the inner dependencies of the observation and prediction tag category and where which one is used.

At first — in subsection 5.1.1 — the tags are separated into different classes. Subsection 5.1.2 explains the design of the `PredictionUnit` step by step. After that, the interface to the outside of the framework is shown in subsection 5.1.3 and 5.1.4.

5.1.1 Observation and Prediction Tags

In general it can be said that making a prediction is transforming a sequence of tags (observation) into a distribution of tags (prediction). As mentioned in chapter 3, the category of observations and predictions can be different. To realise this the prediction unit depends on two independent tag categories. Besides the categories there is a need to know which tags are allowed, because the prediction unit respectively the statistical model needs the possible values to create a probability distribution of them. These are the main information that each `PredictionUnit` needs to be initialised. To describe the design of tags some keywords have to be defined. The skipping behaviour classification of section 3.1 is used to explain these keywords by example. Figure 5.2 visualises all dependencies between the `PredictionUnit` and the tag categories.

Tag A *tag* is a classification feature. It describes an observed or predicted behaviour. The tags to predict the skipping behaviour are the already defined classifications: “no error”, “error”, “insufficient knowledge” and “skipped”. A possible other example is a mark in an exam (e.g. the marks 1,0; 2,3 or 3,7). Generally can be said that a tag is a value of a behaviour.

Tag category Each tag belongs to exactly one *tag category*. The possible values for how the behaviour can be categorised are described by this category. To make a prediction the prediction unit receives a tag sequence where each tag is from the observation tag category. Based on this sequence the prediction unit predicts a probability for each tag out of the prediction tag category.

For the skipping behaviour prediction the tag category is called *ErrorState*. *ErrorState* contains the four afore-mentioned tags. In this case the observations and predictions are of the same category. An example for the use of different tag categories is the prediction of an exam mark based on the submission behaviour. Here the prediction category is represented by a set of all possible marks (*ExamMark*) while the observation category is the *ErrorState*. Possible tags of this category are: {1,0; 1,3; 1,7; 2,0; ...; 4,0; 5,0}. In this use case the prediction unit would transform a sequence of *ErrorState*-tags into a probability distribution of all possible exam marks.

Tag category collection The *TagCategoryCollection* is a collection of all tag categories.

It is necessary due to two reasons. On the one hand a user might want to know which are the available categories to choose from, on the other hand it has technical reasons:

The prediction unit has to know from which tag category the observations should be taken and from which tag category the predictions should be produced. This could be realised by using classical object oriented inheritance. But some other system requirement causes trouble. As imaginable, the tags are supposed to be saved in a database on the integrating system. This concerns the observation sequences as well as the prediction distributions. As will be shown later, the framework components will be implemented as generic classes. To (de-)serialise the tags during runtime the system has to detect the category so that the correct serialisation method can be chosen. Exactly this is done by the *TagCategoryCollection*. It provides methods to serialise and deserialise the different tag categories.

Summarised the *TagCategoryCollection* is a collection to define the used tag category in different objects during the system runtime.

The trait *Tag* is a wrapper to determine different implementations of observation- and prediction-values. The implemented tags are bound to a tag category. A tag category is not explicit implemented class or trait. Instead it is a kind of *Enumeration* where each possible tag is added as value. The *TagCategoryCollection* is a Scala *Enumeration* to which each realised category is added. Figure 5.3 visualises the components of the *ErrorState* and *ExamMark* implementation with entries in the *TagCategoryCollection* enumeration.

A problem of Scala's *Enumeration* (enum) is that "Enumerations have the same type after erasure" (Rijo (2016) and Dallaway (2014)) which means "you do not end up with a class per member" as said by Dallaway. Unfortunately this is what we need for the tag category. Each value of the tag category has to be an instance of the trait *Tag* to enable a generic implementation of the *PredictionUnit*. Regarding to Rijo and Dallaway sealed case objects can be used instead. With sealed case objects it would be possible to ensure that each value of the tag category is an instance of *Tag*, but there would be no possibility to

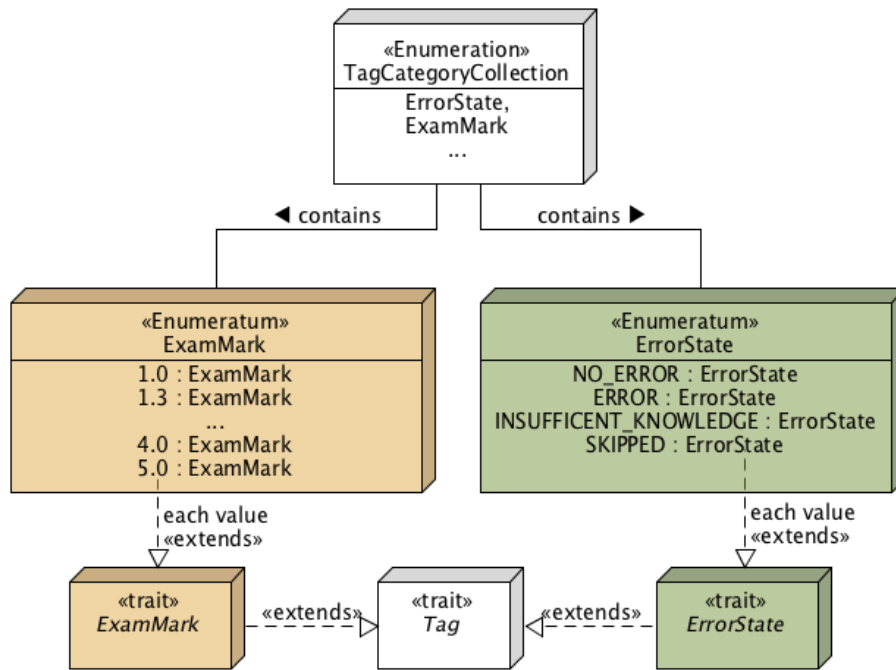


Figure 5.3: Reduced UML diagram of the tag components based on the `ErrorState` and `ExamMark` categories.

get all values of the category at a time. This again is needed to instantiate the `PredictionUnit` as seen in figure 5.2. Additionally it can be imagined that it is helpful to have one method which provides each possible value of a tag category when the students' behaviour should be described. Thankfully there is a library called *Enumeratum* by Chan⁴ which offers these functionalities. Besides that it provides “exhaustive pattern match warnings” which might be useful when checking the prediction distributions. Further it adds a good integration into the *Play Framework* which is used at the project anyway. Listing 5.1 shows the usage of the `enumeratum` for the `ErrorState` tag category. At line 1 a tag category specific `Tag` for the entries of the `enumeratum` is introduced, called `ErrorState`. This sealed `trait` is used to instantiate the `enum` object `ErrorState` (line 3). Hereby the name of the `EnumEntry` and the `PlayEnum` itself have to be equal which might be a little confusing when reading the code. For a better understanding in the later work the `enumeratum` itself — the extended `PlayEnum` — will be called *ErrorStates* or *ErrorState category* and one value of this `enum` will be called *ErrorState*. As seen in line 6–9, each value of the tag category is extended with the specific `Tag ErrorState`.

5.1.2 Prediction Unit

Since the design of tags is explained now, the prediction unit can be described. As already mentioned, the prediction unit depends on the observation and prediction category. So

⁴Chan at github: <https://github.com/lloydmeta/enumeratum>


```

1 sealed trait ErrorState extends EnumEntry with Tag
2
3 object ErrorState extends enumeratum.PlayEnum[ErrorState] {
4   val values = findValues // mandatory due to Enum extension
5
6   case object NO_ERROR           extends ErrorState
7   case object ERROR              extends ErrorState
8   case object INSUFFICIENT_KNOWLEDGE extends ErrorState
9   case object SKIPPED           extends ErrorState
10 }

```

Listing 5.1: ErrorState implementation that shows the usage of the Enumeratum library.

the PredictionUnit needs to be generic for these both types (line 1 in listing 5.2). The TagCategoryCollection is not used in the PredictionUnit itself although it is completely build up on generics, but it has not be serialised. Anyway the PredictionUnit additionally requires all possible values of both — the observation and prediction category — as already seen in figure 5.2. The possible values are defined inside the inner trait Options (line 5–8 in listing 5.2). Line 2 in listing 5.2 instantiates the Options directly. This forces the developer of a new class which implements the PredictionUnit to define the sets, since each class has to override abstract fields and methods of its parents traits.

```

1 trait PredictionUnit[O <: Tag, P <: Tag] {
2   val options : Options[O, P]
3   ...
4
5   trait Options[O <: Tag, P <: Tag] {
6     val observationSet : Seq[O]
7     val predictionSet : Seq[P]
8   }
9   ...
10 }

```

Listing 5.2: PredictionUnit implementation (Part 1/3). It shows the definition of the inner Options trait.

Listing 5.3 shows such an implementation of a Hidden Markov Model to predict students' skipping behaviour. The first line of the listing shows the definition of the same tag category for observations and predictions. As described before, the options have to be implemented. Here this happens as an anonymous class (line 3 in listing 5.3). The observationSet and predictionSet is defined by retrieving all values of the ErrorState category (line 4–5 in listing 5.3). Inside the inner class it is possible to instantiate additional properties which may be needed by the used statistical model (line 8–10 in listing 5.3).

So far a concrete prediction unit implementation could be initiated. But there is still no statistical model yet. Listing 5.4 shows the next part of the PredictionUnit. The Model is likely to the Options implemented as inner trait (line 8–11 in listing 5.4). Because of a

```

1 class HMMSkippingPrediction extends PredictionUnit[ErrorState, ErrorState] {
2
3   override val options = new Options[ErrorState, ErrorState] {
4     override val observationSet: Seq[ErrorState] = ErrorState.values
5     override val predictionSet: Seq[ErrorState] = ErrorState.values
6
7     // additional options for the HMM
8     val iterations = 100 // baum-welch iterations / forward-backward algorithm
9     val startProbability = Array(0.8, 0.2) // motivated / unmotivated
10    ...
11  }
12  ...
13 }

```

Listing 5.3: HMMSkippingPrediction implementation (Part 1/2). It shows an anonymous class implementation of the Options trait. Besides the needed set overrides it defines additional options which are used to initialise the Hidden Markov model.

possible reinitiation of the model its field is not declared as constant (line 3 in listing 5.4). For the same reason the PredictionUnit provides an abstract method initModel() which has to be implemented by its child classes (line 13 in listing 5.4).

```

1 trait PredictionUnit[O <: Tag, P <: Tag] {
2   val options : Options[O, P]
3   protected var model : Model[O, P]
4   var histories: Seq[History[O]] = Seq[History[O]]()
5
6   trait Options[O <: Tag, P <: Tag] { ... }
7
8   trait Model[O <: Tag, P <: Tag] {
9     def train(history : Seq[Seq[O]]) : Unit
10    def predict(behaviour : Seq[O], predSet : Seq[P]) : Prediction[P]
11  }
12
13  def initModel()
14  ...
15 }

```

Listing 5.4: PredictionUnit implementation (Part 2/3). It shows the definition of the inner Model trait.

As with the Options, the Model has to be defined when extending the PredictionUnit, but in this case the implementation is not an anonymous class as seen in listing 5.5 on line 6. Instead the inner class is implemented regularly to use it twice (line 29 and 4 in listing 5.5).

Since the model can be initialised by creating a concrete implementation of the PredictionUnit, it still has to be trained. The model is trained by sequences of observations

```

1 class HMMSkippingPrediction extends PredictionUnit[ErrorState, ErrorState] {
2
3   override val options = new Options[ErrorState, ErrorState] { ... }
4   override protected var model: Model[ErrorState, ErrorState] = new HMMModel
5
6   class HMMModel extends Model[ErrorState, ErrorState] {
7     var hmm: HMM[ErrorState] = new HMM[ErrorState](options.startProbability, ...)
8
9     override def train(history: Seq[Seq[ErrorState]]): Unit = {
10       val histArray = seqToArray(history)
11       hmm = hmm.learn(histArray, options.iterations)
12     }
13
14     override def predict(behaviour: Seq[ErrorState], predSet: Seq[ErrorState]):
15       Prediction[ErrorState] = {
16       val possibleSeqs = getPossibleSequences(behaviour, predSet)
17       val p = if (behaviour.size > 0) hmm.p(behaviour.toArray) else 1
18       val dist = mutable.Map(possibleSeqs.map({
19         case (sym, seq) => (sym -> hmm.p(seq) / p) }).toSeq: _*)
20       new Prediction(dist)
21     }
22
23     protected def getPossibleSequences(behaviour: Seq[ErrorState], predSet: Seq[
24       ErrorState]): Map[ErrorState, Array[ErrorState]] = {
25       (predSet.map(sym => sym -> (behaviour.toArray.clone() :+ sym))).toMap
26     }
27     ...
28
29     override def initModel() = {
30       model = new HMMModel
31     }
32 }

```

Listing 5.5: HMMSkippingPrediction implementation (Part 2/2). It shows the implementation of an inner HMMModel class.

— the so called History. A History is a simple case class as seen in listing 5.6. Attention should be paid to the tagCategory. As already described, the generic classes which perhaps should be serialised have to know about its categories during runtime — so does the History. How the serialisation is done is shown in section 5.2. In section 3.2 was shown that it has to be possible to change the training data of an unit. Therefore the histories are saved in the PredictionUnit (line 4 in listing 5.4).

```

1 case class History[+O <: Tag](title : String,
2                               entries: Seq[Seq[O]],
3                               tagType : TagType.Value)

```

Listing 5.6: Implementation of the case class History.

The `predict` method does the transformation from an observed behaviour sequence to a prediction distribution based on the `predictionSet`. As already described in section 4.2, the prediction is made (line 14–24 in listing 5.5).

The last part of the `PredictionUnit` is shown in listing 5.7. A benefit of traits “is that they can be partially implemented, like abstract classes” in Java (Harrison, 2011). Through this, some operations can be predefined inside the `PredictionUnit` so that the developer has not to take care of them. These operations are the concatenation of histories and the insertion of the `predictionSet` into the `predict` method. The `PredictionUnit` itself can be seen as a wrapper or container which delegates the request to the model. By this, the interface of the `PredictionUnit` is very clean and no interaction with the model itself is needed from the outside.

```

1 trait PredictionUnit[O <: Tag, P <: Tag] {
2   val options : Options[O, P]
3   protected var model : Model[O, P]
4   var histories: Seq[History[O]] = Seq[History[O]]()
5
6   trait Options[O <: Tag, P <: Tag] { ... }
7   trait Model[O <: Tag, P <: Tag] { ... }
8
9   def initModel()
10
11  def trainModel(history: History[O]) {
12    trainModel(Seq(history))
13  }
14
15  def trainModel(histories: Seq[History[O]]) {
16    initModel()
17    this.histories = histories
18    model.train( this.histories.map(_.entries).reduceLeft(_ ++ _) )
19  }
20
21  def predict(behaviour: Seq[O]): Prediction[P] = {
22    model.predict(behaviour, options.predictionSet)
23  }
24 }
```

Listing 5.7: `PredictionUnit` implementation (Part 3/3). It shows the delegation to the concrete implemented models.

5.1.3 Behaviour Log

Since the predictions that should be made are related to a course, there has to be a place to summarise all needed information for the prediction unit. Mainly these information are:

- `TagCategory` of the observations and predictions.
- Observation and prediction sets.
- Observations respectively the personal recent behaviour sequences of each student.
- Predictions of each student which already were predicted.
- Histories that should be used to train the prediction units
- The prediction units themselves which should be used for the predictions

In addition some other data might be useful:

- Names for each step / assignment.
- A possibility to compare two predictions per step, e.g. by subtract one prediction distribution from the other — called discrepancy.

The `BehaviourLog` is the one point in the framework where all the components per course come together. Figure 5.4 shows its class diagram. As already seen at the `History`, the `BehaviourLog` also holds a reference to its tag category. Reason for this is the serialisation and deserialisation as well. `ObservationSet` and `predictionSet` are needed to initialise the prediction units. The observations will be updated from the outside of the framework. They are represented by a key/value pair (`Map`). Key of this `Map` is the ID of the user which belongs to the sequence of observations (value). Once a observation is updated the predictions have to be updated. Since there can be multiple predictions per user (from different prediction units), the key of the outer `Map` is an unique identifier of the prediction unit. The inner key/value pair maps the user to a sequence of predictions. That sequence represents the predictions of the different steps/assignments. In summary, this means: `Map[<id of prediction unit>, Map[<id of user>, Sequence of predictions per step]]`. The `StepNames` are just a simple list in which the mapping to the related observations and predictions are identified by the index of the step. The `discrepancy` is a list with at most two entries. If the discrepancy of two prediction distributions should be calculated, it contains the IDs of the two prediction units which are subtracted. Histories are referenced by unique titles that are represented by lists of strings. Each behaviour log holds an instance of its prediction units which can be initialised by the `initUnit(...)` method using reflection. There are various further methods that enable all needed interactions to make predictions. Many of these delegate calls to the prediction units. By this structure the framework provides one main interaction point since all these methods are called from the outside of the framework.

5.1.4 Prediction Unit Factory

Another object that is directly used from outside the framework, is the `PredictionUnitFactory`. It is used to define which `PredictionUnit` and `BehaviourLog` implementations are available. Therefore it is an object to be accessible from everywhere in the system. Two maps are saved in this object. The prediction unit implementations are saved by

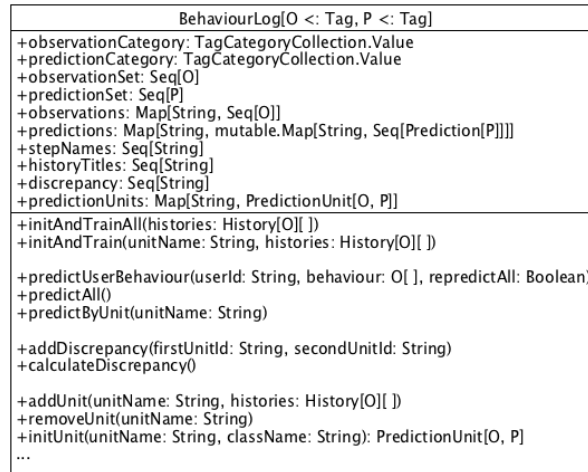


Figure 5.4: UML class diagram of the `BehaviourLog` which unites all components of the prediction framework.

its unique identifier and its corresponding qualified class path. In this way it is possible to create an instance of these prediction units applying reflection as used by the `BehaviourLog`. Available behaviour logs are presented by a second map that refers to a simplified behaviour log object (`SimpleBehaviourLog`). It only contains the most important information to initialise behaviour logs (figure 5.5). Besides the two kinds of categories it encloses the prediction units which can be used with this behaviour log.

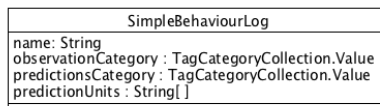


Figure 5.5: UML class diagram of the `SimpleBehaviourLog`. It defines the prime data which are needed to initialise a behaviour log.

There are some functions provided by the factory that can be used by the developer to (de-)register newly implemented prediction units and behaviour logs. Normally this registration is done at the server start up and might be defined in a configuration file. How this could look like is shown in the next section.

Figure 5.6 summarises the different components of the prediction framework. Besides the classes and traits, which are provided by the framework, the figure contains the implementation of the `ErrorState` tags and two prediction units.

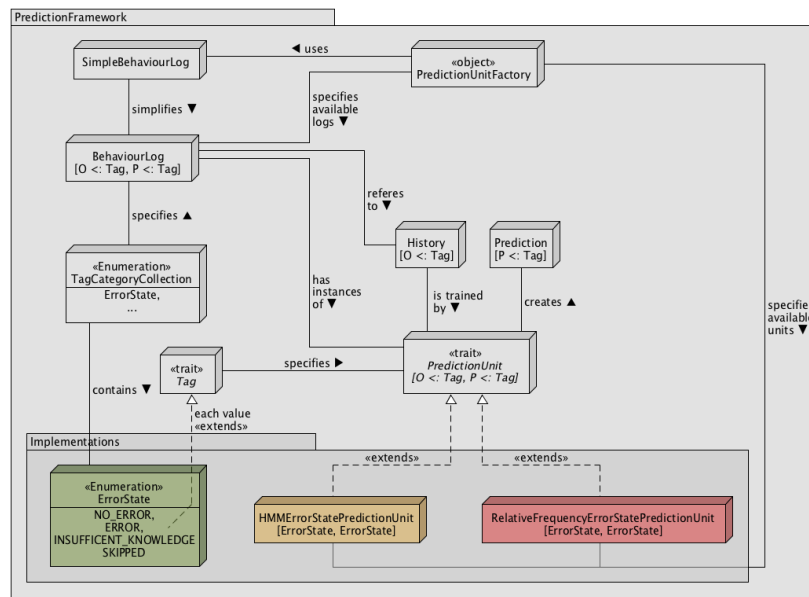


Figure 5.6: Simplified structure of the prediction framework including an implemented `ErrorState` category and two `PredictionUnit` implementations.

5.2 Integration

To integrate the prediction framework properly in an existing system some questions need to be answered.

- How are the available prediction units and behaviour logs registered to be used in different courses and projects?
- How are the histories and behaviour logs stored and reinitialised to still be available on server restart?
- How is the interaction between the front-end and the back-end managed and which requests are needed?

To address these questions we need to have a look at the design of the existing system in which the framework has been integrated. Like already described the system uses the Play Framework which provides *RESTful Web services* as interface between front-end and back-end. On server-side *Controllers* handle the requests of the client. *Actions*⁵ define the possible payload of these requests. The database of the system is MongoDB⁶ that saves BSON⁷ objects. BSON “is a binary-encoded serialization of JSON-like documents” (BSON - Binary JSON). This is convenient since the RESTful Web service can use JSON objects as

⁵<https://www.playframework.com/documentation/2.5.x/ScalaActions>

⁶MongoDB, version 2.8. Available at <https://www.mongodb.com>

⁷BSON - Binary JSON. Available at <http://bsonspec.org/>

response format. Through this the same deserialisation and serialisation methods can be used to transform the Scala objects when saving the data on the database and when sending the data to the front-end. The current system organises learning activities in projects. A `Project` can be used to represent a course. To integrate the prediction framework the important fields of this model are the unique identifier (`id`) and a list of all participants of the course. All project instances are fully serialisable to be saved in the database and to be sent to the front-end. The following subsections show how the integration into this existing system has been done.

5.2.1 Server Start Up

As already mentioned in subsection 5.1.4, the `PredictionUnitFactory` of the framework provides methods to register behaviour logs and prediction units. On the given system the available units and logs are defined in the application configuration as seen in listing 5.8. Each prediction unit needs an unique name and a fully qualified class path to the implemented class to be initialised by the `PredictionUnitFactory`. A behaviour log requires, besides an unique name, the prediction units it is using as well as the tag category of the observations and predictions. These are exactly the information which are needed to initialise a behaviour log (see figure 5.5). This configuration is used on server start up to register the prediction units and behaviour logs at the `PredictionUnitFactory`. After that the prediction units and behaviour logs can be initialised by the `PredictionUnitFactory` during runtime. Furthermore the *Singleton* `BehaviourLogFactory` is created on server start up which takes care of all behaviour log instances.

```

1 predictionUnits += { name = "HMM",
2   class = "predictionFramework.units.HMMSkippingPrediction" }
3 predictionUnits += { name = "Relative Frequency",
4   class = "predictionFramework.units.RFSkippingPrediction" }
5
6 behaviourLogs += { name = "Skipping Prediction",
7   predictionUnits = ["HMM", "Relative Frequency"],
8   observationCategory = "ErrorState", predictionCategory = "ErrorState" }
```

Listing 5.8: Scala Play configuration entries of the integrated system to add two prediction unit implementations and one behaviour log entry.

5.2.2 History and Behaviour Log Integration

To have a connection between behaviour logs and a course the existing `Project` model needs to be adjusted. It has to be said again that the `Project` has to be serialisable for the afore-mentioned reasons. Since the behaviour log should also be sent to the front-end to be displayed and adjusted, the behaviour log referenced in the project must be serialisable as well. This makes also sense considering that all predictions of a project are saved in the behaviour log. If the behaviour logs would not be serialised and stored in the database, the

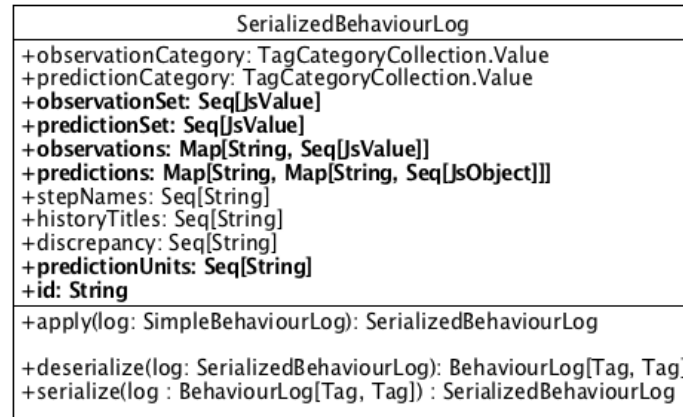


Figure 5.7: UML class diagram of the `SerializedBehaviourLog` which acts as a serialisable version of the `BehaviourLog` (Figure 5.4). Differences between these two classes are highlighted.

predictions of all courses would have to be recalculated when the server starts. In case of a slow predicting process this would have a delaying effect on server start up.

Since each `BehaviourLog` holds references to its prediction units, the prediction unit implementations would have to be serialisable as well, which might be a problem for the developer. To prevent this, a serialisable version of the `BehaviourLog` is introduced. This so-called `SerializedBehaviourLog` is saved in the `Project`. By saving the unique name of its prediction units instead of a reference, the prediction unit itself does not need to be serialisable. Furthermore the `SerializedBehaviourLog` is no generic class implementation which means that it is independent of the implementations of the tag categories. Instead it saves the different tags as JSON objects. Figure 5.7 shows the UML diagram of the `SerializedBehaviourLog`. By using this serialisable version, the `Project` can be extended and is still able to be saved in the database and to be sent to the front-end.

However the new class adds another issue — the transformation between `BehaviourLog` and `SerializedBehaviourLog`. To this end the `SerializedBehaviourLog` has two additional methods as seen in figure 5.7. The `serialize` and `deserialize` methods are used to convert the tags based on the `observationCategory` and `predictionCategory`. Exactly this is the reason why tag categories have to be saved as variables in the `BehaviourLog`. Without doing this it would not be possible to know to which tag the JSON data should be converted. How a respective tag of a category is converted is task of the developer who introduces a new tag category. As the developer should not have to adjust the nested entries of the `SerializedBehaviourLog`, the framework provides one point where a converter function can be implemented so that the developer need not adjust the nested entries of the `SerializedBehaviourLog`. The `TagCategoryCollection` provides methods — for serialisation and deserialisation — which return a converter function based on the category. How these methods look like is shown in subsection 5.3. Anyway the `SerializedBehaviourLog` calls these methods and uses the converter functions

when converting between the two behaviour log implementations.

Unfortunately the conversion from `SerializedBehaviourLog` to `BehaviourLog` should only be done once at the server lifetime, since it holds the reference to prediction unit instances. For this purpose the `BehaviourLogFactory` is used. It adds the behaviour log instances to the projects of the system. At server start up the `BehaviourLogFactory` initialises all behaviour logs by finding all projects in the database which include at least one behaviour log. For each of these projects the `BehaviourLogFactory` deserialises the `SerializedBehaviourLog` which creates a `BehaviourLog` instance. During the deserialisation the prediction units of the behaviour log are initialised as well. Once a `BehaviourLog` instance is created the `BehaviourLogFactory` trains its prediction units by its histories. After that all `BehaviourLog` instances are saved in a map. The key of an entry is the concatenation of the project ID and the behaviour log name defined in the configuration file. By doing this the `BehaviourLogFactory` can handle requests on an specific behaviour log and change it accordingly.

```

1 def deserialize(h : SerializedHistory): History[Tag] = {
2   val converter = TagCategoryCollection.deserialize(h.category)
3   new History[Tag](h.title, h.entries.map(_.map(converter)), h.category)
4 }
5
6 def serialize(h : History[Tag]) : SerializedHistory = {
7   val converter = TagCategoryCollection.serialize(h.category)
8   new SerializedHistory(h.title, h.entries.map(_.map(converter)), h.category)
9 }

```

Listing 5.9: Implementation of the `serialize` and `deserialize` method of the `SerializedHistory`. It shows the call of the `TagCategoryCollection` methods which return converter functions based on the category.

As already described in subsection 5.1.2, the histories should be serialised as well. Since the `History` is a generic class which depends on the observation tag category, there is also a need of a serialised version of this class. The `SerializedHistory` is similar to the `SerializedBehaviourLog`. It also saves its entries as JSON object and provides a `serialize` and `deserialize` method. Listing 5.9 shows the implementation of these methods. Due to the fact that each `History` can be used by multiple behaviour logs respectively projects, it is saved in an own database collection.

5.2.3 Client-Server Communication

As mentioned, the Play Framework uses *Controllers* and *Actions* to handle client requests. To handle the requests which update the behaviour log an own Controller and Action is written. The Action defines the values possibly send with a request. It is called `PredictionUpdate`. Figure 5.8 shows the UML diagram. In order to know which behaviour log is addressed the `projectId` and `behaviourLogTitle` are mandatory. If the further

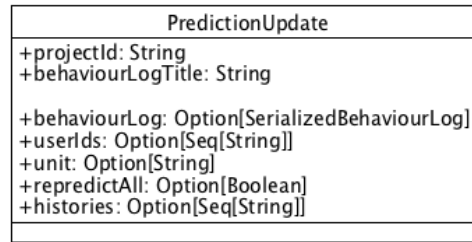


Figure 5.8: UML class diagram of the `PredictionUpdate` Action. It contains the project ID and the title of the current behaviour log as required fields. All further fields are optional.

fields are set depend on the request. Possible requests are defined by the RESTful API as shown in table 5.1.

GET	/predictions/logs	get all available behaviour logs
PUT	/predictions/project/behaviourlog	update the complete behaviour log
PUT	/predictions/project/histories	update the histories of the behaviour log
PUT	/predictions/project/addLog	add a new behaviour log to the project
PUT	/predictions/project/add	add a prediction unit to the behaviour log
PUT	/predictions/project/del	remove a prediction unit from the behaviour log

Table 5.1: An extracted version of the RESTful API which is used to modify or visualise the behaviour log.

The incoming requests are forwarded to the corresponding methods of the prediction controller. This controller extracts the values from the `PredictionUpdate` and delegates the task to the `BehaviourLogFactory`. The `BehaviourLogFactory` receives the appropriate `BehaviourLog` instance based on the project ID and behaviour log title from its `Map`. The `BehaviourLog` instance itself processes the request. Once the behaviour log is updated, it is serialised and saved in the `Project` by the `BehaviourLogFactory`. The `Project` is returned to the controller in which the `Project` is saved in the database. If everything went well, a response with the changed serialised behaviour log is sent to the client.

5.3 Framework Extension

This section summarises what needs to be done to extend the framework and integrate it into the system. For this purpose the *ExamMark* example is used which requires an additional tag category and a prediction unit. Short recapitulation: The students' exam mark should be predicted based on the assignment behaviour of that student.

Implement Tag At first a new tag category implementation is needed. Each possible tag of this category needs to extend the new *ExamMark* tag. Listing 5.10 shows this.

```

1 sealed trait ExamMark extends EnumEntry with Tag
2
3 object ExamMark extends enumeratum.PlayEnum[ExamMark] {
4   val values = findValues // mandatory due to Enum extension
5   case object mark_1_0 extends ExamMark
6   case object mark_1_3 extends ExamMark
7   ...
8 }

```

Listing 5.10: Implementation of the Enumeratum *ExamMark*.

Extend TagCategoryCollection To make the new tag category available in the framework the *TagCategoryCollection* has to be extended. This is done by adding the new enumeration to the values of this class (listing 5.11 in line 2). Furthermore the methods to serialise and deserialise the tags need to be implemented. As described in subsection 5.1.3, they are used when transforming the *BehaviourLog* into the *SerializedBehaviourLog* and vice versa. Moreover the method *getAllTags* should be extended which simply returns all tags based on the category.

```

1 object TagCategoryCollection extends Enumeration {
2   val ErrorState, ExamMark = Value
3
4   def deserialize(category: TagCategoryCollection.Value) : (JsValue) => Tag =
5     category match {
6       case ExamMark => ((jv: JsValue) => tags.ExamMark.withName(jv.as[String]))
7       case ErrorState => ((jv: JsValue) => tags.ErrorState.withName(jv.as[String]))
8       case _ => throw new NoSuchElementException(...)
9     }
10  def serialize(category: TagCategoryCollection.Value) : (Tag) => JsValue =
11    category match {
12      case ExamMark => o => JsString(o.toString)
13      ...
14    }
15  def getAllTags[P <: Tag](category: TagCategoryCollection.Value) : Seq[P] = ...
16 }

```

Listing 5.11: Extended version of the *TagCategoryCollection* which includes the *ExamMark* und *ErrorState* tag categories.

Implement a Prediction Unit Next a new `PredictionUnit` implementation is needed. The new class needs to extend the `PredictionUnit` trait. Here the category of observations and predictions are different as seen in listing 5.12. Through the inheritance the inner classes `Options` and `Model` as well as the `initModel` method have to be implemented.

```

1 | class ExamMarkPrediction extends PredictionUnit[ErrorState, ExamMark] {
2 |     override val options: Options[ErrorState, ExamMark] = ...
3 |     override protected var model: Model[ErrorState, ExamMark] = ...
4 |     override def initModel(): Unit = ...
5 | }

```

Listing 5.12: Shortened implementation of an `ExamMarkPrediction` unit.

Extend Configuration At last the extension of the configuration file is needed to register the new prediction unit in the system. Additionally the unit has to be added to an existing behaviour log or a new one is needed. Since there is no existing behaviour log which has the same tag categories as the `ExamMarkPrediction`, a new behaviour log is required.

```

1 | predictionUnits += { name = "ExamMarkUnit",
2 |     class = "predictionFramework.units.ExamMarkPrediction" }
3 | behaviourLogs += { name = "Exam Mark Log", predictionUnits = ["ExamMarkUnit"],
4 |     observationCategory = "ErrorState", predictionCategory = "ExamMark" }

```

Listing 5.13: Needed entries in the configuration file to register the `ExamMarkPrediction` unit and a new behaviour log.

5.4 Summary

In this chapter the architecture of the framework has been explained. Furthermore the integration in an existing system has been described. It has been demonstrated that the framework is simply extendable for further statistical models and additional tags. These are also mostly the reasons for the design of the framework, since it had to be independent of the statistical model and the tagged data as shown in chapter 3 and chapter 4.

CHAPTER 6

User Interface

The focus of this thesis lies on the server implementation. Therefore less emphasis has been put on the client-sided implementation of the user interface and the usage will be described instead. A front-end view for examining and getting predictions as well as for modifying behaviour logs was implemented and integrated in the existing system. As the rest of the platform, this view was implemented in *Angular JS*¹ and *Bootstrap*². The user interface represents exactly one behaviour log of the current project at a time. It consists of four main elements which will be explained one by one in the following sections.

6.1 Students' Behaviour Table




Name	Assignment 1	Assignment 2	Assignment 3	Assignment 4	
bob	NO_ERROR ▾	SKIPPED ▾	ERROR ▾		  
	HMM	HMM	HMM	HMM	
	ERROR: 47.77%	ERROR: 46.64%	ERROR: 26.83%	ERROR: 42.34%	
	SKIPPED: 18.93%	SKIPPED: 20.89%	SKIPPED: 55.39%	SKIPPED: 28.38%	
	NO_ERROR: 17.63%	NO_ERROR: 17.18%	NO_ERROR: 9.25%	NO_ERROR: 15.46%	
	INSUFFIC: 15.67%	INSUFFIC: 15.29%	INSUFFIC: 8.52%	INSUFFIC: 13.82%	

Figure 6.1: Students' Behaviour Table including one student and three assessed assignments.

To visualise the students' behaviour and their predictions a table has been chosen (figure 6.1). Each row represents a student and each column represents the students behaviour per step. The table header provides the possibility for naming the steps. So for example a step could be an assignment during a running course. An entry of the table visualises the

¹Angular JS, version 1.6. Available at <https://angularjs.org>

²Bootstrap, version 3.3. Available at <http://getbootstrap.com>

students' behaviour at a step. This is represented by a tag and can be changed using a drop-down menu. Changing a tag can be useful since the assessment for a tag of an assignment could be adjusted during a running course or if the tutor committed a wrong entry. Furthermore predictions per assignment are shown. It represents the predicted behaviour of the corresponding prediction unit for this assignment. How many different predictions are possibly shown depends on the available prediction units of this behaviour log. The last column of each row shows the predictions for the next non assessed assignment. Moreover the last column contains interaction elements for adding or removing an entry, or to reset all changes of this row. Once a change is done, either by adding/removing an entry or by editing one, the row is highlighted to visualise a change which needs to be saved.

On pressing the save button, which is available in the view control above the table, all current changes in the behaviour log are saved. By saving, the predictions of the students, whose behaviour has been changed, are recalculated and the behaviour log is saved in the database. Once this is done, the view is updated. Besides the saving button the view control contains a button to reset all current changes. These two buttons are shown on the top right of figure 6.2.

6.2 View Control

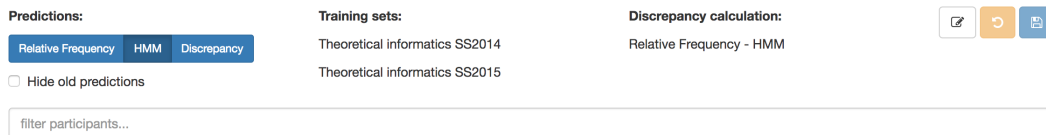


Figure 6.2: View control which visualises the current settings of the behaviour log and provides options to update the view.

The view control displays the behaviour log settings (figure 6.2). On the left the active prediction units of the current behaviour log are shown. At centre the histories respectively training data which are used to train the statistical models of the prediction units are represented by their unique name. The prediction units which are used for the discrepancy calculation are shown on the right.

Furthermore the user is able to select which predictions are visible in the students' behaviour table. On the one hand the prediction units can be chosen and on the other hand it is possible to decide whether to show the predictions of each assignment or only these of the future assignment ("hide old predictions"). Additionally the students' behaviour table can be filtered by the name of its students using the input field. Summarised, the view control offers possibilities to adjust the view of the students' behaviour table. In order to change the behaviour log itself the behaviour log control is used that can be opened by pressing the edit button in the right upper corner.

6.3 Behaviour Log Control

Figure 6.3 shows the 'Edit behaviour log options' dialog box. It includes the following settings:

- Select prediction units:** HMM (checked), Relative Frequency (checked)
- Select training sets:** Theoretical informatics SS2016 (unchecked), Theoretical informatics SS2014 (checked)
- Discrepancy:** Relative Frequency (selected), HMM (available)
- Predict complete time series:** (checked)
- Save observations as history:** history title (input field), [Save icon]

Figure 6.3: Possible behaviour log settings (Behaviour Log control)

The behaviour log control enables the functionality to change the behaviour log settings (figure 6.3). In this case changing one value automatically sends a server request and perhaps recalculates the predictions if they are affected.

All prediction units of this behaviour log are available for selection. The predictions for each activated prediction unit is calculated when updating the behaviour log. This means that it effects the duration of the server requests. The selection of the training set decides which histories are used for training the statistical model of all selected prediction units. Which histories are shown depends on the observation tag category. At least one history needs to be activated. The discrepancy calculation can be defined by selecting prediction units based on the active ones. The last checkbox determines if the complete time series should be predicted again when the behaviour log is updated. If not activated, only the future behaviour is predicted and earlier assignment predictions remain unchanged even if previous step behaviours have changed. In order to create a new history from the current behaviour log the input field on the bottom of the behaviour log control can be used. The new history is then available for selection in the training set list.

6.4 Behaviour Log Selection

Figure 6.4 shows the 'Active behaviour log' selection interface. It includes the following components:

- Active behaviour log:** A dropdown menu for selecting the current log.
- Skipping Prediction:** A button with a dropdown arrow.
- + Behaviour log:** A button to add new logs to the project.

Figure 6.4: Selection of the currently visible behaviour log.

Since each project is able to handle multiple behaviour logs, the behaviour log selection is needed (figure 6.4). The active behaviour log is set by a drop-down menu. By selecting a new one the view is updated instantly. Furthermore it is possible to add new behaviour logs to the project by clicking the “add” button. Figure 6.5 gives a complete overview about the user interface and its described components.

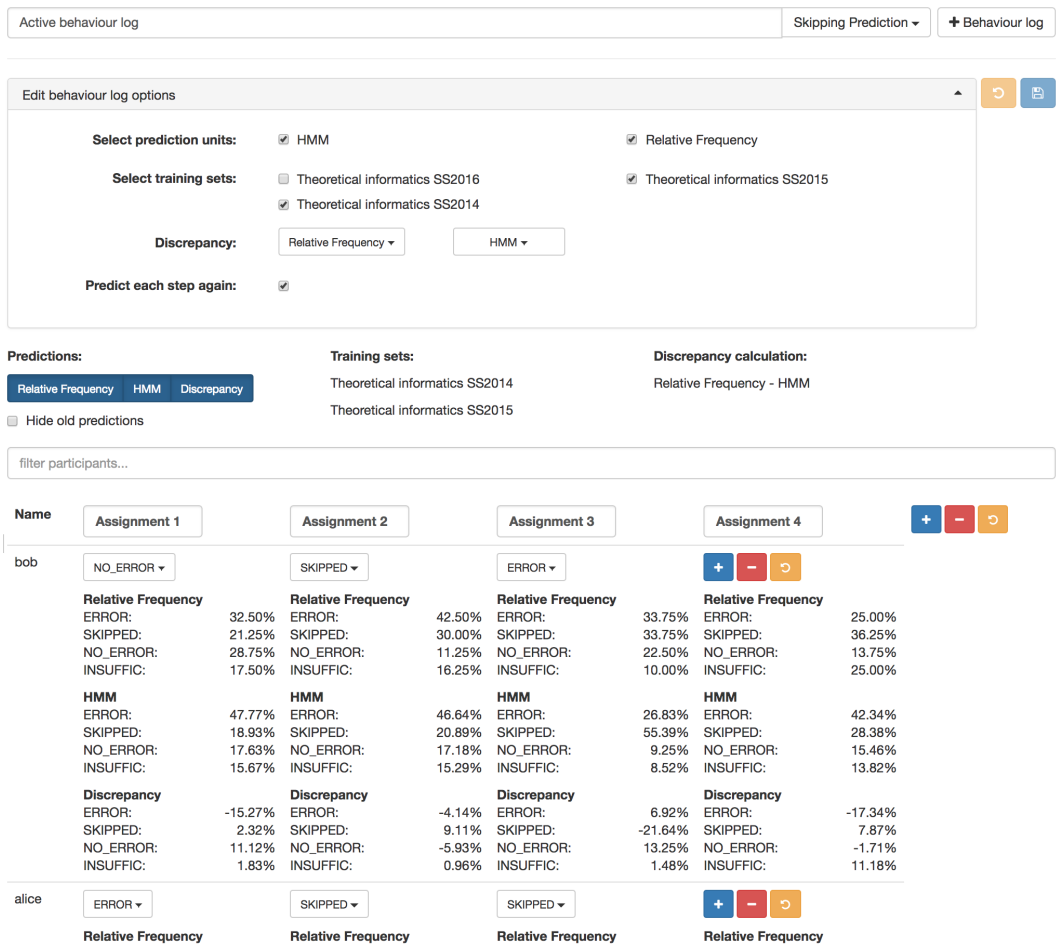


Figure 6.5: Screenshot of the complete user interface with expanded behaviour log control. All predictions are visible in the behaviour log table (HMM, Relative Frequency and Discrepancy).

In this chapter, the two implemented prediction units are evaluated. Each use a specific statistical model — on one hand the Relative Frequency Estimation (RFE) and on the other the Hidden Markov Model (HMM). The predictions of the Relative Frequency Estimation (RFE) and the Hidden Markov Model (HMM) are compared. The evaluation is performed by comparing the predictions of these two models. This makes sense because the RFE describes the behaviour for most of the students in a given situation while the HMM predicts the students' future behaviour based on their recent behaviour. As seen before in section 5.1, the prediction units provide a probability distribution of all tags instead of one specific tag. That is why the tag with the maximum value of this distribution is defined as prediction (or predicted tag) in this chapter.

Since there is only one course in which the ErrorState tags have been assigned, this data has to act as training data and test data (see chapter 3). This is done by splitting the data into subsets (one set used for training and the other for evaluating). In order to implement this evaluation, cross-validation is introduced in section 7.1. Furthermore this section explains the metrics used for measuring the prediction performance and how they have been applied. Section 7.2 points out the key results of these metrics. In section 7.3 these results are summarised and discussed.

7.1 Method

To evaluate the prediction performance different metrics are used. These metrics are applied on different parts of the test set. Cross-validation is used to receive several training and test sets. Three main aspects have been considered to evaluate the two implemented prediction units.

7.1.1 Metrics for Measuring Prediction Performance

There are two types of measurements which are distinguished in this thesis. On the one hand *tag independent measurements* and on the other hand *tag dependent measurements*. At both measurements the performance depends on the *predicted tag* and the *actual tag*. The following subsection is based on Powers (2011) and Fawcett (2006).

The tag independent measurements simply address the question if a prediction is correct or not (tag independent accuracy). This is done by comparing the actual tag with the predicted tag. If both tags are the same, the number of correct predictions is increased otherwise the number of incorrect predictions. The marginal probabilities are calculated as shown by equation 7.1 and 7.2.

$$\text{correct prediction} = \frac{|\text{Correct Predictions}|}{|\text{All Predictions}|} \quad (7.1)$$

$$\begin{aligned} \text{incorrect prediction} &= \frac{|\text{Incorrect Predictions}|}{|\text{All Predictions}|} \\ &= 1 - \text{correct prediction} \end{aligned} \quad (7.2)$$

Tag dependent measurements are calculated by considering exactly one tag type at a time. For better understandings the measurement is explained by the tag (classifier) “skipped”. This is reasonable since the main focus of this thesis is on predicting students’ skipping behaviour. But of course the classifier can be each possible tag. For the actual tag two classes $\{A^+, A^-\}$ are given, either the actual tag is positive (A^+ : *actual tag = skipped*) or it is negative (A^- : *actual tag \neq skipped*). Something similar applies to the predicted tag $\{P^+, P^-\}$. It also can be classified as positive regarding the tag “skipped” (P^+ : *predicted tag = skipped*) or negative (P^- : *predicted tag \neq skipped*). Given an actual tag and a predicted tag it is possible to assign the prediction to one of four groups/counts:

True positive (TP) Actual tag is positive (A^+) and predicted tag is positive (P^+)

True negative (TN) Actual tag is negative (A^-) and predicted tag is negative (P^-)

False positive (FP) Actual tag is negative (A^-) and predicted tag is positive (P^+)

False negative (FN) Actual tag is positive (A^+) and predicted tag is negative (P^-)

The “true values” (TP/TN) are good relating to predictions since they describe a correct predicted tag, while the “false values” (FP/FN) describe exactly the opposite of it. All four values are counts¹. So given a classifier (e.g. “skipped”) and a test set (actual tags and regarding predicted tags) a two-by-two *confusion matrix* can be constructed as visualised in figure 7.1. How the test sets are set up is shown in the next sections. Once the predictions are assigned to the different groups, and thus the matrix is build, many common metrics can be calculated from it to evaluate the prediction performance. All these common metrics provide a “joint and marginal probability” (Powers, 2011). These metrics are shown in figure 7.1 and can be described as follows:

¹The UPPER CASE variables describe counts and the lower case variables describe proportions relative to N (number of all predictions) or the marginal probabilities

Classifier (e.g. skipped)		Actual Tag		
		Total $N = A^+ + A^-$ $= P^+ + P^-$	Actual Positive $A^+ = TP + FN$	Actual Negative $A^- = FP + TN$
Predicted Tag	Prediction Positive $P^+ = TP + FP$	True Positive (TP)	False Positive (FP)	$accuracy = \frac{ TP + TN }{ N }$ $precision = \frac{ TP }{ P^+ }$
	Prediction Negative $P^- = FN + TN$	False Negative (FN)	True Negative (TN)	
		$tp\ rate = \frac{ TP }{ A^+ }$ $fn\ rate = \frac{ FN }{ A^+ }$	$fp\ rate = \frac{ FP }{ A^- }$ $tn\ rate = \frac{ TN }{ A^- }$	

Figure 7.1: Confusion matrix (also called a contingency table) and common performance metrics calculations (Source: adapted from Fawcett (2006))

True positive rate (tp rate) / sensitivity / recall / hit rate The *true positive rate* is the proportion of Actual Positive (A^+) cases which are correctly Predicted Positive (P^+). It describes how many Actual Positive cases are actually being detected.

False negative rate (fn rate) / miss rate The *false negative rate* is the proportion of Actual Positive (A^+) cases which are incorrectly Predicted Negative (P^-). It describes how many Actual Positive cases are not being detected.

True negative rate (tn rate) / specificity / inverse recall The *true negative rate* is the proportion of Actual Negative (A^-) cases which are correctly Predicted Negative (P^-). It describes how many Actual Negative cases are actually being detected.

False positive rate (fp rate) / false alarm rate / fallout The *false positive rate* is the proportion of Actual Negative (A^-) cases which are incorrectly Predicted Positive (P^+). It describes how many Actual Negative cases are not being detected.

Precision / positive predictive value The *precision* is the proportion of Predicted Positive (P^+) cases which are correctly Actual Positives (A^+). It describes the accuracy of Predicted Positives regarding the Actual Positives.

Accuracy The *accuracy* is the proportion of correct predictions (TP+TN) regarding all predictions (N).

F1-score / F-measure The *f1-score* is a combined metric that considers the precision and the recall. It is the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (7.3)$$

The importance and relevance of these metrics to evaluate the prediction performance regarding students' skipping behaviour is focused in section 7.2. But first it is shown how the test sets are built and on which data the metrics are applied.

7.1.2 Cross-Validation

If tagged data would be available for several courses, one or more courses could be used to train our model and one other course could be used to evaluate the performance of this model. Unfortunately at the time of this report there was only one tagged data set available. Therefore *K-fold cross-validation* is taken to use one "part of the available data to fit the model, and a different part to test it" (Hastie et al., 2009, 241ff.). *K-fold cross-validation* splits the data into "*K* roughly equal-sized parts". The length of the whole data set is defined as *N*. One of the splitted parts is tested while the other *K* − 1 parts train the model. The part which is used as test data is defined as *k*th part. This is done *K* times while *k* is increased per round (*k* = 1, 2, ..., *K*). In each of these rounds the previously described metrics are applied and their results are combined to the results of the steps before. The case where *K* = *N* is called *leave-one-out cross-validation*. Hereby exactly one entry of the data set is used as test set while all remaining entries are used to set up the prediction model.

The tagged data set consists of 82 entries, which is already pretty small to train a prediction model — even without leaving entries out to take them as test data. Therefore *leave-one-out cross-validation* is applied to train the model with as much entries as possible.

During each cross-validation step each subsequence of the current entry has to be predicted and evaluated. These subsequences always start at assignment/step 1 which means for an exemplary sequence

"no_error", "error", "error", "skipped"

that the evaluation performance of the following subsequences has to be measured:

Step 1 "no_error"

Step 2 "no_error", "error"

Step 3 "no_error", "error", "error"

Step 4 "no_error", "error", "error", "skipped"

For each of these sequences the grouping of subsection 7.1.1 is applied so that the confusion matrix is built up to calculate the metrics. This has to be done for the both implemented prediction units independently — Relative Frequency Estimation and Hidden Markov Model.

Besides a composed evaluation of all steps, each step can be evaluated separately. This is useful since the students' recent behaviour and the passed duration of the course have an impact on the predictions and thus on the prediction performance. So it can be assumed that the HMM predictions of step 1 are less accurate than the predictions of step 6, since there is no recent behaviour given before step 1 and thus the personal characteristics are missing. Additionally due to the trend of skipping during a running course an uneven distribution is given towards the end of the course which could distort the predictions as

well. To evaluate these assumptions the prediction performance of step 1, 6 and 11² are examined.

7.1.3 Evaluation Aspects

The evaluation of the two implemented prediction units is done considering three different aspects:

1. Comparison of Hidden Markov Model and Relative Frequency Estimation regarding tag independent measurements.
 - a. Considering the best prediction.
 - b. Considering the two best predictions.
2. Comparison of Hidden Markov Model and Relative Frequency Estimation regarding tag dependent measurements.
 - a. Step-based evaluation
 - b. Composed evaluation for all steps
3. How important are the different classifications? Which influence has the tag “insufficient knowledge” when predicting the students’ skipping behaviour?

7.2 Results

This section shows the results received by performing the cross-validation on the given data set. It is separated into the different evaluation aspects.

7.2.1 Tag Independent Evaluation

The tag independent evaluation describes the tag independent accuracy. Here the evaluation is done for all steps composed which means that there is no evaluation per step. Using cross-validation 902 predictions are made. Each prediction unit is assessed by counting the correct — respectively incorrect — prediction regarding the actual tag. This is done in two different ways. For one thing by considering only the tag with the maximum value out of the probability distribution provided by the prediction unit (*best prediction*, table 7.1), for another thing by considering the two highest tags regarding their probability (*two best predictions*, table 7.2).

<i>Best prediction (N = 902)</i>	Correct Predictions	Incorrect Predictions
Relative Frequency Estimation	0.45898 (414)	0.54102 (488)
Hidden Markov Model	0.60643 (547)	0.39357 (355)

Table 7.1: Results of tag independent measurements for the best prediction.

²last step of the given data set

<i>Two best predictions (N = 902)</i>	Correct Predictions	Incorrect Predictions
Relative Frequency Estimation	0.77384 (698)	0.22616 (204)
Hidden Markov Model	0.78271 (706)	0.21729 (196)

Table 7.2: Results of tag independent measurements for the two best predictions.

For the best prediction the correct prediction accuracy is relatively low for both prediction units (RFE: 46% vs. HMM: 61%), but at least the HMM provides better results. By considering the best two predictions the accuracies are almost equal ($\sim 78\%$).

7.2.2 Tag Dependent Evaluation

This subsection treats the metrics based on the different tags. The focus is on the tag “skipped”. Each tag can be considered step-based or as a combination of all steps. The step-based evaluation is done for the first step (step 1), one in the middle (step 6) and the last (step 11).

Step-based evaluation As represented in table 7.3, the step-based results of the Relative Frequency Estimation are very inaccurate. Due to the statistical model the likeliest tag is always the same at a specific step. This means that predicting the tag “skipped” is very likely because of the students’ propensity to skip. As a consequence, the sensitivity and specificity — and with it also the miss rate and false alarm rate³ — only are one or zero. Through this characteristic the step-based results for the RFE are useless.

As expected, step 1 of the Hidden Markov Model is also very inaccurate since there is no recent behaviour available at this time. Step 6 and 11 are very similar although step 11 is more accurate due to the skipping trend. But both have a high precision and accuracy as well as an associated f1-score ($\geq 75\%$).

Step-based "skipped"	RFE			HMM		
	Step 1	Step 6	Step 11	Step 1	Step 6	Step 11
<i>sensitivity</i>	0.0	1.0	1.0	0.0	0.7895	0.7869
<i>specificity</i>	1.0	0.0	0.0	1.0	0.7727	0.8095
<i>miss rate</i>	1.0	0.0	0.0	1.0	0.2105	0.2131
<i>false alarm rate</i>	0.0	1.0	1.0	0.0	0.2273	0.1905
<i>precision</i>	0.0	0.4634	0.7439	0.0	0.75	0.9231
<i>accuracy</i>	0.7927	0.4634	0.7439	0.7927	0.7805	0.7927
<i>f1-score</i>	0.0	0.6333	0.8531	0.0	0.7692	0.8496

Table 7.3: Results depending on tag “skipped” based on the steps 1, 6 and 11.

³Reminder: $miss\ rate = 1 - sensitivity$ and $false\ alarm\ rate = 1 - specificity$

Composed evaluation of all steps Table 7.4 shows the composed results of all steps and all described metrics for the tag “skipped”. As already said, the sensitivity of the Relative Frequency Estimation is comparatively high (83%) for predicting the tag “skipped”, since most of the students tend to skip during the running course. In contrast to that the specificity is very low (25%) — or the false alarm rate very high (75%) — which makes sense considering that nearly each step predicts skipping. This explains the moderate precision and accuracy of about 50%.

Although the Hidden Markov Model provides better overall performances. Even if the sensitivity is lower than the sensitivity of the RFE the specificity is sharply higher. This has an impact on the precision and accuracy which are both about 80%.

<i>All steps "skipped"</i>	RFE	HMM
<i>sensitivity</i>	0.8341	0.7299
<i>specificity</i>	0.2542	0.8479
<i>miss rate</i>	0.1659	0.2701
<i>false alarm rate</i>	0.7458	0.1521
<i>precision</i>	0.4958	0.8084
<i>accuracy</i>	0.5255	0.7927
<i>f1-score</i>	0.6219	0.7671

Table 7.4: Composed results of all steps depending on tag “skipped”.

7.2.3 Classification Evaluation

So far only the tag “skipped” has been considered. Now the other tags are taken into account. This is done by comparing the performance results of the Hidden Markov Model. As seen in table 7.5, besides the tag “skipped” only “error” provides relevant values. The reason for this is that “no error” and “insufficient knowledge” are never predicted using cross-validation and the given data set. This also has an influence on the results of the tag “error” by which the false alarm rate is higher (46%) than for the tag “skipped”.

<i>All steps All tags</i>	Hidden Markov Model			
	skipped	error	no error	insufficient knowledge
<i>sensitivity</i>	0.7299	0.8241	0.0	0.0
<i>specificity</i>	0.8479	0.5392	1.0	1.0
<i>miss rate</i>	0.2701	0.1759	1.0	1.0
<i>false alarm rate</i>	0.1521	0.4608	0.0	0.0
<i>precision</i>	0.8084	0.4587	0.0	0.0
<i>accuracy</i>	0.7927	0.6308	0.8891	0.9002
<i>f1-score</i>	0.7671	0.5894	0.0	0.0

Table 7.5: Composed results of all steps visualised for each tags.

7.3 Discussion

The results using a Hidden Markov Model are valuable for the tag “skipped” which means the students’ skipping behaviour can be predicted relatively accurate. Step-based evaluation is senseless using Relative Frequency Estimation because the students’ recent behaviour has no influence and thus predictions are identical for all students at each step. However the RFE describes the “default behaviour” which most students showed in a given situation. This means that it is a good comparative measurement that can perfectly be used for comparing with another statistical model using the discrepancy function of the prediction framework.

For both units the tags “no error” and “insufficient knowledge” have less influence, since they are not very common to be predicted — or to be precise they are never predicted using cross-validation and the given data set. This means that the most likely predictions are more accurate without these tags. As a consequence, the assignments could also be classified with “submitted” (including “no error”, “error” and “insufficient knowledge”) and “not submitted” (or “skipped”). Table 7.6 represents the results of this classification using Hidden Markov Model. It shows that the correct predictions rise to 78% for tag independent evaluation. For tag dependent evaluation with combined steps the “not submitted” values are nearly the same compared to the results of the tag “skipped” in table 7.5. Additionally the “submitted” results are better compared with the results of the tag “error” (e.g. precision 80% vs. 46%). Due to these results (based on one course and the most likely prediction) it can be said that it is not necessary to assign detailed tags to predict the skipping behaviour of the student. But indeed the detailed classification might be important when considering the two best predictions or even the complete probability distribution.

<i>Submitted / Not Submitted</i>	Hidden Markov Model	
	<i>Tag independent</i>	
<i>Correct predictions</i>	0.7827 (706/902)	
<i>Incorrect predictions</i>	0.2173 (196/902)	
<i>All steps combined</i>	<i>Tag dependent</i>	
	Submitted	Not Submitted
<i>sensitivity</i>	0.8479	0.7085
<i>specificity</i>	0.7085	0.8479
<i>miss rate</i>	0.1521	0.2915
<i>false alarm rate</i>	0.2915	0.1521
<i>precision</i>	0.7679	0.8038
<i>accuracy</i>	0.7827	0.7827
<i>f1-score</i>	0.8059	0.7531

Table 7.6: Hidden Markov Model results of tag independent and tag dependent evaluations using the tags “submitted” and “not submitted”

Summarised, the evaluation shows that the Hidden Markov Model provides good results when predicting the skipping behaviour of students with the given data set. It respects the students' recent behaviour whereas the Relative Frequency Estimation supplies the same predictions for each student in a specific step.

CHAPTER 8

Conclusion

This work addresses the problem that students of a course tend to skip their weekly work assignments as time goes on. To prevent the students from doing this it was shown that it is possible to predict this behaviour to a certain degree. This was done by introducing a classification scheme to tag the students' submissions. The tags of this classification are: "no error", "error", "insufficient knowledge" and "skipped". Based on these tags statistical models were investigated which also consider personal characteristics of students. As a result of this research, the simple Relative Frequency Estimation, which describes the students' "default behaviour" in a given situation, and the more detailed Hidden Markov Model, which respects the students' recent behaviour, have been found. This information was used to design and implement a prediction framework which is independent of the classification and statistical model, so that it can be applied for other use cases as well. The main parts of this framework are the prediction units, which are responsible for the predictions, and the behaviour logs, which provide several functions to interact with the prediction units. To predict the students' skipping behaviour regarding weekly work assignments the framework was integrated into an existing system whereby the usage was demonstrated. Furthermore it was shown how the framework can be extended with additional tags and prediction units to cover further use cases. On top of this, a user interface, which is usable for varying kinds of predictions, was conceptualised and its usage described. The predictions have been evaluated regarding different aspects.

The evaluation has shown that the Hidden Markov Model results in good predictions regarding skipping, whereas the Relative Frequency Estimation provides good comparative measurements. A further result is that the given classification has less influence when using these statistical models and the given test data, since the tags "no error" and "insufficient knowledge" are never predicted. Under these circumstances a less detailed classification with tags such as "submitted" and "not submitted" would be enough to obtain predictions of a similar quality — or even better ones. Through this knowledge further

test data could be exported from existing systems (e.g. UniWorX¹) and act as training sets for the statistical models, since such a system already maintains information about those submitting tags.

Anyway the work on the more detailed classification should be continued as well. Perhaps other statistical models or further training sets would provide better predictions for the more detailed tags. Also the predictions could be improved by comparing the predicted behaviour with the real future behaviour of the students during a running course. Furthermore providing information to students about their future behaviour could counteract the actual one which in turn could change the predictions so that the tags “no error” and “insufficient knowledge” become important again.

It will be interesting to see if and how the students can be incited to a sufficient work level by the predictions. Thereto different approaches of feedforward need to be tested. This could be realised with an A/B test where two groups are introduced during a course. One test group in which the students receive information about their possible future behaviour (feedforward) and one control group which receive nothing. In this way a comparison is possible whether the feedforward has an impact or not. In case of a high likelihood in the prediction the system could automatically produce feedforward for the student. Additionally an indicator could be introduced which describes how strongly the student has been influenced by this information. Thus, it could be evaluated if the students’ exam mark correlates with this indicator. Lastly, the reaction by the student to the feedforward could be used for further training of the prediction unit. This approach could result in personalised predictors if this is striven for.

Further statistical models and other classifications should be implemented and tested. The idea here is to implement good and accurate predictors. Using these predictors to inform the students about their possible future behaviour and thus to have an impact on them the predictions should not come true if the student counteracts against the prediction. As a consequence, the predictors in a running course would turn bad. This in turn could be used to motivate the student by telling him that he has tricked the system.

As a conclusion, it can be stressed that predicting the human behaviour is a very interesting and up-to-date topic which not only concerns the university. In almost every sector one tries to predict the future. For example, many companies try to predict the behaviour of their customers. And in the end both sides benefit from these predictions. So, for example, if a supermarket can predict the rush hours, the customers can react to these predictions and change their shopping behaviour to have a more relaxed shopping experience. Besides, the supermarket can do their staffing based on the predictions.

Anyway, since it is not possible for the professors and tutors to take care of each student individually, the problem needs to be abstracted, so that both sides benefit. On the one hand the professors and tutors are able to educate the students in the best possible way, on the other hand the students are prevented from cramming and they are perfectly prepared for their professional life. In the end, this is a win-win situation for both sides.

¹UniWorX: <https://uniworx.ifi.lmu.de/>

Appendix

The source code of the implemented software is available at:
<https://gitlab.pms.ifi.lmu.de/niels/cwdl-projects>
on branch *behaviourlog* (Last commit: c5b76f15).

Bibliography

Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, pages 121–130, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3630-7. doi: 10.1145/2787622.2787717. URL <http://doi.acm.org/10.1145/2787622.2787717>.

Sattar Ameri, Mahtab J. Fard, Ratna B. Chinnam, and Chandan K. Reddy. Survival analysis based framework for early prediction of student dropouts. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 903–912, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4073-1. doi: 10.1145/2983323.2983351. URL <http://doi.acm.org/10.1145/2983323.2983351>.

Angular JS, 2017. URL <https://angularjs.org/>. [Online: accessed 23-July-2017].

L. Aulck, N. Velagapudi, J. Blumenstock, and J. West. Predicting Student Dropout in Higher Education. *ArXiv e-prints*, June 2016.

Girish Balakrishnan. Predicting student retention in massive open online courses using hidden markov models. Master's thesis, EECS Department, University of California, Berkeley, May 2013. URL <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-109.html>.

Demis Basso and Marta Olivetti Belardinelli. The role of the feedforward paradigm in cognitive psychology. *Cognitive Processing*, 7(2):73–88, 2006. ISSN 1612-4790. doi: 10.1007/s10339-006-0034-1. URL <http://dx.doi.org/10.1007/s10339-006-0034-1>.

Bootstrap, 2017. URL <http://getbootstrap.com/>. [Online: accessed 23-July-2017].

François Bry and Niels Heller. Research project: Backstage 2. personal communication, 2017.

- François Bry and Alexander Pohl. Backstage: A digital backchannel for large class lectures, 2009. URL <http://www.en.pms.ifi.lmu.de/research/backchannels/index.html>. [Online: accessed 23-July-2017].
- BSON - Binary JSON, 2017. URL <http://bsonspec.org/>. [Online: accessed 23-July-2017].
- Lloyd (lloydmeta) Chan. Enumeratum, 2017. URL <https://github.com/lloydmeta/enumeratum>. [Online: accessed 23-July-2017].
- Richard Dallaway. Enumeratum, 2014. URL <http://underscore.io/blog/posts/2014/09/03/enumerations.html>. [Online: accessed 23-July-2017].
- Peter W. Dowrick. Self model theory: learning from the future. *Wiley Interdisciplinary Reviews: Cognitive Science*, 3(2):215–230, 2012. ISSN 1939-5086. doi: 10.1002/wcs.1156. URL <http://dx.doi.org/10.1002/wcs.1156>.
- Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006. ISSN 0167-8655. doi: 10.1016/j.patrec.2005.10.010. URL <http://dx.doi.org/10.1016/j.patrec.2005.10.010>.
- S. Halawa, D. Greene, and J. Mitchell. Dropout prediction in MOOCs using learner activity features. *eLearning Papers*, 37, March 2014.
- Mark Harrison. Cake pattern, 2011. URL <http://www.cakesolutions.net/teamblogs/2011/12/19/cake-pattern-in-depth>. [Online: accessed 23-July-2017].
- Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, New York, 2009. ISBN 978-0-387-84857-0. doi: 10.1007/978-0-387-84858-7.
- Shelby H. McIntyre and J. Michael Munson. Exploring cramming. *Journal of Marketing Education*, 30(3):226–243, 2008. doi: 10.1177/0273475308321819. URL <http://dx.doi.org/10.1177/0273475308321819>.
- MongoDB, 2017. URL <https://www.mongodb.com>. [Online: accessed 23-July-2017].
- Tomas Obsivac, Lubos Popelínský, Jaroslav Bayer, Jan Geryk, and Hana Bydzovská. Predicting drop-out from social behaviour of students. In Kalina Yacef, Osmar R. Zañe, Arnon HersHKovitz, Michael Yudelson, and John C. Stamper, editors, *EDM*, pages 103–109. www.educationaldatamining.org, 2012. ISBN 978-1-74210-276-4. URL <http://dblp.uni-trier.de/db/conf/edm/edm2012.html#ObsivacPBGB12>.
- Play Framework, 2017. URL <https://www.playframework.com/>. [Online: accessed 23-July-2017].
- D. M. W. Powers. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

- Jiezhong Qiu, Jie Tang, Tracy Xiao Liu, Jie Gong, Chenhui Zhang, Qian Zhang, and Yufei Xue. Modeling and predicting learning behavior in moocs. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining, WSDM '16*, pages 93–102, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3716-8. doi: 10.1145/2835776.2835842. URL <http://doi.acm.org/10.1145/2835776.2835842>.
- Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *PROCEEDINGS OF THE IEEE*, pages 257–286, 1989.
- Hendrik Radatz. Error analysis in mathematics education. *Journal for Research in Mathematics Education*, 10(3):163–172, 1979. ISSN 00218251, 19452306. URL <http://www.jstor.org/stable/748804>.
- Pedro Rijo. Scala enumerations, 2016. URL <http://pedrorijo.com/blog/scala-enums/>. [Online: accessed 23-July-2017].
- Henry L Roediger and Jeffrey D Karpicke. Test-enhanced learning taking memory tests improves long-term retention. *Psychological Science*, 17(3):249–255, 2006.
- Alina Selyukh. Time management, avoiding procrastination key in evading end-of-semester stress, 2007. URL http://www.dailynebraskan.com/time-management-avoiding-procrastination-key-in-evading-end-of-semester/article_14c07b5d-a795-5691-aeaf-74bb72c7eb94.html. [Online: accessed 23-July-2017].
- Smile - Statistical Machine Intelligence and Learning Engine, 2017. URL <http://haifengl.github.io/smile>. [Online: accessed 23-July-2017].
- The Scala Programming Language, 2017. URL <https://www.scala-lang.org/>. [Online: accessed 23-July-2017].
- UniWorX, 2017. URL <https://uniworx.ifi.lmu.de/>. [Online: accessed 23-July-2017].
- Weka: Data Mining Software in Java, 2017. URL <http://www.cs.waikato.ac.nz/~ml/weka/>. [Online: accessed 23-July-2017].
- S. K. Yadav, B. Bharadwaj, and S. Pal. Data Mining Applications: A comparative Study for Predicting Student’s performance. *ArXiv e-prints*, February 2012.