INSTITUT FÜR INFORMATIK
der Ludwig-Maximilians-Universität München

Predicting Students' Assignment Performance to Personalize Blended Learning

Andreas Born

Zulassungsarbeit

Aufgabensteller	Prof. Dr. François Bry
Betreuer	Prof. Dr. François Bry, Niels Heller
Abgabe am	1. Oktober 2017

Erklärung

Hiermit erkläre ich, dass die vorliegende Hausarbeit von mir selbstständig verfasst wurde und dass keine anderen als die angegebenen Hilfsmittel benutzt wurden. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder Sinn nach entnommen sind, sind in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht.
Diese Erklärung erstreckt sich auch auf etwa in der Arbeit enthaltene Zeichnungen, Kartenskizzen und bildliche Darstellungen.

München, den 1. Oktober 2017 Andreas Born

Abstract

In large university courses, and particularly in Massive Open Online Courses (MOOCs), interventions to prevent students' disengagement, and eventually dropout, often conflict with limited resources, as compared to the number of participating students. In previous research many prediction models have been proposed to identify students at risk of dropout at an earlier stage. However, just the identification of at-risk students is not enough. Effectively, dropout and disengagement of students should be prevented in the first place. This thesis presents a prediction model, which uses multiple, differentiated labels to describe the assignment performance of students. For that purpose an existing dataset from an introductory lecture of theoretical informatics was used. The course required students to submit weekly assignments, which were assessed and with feedback returned to the students. For the dataset this process was extended for 82 randomly selected students and each assignment was additionally tagged out of a total 27 labels. Generally, more commonly given feedback has been coded into labels. The simplest label is just "good". In the opposite case of "bad" exercises more differentiated labels have been selected, e.g. "Understanding of implications insufficient" or "Syntax for grammars invalid".

To provide directions to students based on anticipated labels, this thesis presents a prediction model, which has been inspired by recommender systems and relies on Collaborative Filtering. More specifically, a distributed implementation of the Alternating Least Squares method with weighted-λ-regularization (ALS-WR) has been selected. For every student, assignment and exercise this results in a predicted (confidence) rating of their co-occurrence. The evaluation relies on ten-fold Cross-Validation and a variety of metrics to account for top-k-recommendation and binary label prediction use cases. After identifying suitable hyperparameters for ALS-WR using a grid-based method the model obtained results that seem feasible for future applications.

These applications have been prepared by integrating the prediction model with the *Backstage 2* learning environment developed at the Ludwig-Maximilians-Universität München. This implementation was facilitated by the prediction framework developed with the earlier master thesis of Steven Dostert. In the future *Backstage 2* could be used to either directly communicate prediction results to students or to otherwise indirectly personalize their learning experience based on predicted labels. In their discussion these applications will be subsumed as feedback, or more specifically Feed Forward. Lastly, four criteria for the effectiveness of these Feed Forward applications are proposed based on research about recommender systems and effective feedback.

Zusammenfassung

In großen Vorlesungen, und insbesondere in “Massive Open Online Courses” (MOOCs), scheitern Maßnahmen, um das Zurückfallen, und letztendlich den Kursabbruch, von Studenten zu verhindern, oft an den begrenzten Mitteln im Vergleich zur Teilnehmerzahl. In früherer Forschung wurden viele Vorhersagemodelle vorgeschlagen, um Studenten, bei denen die Gefahr eines Kursabbruchs besteht, frühzeitiger zu erkennen. Jedoch ist das bloße Erkennen gefährdeter Studenten nicht ausreichend. Sinnvollerweise sollte der Kursabbruch und das Zurückfallen von Studenten von vorneherein verhindert werden. Diese Zulassungsarbeit stellt ein Vorhersagemodell vor, das eine Vielzahl differenzierter Label nutzt, um die Leistung der Studenten in Übungsaufgaben zu beschreiben. Für diesen Zweck wird ein existierender Datensatz aus einer Einführungsvorlesung in die Theoretische Informatik verwendet. In diesem Kurs mussten Studenten wöchentliche Übungsblätter abgeben, welche beurteilt und mit Rückmeldungen versehen an die Studenten zurückgingen. Für den Datensatz wurde dieser Ablauf für 82 zufällig ausgewählte Studenten ergänzt und jede Abgabe zusätzlich mit einem oder mehreren von insgesamt 27 Labels versehen. Im Allgemeinen wurden häufiger vergebene Rückmeldungen durch Labels kodiert. Hierbei ist das einfachste Label schlicht “gut”. Im gegenteiligen Fall “schlechter” Abgaben wurden differenziertere Label ausgewählt, z.B. “Implikationen nicht richtig verstanden” oder “Syntax von Grammatiken nicht eingehalten”.

Um den Studenten zielführende Hinweise basierend auf ihren zu erwartenden Labels zu geben, wird mit dieser Arbeit ein Vorhersagemodell vorgestellt, das von Empfehlungssystemen inspiriert wurde und auf Collaborative Filtering setzt. Dafür wurde eine verteilte Implementierung der Methode alternierender kleinster Quadrate mit gewichteter λ-Regularisierung (ALS-WR) ausgewählt. Für jeden Studenten, jedes Übungsblatt und jede Aufgabe ergibt sich dabei eine vorhergesagte Bewertung ihres gemeinsamen Auftretens. Deren Auswertung beruht auf 10-facher Kreuzvalidierung und einer Vielfalt von Metriken, um die Anwendungsfälle Top-k-Empfehlungen und binäre Labelvorhersage zu berücksichtigen. Nach der Bestimmung geeigneter Hyperparameter für ALS-WR mit einer Rastermethode erreichte das Modell Ergebnisse, die für zukünftige Anwendungen praktikabel erscheinen.

Solche Anwendungen wurden durch die Einbindung des Vorhersagemodells in die an der Ludwig-Maximilians-Universität München entwickelte “*Backstage 2*”-Lernumgebung vorbereitet. Diese Implementierung wurde durch ein Vorhersage-Framework erleichtert, das in der früheren Masterarbeit von Steven Dostert entwickelt worden ist. Zukünftig könnte *Backstage 2* genutzt werden, um Studenten die Vorhersageergebnisse entweder direkt zu kommunizieren oder andernfalls indirekt ihr Lernerlebnis basierend auf vorhergesagten Labels zu personalisieren. In der Diskussion dieser Anwendung werden diese als Rückmeldung, oder genauer gesagt als Feed Forward, subsumiert. Zuletzt werden vier Kriterien für die Effektivität solcher Feed Forwards anhand von Forschung über Empfehlungssysteme und effektives Feedback aufgestellt.

Acknowledgements

Writing this thesis has necessarily not always been straightforward and the result – which you probably plan on reading right now – would not have been the same without the constant support and valuable feedback of my two mentors Prof. Dr. Bry and Niels Heller. Niels furthermore labeled students' feedback in countless hours of tedious work and originally suggested to use some Collaborative Filtering method on this dataset. Every result presented and the topic of this thesis only exist because of him. Prof. Dr. Bry shined with his always to the point criticism, which particularly helped to forge a meaningful title and concise argumentation throughout the following chapters. Both Prof. Dr. Bry and Niels have my deepest gratitude.

Apart from them, I want to explicitly thank my family, who always had an open ear for my thoughts and kept me going at all times. At the busiest times Martina, my sister, boosted my motivation with unexpected, homemade delicacies, that could not have been any better. Many more like my grandma, who celebrated her birthday during the final weeks of writing, would deserve mention. Instead they should know that I am gifted to have them for family.

In many discussions not just with my family but also friends my ideas were reflected allowing them to develop into this thesis. Out of those I want to particularly thank my good friends Fabian, with whom I shared many desk during school, and Bianca, who I have known for my whole university life. They were a tremendous help in the final proofreading. It is a privilege to know both of them as my friends.

Finally, only two words remain to be said to everyone mentioned and unmentioned:

THANK YOU!

Contents

CHAPTER 1

Introduction

This thesis aims to contribute to the development of personalized blended learning environments in order to facilitate university students' learning and to prevent dropout from lectures. In education research and debate the individual needs of students and adapting to them has received increasing attention over the last decades. Unsurprisingly, expectations are that information and communications technology (ICT) could solve this issue of personalization e.g. through "just-in-time' support" [32, p. 40]. However, also for blended learning environments, which blend ICT and face-to-face education [28], personalization poses a major challenge [6, 32, 37].

Unlike a human instructor, who sleeps, eats or gets sick, a sophisticated computer system could theoretically provide close to uninterrupted tutorship even to hundreds of students simultaneously. Practically however, in the case of a human and a computer the balance between the benefit for every individual learner and the cost of personalization must be considered. Indeed, developing and especially maintaining personalization software also depends on human resources like time and experience. For that reason, personalization – even using ICT – is still limited by the access to human experience.

While advances in information technology have not readily been able to generate experience, storage of and ubiquitous access to all types of information has become feasible in recent years [10]. Nowadays many online platforms apply this data in computer recommendation systems aiming to substitute human peer review, of movies for instance, by computer-generated recommendations [33, pp. 4–20]. Like education this is just another context for personalization. Often these recommender systems still require explicit user feedback and do not rely on implicit feedback derived from behaviour such as "time spent watching ..." for instance. Particularly, users need to express their individual preferences of certain products over others, usually called *rating* [3, p. 1]. Subsequently, a compilation of these preferences is digitally stored thus granting everybody access to any preference ever expressed by anybody within the scope of the system.

1.1 Research aims

The proposed prediction model relies on a compilation of preferences as well, and yields predictions for students' assignment performance in large university courses. However, in our application preferences are not expressed by the students themselves. Instead we rely on correctors – since they have more experience than students – to assess and label students' assignment performance. If peer review were used instead, it is challenging to guarantee effective feedback [26, 31] and a consistent application of labels, which is especially important to obtain meaningful predictions.

Foremost, the assessments provide students with differentiated feedback of individual issues in the assignment and conceptual problems. Simultaneously, typical feedback instances are coded into *labels*, which in their combination are considered as assignment performance by the proposed prediction model. For example "good" is the label used most often. In the opposite case of "bad" exercises more differentiated labels describe the specific issue. Those are for instance "Understanding of implications insufficient" or "Syntax for grammars invalid". Chapter 2 will give a more comprehensive overview of labels complemented with an analysis.

For now, the next step is to view predictions as the problem of recommending labels. Much similar to product recommendations, the existence of underlying causes, that is latent factors, which determine product choice or assignment performance, is assumed. More specifically, a Collaborative Filtering algorithm based on matrix factorization using the Alternating Least Squares method with weighted-λ-regularization (ALS-WR) [5, 30, 44] is employed. This results in predictions for labels representing students' assignment performance and ordered by a non-probabilistic confidence rating. Our primary intention is to gain usable hints about individual students needs by anticipating their future behaviour. As one possible application, interventions revolving around the aim to counterbalance students' misconceptions can be implemented. Since labels can manifest differently in followup assignments or connections between labels might exist, the anticipated behaviour, as opposed to past observations, is more fit for that application. That is, while the same application is feasible using past observations, by interleaving a prediction layer the notion of labels becomes more that of "Where to next?" instead of just "How am I going?" [26, p. 87]. This is desirable because the question "How am I going?" is already answered by the assessment of students' assignments.

Therefore, this thesis addresses the development of personalized blended learning environments through three aims. (1) The main focus is to develop and evaluate the aforementioned prediction model. In that course possible alternative machine-learning approaches and the possible effects of different data representations are mentioned. Chapter 2 and particularly chapter 3 focus this part of the thesis. (2) To prepare for future applications a working reference implementation has been integrated with *Backstage 2* an online learning platform for university courses developed by the Teaching and Research Unit Programming and Modelling Languages at the Ludwig-Maximilians-Universität München. In a previous master thesis Steven Dostert developed a generic prediction framework [13] for *Backstage 2*, which served as the foundation. Hence chapter 4 summarizes the implementation of the prediction model with a focus on the enhancements to the framework. (3) Lastly in chapter 5 possible future applications are discussed. For the resulting predictions different transformations and thus eventually different applications are conceivable. One such application has already been sketched and will be built on in chapter 5 with the final discussion.

1.2 Background

For decades now we have seen a substantial rise of everyday information technology. Lately, innovations in this field have especially been driven by applications of machine learning and data mining supported by the historically unprecedented ubiquity of data on almost every aspect of our daily life. While many problems have been solved by ICT, on many other occasions this has not (yet) been possible. This sparked research on human computation, whose applications often use gamification elements for incentive [21].

In education many problems have remained difficult to solve solely by technology. Instead many instances of blended learning have been developed, which is commonly defined as the "integration of online and face-to-face strategies" [28]. At the Ludwig-Maximilians-Universität München *Backstage* has been one such research project, where a backchannel communication platform for presence courses at universities has been conceived, implemented and evaluated. Experiences from the evaluation of *Backstage* [9] eventually lead to the development of the learning platform *Backstage 2*, which also constitutes the context of this thesis.

In the sense of blended learning, the proposed prediction model aims to combine the best of both worlds by using humans for the assessment and technology for the predictions. While applications with computer-assisted feedback have been available for a while, they do not cover any type of exercises. The dataset used throughout this thesis has been collected from an introductory lecture about theoretical informatics. Therefore a major aim is to convey fundamental concepts and notations to students. However, the application of automated proof checking, for example, requires the usage of specific notations, which however are still a major learning objective for the students.

Eventually, the question that should be asked with any learning intervention has been nicely clarified by Hattie [25]. It is not just the question: Does it work? Because based on Hattie's analysis of more than 800 meta-analysis, the answer would be: "everything works" or more precisely 95% of all the learning interventions, which he investigated, have a positive effect on the learning outcome. So instead Hattie suggests to ask a different question by using effect size to compare all the interventions on a common scale: What works better than average?

Following Hattie's terms, when replacing an intervention previously conducted by teachers with a working electronic equivalent, just evaluating whether they both work is not sufficient. Instead both solutions should be compared more closely and from the perspective of learning outcome the better one (or at least the better than average one) is favourable. So in the context of this thesis this is the question to ask, when replacing the human assessment of assignments by another possibly electronic method. And arguably the same question could also be asked with a future application of the prediction model. Except that differently from the assessment, students currently do not receive predictions about their future assignment performance by any other means.

Similar principals have been found to guide former research as well. For instance GraphoGame[1] (also Ekapeli[2]) is a relatively new creation that relies on teacher-software cooperation. Developed at the University of Jyväskylä in Finland it is an "evidence-based learning game [that] helps children to read". Here the word "helps" is of utter importance and emblematic for GraphoGame's self-concept. Clearly it does not say "teaches" because "GraphoGame is not intended for replacing the regular reading instruction children receive at school, but to work as a supplementary tool" [36]. Its roots lay with the Jyväskylä Longitudinal Study of Dyslexia. Meanwhile however, countless Finnish variants, such as for immigrants, have been developed in addition to other language versions.

The core concept of the gameplay is actually really simple. In several minigames the player hears a sound (phoneme or phonemic unit) and has to pick the right letter, syllable or even word from a selection displayed on screen. A key feature of the game is how it adapts to the learner to maintain a success rate of around 80% all the time [34]. The basis is an adapted Hidden Markov Model that optimizes the information gain about the learner's skill by selection of tasks. Thus the underlying assumption is that exactly these tasks are currently also most suitable to the learner.

GraphoGame has many distinctive features when compared to our method such as younger users, different model and the integration of game elements. However, similarly to

[1] `http://info.grapholearn.com/`
[2] `http://www.lukimat.fi/lukeminen/materiaalit/ekapeli/ekapeli-in-english-1`

the prediction approach in this thesis predictions about assignment performance can take one of comparably many values and they are then used to provide a personalized learning experience. Other works in the context of higher education courses focus the prediction of some notion of course completion.

For example Drăgulescu et al. evaluated different machine-learning techniques for a multi-class classification of students' assignment submission behaviour. Based on 11 features collected for every assignment the behaviour of the students is classified into one of three groups: *on time*, *over time* or *no* submission. Since students that did not submit a "graded assignment" [14, p. 244] were more likely to fail the course, they proposed to red-flag students based on this classification. Obviously, the already observed submission behaviour can be used to red-flag students as well and even without the uncertainty of prediction. Assumably however, Drăgulescu et al. recognized the importance of early intervention to better prevent dropout. When assignments are (continuously) not submitted, dropout has arguably already occurred. In the case of high-school students reconstructing engagement of disengaged students has been found to require "intensive measures" [43, p. 53]. Consequently, a prediction model that allows to identify at-risk students early and reliably would be preferable. Other scholars like Halawa et al. have recognized this issue as well and evaluate the timely identification of at-risk students as well [23]. Nevertheless, this specific work is situated in the different context of Massive Open Online Courses (MOOCs) where dropout rates are particularly high [11].

Eventually, when some notion of course completion is predicted the information of at-risk students should be applied in some way. Just automatically informing the relevant students about their imminent dropout requires little effort and is also feasible for large courses up to MOOCs. Any further, more differentiated intervention either requires the usage of another prediction model or human intervention, for which in MOOCs and higher education the resources are limited. This is the gap that further prediction models like the one proposed with this thesis might be able to fill. While the works on dropout prediction allow to identify the need for intervention the differentiated prediction of labelled assignment performance allows to choose between different interventions without further human interaction.

CHAPTER 2

Dataset

The groundwork for this whole thesis is constituted by the dataset used, which was provided by research assistant Niels Heller. Collected in the summer term 2016 from the introductory lecture *Formale Sprachen und Komplexität* (in English: Formal languages and complexity) it provides a fully anonymized and obfuscated binary matrix representation of the relationship between *students*, *exercises* and respective *labels*.

Since a solid understanding of the dataset given by the matrix is key to all the remaining chapters a short analysis of the dataset is included in the end of this chapter. Before that the next section provides the context by detailing the meaning and origin of labels. It is followed by the formalization used for labels, exercises, students and the rating matrix throughout the remaining work.

2.1 Labels

Labels are derived from the feedback given by correctors on students' assignments. Each label codes one common feedback like for the simplest instance "good". That is, that student's returned assignment exhibited neither errors nor evidence for relevant misconceptions. The remaining labels code issues evident in the returned assignments with a main focus on those of conceptual nature. Here are some examples for labels, that have been translated closely from the German counterparts listed on page 47 in the appendix:

Understanding of implications insufficient

When there's a grammar of type n, the language for this grammar is at least of type n. Maybe there is an even *simpler* grammar, that produces the same language.

Remember: The type of a language is always greater or equal to the type of a corresponding grammar.

Particularly: If one is able to specify a type 2 grammar for a language, one cannot defer that the language is regular.

Language incomplete

This grammar does not produce all the words.

Typically: The word "100" is not contained in the language.

Epsilon production ignored
A grammar that contains an epsilon production, is necessarily of type 0.

Syntax for grammars invalid
On the right side of production rules only variables of the language may be denoted.

Not allowed: Regular expression, whole languages, automatons, ...

That is, every label consists of two parts. The first line provides a short summary intended mainly for easy identification by the correctors. Additionally this line may explicitly or implicitly denote a conceptual problem. Explicitly in the example *Understanding of implications insufficient* by giving an interpretation instead of describing a mere observation. Here the student should either revise the concept of implication as opposed to equivalence in general, or the specific instance of an implication relating the type of languages and grammars. Implicitly in the case of *Language incomplete* because in this case an observation is described as a fact not relying on an interpretation. Here one can locate the issue with the concept of formal languages. Finally the following lines provide a more detailed description of the issue and hint at relevant concepts. This part is of most relevance to the student receiving the feedback. For the corrector, on the other hand, collecting these description reduces the workload unlike rewriting them every time. For this reason, we expect high acceptance of a software system that integrates labels into the correction process. Especially so in combination with functionality to complement the label with individual feedback as required on a case to case basis. However, such a system goes beyond the scope of this thesis and will only be grazed when discussing the results in chapter 5.

2.2 Formalization

To develop a prediction model based on Collaborative Filtering the content is of no relevance to the model. Therefore this thesis uses a simple representation of labels, which only comprises of necessary information. Additionally, that representation guarantees students full anonymity and obfuscates the labels. Every student, exercise and label is represented by an index:

- for students: $s \in \mathbb{N}$
- for exercises: $\texttt{u}-v-ij$ with
  - $v \in \mathbb{N}$ the exercise sheet,
  - $i \in \mathbb{N}$ the exercise, and
  - $j \in \{a, \ldots, z, \emptyset\}$ an optional subexercise
- for labels: $\texttt{c}-l$ with $l \in \mathbb{N}$

Furthermore the last two identifiers for an exercise e and label l are combined into one *observation* index

$$o = (e, l)$$

This allows for a representation of the binary relationship between students and observations in a two-dimensional rating matrix $r_{so} \in \{1, 0\}$. For an observation $o = (e, l)$ the rating matrix is encoded as

$$r_{so} = \begin{cases} 1 \mathrel{\hat{=}} \textsc{True} & \text{if label } l \text{ was contained in student's } s \text{ feedback for exercise } e \\ 0 \mathrel{\hat{=}} \textsc{False} & \text{otherwise} \end{cases}$$

A partly visualization of the rating matrix r_{so} is shown in figure 2.1.

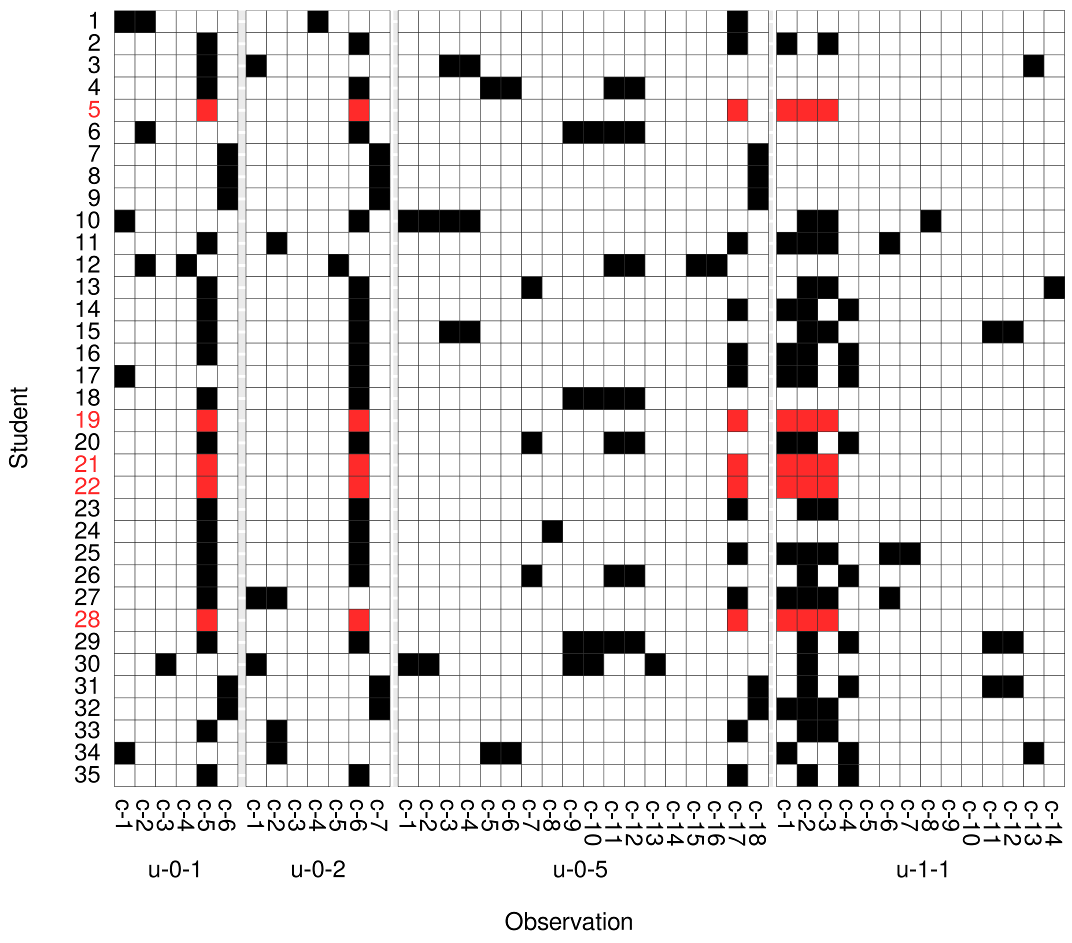


Figure 2.1: Extract from the rating matrix (r_{so}) of the introductory lecture *Formale Sprachen und Komplexität* showing an extract of 35 students and 45 exercise-label pairs, i.e. observations. For a student s an observation o is represented by a filled cell if the rating $r_{so} = 1$ and by an empty cell otherwise. One pattern emerging with students is highlighted by red cells.

2.3 Analysis

The complete rating matrix contains ratings for 82 students on 495 observations derived from 30 exercises and 27 labels. Those 82 students were randomly chosen out of all the 345 participants that handed in the first assignment. For all the students labels have been collected for every single exercise. The sample size is considered sufficient for this work since predictions are made for individuals and not populations. Collaborative Filtering primarily relies on sufficient information about the relation between observations and students among themselves and each other. In a supervised machine-learning setting if the dataset were to be insufficient in size or quality, poor predictions are very likely identifiable by using only a subset of the data in training and the remainder to evaluate the predictions. However, further analysis will reveal, that due to the properties of the dataset this is an unlikely outcome.

Before we continue several of those properties are already evident in figure 2.1, the visualization of the rating matrix. All in all the following four properties can be observed and will be discussed hereafter:

1. Not every label has been observed for every exercise:
   e.g. `c-7`

2. One student can receive multiple ratings for a single exercise:
   e.g. `u-0-1c1` and `u-0-1c2` for student 1

3. Some labels are observed more frequently than others:
   e.g. `c-6` vs. `c-7`

4. For students different patterns seem to emerge based on the labels they received:
   e.g. students 5, 19, 21, 22 and 28

Property 1 leads to the important design decision for the implementation of the prediction model to incorporate an observation set $O = \{o = (l, e) \mid \text{label } l \text{ is observable for exercise } e\}$. With the previously defined formalization of the dataset the observation set would be redundant as the same information can be contained in the rating matrix. However, the implementation uses a sparser representation of the matrix for persistence, where – to preserve all the information from the rating matrix – an observation set can be necessary. The details only relevant to the implementation are described in section 4.3.2.

Next, property 2 is important to the definition of the rating matrix. Due to multiple ratings per exercise for one student the ratings r_{so} from the rating matrix cannot be simplified to $r_{se} = l$ with e, the exercise, and l the label derived from an observation $o = (e, l)$. As a consequence, a software implementation needs to provide the means to record multiple observations for one time step like an exercise in our case. Alternatively, it would be possible to either pick only one label per exercise according to some algorithm or to represent multiple labels as a single one. For the prediction model neither alternative is acceptable. The first one was rejected because it would drastically lower the available information as figure 2.1 shows that multiple observations per exercise are common. The second one, on the other hand, suffers from a different problem. Assuming we have several observations o_i with an index $i \in \mathbb{N}$ and we combine them into one observation $o_a = \{o_1, o_2\}$ and another one $o_b = \{o_1, o_3\}$, then $o_a \neq o_b$. That is, the information that $o_1 \in o_a, o_b$ is not anymore available by an equality comparison. However, if two students receive o_a and o_b respectively, with Collaborative Filtering the information that both actually received o_1 is essential. In other words, joining multiple labels would yield another, more detailed and hence most probably less reliable prediction model, which predicts sets of observations instead of individual observations. Then predicting ratings for every possible set of observations results in a map $\mathscr{P}(O) \to \mathbb{R}$ with O, the observation set as previously defined, and $\mathscr{P}(O)$, its power set. Since the power set easily grows very large and hence is difficult to computationally handle, making predictions for individual observations is the better alternative. Especially since predictions for individual observations can be flexibly converted into a predicted set of observations for example by selecting only those observations with a predicted rating above a suitable threshold.

The conjecture from property 3 about observation frequencies is confirmed by the label frequencies as shown in figure 2.2. The by far most frequent label `c-1` is known to code "good", which makes it the opposite of all the other labels coding some sort of problem with the assignment. Hence `c-1` appears more often. Furthermore, the overall observable 3,975 exercise-label pairs are relatively equally distributed over the remaining labels. Hence predictions about them should be feasible and reliable. In that regard most problematic are `c-19`, `c-27` and `c-22` with less than 50 observations. In the evaluation the quality of their predictions should be especially scrutinized. On the other hand however, exactly due to their rareness these labels provide the highest information gain when observed for a student. Hence these labels are not going to be removed and are instead just given more attention in the evaluation.

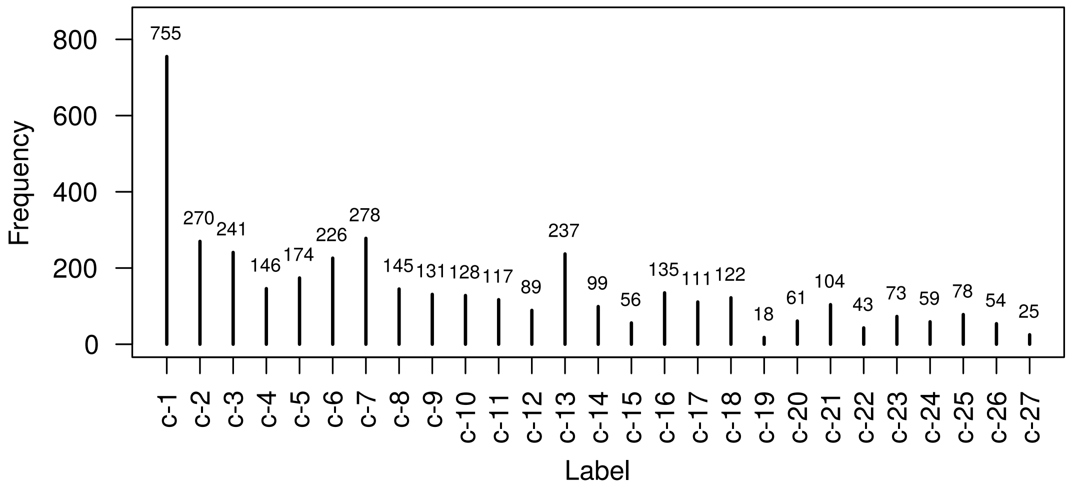


Figure 2.2: Frequencies of labels for the rating matrix r_{so} accumulated throughout all exercises

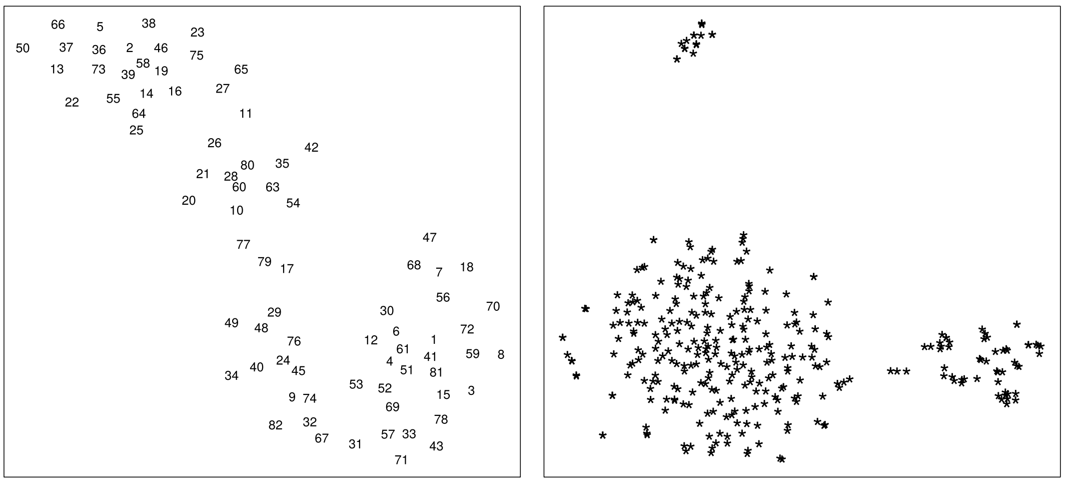
For property 4 intuition suggests that if patterns indeed exist, that they would also visually materialize themselves in clusters. To obtain a plot suitable for such analysis t-distributed stochastic neighbour embedding [42, t-SNE] was applied to the rating matrix mapping the original 495 observation- and 82 student-dimensions onto respectively one two-dimensional scatterplot. The results are presented in figure 2.3. The choice for t-SNE was made, because out of other unsupervised dimensionality reduction techniques it was found to result in better visualizations "on almost all of the data sets" [42, p. 2579], which van der Maaten tested it on. Usually t-SNE like its predecessor SNE [29] is input already dimensionality-reduced data by for example first applying principal component analysis (PCA) [24, ch. 11]. Apart from that the most important parameter for t-SNE is the perplexity, which for a discrete probability distribution p and its Shannon-Entropy $H(p)$ is defined by:

$$Perp(p) = 2^{H(p)} = 2^{p(x)\log_2 p(x)}$$

According to van der Maaten and Hinton "perplexity can be interpreted as a smooth measure of the effective number of neighbours" [42, p. 2582]. Therefore in our case, because of the small number of samples in the rating matrix, low perplexity is expected out of the typical range "between 5 and 50" [42, p. 2582]. Sampling several settings lead to a perplexity of 10 and an initial dimensionality reduction by PCA to 80 for figure 2.3a and a perplexity of 20 and an initial 82 dimensions by PCA for figure 2.3b.

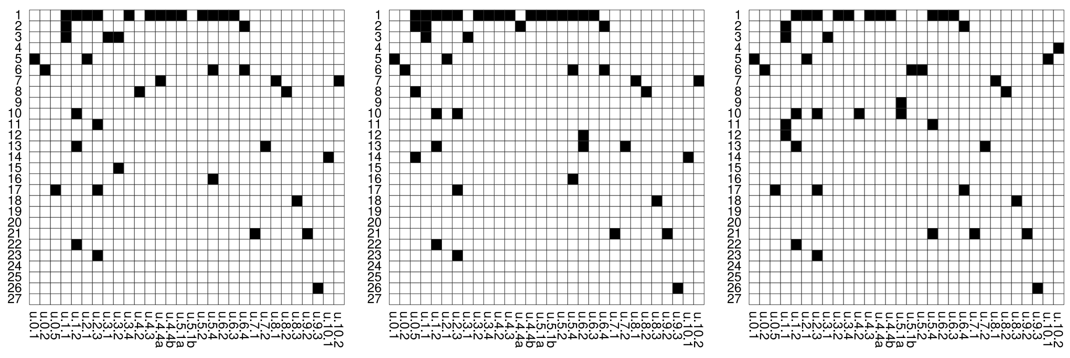
Both figures show that the data contains local and global structure. While bigger, more obvious clusters exist for observations, some small groups of students are significantly more similar among each other than compared to the remainder. We do not see a striking neighbourhood relation for the students initially mentioned in item 4, which is unsurprising when selecting students based on a small subset of their observations. Students 28, 60 and 80 are according to figure 2.3a very similar. This similarity is confirmed by a visualization of all their respective observations in figure 2.4. All three bear a striking resemblance.

When generating figure 2.3 it also became obvious, that the dataset contained duplicates both in rows and columns of (r_{so}). Since the PCA transformation may subtly changes duplicates to become distinct, they have been removed for the purpose of visualization only. That is, students 44 and 61 are solely depicted by student 41. However for the prediction model, neither duplicate students nor observations are removed. In the latter case, for observations, they actually could provide additional information when combining new data



(a) students / rows of (r_{so}) (b) observations / columns of (r_{so})

Figure 2.3: Scatterplots of the rating matrix (r_{so}) using t-SNE for dimensionality reduction



(a) Student 28 (b) Student 60 (c) Student 80

Figure 2.4: Visualizations of all observations from the rating matrix (r_{so}) for three students. Labels are indexed by row and exercises by column.

with our dataset used as training data. In the former case, for users, the explanation is similar. In this case during evaluation it is perfectly valid for test data to contain users with the exact same observations as in the training data. Additionally, dropping duplicates would necessarily impede a computational and developmental overhead, which is superfluous when unlike t-SNE Collaborative Filtering can handle duplicates.

In summary, the aforementioned properties provide strong support that our dataset does actually contain sufficient information to make the predictions we aim for. On these grounds and with a solid understanding of the dataset there is a foundation for the following chapters, in which a prediction model is developed and implemented.

Finally, so far observations were viewed as exercise-label pairs. Another approach would be to use labels as observations regardless of the exercise and to assign ratings based on their frequency for one student. This would most likely reduce the available information but also significantly reduces the dimensionality of the data. Furthermore, predicted ratings would represent frequencies instead of just confidence, which could be of interest to some use cases. Future works could evaluate the effects of this different representation using explicit and implicit Collaborative Filtering methods.

CHAPTER 3

Prediction model

Because the proposed prediction model is not limited to one software platform such as *Backstage 2*, this chapter only gives a formal description. After a short introduction to Collaborative Filtering and recommender systems in section 3.1, the structure used to present the proposed prediction model is loosely built around the input-process-output model. Hence in section 3.2 two important properties of the input lead to a short formalization of the input and expected output. Next section 3.3 describes the prediction process connecting input and output. The specific method is called Alternating Least Squares with weighted-λ-regularization (ALS-WR), which is one approach to Collaborative Filtering and recommender systems described by Zhou et al. [44]. Lastly, an introduction to the evaluation tools follows, which are subsequently used to interpret the results in section 3.5. As a reminder, the aim is to make predictions about students' future assignment performance to enable personalization applications in blended learning environments such as *Backstage 2*.

3.1 Collaborative Filtering

Since personalization is also a problem of recommendation, recommender systems have been the research focus for this thesis. Collaborative Filtering, as the method of choice, is but one approach to recommender systems in the bigger ensemble of machine-learning techniques. Recommender systems may roughly be split into two groups [3]. Common to both groups are *users*, *items* and *ratings*, whereas ratings connect the former two. For this thesis they correspond to students, observations or labels, and ratings. The distinction of recommender systems lies with their conceptualization. For *Content-based recommender* methods ratings are of lesser importance and can be mere results. Instead users and items are represented by "attribute information" such as "textual profiles or relevant keywords" [3, p. 8]. Therefore, it is *content* connecting users and items into a rating relationship. *Collaborative Filtering* relies only on "the collaborative power of the ratings provided by multiple users" [3, p. 8]. Users are represented by feature vectors containing their ratings on items. Usually, where ratings are missing, no implicit assumptions are made and the vector is left sparse. To obtain predictions, the basic idea with Collaborative Filtering is to assume, that proximity of users in this feature space implies similar taste. Based on that recommendations are derived from the remaining dissimilarities of similar users. Next, we will relate this general understanding of Collaborative Filtering to the application of predicting assignment performance.

3.2 Context

In this thesis the input, that is the rating matrix, has two special properties not usually seen in Collaborative Filtering [3]. First, at one point in time only a subset of all observations have a rating record. Taking lecture week three for example, at that point all observations for exercises from weeks one through three have a rating record. Contrarily, observations for exercises from weeks four until the last week do not have a rating record. Only for that exact reasons exists in our application a distinct boundary between two datasets. For this thesis they are referred to as *training data*, which is collected from past lectures, and as *runtime data*, which is collected from a current lecture. Both of them are of the same matrix format, whereas only the latter, the runtime data, is incomplete from a certain point. and the former, the training data, fills its gaps thereby enabling prediction. Effectively however, Collaborative Filtering is only run on one matrix, where training and runtime data are concatenated by rows, that is by users.

Second, this concatenated rating matrix is not sparse except for the runtime data not yet available at one point in time. During assessment the expectation for correctors is to either record a label if it is applicable to the assignment, or not to record it if it is not applicable. Since positive ratings are explicitly recorded and all observations are known the negative ratings can be implicitly deferred. Additionally, this is based on the assumption that for correctors the decision is clear-cut as for the present dataset. If it were not clear-cut, for any label some notion of "I don't know" could be represented by sparse entries in the matrix. Based on these consideration an implicit approach as presented by Hu et al. [30], that implicitly assumes missing ratings to be FALSE, does not fit our application. Hu et al. would be applicable to non-explicit ratings, such as "time spent viewing" or "number of clicks", where the previous implicit assumption is valid. The implementation from chapter 4 would support the method of Hu et al. as well. However, in our case ratings are indeed explicit and the sparse entries are completely unknown.

3.2.1 Training data

In summary, for the input, that is the training data, a rating matrix is used, where for every student s an observation o is assigned a rating or *NaN* for missing ratings

$$r_{so} = \begin{cases} 1 \mathrel{\hat{=}} \textsc{True} & \text{if } o \text{ was observed} \\ 0 \mathrel{\hat{=}} \textsc{False} & \text{if } o \text{ was not observed} \\ \mathit{NaN} & \text{otherwise} \end{cases}$$

Every student s from this matrix belongs to one of two groups. The first group is the *runtime data*, which contains students of the current lecture and observations up to certain point. Afterwards this group contains only sparse entries given by *NaN*. The second group is the *training data* and it includes students from past lectures and hence ratings on any observation. Their combination is formed by first dropping all the observations from the runtime data not in the training data. Next, the resulting student columns are appended as new students to the training data. As we advance in time the runtime data changes incorporating new ratings for sparse entries, whereas the training data remains unchanged.

3.2.2 Predictions

For the output, i.e. predictions, assignment performance is desired and modelled by observations based on labels as for the input. Furthermore, given that not all observations might be interesting, a prediction set P containing arbitrary observations is used. A prediction model should output a map $p_s : P \longrightarrow \mathbb{R}_0^+$, which maps for a student s every observation $o \in P$ to a predicted confidence rating and particularly not to a probability rating

$$p_s(o) = \begin{cases} x \in B_{\varepsilon_{so}}(1) \cup [1, \infty[& \text{if } o \text{ will be observed} \\ x \in B_{\varepsilon_{so}}(0) & \text{if } o \text{ will not be observed} \\ NaN & \text{otherwise, if no prediction is possible} \end{cases}$$

with the open ball $B_\varepsilon(c) = \{x \in \mathbb{R}_0^+ \mid |x - c| < \varepsilon\}$ and ε_{so} unknown. Moreover, for two observations $o_1, o_2 \in O$ and with a non-strict definition of likeliness the following property is generally desired

$$o_1 \text{ more likely than } o_2 \iff p_s(o_1) > p_s(o_2)$$

for which reason $p_s(o) \in [1 - \varepsilon_{so}, \infty[= B_{\varepsilon_{so}}(1) \cup [1, \infty[$ is used in the case that o will be observed. The above property allows to particularly obtain top-k-recommendations while a clear-cut binary decision for or against an observation is generally impossible since ε_{so} is unknown and dependent on s and o. The results in section 3.5 will show, that, based on the mean prediction error, ε is typically sufficiently small to obtain a good binary prediction as well.

3.3 Alternating Least Squares with weighted-λ-regularization

To connect input and output the matrix factorization approach to Collaborative Filtering using the Alternating Least Squares method with weighted-λ-regularization (ALS-WR) described by Zhou et al. [44] was selected. It has already been implemented for the Apache Spark platform and this section is essentially a formal documentation of that implementation for the case of *explicit* and *non-negative* ratings.

In matrix factorization the idea is, that in our case for the students and observations, latent factors exist, which explain the more frequent co-occurrence of certain observations as compared to others. Intuitively, that matched exactly the expectation that possibly non-observed, relatively time-stable characteristics of the students, such as misconceptions or even personality traits, sufficiently determine observations and thereby aspects of assignment performance.

For a rating matrix $(r_{so}) \in \{0, 1, NaN\}^{n \times m}$, that is with n students and m observations, the resulting factorization yields k student factors $(x_s) \in \mathbb{R}^{n \times k}$ and k observation factors $(y_o) \in \mathbb{R}^{m \times k}$ such that

$$x_s^\intercal y_o = \hat{r}_{so} \approx r_{so} \quad \forall s, o$$

We call $\hat{r}_{so}$ the predicted rating and k the *rank* of the factorization, which signifies the number of latent factors and is dependent on the specific dataset.

As the name Alternating Least Squares suggests, this method alternates between students and observations to optimize the squared error of $\hat{r}_{so} - r_{so}$ in a pre-defined (maximum) number of iterations. For that purpose $X = (x_s) \in \mathbb{R}^{n \times k}$ and $Y = (y_o) \in \mathbb{R}^{m \times k}$ are randomly initialized and iteratively optimized by minimizing the cost function

$$\sum_{s,o} (r_{so} - x_s^\intercal y_o)^2 + \lambda\, \omega (\|x_s\|^2 + \|y_o\|^2)$$

Here $\lambda \in \mathbb{R}$ is a suitably chosen (Tikhonov) *regularization parameter*, that limits the complexity of the factorization into X and Y and a proposed value is 0.02 [20]. As suggested by Zhou et al. [44] λ is weighted by the number of explicit ratings. Hence for this thesis $\omega = n \cdot m - (\delta_T + \delta_R)$, where δ_T is the number of sparse entries in the training data (usually 0) and δ_R the number of sparse entries in the runtime data dependent on the lecture week.

More precisely, in every iteration a better approximation of X and Y is computed. That computation is based on the observation, that with either factors fixed "the cost function becomes quadratic" [30, p.4]. So with either student or observation factors fixed obtaining respectively the observation or student factors represents a least squares problem

$$\min_x \|Ax - b\|^2 + (\lambda\omega)\|x\|^2 \iff \min_x x^\intercal(A^\intercal A + (\lambda\omega)I_k)x - x^\intercal(A^\intercal b)$$

where either $A := X$ or $A := Y$ fixed, and respectively $x := y_o \; \forall o$ and $b := r_{*o}$, or $x := x_s \; \forall s$ and $b := r_{s*}$. Here the *-notation signifies the vectors $r_{*o} = (r_{1,o}, r_{2,o}, \dots)$ and $r_{s*} = (r_{s,1}, r_{s,2}, \dots)$ and I_k is used for the $k \times k$ identity matrix. Finally, this problem is solved with a non-negativity constraint on x using the modified conjugate gradient method implemented by Apache Spark. Shewchuk [39] gives a good summary of the conjugate gradient method in general.

So in summary, how does this relate to the training data (input) and predictions (output) previously defined in section 3.2? For every student s and every observation o we had the rating r_{so} for the input. Applying a factorization of rank k to the resulting matrix $(r_{so}) \in \mathbb{R}^{n \times m}$ yields student factors $(x_s) \in \mathbb{R}^{n \times k}$ and observation factors $(y_o) \in \mathbb{R}^{m \times k}$. Their product $x_s^\intercal y_o$ forms an approximation of r_{so}, which should particularly be defined for sparse entries $r_{so} = NaN$. In our case the sparse entries are mostly in the runtime data, which is the part of (r_{so}) where the students s are currently participating in a lecture. Since the other part, the training data, only contains students from past lectures, its sparse entries are not of interest in prediction (if there are any at all).

In any case, out of all the students s and observations o, there are some of particular interest but $r_{so} = NaN$. For every interesting student s this leads to the restricted definition of a prediction set $P_s = \{o \mid \text{observation } o \text{ is interesting for student } s\}$. Here interesting usually means that for an observation, i.e. the tuple of an exercise and a label, the exercise and label are generally still used with the current lecture and particularly in the near future. Thus using the factorization we obtain the output, which is a prediction $p_s(o) = x_s^\intercal y_o$ for every $o \in P$, where $p_s(o) = NaN$ particularly if a student s has no observations or past lectures did not gather ratings for the observation o. The biggest issue for the prediction model is posed by the latter case. Since observations have been defined as an exercise-label pair one possible remedy is the usage of labels on their own. This has not yet been evaluated and would yield very different results predicting the frequency of labels for students and not their occurrence on exercises. And for the exercise-label pairs the results will show that using this additional information produces good prediction results, which is equally important.

3.4 Evaluation

Before presenting the results in section 3.5 an introduction is given about the method and metrics used to evaluate and tune the parameters of the prediction model. All the results are based on the dataset from chapter 2, from which ten folds were derived for Cross-Validation. Every fold contains a random selection of users, i.e. rows of the rating matrix. Conceptually, the training data is viewed as coming from past lectures, whereas the runtime data is being collected from an ongoing lecture. For that reason, we use so called *steps* [13] to model varying levels of knowledge. For the purpose of evaluation a step is defined as one week and comprises all the observations for exercises of that week. Furthermore, for the first week, where no behaviour has been observed yet, no predictions are possible. Hence out of 11 assignment weeks predictions are computed in 10 steps. In the first step only ratings for observations of the first first week are available and observations from the remaining weeks form the prediction set. Iteratively in every step more ratings for observations from the current week become known, whereas the number of predictions shrinks as the same observations are removed from the prediction set. Thus with ten folds 100 models (one for each step and fold) must be trained and for every fold 35,264 predictions are calculated and evaluated. Due to the resulting high computational overhead the majority of the computations was run on Apache Spark separate from *Backstage 2* and the statistics suite R was used to summarize and plot the resulting data.

3.4.1 Metrics

Metrics are the foundation of the following hyperparameter tuning and for the result evaluation in general. Here, their respective definitions are presented along with a short overview of their interpretation and usage. In this thesis they are generally used with a prediction set P of observations and a prediction map $p_s : P \longrightarrow \mathbb{R}_0^+$, which for a student s yields nonnegative predictions as defined in section 3.2 for the output of the prediction model. Moreover, in this supervised machine-learning setting the correct prediction for a student s and an observation o is given by $r_{so} \in \{0,1\}$, derived from the dataset as described in chapter 2. Hence particularly $r_{so} \neq NaN$ can be presumed, which simplifies the evaluation. Several instances of P, p_s and r_{so} can be collected from different students, steps or folds into one family input to an aggregate metric. Those metrics are now outlined in the following sections and the index sets $I = \{1,\dots,n\}$ and $J = \{1,\dots,m\}$ are used throughout with $n,m \in \mathbb{N}$.

3.4.1.1 Root-Mean-Square-Error (RMSE)

RMSE is computationally the simplest metric that has been included. But RMSE can also be difficult to interpret when used but for the general optimization of a (Collaborative Filtering) algorithm [3, p. 413]. Its definition is based on the *error* of a prediction $p \in \mathbb{R}_0^+$ with a corresponding correct rating $r \in \{0,1\}$

$$\operatorname{err}(p,r) = p - r$$

One might be tempted to believe that $\operatorname{err}(p,r) \in \{-1,0,1\}$, which in fact it is not. Instead since $p \in \mathbb{R}_0^+$ the error $\operatorname{err}(p,r) \in [-1,\infty[$.

To obtain a measure that weights large errors over smaller ones RMSE [3, p. 230] is derived from the plain error in analogy to the standard deviation

$$\operatorname{RMSE}(D) = \sqrt{\frac{1}{n}\sum_{i\in I}(\operatorname{err}(p_i,r_i))^2} \in \mathbb{R}_0^+ \quad \textit{smaller is better}$$

with a family $D = (p_i,r_i)_{i\in I}$ and where p_i the predicted and r_i the correct rating.

The resulting measure is flexible and can be arbitrarily aggregated over the predictions for one student, multiple students, steps or even folds. In every case RMSE yields only a single number. Because of its definition the meaning of the RMSE is analogous to the standard deviation. So particularly 95% of the samples RMSE was aggregated over have an absolute error less or equal to twice the RMSE. For top-k-recommendations, where the k observations with the highest predicted rating for a student are used, its applicability is limited since RMSE constitutes a “global ranking measure” [3, p. 413]. These measures are generally global aggregates over all the predictions of every student contrarily to just the top-k ones for instance.

3.4.1.2 Mean Reciprocal Rank (MRR)

In the case of top-k-recommendations a “top-heavy ranking measure” [3, p. 413] such as the MRR is more typically used. For the prediction set P containing interesting observations and a prediction map $p : P \longrightarrow \mathbb{R}_0^+$ the *rank* of an observation $o \in P$ is defined as

$$\operatorname{rank}_{P,p}(o) = |\{\hat{o} \in P \mid p(\hat{o}) > p(o)\}| + 1$$

In respect to the order defined by the prediction map p on P the everyday meaning of $\text{rank}(o) = n \in \mathbb{N}$ would be *"o is the n-th 'finisher'"*, where in the case of m equally ranking observations $m-1$ ranks are skipped.

To obtain the MRR a family $D = (P_i, p_i, R_i)_{i\in I}$ is needed, where P_i the prediction set, p_i the prediction map and R_i the set of relevant observations. Here, relevant signifies all those observations that should be ideally predicted, i.e. $r_{so} = 1$. Averaging the reciprocal rank of the first relevant observation leads to the definition of MRR [3, p. 246] given by

$$\text{MRR}(D) = \frac{1}{n}\sum_{i\in I}\frac{1}{\min \text{rank}_{P_i,p_i}[R_i]} \in \,]0,1] \quad \textit{bigger is better}$$

As already hinted, this measure is tightly coupled to the top-k-recommendations. Assuming $\text{MRR}(D) \approx \frac{1}{k}$ with $k \in \mathbb{N}$ then on average the top-k-recommendations contain at least one relevant prediction. Particularly for that reason, using MRR does not globally optimize all predictions whether relevant or not. Instead its focus is to provide anything relevant first, that is with a high rating.

3.4.2 Metrics for binary classification

Since for each fixed observation the proposed prediction model could be used as a binary predictor as well, the necessary tools are collected in this section. Assuming $\theta \in \mathbb{R}_0^+$ is a suitable threshold, then by using the decision rule $p >= \theta$ yields a binary prediction based on a predicted rating $p \in \mathbb{R}_0^+$.

Therefore, from now on we suppose binary predictions and let $D = (p_i, r_i)_{i\in I}$ a family where $p_i \in \{\text{TRUE}, \text{FALSE}\}$ a prediction and $r_i \in \{\text{TRUE}, \text{FALSE}\}$ the expected correct prediction. For each index $i \in I$ this results in four cases which can each be counted and summarized in a contingency table [35]:

		Actual rating	
		$r_i = \text{TRUE}$	$r_i = \text{FALSE}$
Predicted rating	$p_i = \text{TRUE}$	**True Positive (TP)**	*False Positive (FP)*
	$p_i = \text{FALSE}$	*False Negative (FN)*	**True Negative (TN)**

TP and TN on the one diagonal consist of correct predictions whereas the other diagonal with FN and FP contains wrong predictions. Ideally for a sample of size n the expectation is no prediction errors $\text{FP} = \text{FN} = 0$ and the remaining predictions to either be rightfully positive or negative $\text{TP} + \text{TN} = n$. Since optimization on multiple values is impractical, many combined measure have been defined for the confusion matrix. The aim for this thesis is to eventually plot ROC and Precision-Recall curves, for which purpose Recall, Fall out and Precision are needed.

Recall / FPR This is the *True Positive Rate* (TPR) also called *Recall* or *Specificity* [35]. Out of all the actually positive ratings it measures the proportion recalled as positive ratings in prediction.

$$\text{TPR}(D) = \frac{\text{TP}}{\text{TP}+\text{FN}} = \frac{|\{i \in I \mid p_i = \text{TRUE} \wedge r_i = \text{TRUE}\}|}{|\{i \in I \mid r_i = \text{TRUE}\}|} \in [0,1] \quad \textit{bigger is better}$$

However, just looking at the TPR is not sufficient since the right half of the confusion matrix is completely ignored. So for perfect $\text{TPR} = 100\%$ the easiest solution is to just always predict TRUE, which is not desirable and for which reason a second metric is needed.

Fall-out / FPR This second metric is the *False Positive Rate* (FPR) also called *Fall-out* [35]. Out of all the actually negative ratings it measures the proportion wrongly predicted as positive.

$$\mathrm{FPR}(D) = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} = \frac{|\{i \in I \mid p_i = \textsc{True} \wedge r_i = \textsc{False}\}|}{|\{i \in I \mid r_i = \textsc{False}\}|} \in [0,1] \quad \textit{smaller is better}$$

Finally, in combination with the TPR the FPR accounts also for actually negative ratings and hence prevents the degenerate predictor previously outlined. Nevertheless, a direct relation between TPR and FPR is still missing. For example, we could suppose the number of actually negative and actually positive ratings is very unbalanced, say many more negatives. In that case although TPR high and FPR low the prediction could still yield many more false positives than true positives in absolute numbers. So a third measure is needed.

Precision / PPV This is the *Positive Predictive Value* (PPV) also called *Precision*[35]. Out of all the positively predicted ratings it measures the proportion rightfully predicted as positive.

$$\mathrm{PPV}(D) = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = \frac{|\{i \in I \mid p_i = \textsc{True} \wedge r_i = \textsc{True}\}|}{|\{i \in I \mid p_i = \textsc{True}\}|} \in [0,1] \quad \textit{bigger is better}$$

Altogether combined with the TPR and FPR the PPV helps to prevent all the previously described degenerate cases of prediction. The remaining problems are finding the suitable threshold $\theta \in \mathbb{R}_0^+$ and particularly for that purpose a means to compare multiple measures in a compact form, such as plots of curves.

3.4.2.1 Receiver Operating Characteristic (ROC)

The ROC was first invented for and named after radar receivers during the second world war, for which the challenge was to tweak parameters to obtain a good balance between true and false positives for the detection of radar echos [19]. For this thesis the parameter is a family of thresholds $(\theta_i)_{i\in I}$ where $\theta_i \in \mathbb{R}_0^+$. As θ_i is varied the ROC visualizes changes in prediction performance by plotting the TPR on the y-axis against the FPR on the x-axis [3, p. 247ff.]. We require for every threshold θ_i a family $D_i = (p_{ij}, r_j)_{j\in J}$ where p_{ij} the binary prediction derived from the threshold θ_i and r_j the true rating. Then the ROC is a non-continuous function graph defined by

$$\mathrm{ROC}((D_i)_{i\in I}) = \{(\mathrm{FPR}(D_i), \mathrm{TPR}(D_i)) \mid \forall i \in I\} \subset [0,1]^2$$

For random predictions all points of the ROC $(\mathrm{FPR}, \mathrm{TPR}) \in \mathrm{ROC}$ are located on the bisector of the axes of the coordinate system because $\mathrm{FPR} = \mathrm{TPR}$. A good threshold, on the other hand, maps above the bisector and as close as possible to $(\mathrm{FPR}, \mathrm{TPR}) = (0,1)$. Finally, an ROC underneath the bisector indicates wrong assumptions because by swapping positive for negative it is mapped above the bisector. Overall, the ROC has two usages. First, as detailed so far, it provides evidence of the prediction quality for varying parameters. Second, as a result it allows to pick an optimal threshold parameter in respect to the balance between FPR and TPR.

3.4.2.2 Precision-Recall curve (PRC)

Likewise for the PRC the PPV on the y-axis is plotted against the TPR on the x-axis [3, p. 247ff.]. As previously defined for the ROC, we have a family of thresholds $(\theta_i)_{i\in I}$ and corresponding families $D_i = (p_{ij}, r_j)_{j\in J}$. The PRC is another non-continuous curve given by

$$\mathrm{PRC}((D_i)_{i\in I}) = \{(\mathrm{PPV}(D_i), \mathrm{TPR}(D_i)) \mid \forall i \in I\} \subset [0,1]^2$$

Typically the PRC origins in the top-left corner and ends in the bottom-right. Unlike for the ROC neither a line connecting those corners nor the bisector of the coordinate axes are of relevance. Instead the PRC allows evaluation of the precision as the threshold is changed to increase or decrease the FPR. For that reason ideally the curve remains high and close to constant, which signifies high precision with stability throughout different thresholds.

Lastly, we note that comparing either curve (ROC or PRC) for different prediction models suffices to identify the better model. Herein the definition of better is that one curve dominates the other. This is because ROC and PRC determine the contingency table and hence also the other curve [12]. Therefore in this thesis the ROC is used for the hyperparameter optimization, which involves training and comparing multiple models.

3.4.3 Hyperparameter optimization

For machine-learning algorithms hyperparameters are those parameters, which "change the way the learning algorithm [...] works" [41] and thus ultimately influence what is learned. They depend on properties of the specific dataset and can be viewed as part of the model selection. Parameters on the other hand are not part of the model selection. For example with our prediction model one parameter is the prediction set, which only limits the scope to which the model is applied and does not influence what was learned. However, our interest in this section is the selection of suitable hyperparameters for the proposed ALS-WR prediction model. Reviewing the algorithm from section 3.3 there are three hyperparameters: rank, λ (Lykhonov regularization parameter) and the maximum number of iterations.

Rank (k) The rank $k \in \mathbb{N}$ of the factorization of a rating matrix into k student factors and k observation factors. For an ideal factorization k could be chosen as the rank of the rating matrix. Practically however, the rating matrix contains noise and hence requires approximation by using a lower rank [44]. Therefore k should be close to (or higher) than the actual number of latent factors which determine the behaviour of students resulting in observations. For that reason it could grow with the number of students or when incorporating new observations through new exercises or labels. Since those latent factors are usually unknown we estimate the rank based on cross validation. The default value for the Apache Spark implementation of ALS-WR is $k = 10$. For the Netflix dataset Zhou et al. [44] report values between 20–60 and up to 1000, as they could not observe any overfitting. Based on that and some initial experimentation ranks ranging from 5 to 100 are proposed for evaluation of the prediction model.

Regularization parameter (λ) The Lykhonov regularization parameter λ is introduced to prevent overfitting of the model. Overfitting in the case of matrix factorization means that single factors gain too much weight, for which reason λ is used to limit their magnitude. Due to the weighted regularization approach of ALS-WR λ is less dependent on the dataset. Again for the Netflix dataset Funk [20] proposed $\lambda = 0.02$ with Zhou et al. [44] reporting good results slightly higher with λ between 0.03 and 0.075. Since additionally the default with Apache Spark is set to $\lambda = 0.01$, λ-values from 0.001 to 0.1 are evaluated.

Iterations (N) Since ALS is an iterative method and we do not expect to find a non-approximate solution to the matrix factorization, the third parameter is the maximum number of iterations. In every iteration student and observation factors are optimized by using a conjugate gradient method on each set of factors separately with the other one fixed. Therefore the approximation should improve in every step unless it is already ideal or overfitting occurs. At some point increases should become sufficiently small while the computation cost is still acceptable. This point is dependent on the dataset and other

k		λ		N
$\{5,10,20,40,60,70,80,90,100\}$	$\times$	$\{0.001,0.005,0.01,0.02,0.05,0.1\}$	$\times$	$\{5,40\}$
$\{10,40,70,80,90,100\}$	$\times$	$\{0.005,0.02,0.05,0.065,0.075\}$	$\times$	$\{10\}$
$\{10,40.70,80,90,100,120\}$	$\times$	$\{0.065,0.075\}$	$\times$	$\{5\}$

Table 3.1: The three grids evaluated to identify suitable hyperparameter settings

hyperparameters. Diverging from the default value of 10 iterations we will compare 5 and 40 iterations. Earlier experiments showed that both settings lead to acceptable albeit different quality. Higher values were not systematically evaluated for all parameters due to the previously detailed high computation cost of evaluation. Either, the expectation is that performance differences already manifest themselves between 5 and 40 iterations or that they are small. In the latter case small differences imply that the algorithm has already converged close to an optimum with few iterations.

Conclusively, the the first grid from table 3.1 was constructed. For every grid the settings of any hyperparameter are combined with every setting of the other hyperparameters. After some literature research and pre-experiments the ranges given in the description of every hyperparameter were selected. Only few distinct values were picked from these ranges such that the evaluation would show tendencies for possibly further optimization without becoming overly expensive. Just the first grid identified $\lambda = 0.05$ and $N = 5$ as good settings. Based on this tendency the second and third grid evaluate some more options. For every combination models were trained and predictions calculated as initially outlined in this section. The comparison is done using single value metrics only, that is RMSE, MRR and instead of ROC its area under the curve (AUC). For the AUC [35] the following (approximative) definition is used

$$\mathrm{AUC}\left((x_i, y_i)_{i\in I}\right) = \sum_{i\in I} y_i(x_{i+1} - x_i)$$

where $(x_i, y_i)_{i\in I}$ is a family of points belonging to a curve such as the ROC and ordered such that $x_i < x_{i+1}\ \forall i \in I$.

3.5 Results

The evaluation results are presented in two steps. First, the prediction model is optimized by tuning the hyperparameters. Based on this a good hyperparameter combination for the current dataset is proposed. For different datasets suitable hyperparameters can be selected by a likewise process. The result is a final model concluding the model selection process. Second, the final model is evaluated in more detail.

3.5.1 Model selection

The foundation for the final model selection are the metrics from section 3.4 that were calculated for all the 152 models derived from the parameter grid and every fold. For further analysis of the still very large dataset the metrics were next summarized by common statistical measures such as mean and quantiles into one summary of all the 10 folds for every model. Table 3.2 shows the results for the six best models and an overall summary for all 1520 evaluations performed for any model and fold.

For selecting the best model a combined rank based on the mean of MRR, RMSE and ROC AUC was chosen. PRC AUC was particularly not included because it yields the same

Parameters								
	k	40	70	90	10	80	90	
	N	10	10	10	10	5	10	Overall
	λ	0.065	0.065	0.065	0.065	0.065	0.075	
MRR	Min.	0.5929	0.5909	0.5844	0.5629	0.5800	0.5903	0.5070
	1st Qu.	0.7354	0.7499	0.7619	0.7757	0.7481	0.7532	0.7043
	Median	0.8289	0.8303	0.8263	0.8229	0.8310	0.8222	0.7736
	Mean	*0.7982*	*0.7981*	*0.7980*	*0.7982*	*0.7970*	*0.7975*	*0.7571*
	3rd Qu.	0.8712	0.8628	0.8642	0.8623	0.8601	0.8625	0.8212
	Max.	0.8931	0.8902	0.8915	0.8916	0.8874	0.8904	0.9016
RMSE	Min.	0.2485	0.2490	0.2487	0.2497	0.2491	0.2516	0.2467
	1st Qu.	0.2546	0.2544	0.2547	0.2538	0.2550	0.2567	0.2608
	Median	0.2606	0.2609	0.2609	0.2600	0.2607	0.2619	0.2690
	Mean	*0.2604*	*0.2605*	*0.2605*	*0.2605*	*0.2605*	*0.2622*	*0.2770*
	3rd Qu.	0.2629	0.2628	0.2630	0.2628	0.2627	0.2653	0.2786
	Max.	0.2751	0.2745	0.2748	0.2757	0.2734	0.2739	0.8406
ROC AUC	Min.	0.6272	0.6279	0.6298	0.6180	0.6310	0.6309	0.4832
	1st Qu.	0.7255	0.7254	0.7258	0.7172	0.7265	0.7262	0.6879
	Median	0.7307	0.7327	0.7327	0.7302	0.7350	0.7339	0.7211
	Mean	*0.7387*	*0.7392*	*0.7395*	*0.7326*	*0.7408*	*0.7393*	*0.7128*
	3rd Qu.	0.7665	0.7662	0.7667	0.7582	0.7655	0.7649	0.7498
	Max.	0.8160	0.8138	0.8160	0.8042	0.8185	0.8142	0.8186
PRC AUC	Min.	0.2769	0.2786	0.2817	0.2784	0.2795	0.2798	0.0609
	1st Qu.	0.3409	0.3429	0.3425	0.3488	0.3385	0.3263	0.3131
	Median	0.3899	0.3821	0.3828	0.3802	0.3830	0.3587	0.3556
	Mean	*0.3773*	*0.3774*	*0.3796*	*0.3820*	*0.3770*	*0.3654*	*0.3506*
	3rd Qu.	0.4169	0.4224	0.4237	0.4249	0.4194	0.4134	0.3935
	Max.	0.4441	0.4421	0.4426	0.4586	0.4521	0.4407	0.4808
Samples		10	10	10	10	10	10	1520

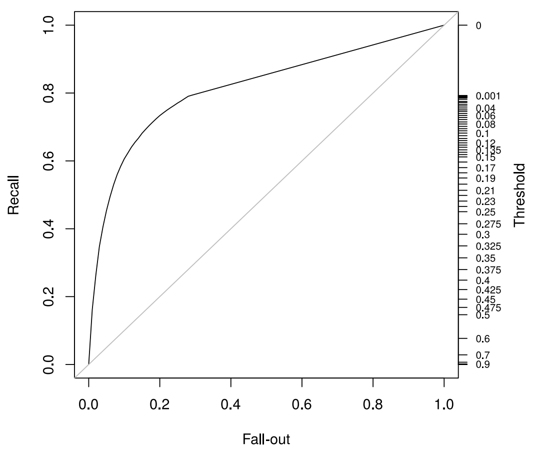
Table 3.2: Summary statistics for the six best as well as for all evaluated models and folds overall. The best models have been selected from a combined rank of the means with weights 3× MRR, 1× RMSE and 2× ROC AUC.

ranks as ROC AUC (see section 3.4.2.2). The combined rank is defined as

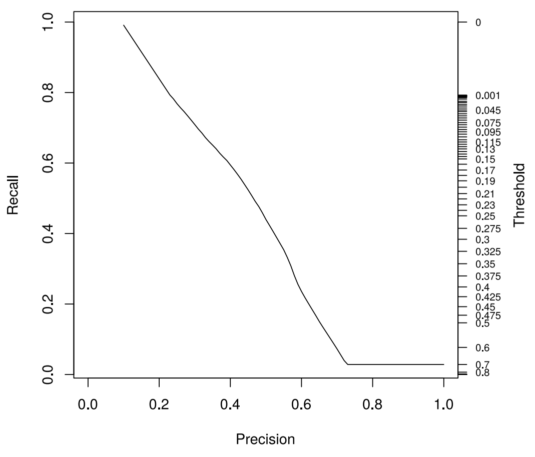
$$\frac{3 \cdot \mathrm{rank}(\mathrm{MRR}) + \mathrm{rank}(\mathrm{RMSE}) + 2 \cdot \mathrm{rank}(\mathrm{ROC\ AUC})}{4}$$

where rank(M) is used to denote the rank of the mean of the metric M. How every metric is weighted depends much on how the predictions will be used. With a top-k-recommendation approach the MRR is most relevant measuring the average position at which the first relevant prediction occurs. When a plain binary prediction instead of recommendations is desired, the ROC provides the means to identify a suitable threshold to convert predicted into binary ratings. Foremost the combined rank is an effort to find a feasible tradeoff for both use cases. Additionally its aim is to prevent overly global optimization when eventually only one threshold is used in the case of binary predictions. For that purpose the MRR received the most weight whereas the global measures RMSE and ROC AUC were reduced in their weight.

This leads to a final model selection with hyperparameters tuned to $k = 40$ for the rank, $N = 10$ for the maximum number of iterations and $\lambda = 0.065$ for the regularization parameter. It is worth noting that the rank significantly varies between the top-ranking models and that 40 is situated in the middle of that range. Furthermore, a maximum number of iterations set to 10 was not expected to yield better results than 40 iterations. Other studies on the much sparser and much larger Netflix dataset showed consistently increasing performance with an increasing number of iterations as measured by RMSE [44]. Possibly the lack of sparseness and the comparably small size of our dataset fosters overfitting with higher iteration settings.



(a) Receiver Operating Characteristic



(b) Precision-Recall Curve

Figure 3.1: Performance of the final model ($k = 40$, $N = 10$, $\lambda = 0.065$) in the case of binary predictions evaluated with different thresholds

On the other hand, the evaluation identified an optimal registration parameter of $\lambda = 0.065$ equal to the optimum identified by Zhou et al.

Nevertheless, the performance of all the top ranking models is consistently above average as compared to the overall column of table 3.2. Indeed differences between them are so small that either one should be suitable for top-k-recommendations and binary predictions alike. A mean MRR of 0.7970 for the final model corresponds to a rank of $\frac{1}{0.7982} = 1.2528$ for the first correct prediction on average. Assuming that predictions can be connected to content relevant for the individual student, it is a very good result. Essentially for every student only the two recommendations with highest predicted rating have to be presented to him to typically provide at least one relevant recommendation. Furthermore with a mean RMSE of about 0.25, approximately 95% of the predicted ratings have an error of $2 \cdot 0.25 = 0.5$ since the RMSE is actually the standard deviation of the error. In those cases using the interval $[0.5, 1.5]$ allows for separation of positive and negative predictions with certainty because a predicted rating of $1 \mathrel{\hat{=}}$ TRUE and a predicted rating of $0 \mathrel{\hat{=}}$ FALSE. Additionally, the ROC AUC is well above 0.5, which is the expected value for random predictions mapping the ROC to the bisector of the coordinate axes.

3.5.2 Final model

For further analysis of the final model the Receiver Operating Characteristic and the Precision-Recall Curve have been plotted by varying the threshold used to convert predicted ratings into a plain "yes" and "no". The result is shown in figure 3.1. Instead of the left-side continuous constant approximation used for the AUC the curves were generated by a linear approximation between the distinct results. Therefore, while the specified AUC values are necessarily underestimates the plot from figure 3.1 could slightly overestimate the performance.

The ROC suggests that a good tradeoff between Recall and Fall-out is at 0.08 as a threshold setting. However, since for one student the number of actually applicable observations is usually much lower than the overall number of possible observations, a comparison with the PRC is important. The PRC shows mostly a near linear descent of the Recall with increasing Precision. In fact at the 0.08 threshold the Precision is only at approximately 0.3 (unlike the apparently good Fall-out of approx. 0.2) with approx. 0.7 for the Recall. Therefore

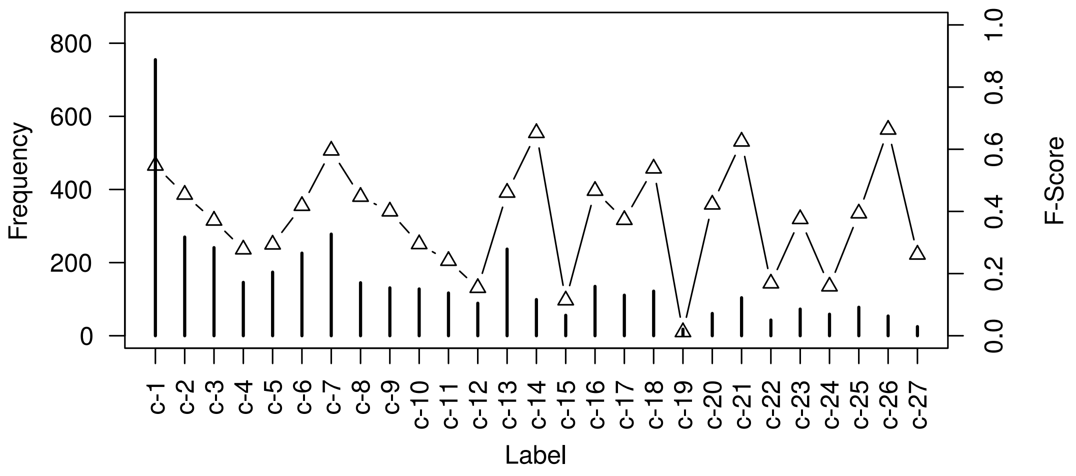


Figure 3.2: Frequency (bars) and F-Score (triangles) with the final model ($k = 40$, $N = 10$, $\lambda = 0.065$) and a threshold setting of $\theta = 0.25$

a threshold setting of 0.25 could be a better tradeoff yielding 0.5 for Precision and Recall and less than 0.1 for the Fall-out. Eventually every individual user will want many correct and few incorrect predictions regardless whether there are few or many possibilities for the prediction model to make wrong predictions. That is, users want high Recall and high Precision. Low Fall-out on the other hand is only relevant to the developer of a prediction model that struggles to select the few correct predictions out of all the wrong ones. Therefore, the Precision-Recall curve should be used to identify a suitable threshold setting on the imbalanced dataset used [38].

Lastly, the proposed value $\theta = 0.25$ was selected to analyze the performance of individual labels. For this purpose an unweighted F-Score was used that combines Recall and Precision like the Precision-Recall curve

$$\text{F-Score}(\text{PPV}, \text{TPR}) = 2 \cdot \frac{\text{PPV} \cdot \text{TPR}}{\text{PPV} + \text{TPR}} \in [0, 1] \quad \textit{bigger is better}$$

The results in figure 3.2 show that the prediction performance varies to a great degree between labels. Initially, when analysing the dataset the labels `c-19`, `c-22` and `c-27` were identified as least frequent. None of them are among the top-performing labels based on the F-Score. But only label `c-19` is very clearly predicted worse than the rest. There seem to exist different explanations for the variance in prediction performance, because `c-27` with 25 occurrences is not much more frequent than `c-19` with 18 occurrences but still `c-27` performs better than `c-22` with 43 occurrences. Regardless of such different explanations the overall tendency with all labels in figure 3.2 is, that with increasing frequency the prediction performance improves. The current dataset only contained 82 out 345 participating students. Therefore most likely the prediction performance would increase when applying the model to even more students or when data from different lectures or terms is consistently combined.

CHAPTER 4

Implementation

To work towards the greater aim of personalized blended learning this chapter details the integration of the proposed prediction model with the *Backstage 2* learning platform. In great parts the implementation has been prepared for by the prediction framework from the master thesis of Dostert [13]. Nevertheless, in other parts modifications were required while the structure of the framework remained mostly unchanged and was preferably supplemented. Since the original implementation of Dostert included only one prediction model quite different from the one proposed now, enhancements had already been anticipated beforehand. Specifically Dostert used the four tags NO ERROR, ERROR, SKIPPED and INSUFFICIENT KNOWLEDGE assigned to all past assignments of a student to predict the exact same tags for future assignments. For that purpose he trained and compared Relative Frequency and Hidden Markov Models using students' time series of these four tags from past lectures. On the one hand, like the now proposed model a time series of tags is used to predict the same tags. On the other hand, in Dostert's case the tags are (1) sealed, (2) known at compile-time and (3) all four are always applicable to any assignment. This differs from the requirements of the new model, which are outlined in the next section and which constitute the software specification. The remaining sections first give an overview of Dostert's framework and identify relevant limitations. Afterwards the resulting general enhancements to the framework are summarized, which are lastly connected by the implementation of the prediction model.

4.1 Specification

With the now proposed prediction model tags, that is observations or labels, are contrarily to Dostert's model (1) not sealed, (2) only just known at run-time, (3) not applicable (= observable and predictable) to every assignment and (4) prediction as well as training is computationally more expensive. These four differences guided every enhancement except for the serialization changes described in section 4.3.6. These serialization changes are optional and provide a general improvement to the prediction framework independent of the proposed model and its implementation requirements.

Next the specific requirements of the proposed prediction model towards a framework like Dostert's are defined. These are the resulting requirements in the style of a user specification:

1. Labels and exercises can be collected from different sources.
2. Labels and exercises are viewed as one unit and called observation as previously in this thesis.
3. Students can have a record of multiple distinct observations for one exercise.
4. Not every label can be recorded for any exercise, that is some of their combinations do not form a valid observation.
5. Used labels and exercises may change throughout lectures but remain identifiable if unchanged.
6. Predictions can be restricted to a subset of interesting ones out of all the predictable observations, such that the computational cost is also reduced. This set of interesting predictions may change throughout the lecture.
7. Predictions for a label are either assigned an anticipated rating or no rating at all in case a prediction was not possible.
8. The storage space required by the data types used should remain within sensible bounds.

In addition to the above specification there are some general requirements not just specific to the proposed model. Since they are exactly what Dostert's prediction framework was developed for, this thesis does not include a list of them. Instead the interested reader may find more information in the master thesis of Dostert [13]. Since the following sections explain how the specification was met, the relevant requirements are referenced by their index in round braces, e.g. (1) for requirement 1.

4.2 Existing prediction framework

By now giving a short overview of Dostert's prediction framework we will be able to isolate those parts of the specification that are not yet sufficiently supported by the present implementation. This results in the limitations of the existing framework for the new use case, which conclude this section.

4.2.1 Overview

The framework was built on top of *Backstage 2* which uses various technologies of which the most relevant ones are:

Scala[1] The programming language used for the backend. It is a functional and object-oriented language compiled to Java bytecode. Therefore it runs on the JVM and is fully compatible to Java code. Its type system is strong and provides automatic type inference. The sophisticated collections API of Scala is of great relevance to the prediction framework [18, scala.collections].

Play Framework[2] The web application framework used for the server-side backend. With Play the interface, application and domain logic are separated using the model-view-controller (MVC) software pattern.

[1]`https://www.scala-lang.org/`
[2]`https://www.playframework.com/`

MongoDB[3] The database providing persistence for objects in the JavaScript object notation (JSON). It is a NoSQL database that stores documents as binary JSON (BSON), a binary format derived from JSON. Since for frontend communication JSON is also used, (de-)serialization is only implemented once for the JSON format [13].

JavaScript The programming language used for the frontend. It is an interpreted language that is functional as well as imperative and weakly typed unlike Scala. *Backstage 2* uses functionality introduced with ECMAScript 6[4].

AngularJS[5] The second web application framework used for the client-side frontend. In addition to the MVC pattern AngularJS implements the Model-View-ViewModel (MVVM) design pattern for updates of the user interface.

At the time of writing this thesis the versions Scala 2.11, Play 2.5 and AngularJS 1.6 defined the most relevant APIs.

The overall structure of *Backstage 2* divides the prediction framework into the two main components: backend and frontend. The backend runs on a server and handles predictions, whereas the frontend shows the results and allows to control the data and prediction models used. Since with this thesis a new prediction model was implemented, the backend is of most relevance. Indeed, one aim of this thesis was not to modify the frontend. Nevertheless, enhancements of the user interface should be considered to improve the usability for the new use case. To keep the overview as short as possible the following paragraphs are a walk-through for the most relevant components of the prediction use case. An overview of the relationships between the explained components is given in the class chart from figure 4.1.

The class **`BehaviourLog`** is the entry point to most functionality of the backend of the prediction framework. With *Backstage 2* lectures are represented as **`Project`**s containing an arbitrary number of **`BehaviourLog`**s. Additionally a **`Project`** provides information about **`User`**s participating in different roles and various types of content, particularly such as assignments and the answers returned by students.

Nevertheless, this information is not directly used from a **`Project`** in the prediction framework (1). Instead there are **`Tag`**s of different types that represent data from which and of which to make predictions. A **`BehaviourLog`** contains for each participating student a sequence of **`Tag`**s observed for that particular student which is called behaviour. For one **`BehaviourLog`** every student's behaviour is of the same **`Tag`** type called the observation type. Furthermore, as new observations are made the **`BehaviourLog`** of suitable observation type is only on the receiving end and the corresponding student's observation sequence is added to (1). Apart from the observation type every **`BehaviourLog`** has a defined prediction type. Predictions are **`Tag`**s of this type and it can be equal – as for the proposed model – or non-equal to the observation type. Thus **`BehaviourLog`**s not only contain all the information required for predictions, they also act as gateways to that functionality.

However, to make predictions training data is required, which is provided by a **`History`** or a set of them. Every **`History`** contains sequences of tags (i.e. behaviour) from past lectures like **`BehaviourLog`**s but without the association to a specific user. Furthermore, tags are all of the same observation type, whereas a prediction type is not specified by a **`History`** since it can be used for any prediction based on tags of its observation type.

Next, predictions are provided by **`PredictionUnit`**s which are configured for each **`BehaviourLog`**, trained using at least one **`History`** and transform a sequence of observations into predictions. Therefore much like the **`BehaviourLog`**s supported **`Tag`**s are specified by an observation type for the training data and by a prediction type for the resulting

[3]https://www.mongodb.com/
[4]http://es6-features.org/
[5]https://angularjs.org/

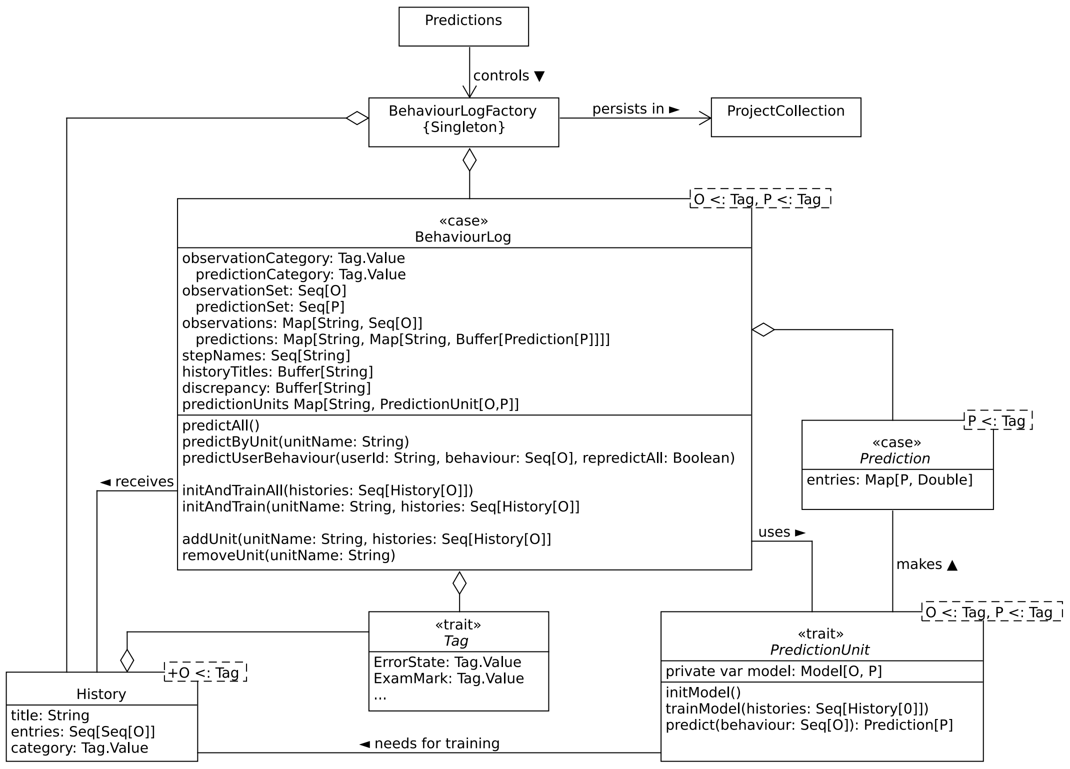

Predictions
controls ▼
BehaviourLogFactory {Singleton}
persists in ►
ProjectCollection
O <: Tag, P <: Tag
«case» BehaviourLog
observationCategory: Tag.Value
predictionCategory: Tag.Value
observationSet: Seq[O]
predictionSet: Seq[P]
observations: Map[String, Seq[O]]
predictions: Map[String, Map[String, Buffer[Prediction[P]]]]
stepNames: Seq[String]
historyTitles: Buffer[String]
discrepancy: Buffer[String]
predictionUnits Map[String, PredictionUnit[O,P]]
predictAll()
predictByUnit(unitName: String)
predictUserBehaviour(userId: String, behaviour: Seq[O], repredictAll: Boolean)
initAndTrainAll(histories: Seq[History[O]])
initAndTrain(unitName: String, histories: Seq[History[O]]
addUnit(unitName: String, histories: Seq[History[O]]
removeUnit(unitName: String)
◄ receives
P <: Tag
«case» Prediction
entries: Map[P, Double]
uses ►
makes ▲
«trait» Tag
ErrorState: Tag.Value
ExamMark: Tag.Value
...
+O <: Tag
History
title: String
entries: Seq[Seq[O]]
category: Tag.Value
O <: Tag, P <: Tag
«trait» PredictionUnit
private var model: Model[O, P]
initModel()
trainModel(histories: Seq[History[0]])
predict(behaviour: Seq[O]): Prediction[P]
◄ needs for training

Figure 4.1: Overview of the main components and their relations in Dostert's original prediction framework. Included members are all public unless specified differently.

`Prediction`s. A **`Prediction`** in turn is simply a map from all the the **`Tag`**s of the prediction type to a **`Double`** value, which usually represents the predicted probability for that **`Tag`** (7). **`PredictionUnit`**s may be provided with a prediction set to restrict the **`Tag`**s contained in the output **`Prediction`** (6).

Finally, a major technical challenge Dostert faced was the (de-)serialization of the case class **`BehaviourLog`**. For that purpose it is not sufficient to implement a generic case class **`BehaviourLog`**, as the generic type information is stripped at compile-time and thus not available during runtime. However, (de-)serialization is performed at runtime and thus the right method cannot be selected with the generic type information missing. Furthermore, **`TypeTag`**s are also not a viable solution. They are a non-experimental part of the Scala Reflection Library [17, scala.reflect.api.TypeTags] and preserve generic type information at runtime. Of actual interest is not just the type, but the particular class of which to create instances. Therefore, Dostert introduced an enumeration **`TagCategoryCollection`** of all **`Tag`**s that connects each entry with a suitable (de-)serialization method yielding instances of the correct class. Conclusively, **`BehaviourLog`**s store the prediction and observation type in two places. First, two generic types determine what the **`BehaviourLog`** should be viewed as, which is typically just **`BehaviourLog[Tag, Tag]`** without specifying a more specific **`Tag`** type. Second, two fields store an entry of the **`TagCategoryCollection`**, which define the (de-)serialization method for observations respectively predictions. Finally, for the following implementation the enumeration has been moved from **`TagCategoryCollection`** to the trait **`Tag`**, which used to be nothing but a declaration anyway. Instead of the long **`TagCategoryCollection.Value`** the class chart from figure 4.1 already uses the much shorter but equivalent **`Tag.Value`**.

4.2.2 Limitations

For all the remaining requirements not mentioned in the previous section there are at least partial limitations with the original prediction framework. For requirement 7 (see [6]) the existing **`Prediction`** implementation works perfectly for the new prediction model but no value has been defined for unavailable predictions. *NaN* could be used for that purpose but is not supported for serialization and by the user interface.

Requirement 3 (see [7]) creates a conflict with a concept named step in the **`BehaviourLog`**. A step is merely an index which can be assigned a name for the user interface. Typical names are weeks or exercises, since steps represent exactly that. The behaviour of any user stored at the index $i \in \mathbb{N}_0$ is said to belong to the step i and the behaviour is displayed alongside the corresponding name in the user interface. Based on that, incremental predictions have been implemented, where only behaviour up to the index i is provided to the **`PredictionUnit`**. Since for the new prediction model multiple observations can be collected for one step either the concept of steps is not applicable or multiple observations must be combined into a single entry of the behaviour sequence.

The second option leads to problems with the storage space as from requirement 8 (see [8]). Assuming b_i is an entry of the behaviour sequence and $b_i \in O$ where O is just some set of valid entries. Now when we combine multiple entries into one we have an entry $\hat{b}_i \subseteq O$. Consequently, to reestablish the original semantics of O, that is $\hat{b}_i \in O$, we need the much larger power set $\mathscr{P}(O)$ resulting in $\hat{b}_i \in \mathscr{P}(O)$. Since storing the power set without any additional information gain is not desirable, a better solution would be to have flexible semantics with either $b_i \in O$ or $b_i \subseteq O$.

This leads us straight to requirement 4 (see [9]) since the information about the validity of observations is not yet stored throughout in Dostert's implementation: A comparison with figure 4.1 shows that the **`History`** does not yet include an observation set used to store that information. Furthermore, analysing the code of the framework reveals that the observation set from **`BehaviourLog`**s is only provided to the user interface but not to **`PredictionUnit`**s.

Next, requirement 5 (see [10]) is not yet of much interest. Essentially, we need any source which provides tags to guarantee object equality between tags that should represent equal observations or predictions. Otherwise the current semantics of the prediction framework do not consider them to be equal and the resulting predictions are not as desired.

Dostert's implementation uses in the **`PredictionUnit`**s and **`BehaviourLog`**s already observation sets to document all the observable observations regardless of their occurrence. But this information is stored independently from each other, and particularly the observation set of a **`BehaviourLog`** is not passed to a **`PredictionUnit`** alongside the behaviour when requesting a **`Prediction`**. Instead every **`PredictionUnit`** uses its own prediction and observation set. However, since with the new implementation **`Tag`**s are not sealed nor all their possible instances known at compile-time, it is impossible for the developer of a **`PredictionUnit`** to define either set for such **`Tag`**s. As **`BehaviourLog`**s are expected to be modified at runtime, their observation and prediction set may be updated as well and passed on to a **`PredictionUnit`** when requesting predictions.

Apart from these aforementioned limitations the existing prediction framework with *Backstage 2* remained applicable to the new prediction model.

[6] *Requirement 7:* Predictions for a label are either assigned an anticipated rating or no rating at all in case a prediction was not possible.

[7] *Requirement 3:* Students can have a record of multiple distinct observations for one exercise.

[8] *Requirement 8:* The storage space required by the data types used should remain within sensible bounds.

[9] *Requirement 4:* Not every label can be recorded for any exercise, that is some of their combinations do not form a valid observation.

[10] *Requirement 5:* Used labels and exercises may change throughout lectures but remain identifiable if unchanged.

4.3 Framework enhancements

Before describing the details of the prediction model as a new `PredictionUnit` this section gives a summary of the general enhancements to the framework. Most of them come straight from the list of limitations and thus from the specification. In the case of the task-based prediction interface (see section 4.3.5) the implementation was not absolutely necessary but greatly benefited performance (possibly for future implementations too) with reasonable effort. In any case great care was taken with all the enhancements to remain compatible with the existing framework and all the already implemented `PredictionUnit`s. Furthermore code reuse was maximized and clear responsibilities aimed for to provide a solid foundation for future enhancements. The only incompatible change to the database schema is described with section 4.3.2 and located in the `History`.

4.3.1 Labels as Tags

The first step when implementing a new `PredictionUnit` is to implement at least one new `Tag` to represent the input (behaviour) and output (predictions) of the unit, unless a suitable `Tag` already exists. For this thesis the three new tags `Label`, `AttributedLabel` and `AttributedLabelSeq` shown in figure 4.2 have been added to the framework.

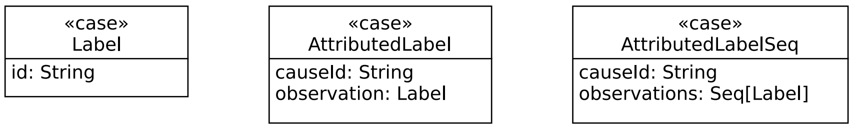


Figure 4.2: Implementations of the trait `Tag` used by the new `PredictionUnit`

The class `Label` is used nowhere else but in the definition of the two other `Tag`s. It only consists of a single `String` that defines the equality relation between `Label`s. Therefore, unlike storing a `String` directly with the other two `Tag`s, in the future more information could be added to `Label`s. For now its only `id` field may be used arbitrarily as long as requirement 5 (see [11]) is met in respect to the equality relation. In Scala terms `equals` and not `eq` [18, scala.AnyRef]. `AttributedLabel`s represent exercise-label pairs as per requirement 2 (see [12]). Since more generally a label is to a cause such as an exercise they have been named `AttributedLabel`. Predictions with the proposed model are of this type. Finally, `AttributedLabelSeq` is used with the `BehaviourLog`, such that only one instance of it is stored for each step.

4.3.2 Observation and prediction set

The concepts of observation and prediction set have already been introduced in section 2.3 respectively section 3.2.2. However, the matter of implementation had not yet been discussed in addition to the necessity of the observation set. The definition of both sets remains unchanged

$$O = \{(e,l) \mid \text{label } l \text{ is observable for exercise } e\}$$
$$P = \{(e,l) \mid \text{predictions for observation } (e,l) \text{ are of interest}\}$$

Prediction set The benefits of a prediction set are obvious as it allows to reduce the runtime cost of predictions pertaining to memory and CPU cycles. Its original implementation

[11] *Requirement 5:* Used labels and exercises may change throughout lectures but remain identifiable if unchanged.
[12] *Requirement 2:* Labels and exercises are viewed as one unit and called observation as previously in this thesis.

remained almost unchanged. With the new framework the generic `predict` method of the trait **`PredictionUnit`** only ensures that every entry from the observation set is contained in the resulting **`Prediction`** returned by one of its implementations. Missing entries are just mapped to *NaN*.

Since the default JSON serialization format of the Play framework does not include support for **`Double`** values the new format from listing 1 was added. It received the distinctive name `naNDoubleFormat` and maps *NaN* to the exact same string. To prevent incompatibilities its implicit declaration has been wrapped by the object **`NaNDoubleFormat`** such that only an explicit import of that object replaces the original format. It is so far only imported by the format for **`Prediction`**s. Furthermore the value `originalDoubleFormat` ensures access to the original `Format[Double]` regardless of the import.

```scala
private val originalDoubleFormat = implicitly[Format[Double]]
object NaNDoubleFormat {
  implicit val naNDoubleFormat: Format[Double] = new Format[Double] {
    override def reads(json: JsValue): JsResult[Double] = json match {
      case JsString("NaN") => JsSuccess(Double.NaN)
      case _ => originalDoubleFormat.reads(json)
    }
    override def writes(o: Double): JsValue = o match {
      case x if x.isNaN => JsString("NaN")
      case _ => originalDoubleFormat.writes(o)
    }
  }
}
```

Listing 1: Implementation of a Play **`Format`** for (de-)serialization of **`Double`**s including *NaN*

The frontend, which already employs the MVVM pattern [40, pp. 27–31], was extended using a variant of the Decorator pattern [16, pp. 70–73] implemented by AngularJS. Essentially, the View uses a decorator to intercept and convert data from the ViewModel. Thus with AngularJS a *filter* named `probability` was introduced. It is used to display **`Prediction`**s and either outputs the **`Double`** value as a percentage or -/-% in the case of *NaN*. In summary, the combination of the new **`Format`** and *filter* fulfils requirement 7 (see [13]) and additionally can be easily extended to new use cases like **`Infinity`** due to the application of established software patterns.

Observation set Next, for the observation set we need to understand how the prediction framework persists observations as behaviour. It is stored per student as a sequence where every entry corresponds to a step (such as week or exercise)

$$(b_1, b_2, \ldots, b_n)$$

and where every entry $b_i \subseteq O$ (the observation set), since per requirement 3 (see [14]) any number of observations can be observed for one exercise and consequently also for one step. The semantic is given by

$$l \in b_i \iff \text{label } l \text{ was observed for step } i$$

This representation does not explicitly store labels, which have not been observed. For the prediction model it would be most straightforward to explicitly store non-observed labels,

[13] *Requirement 7:* Predictions for a label are either assigned an anticipated rating or no rating at all in case a prediction was not possible..

[14] *Requirement 3:* Students can have a record of multiple distinct observations for one exercise.

too, similar to the rating matrix. Then every entry of the behaviour sequence is instead given by $\hat{b}_i \in \{0,1\}^m$ with the same semantics defined for the rating matrix.

However, the option $\hat{b}_i$ was rejected due to two main disadvantages. First, explicitly storing every entry would most likely increase the persistent storage required. During prediction a matrix representation is used anyway so the runtime memory usage remains unaffected. Second and more importantly, as assignments are corrected to incorporate new observations continuous updates of the behaviour sequence are necessary. Particularly, updating a vector $\hat{b}_i$ requires first an index lookup and only then the observation is added by flipping the bit at that index to 1. This is still perfectly possible, but issues arise once the lookup table must be updated with new labels. Because then all the existing vectors $\hat{b}_i$ need to be updated correspondingly, which, when performed often, can be costly and is easily done wrong corrupting the data.

So with b_i chosen instead one could believe that an observation set is redundant, because every label is explicitly stored. Indeed, the observation set can be reconstructed from all the collected behaviour but this reconstruction could miss possible but non-observed behaviour. The reconstruction would lack a label l where $l \notin b_i \, \forall i$, i.e. no students' behaviour sequence contains the label l. That is we have a reconstructed observation set O with $l \notin O$. This is relevant because at some point the lecture, from which the behaviour was collected, ends, and from now on the behaviour could be used as training data to make predictions. Indeed, now with another lecture there could be a student s in the runtime data that actually received the label l. To relate the training data to the runtime data both datasets must be reduced to the common dimensions of their feature space, i.e. their common labels. That common feature space will not contain a dimension for label l since we assumed no reference of it exists in the training data. Therefore the information that the one student s differs from all the students in the training data by the label l is lost. In conclusion the implementation needs to maintain and persist an observation set with **`BehaviourLog`** and **`History`** such that the new implementation of a **`PredictionUnit`** can use that information to correctly determine the common feature space between **`BehaviourLog`** and **`History`**.

Finally, one of the **`Tag`**s, the **`AttributedFeedbackSeq`**, has a peculiarity when used as a member of the observation set in respect to its usage and semantics. As introduced in figure 4.2 the **`AttributedFeedbackSeq`** is a tuple of one `causeId` and a set of labels. First of all, the observation set should only contain one **`AttributedFeedbackSeq`** instance for each `causeId`. Second, that one object has power set semantics to circumvent storing an actual power set as the observation set. This means that for two instances of the **`AttributedFeedbackSeq`** which are given by $t_1 = (\texttt{causeId}_1, \texttt{observations}_1)$ and $t_2 = (\texttt{causeId}_2, \texttt{observations}_2)$

$$t_1 = t_2 \iff \texttt{causeId}_1 = \texttt{causeId}_2 \wedge (\texttt{observations}_1 \subseteq \texttt{observations}_2 \vee \texttt{observations}_2 \subseteq \texttt{observations}_1)$$

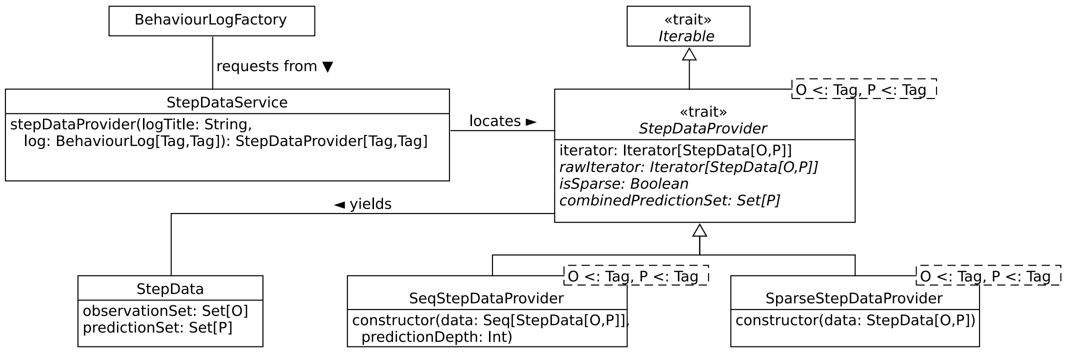
Since object equality cannot be defined depending on the context this has been documented with the **`AttributedLabelSeq`**. In fact with the current implementation such subset comparison is not required, because the implemented **`PredictionUnit`** simply explodes the input **`AttributedLabelSeq`** instances into **`AttributedLabel`**s, which store just one `observation` per instance and hence require no power set semantics. This is generally the proposed solution since the **`AttributedLabelSeq`** was only implemented to store multiple observations for one step as per requirement 3 (see [15]). In the future this problem could be circumvented altogether by enhancing the backend and frontend of the prediction framework to support the aggregation of multiple **`Tag`**s for one step in general. Since this presents a quite fundamental assumption with the prediction framework, many parts of the framework would

[15] *Requirement 3:* Students can have a record of multiple distinct observations for one exercise.

have to be significantly modified or even completely redesigned, which was not acceptable for this thesis.

4.3.3 Step data provisioning

As per requirement 4 (see [16]) valid observations and hence necessarily interesting predictions change with the exercise. This may be nicely observed with the extract from the dataset in figure 2.1 on page 7. These changes of the prediction set could be derived from the overall observation and prediction set introduced in the previous section. They summarize this information for all exercises and thus all steps but by defining the relationship between steps and exercises those two sets could be used as the foundation. So either every exercise constitutes its own step or multiple exercises (such as from one week's assignment) are combined into one step. But since a `BehaviourLog[Tag,Tag]` does not know the specific type stored in its observation set, it cannot (type-safely) access the exercise, i.e. `causeId`, stored with its entries of just the plain type `Tag`s.



BehaviourLogFactory
requests from ▼
StepDataService
stepDataProvider(logTitle: String, log: BehaviourLog[Tag,Tag]): StepDataProvider[Tag,Tag]
locates ►
«trait» Iterable
O <: Tag, P <: Tag
«trait» StepDataProvider
iterator: Iterator[StepData[O,P]]
rawIterator: Iterator[StepData[O,P]]
isSparse: Boolean
combinedPredictionSet: Set[P]
◄ yields
StepData
observationSet: Set[O]
predictionSet: Set[P]
O <: Tag, P <: Tag
SeqStepDataProvider
constructor(data: Seq[StepData[O,P]], predictionDepth: Int)
O <: Tag, P <: Tag
SparseStepDataProvider
constructor(data: StepData[O,P])

Figure 4.3: Overview of the components for Step Data provisioning. Included members are all public and the generic types `O <: Tag` as well as `P <: Tag` have been left out. Members in italics are abstract.

The solution is summarized in figure 4.3 and the new trait `StepDataProvider` constitutes the core component. A `StepDataProvider` is an `Iterable` whose `Iterator` yields `StepData` to be sequentially used for each step. Its users do not need to call `hasNext()` since the iterator does not terminate and just returns empty `StepData` when the `rawIterator` of an implementing class terminates. That way responsibilities are clearly defined. Specifically, the `StepDataProvider` manages the provision of `StepData` whereas the `BehaviourLog` is independent from those implementation details and just uses `StepData` wherever needed such as to initiate prediction. For now `StepData` contains an observation and a prediction set for the corresponding step. For empty `StepData` both sets are also just empty.

Two implementations for `StepDataProvider`s are available. On the one hand, the `Iterator` of a `SparseStepDataProvider` returns the same `StepData` infinitely for every step. On the other hand, a `SeqStepDataProvider` is initialized with a `Seq[StepData]` and a `predictionDepth` of type `Int`. It offsets the observation sets one step. For example when it is initialized with a `Seq[StepData]` given by

$$((O_1, P_1), (O_2, P_2), \ldots, (O_n, P_n), (\emptyset, P_{n+1}))$$

its `iterator` instead yields the following sequence

$$((\emptyset, P_1), (O_1, P_2), (O_2, P_3), \ldots, (O_n, P_{n+1}), (\emptyset, \emptyset), \ldots)$$

[16] *Requirement 4:* Not every label can be recorded for any exercise, that is some of their combinations do not form a valid observation.

This example additionally assumes `predictionDepth = 1` otherwise for every step the prediction sets of the `predictionDepth - 1` subsequent steps are joined into one prediction set. This allows to easily change how far into the future predictions are computed without modifications to the underlying **`StepData`**.

Another class **`StepDataService`** handles the requests for **`StepDataProvider`**s by the **`BehaviourLogFactory`** given a **`BehaviourLog`** alongside its `logTitle`. That `logTitle` origins in the configuration of available **`BehaviourLog`**s as described in Dostert's master thesis [13, Framework Extension]. The relevant key is `behaviourLogs` in the configuration file `application.conf`. With the current implementations the **`StepDataService`** chooses from two alternatives. If no configuration is available for the **`BehaviourLog`** based on its `logTitle`, it returns a **`SparseStepDataProvider`** initialized with the observation and prediction set of the **`BehaviourLog`**. This is consistent with the original prediction framework of Dostert. Otherwise the `application.conf` contains an entry `stepData.<logTitle>`. In that case that configuration object is parsed and used to initialize and return a **`SeqStepDataProvider`**. A commented example configuration is provided with the `application.conf` and has been added on page 48 to the appendix. Furthermore, listing 2 shows the beginning of the configuration used with the dataset from chapter 2.

```
stepData."Label Prediction" {
    predictionDepth = 2,
    steps = [ // derived from files/complete_record_fsk_16ss.csv
      {
        id = "u-0-1"
        observations = ["c-1", "c-2", "c-3", "c-4", "c-5", "c-6"]
      }
      {
        id = "u-0-2"
        observations = ["c-1", "c-2", "c-3", "c-4", "c-5", "c-6", "c-7"]
      }
```

Listing 2: Beginning of the Step Data configuration for the dataset from chapter 2 from the file `conf/application.conf`

In the future the **`StepDataService`** could be extended to actually just read `causeId`s from the configuration. Then the fields `observationSet` and `predictionSet` of the class **`BehaviourLog`** may be split into sets for the corresponding steps by selecting the one **`AttributedLabelSeq`** with the configured `causeId`.

4.3.4 Apache Spark backend

The implementation of the prediction model and great parts of the evaluation use Apache Spark[17], a cluster-computing "engine for large-scale data processing" with APIs for Scala, Java and R programming languages. For this thesis the at that time latest version 2.2.0 was used [2]. Apache Spark is built around Hadoop[18] and its MapReduce functionality, which it extends with caching functionality for acyclic dataflows and over 80 functional high-level operators such as `map`, `flatMap`, `groupBy`, `aggregate` or `union`. For their implementation Apache Spark uses an execution engine based on directed acyclic graphs. They operate on **`Dataset`**s or resilient distributed datasets (**`RDD`**), whereas for this thesis mainly the former is used since this newer API is meant to replace the older one. A **`Dataset`** is either weakly typed as a **`Dataset`**`[`**`Row`**`]`, which is then referred to as **`DataFrame`**, or strongly typed, for

[17]`https://spark.apache.org/`
[18]`https://hadoop.apache.org/`

example as `Dataset[(Int, String, Double)]` or `Dataset[Model]` where `Model` must be a case class for Scala. While the weakly-typed `DataFrame`s implement an SQL-like API, on strongly typed `Dataset`s, typed Scala functions can be executed using the aforementioned high-level operators similar to the Scala Collections API. However, to optimize and distribute the computation `Dataset`s are evaluated lazily computing results only when needed comparable to a `Stream` or `View` in Scala.

Furthermore, Apache Spark ships with a machine-learning library implementing two APIs: *ML* based on the `Dataset`-API and *MLlib* for the `RDD`-API. With this thesis exclusively the former *ML*-API is used. Since recommender systems typically involve large numbers of ratings and ALS-WR describes a parallelizable matrix factorization method and Apache Spark implements a Scala API, it was the logical choice for the implementation of the proposed prediction model. Indeed, the machine-learning API of Apache Spark already includes an implementation of ALS with support for implicit and explicit ratings. Section 3.3 essentially fomalized that implementation for the case of explicit ratings.

Apache Spark proved most invaluable in the evaluation, which required several thousand factorizations and countless predictions. In the future the rating matrix could easily grow both in the number of students (only 82 out of 345 were selected) as well as in the number of labels (so far 27). In that case Apache Spark can provide the means to scale the proposed and possibly other prediction models not just to multiple processor cores but to a whole cluster. Obviously there are limitations since the parallelization relies on distributing the finite number of factors. For the dataset used for this thesis increasing the number of partitions, by which the factors are distributed, well beyond 40 actually increased the runtime. Most probably this is due to the amount of data exchanged between Apache Spark instances, for which an increase in no proportion to the size of the dataset was observed. To counteract this effect multiple steps or `PredictionUnit`s with fewer partitions could be run in parallel, which could be easily implemented with the prediction interface from the next section.

In any case Apache Spark is not necessarily just used with this prediction model but possibly in other places of *Backstage 2* such as further `PredictionUnit`s. *Backstage 2* uses the Guice[19]-based dependency injection provided by the Play framework to make *services* available to other components (including other services). This solution has two advantages. First, services can be easily swapped for another implementation such as a mock. Second, services are loaded as part of a module, which can be enabled or disabled at will. Since Apache Spark requires a server to connect to (or depending on its configuration automatically starts one), it is preferably left disabled until needed. Furthermore, only one `SparkSession` may be started inside a single Java Virtual Machine (JVM) and that session is otherwise reused and the configuration shared. Therefore Apache Spark has been integrated with Play as a very simple, injectable service named `SparkService` loaded by the `SparkModule`. Since that service is a singleton it ensures that only one `SparkSession` with consistent settings is started for the whole *Backstage 2* application. In theory the `SparkModule` would only need to be enabled when a `PredictionUnit` requires it. In practice `PredictionUnit`s are not loaded using injection and hence cannot be injected dependencies themselves. For that reason the `BehaviourLogFactory` is injected with the `SparkService` instead. It registers the `SparkService` with the `PredictionUnitFactory`, which is declared as a Scala `object`. Therefore `PredictionUnit`s can request the `SparkService` on demand by `PredictionUnitFactory.sparkService`. The `SparkService` stores the `SparkSession` with a lazy value such that a session is only started once needed for the first time. Consequently, Apache Spark is only started when the `FeedbackPrediction` `PredictionUnit` is used for the first time.

For the configuration of the `SparkSession` different options are available. First of all,

[19]https://github.com/google/guice

```
spark = {
  master = "local[*]"
  app.name = "Crowdlearning platform"
}
```

Listing 3: Configuration for Apache Spark in the file `conf/application.conf`

the key `spark` in the file `application.conf` is read by the **`SparkService`** and passed to a **`SparkSession`** during initialization. It only supports the two most important options `master` and `app.name`. The former identifies (the master of) the cluster to connect to. It can be set to `local` to automatically start a local cluster with one worker and to `local[k]`, where *k* workers are started. For `local[*]` *k* is set to to the number of cores on the local system. URLs such as `spark://...` for a standalone cluster or `yarn://...` are used otherwise for separately deployed cluster. The second key `app.name` is just used as a human-readable identifier in the cluster frontend. Further options may be configured using Java system properties[20].

Finally, the machine-learning library requires a netlib implementation, which provides various optimized vector and matrix operations. For that purpose the netlib-java[21] wrapper is used. It ships with a slower pure Java implementation and faster native reference implementations for common 32- and 64-bit architectures as well as operating systems. Instructions to configure native system implementation with even better performance (such as ATLAS, openBLAS or Intel MKL) is available with the GitHub repository of netlib-java. For this thesis the undocumented blis[22] implementation with support for recent AMD processors was also successfully used. However, such configuration is optional and the implementation provided with this thesis should work on any system with sufficient memory without further configuration changes falling back to the Java implementation in the worst case. When a **`SparkSession`** is started the actually loaded implementation is logged with the **`DEBUG`** level configured.

4.3.5 Task-based prediction interface

One shortcoming with the ALS-WR implementation of Apache Spark is that the student and observation factors resulting from the factorization cannot be reused. Otherwise the training data could be factorized in advance and during prediction only the runtime data generated from the **`BehaviourLog`** would still require factorization. Modifying the ALS implementation of Apache Spark could resolve this issue but unless maintained upstream could break with future releases of Apache Spark.

Since presently with the prediction framework performance is not of great importance a different solution was implemented to limit the scope of this issue. Dostert's implementation of the prediction framework provided with the trait **`PredictionUnit`** only an interface to predictions for each student individually. Therefore the runtime data used in factorization is only the behaviour sequence of a single user. Therefore for *n* users *n* factorizations would be performed for every single step. The number of students *n* could grow very large despite the support to make predictions for multiple users simultaneously. The obvious solution is to extend the existing interface to simultaneously predict multiple users.

For that purpose a requested prediction for one user is viewed as a **`PredictionTask`**. Every **`PredictionTask`** is used to request predictions for multiple users at once using a common observation and prediction set. The original trait **`Model`** was refactored to

[20]`https://spark.apache.org/docs/latest/configuration.html`
[21]`https://github.com/fommil/netlib-java`
[22]`https://github.com/amd/blis`

`TasklessModel` and superseded by the new task-based interface named **`Model`**. In one trait the **`TasklessModel`** implements a wrapper for the new interface while specifying the old interface. Since one interface is converted to another its an application of the Adapter design pattern [16, p. 66f.]. Figure 4.4 gives an overview of the components of the new task-based prediction interface.

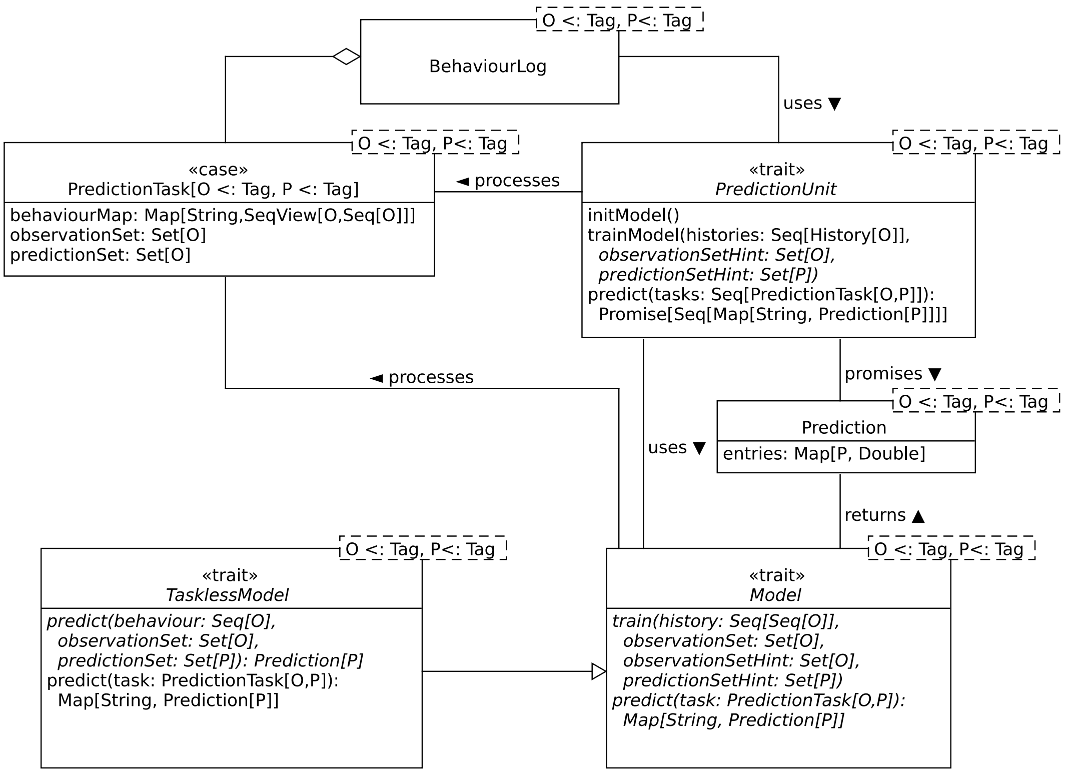

O <: Tag, P<: Tag
BehaviourLog
uses ▼
«case»
PredictionTask[O <: Tag, P <: Tag]
behaviourMap: Map[String,SeqView[O,Seq[O]]]
observationSet: Set[O]
predictionSet: Set[O]
◄ processes
«trait»
PredictionUnit
initModel()
trainModel(histories: Seq[History[O]],
observationSetHint: Set[O],
predictionSetHint: Set[P])
predict(tasks: Seq[PredictionTask[O,P]]):
Promise[Seq[Map[String, Prediction[P]]]]
◄ processes
promises ▼
Prediction
entries: Map[P, Double]
uses ▼
returns ▲
«trait»
TasklessModel
predict(behaviour: Seq[O],
observationSet: Set[O],
predictionSet: Set[P]): Prediction[P]
predict(task: PredictionTask[O,P]):
Map[String, Prediction[P]]
«trait»
Model
train(history: Seq[Seq[O]],
observationSet: Set[O],
observationSetHint: Set[O],
predictionSetHint: Set[P])
predict(task: PredictionTask[O,P]):
Map[String, Prediction[P]]

Figure 4.4: Overview of the interface and components with the new task-based prediction interface. Italic members are abstract.

In the current implementation the **`BehaviourLog`**, upon request from the **`Predictions`** controller, collects a sequence of tasks for every step. Every **`PredictionUnit`** receives that sequence containing only references to immutable object and returns a Promise for the result. Since every **`PredictionTask`** is a request for multiple users the result contains an entry for every step that maps user identifiers to the computed **`Prediction`**. Since **`PredictionUnit`**s are independent and a **`PredictionTask`** only references immutable objects running the predictions for all **`PredictionUnit`**s in parallel is unproblematic. For each **`PredictionUnit`** further parallelization of multiple **`PredictionTask`**s or of the predictions for multiple users from one task remains the responsibility of the implementation. The reason is that the implementation of a **`PredictionUnit`** is not necessarily stateless (at least the trained model is currently stateful). Thus only individual implementations can make the necessary guarantees for parallelization. Finally, in the current implementation a **`Model`** is only responsible for the synchronous prediction of individual tasks instead of a sequence of tasks as for the **`PredictionUnit`**. This simplifies the implementation of the model while code from the **`PredictionUnit`** may additionally be reused for different models. Unchanged is that a **`Model`** also represents a prediction model generated during training. Hence future implementations could for example opt to persist them to avoid repeated training.

4.3.6 Serialization

Here an improved (de-)serialization solution for the case class **BehaviourLog** is proposed. It is implemented as a **Format**, which is a trait used for (de-)serialization by the Play framework. As shown in figure 4.5 every **Format** is a **Reads** and **Writes** using mix-in traits. **Reads** and **Writes** define abstract methods `reads` respectively `writes` and come with some more non-abstract members such as `map`, which "[c]reate a new **Reads** which maps the value produced by this **Reads**" [1, play.api.libs.json.Reads]. Furthermore, for any non-generic case class Play can generate the correct **Format** implementation using Scala compile-time macros. This presumes that for every member of type `A` a `Format[A]`, `Writes[-A]` or `Reads[A]` exists. To locate a `Format[A]` of suitable type `A` implicit parameters are used. When such an implicit parameter is left undefined, the Scala compiler tries to locate an implicit declaration with matching type `A`.

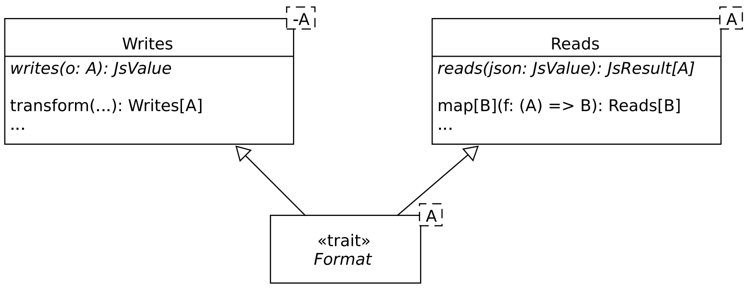

Writes
-A
writes(o: A): JsValue
transform(...): Writes[A]
...
Reads
A
reads(json: JsValue): JsResult[A]
map[B](f: (A) => B): Reads[B]
...
«trait»
Format
A

Figure 4.5: Format interface used for (de-)serialization to JSON by the Play framework

Dostert's prediction framework does not implement `Format[BehaviourLog[Tag,Tag]]` but triggers the autogeneration of a `Format[SerializedBehaviourLog]`. Unlike what the name **SerializedBehaviourLog** suggests it is only partially serialized. Effectively, Dostert implemented a two step serialization, where first every **Tag** is converted into a **JsValue** and stored in the **SerializedBehaviourLog**, which is no **JsValue** itself. Since all **Tag**s are now of the common type **JsValue** the `Format[SerializedBehaviorLog]` is autogenerated using the macro provided by Play. Second, this macro-generated format fully serializes the partial serialization from step one into a single **JsValue**. This solution works, but when the signature of the **BehaviourLog** is change, the lines responsible for the (de-)serialization of one specific field are not easily identified. Furthermore, the serialization and deserialization code are almost identical but need to be implemented redundantly in different methods.

The proposed new serialization solution aims to solve those problems. For the redundancy between serialization and deserialization Play provides a solution using **JsPath**, that allows to define symmetric serialization and deserialization in one place. Listing 4 shows that part of the new serialization solution. It is part of the class **BehaviourLogFormat**, which implements a `Format[BehaviourLog[Tag,Tag]]` and thus renders Dostert's class **SerializedBehaviourLog** unnecessary. When the signature of **BehaviourLog** is changed only lines 37 to 52 from listing 4 require modification, where every single **JsPath**, such as `(__ \ "observationSet")`, is responsible for exactly one field. For one **JsPath** the association with a **Format** is established by a call to `format[T](implicit f: Format[T])`, which yields an **OFormat**, that, using implicit conversions, may be coerced to **InvariantFunctor** or to **InvariantFunctorOps**. These in turn implement a bidirectional map given by `inmap[B](f: (A) => B, g: (B) => A): M[B]`. Lines 47 to 51 of listing 4 show an application of `inmap`, where a bidirectional mapping between a **Buffer** and a **Map** is established.

Finally, how does the **BehaviourLogFormat** solve the problem that the specific type of **Tag**s and thus the needed `Format[A <: Tag]` is unknown at compile-time? This is provided by lines 33 to 36 from listing 4. Listing 5 shows their application in the `reads`-implementation

```
33 private val behaviourLogTypeFormatBuilder = {
34   (__ \ "observationCategory").format[Tag.Value] and
35   (__ \ "predictionCategory").format[Tag.Value]
36 }
37 private def logFormatBuilder(ot: Tag.Value, pt: Tag.Value) = {
38   val (of, pf) = (Tag.format(ot), Tag.format(pt))
39   behaviourLogTypeFormatBuilder and
40   (__ \ "observationSet").format(seqFormat(of)) and
41   (__ \ "predictionSet").format(seqFormat(pf)) and
42   (__ \ "observations").format(stringMapFormat(seqFormat(of))) and
43   (__ \ "predictions").format(stringMapFormat(stringMapFormat(
     ↪  bufferFormat(predictionFrontendFormat(pt)))))
     ↪  and
44   (__ \ "stepNames").format[Seq[String]] and
45   (__ \ "historyTitles").format[Buffer[String]] and
46   (__ \ "discrepancy").format[Buffer[String]] and
47   (__ \ "predictionUnits").format[Buffer[String]]
48     .inmap[Map[String, PredictionUnit[Tag, Tag]]](
49     _.map(name => name -> PredictionUnitFactory.getUnitForName[Tag,
       ↪  Tag](name))(collection.breakOut),
50     _.keys.to
51   )
52 }
```

Listing 4: Definition of the **`Format`**`[`**`BehaviourLog`**`[`**`Tag,Tag`**`]]` using **`JsPath`** where a **`Format`**`[`**`A <: Tag`**`]` for the observation and prediction category is looked up at runtime

```
86 override def reads(json: JsValue): JsResult[BehaviourLog[Tag, Tag]] = {
87   try {
88     val (ot, pt) = json.as(behaviorLogTypeFormat) // <-- exception
89     return logFormatByCategory(ot, pt).reads(json)
90   } catch {
91     case e: JsResultException =>
92       return JsError(JsError.merge(
93         JsError("could not read tag categories for Behaviour").errors,
94         e.errors
95       ))
96   }
97 }
```

Listing 5: Implementation of `reads` that uses the definitions from listing 4 to build a **`Format`**`[`**`BehaviourLog`**`[`**`Tag,Tag`**`]]` at runtime

of the format. The aforementioned lines are used to construct a format just to read the categories. Applying this **`Format`** to the input JSON yields the required categories. Thereafter the `logFormatBuilder` on line 37 of listing 4 is passed both categories and looks up the correct two (macro-generated) **`Format`**s for observation and prediction categories, which have previously been registered with **`Tag`** for each **`Tag`**`.`**`Value`**, i.e. each category. Eventually, it just chains together the returned **`Format`**s into an overall **`Format`**`[`**`BehaviourLog`**`[`**`Tag,Tag`**`]]`, which is used to read the whole **`BehaviourLog`**. The `writes` implementation works analogously but uses the prediction and observation category from the **`BehaviourLog`** directly, which do not require prior deserialization.

Overall this new serialization solution solves both problems initially identified. First, the less than 20 lines (as compared to originally 50 lines for separate serialization and deserialization) specify serialization and deserialization in one place. Second, the syntax is more expressive and concise. Every **JsPath** is responsible for exactly one field of the **BehaviourLog**. Therefore modifications can be easily propagated to the (de-)serialization implementation and consequently are less error-prone. As of now, this new solution has not been applied to the **History**, which still uses Dostert's analogously implemented **SerializedHistory** for (de-)serialization. The same solution works for the **History** as well and if desirable, could be similarly implemented.

4.4 Prediction model

Lastly the new **PredictionUnit**[**AttributedLabelSeq**, **AttributedLabel**], which was named **LabelPrediction**, connects all the previous parts. The core of its implementation lies with the **Model**[**AttributedLabelSeq**, **AttributedLabel**], which is based on the formalization of the ALS-WR method presented in section 3.3 on page 13. During initialization the **Model** receives a **SparkSession**, which triggers the startup of Apache Spark as described in section 4.3.4. To ensure that data from training can be persisted for prediction with the **SparkSession**, it could not have been set or requested with every call to `train` or `predict`.

4.4.1 Training

A call to `train(history, observationSet, ...)` on the **Model** begins its training. The ellipsized parameters from the call are the `observationSetHint` and `predictionSetHint` and so far not used by any **PredictionUnit**. Their proposed usage is to provide a hint during training about the intended usage of `predict`. Specifically, the **Tag**s from the `observationSetHint` will be (most probably) available as behaviour to `predict`. The `predictionSetHint` defines the predictions that will (most probably) be requested from `predict` as the `predictionSet`. The idea is, that a **PredictionUnit**, which drops parts of the training data, might drop exactly those parts later used in calls to `predict`. For such cases the usage of these two hints is proposed and communicates in advance the anticipated usage of the `predict`- to the `train`-method. Like a `sizeHint` with Scala's class **Buffer** for example [18, scala.collection.mutable.Buffer], implementations should still not fail in case the hint was wrong. If they cannot make a prediction, they shall map it to **NaN** or leave that task to the trait **PredictionUnit** by dropping the entry altogether. With the current implementation observations are never removed, for which reason these hints may be safely ignored and are hence definitely used correctly.

The remaining parameters `history` and `observationSet` are used to train the model. They may already combine more than one **History**, since that generic part is already handled by the trait **PredictionUnit**. The actual training process consists of multiple steps and data is transferred to Apache Spark as early as possible. In the first step the `observationSet` is indexed into a map. For that purpose every **AttributedLabelSeq** is first exploded into a **Seq**[**AttributedLabel**] since predictions about individual observations and not about sets of observations are aimed for. The concatenation of all the exploded sequences defines the feature space for every user. Therefore every entry of type **AttributedLabel** is joined with its index and once transmitted to the **SparkSession** yields the `obsMap` of type **Dataset**[(**AttributedLabel**,**Int**)]. As this map is used several times, and particularly for the reverse mapping during prediction, it is cached with Apache Spark and stored with a field of the **Model** implementation.

In the second step the `history` is converted into `ratings` using the `obsMap`. The `history` is a sequence of `behaviour` whereas every entry is of type **Seq**[**AttributedLabelSeq**]. Indexing every entry of the `behaviour` produces the `user`-ids of type **Int** and transforms the `behaviour` into a `Dataset[(Int, Seq[AttributedLabelSeq])]`. Then the resulting `Dataset[(Int, Seq[AttributedLabelSeq])]` is converted into a `Dataset[Rating]` by using the declaration

```
case class Rating(user: Int, label: Int, rating: Float)
```

For the conversion a `Dataset[(Int,AttributedFeedback,Float)]` is built first. Its first entry is the `user` (i.e. student) and the last entry the `rating = 1.0f` $\hat{=}$ TRUE. This `Dataset` is then joined with the `obsMap` on `AttributedFeedback` into the desired `ratings` of type `Dataset[Rating]`.

In the third step the `Dataset[Rating]` is complemented with the implicit ratings, which is any `AttributedLabel` not observed for a student. They are essentially added using a right outer join with the `obsMap` for the right-hand side and the `ratings` for the left-hand side. Moreover, this step may be disabled, in which case the predicted ratings were observed to be always of approximate value 1.0 since all the explicit ratings input to ALS never take on another value.

In fact, no actual model fitting happens during training since the matrix for the factorization still lacks the observed behaviour provided during prediction. And because the `ALS` implementation of Apache Spark does not allow reusal of the factors from the factorization without modifications. Since modifications not merged upstream could break in the future, this was avoided.

4.4.2 Prediction

A call to `predict(task: PredictionTask[AttributedLabelSeq, AttributedLabel])` requests `Prediction`s for multiple students. Those students are indexed using `Int`s with a offset that does not overlap with the `user`s from the training step. Then each users behaviour is transformed into a `Dataset[Rating]` using the same procedure as during training. For the addition of implicit ratings the `obsMap` is filtered using the `observationSet` from the `PredictionTask` since only the intersection of both sets can be implicitly inferred as not observed. The union of the resulting `Dataset[Rating]` and the one from training is used to fit an `ALSModel`. Since with the Apache Spark implementation predictions are requested by a `Dataset[Rating]` the `predictionSet` is also transformed into one. Lastly, the predictions are requested and the result is reverse transformed using the `obsMap` and split into a `Prediction[AttributedLabel]` for each user.

For optimization the main focus was on the hyperparameters of ALS-WR and hence optimal prediction results. Currently, with a native netlib implementation a call to `predict` returns within acceptable 1-10 seconds mostly depending on the maximum number of iterations for the factorization. Nevertheless, this is still significantly slower than for instance Dostert's HMM model, whose computation for all steps finishes within one second. A first measure to improve the response time was to introduce the task-based prediction model. As a result the aforementiond 1-10 seconds per step are the response time for all the students as opposed to previously per student. For all the 82 or even more students the runtime reduction is therefore already significant. In the future this could be further optimized by modifying the `ALS` implementation of Apache Spark, such that a complete factorization of the training data is performed only once or by increasing the parallelization alongside the Apache Spark cluster size.

CHAPTER 5

Discussion

Every result only becomes meaningful in its application. In the case of this prediction model three possible applications were conceivable from the beginning. First, the application of the prediction model to data from an actually running lecture. Second, the prediction model could reveal unknown properties of the data, i.e. new domain knowledge about relationships between labels. Or third, the general results and experiences may provide useful directions for future research. The following sections discuss them in this order, whereas three, future research, is incorporated throughout.

5.1 Proposed applications

For the practical application of predictions the interdependence between an application and the prediction model is the main concern. Focusing an aspect of either one, such as the performance of the prediction model, is not sufficient. Indeed with recommender systems some instances of 'bad predictions' could even be beneficial to prevent the "filter bubble" phenomenon [8, 22]. For the proposed prediction model performance has already been discussed and what now remains to be identified is a suitable application inherently linked to the performance. In the beginning the overall aim was defined as the personalization of blended learning to reduce dropout rates and to improve the learning outcome. In that sense two general applications of the prediction model have been conceived. It is argued that learning platforms such as *Backstage 2* may either *directly* communicate predicted labels to students or *indirectly* personalize their learning experience based on labels. Starting with the direct case this is a straightforward example:

> Based on your previous assignments: Did you ignore epsilon productions?

This example intentionally leaves much room for criticism such as: Is a student that ignores epsilon productions even capable to know that he ignored epsilon productions?

This question has been discussed as the Dunning-Kruger effect, which observes that "people tend to be blissfully unaware of their incompetence" [15, p. 83]. The interpretation of Dunning and Kruger is that incompetence prevents the recognition of itself as the missing skills are exactly those necessary to recognize the lack of competence. Hence where relevant competences have not yet been acquired by the students, any direct communication of their anticipated mistakes based on labels could be ineffective. Nevertheless, sometimes mistakes only happen carelessly without a fundamental lack of knowledge. For such cases and labels the Dunning-Kruger effect does not apply. In this case, and if the student is willing to act, the above example question could prevent the small mistake. Questionable remains whether a learning outcome would be observable, when just a small, careless mistake is corrected.

Next, by instead assuming the existence of knowledge gaps the question necessarily becomes how to close them. The "power of feedback" is proposed for this purpose. It is an extensive review of research regarding the properties of effective feedback created by Hattie and Timperley [26]. And for this thesis it is argued, that from the student's viewpoint directly and indirectly provided predictions are like feedback "information provided by an agent [...] regarding aspects of one's performance or understanding" [26, p. 81]. And since labels reflect the "performance" or "understanding" of students, validly predicted labels are equally feedback. For effective feedback Hattie and Timperley claim that three questions and notions of feed up, feed back and feed forward should be answered:

Feed Up: Where am I going? (goals)

Feed Back: How am I going?

Feed Forward: Where to next?

Hence they particularly create a distinction between feedback, as the general information provided to a student, and Feed Back, as one part of feedback, which answers only one of the questions. Next, a more detailed description of these questions based on the work of Hattie and Timperley [26] and an interpretation for the prediction context is given.

The question of *Feed Up* is usually answered by the lecture staff and eventually students construct their individual goals from that. These goals are not provided by software in the case of *Backstage 2*.

Feed Back instead is provided by correctors and hence also not by technology. Unlike for Feed Up concepts for software-generated Feed Back exist and also have been used in real-world applications. For example unit tests [13] may "evaluate the correctness of a response" [26, p. 81] to a programming task. And as Hattie and Timperley specifically mention, students may develop strategies to obtain Feed Back themselves. For example for a programming task by anticipating the expected output and comparing it to the actual result output by their own implementation. In large university courses, where students do not receive constant Feed Back, developing these "effective error detection skills" of students seems very desirable. At least if the feedback is applicable and if the knowledge level required for its understanding does not trigger the Dunning-Kruger effect.

Finally, for the question of *Feed Forward* software-generated predictions are proposed for the future and the necessary foundation has been developed with this thesis. The reason is not, that correctors would not be able to provide Feed Forward. Instead one corrector is often responsible for a large number of students. Even with and regardless of a great amount of experience, human working memory remains limited to about half a dozen items [4]. A prediction model may not have the same experience as a corrector, but for computer software combining the labels of any student on any exercise into one prediction is no problem even when the number of students rises to hundreds or thousands. Proof is the prediction model introduced with this thesis. Now for a true Feed Forward the question is "Where to next?", which is not about the present Feed Back coded into the already known labels. Therefore predictions, which anticipate students' Feed Back on future assignments, are required. This is particularly because different assignments typically cover different topics, where just the observed labels from previous exercises can be useless. Predictions build a bridge between such possibly incompatible domains by answering the question, what Feed Back represented as labels to expect next. And since they anticipate future, so far unobserved behaviour, predictions are not actually Feed Back but Feed Forward.

The result can be communicated to students by the aforementioned direct and indirect means. In the direct case future research could evaluate phrases used to present the labels and when, what labels are shown to whom. For example, based on Dostert's HMM prediction model [13] students with insufficient knowledge, or which are expected to skip assignments, could be the most interesting subjects. In the indirect case for every label relevant material could be collected or newly written, which is especially helpful to students with a prediction of that label. For example, students with a predicted label 'Syntax for grammars invalid' could be offered a summary of the syntax or with a select reference to relevant resources. For the effectiveness and the acceptance of any of the previously described direct or indirect Feed Forward four criteria were collected from related research:

1. constructiveness
2. justification
3. specificity
4. exclusivity

Their detailed meanings are described hereafter alongside their origin and the reasoning for their inclusion.

Constructiveness is created with Hattie and Timperley [26] by answering the questions of Feed Up, Feed Back and particularly Feed Forward. Like any part of the feedback Feed Forward may be generated by students themselves if they possess the necessary skills and domain knowledge to do so. Otherwise the Feed Forward can only be provided by another agent in the form of constructive feedback.

Justification is demanded, since predictions will certainly be wrong at some point. As Herlocker [27] observes, humans in general are used to deal with imperfect recommendations and hence also feedback used as such. If doubt about the validity of a recommendation arises, the question 'Why?' will be asked. Hence providing explanations about the origin of feedback (e.g. on exercise 2 your knowledge of implications was insufficient) empowers students to deal with the ultimately never completely avoidable prediction errors themselves. For this reason justification has according to Herlocker become accepted for recommender systems in general. Future research could evaluate possibilities to integrate such explanations with the prediction model based on ALS-WR. For ALS with implicit ratings Hu et al. [30] have already proposed a solution to generate explanations that link predictions to the causative observations.

Specificity guarantees that eventually some prediction will be useful, which is necessary for students to even keep considering the provided Feed Forward. This criterion may be interpreted as the binary prediction metric specificity, as its close relative the precision metric and in respect to the content of the provided Feed Forward. Performance metrics may yield stunning but meaningless results simply just by labels, which represent obvious characteristics of students such as gender or hair colour. Therefore only the combination of all these notions of specificity results in effective feedback, which needs to be "clear, purposeful [and] meaningful" [26, p. 104].

Exclusivity refers to a non-inflational supply of feedback to the student. If too much irrelevant feedback is provided to the individual student, difficulties to isolate the relevant one become the consequence. Furthermore, among relevant feedback the level of relevance for a specific student will vary. Feed Forward asks the question "Where to next?" and not just "Where to?". Exclusive Feed Forward accounts for this by filtering the predictions. For this purpose the MRR presented during evaluation is one useful tool. Based on the ordering defined by the predicted rating on a subset of observations, the rank at which the first relevant prediction appears can be measured. For a MRR above 0.5 like for our prediction model only two predictions need to be presented to students, such that on average one of them is correct. Here, future research would have to evaluate whether *correct* predictions are actually those that produce *relevant* Feed Forward, which requires unobfuscated, meaningful labels in addition to a formative assessment of their effect on students when used as direct or indirect Feed Forward.

5.2 Frequent-pattern mining

To possibly generate new domain knowledge about labels Collaborative Filtering, such as ALS-WR is not the best choice, as the computed factor relationship between students and observations is not explicitly accessible. Particularly, the student and observation factors resulting from the matrix factorization are not easily connected to a specific meaning such as 'insufficient knowledge of ...', 'busy' or 'careless'. This reason, that caused the observed behaviour, is in general unobservable and possibly even to the observed student unknown.

In psychology, what has been referred to as 'reason', is conceptualized as latent factors (also: latent variables). By definition they "[explain] a wide range of behaviour by invoking a limited number of latent variables" [7, p. 203]. Since ALS-WR maps students onto factors to predict (i.e. explain) observations, it is argued that the factor space of ALS-WR is at least conceptually close to latent variables. The factors are not necessarily independent, although the term 'rank' from linear algebra suggests exactly that notion. Regardless of the independence, if a rank is identified, where higher ranks do not yield significantly better results, it constitutes an upper bound estimate of the actual rank, i.e. the number of (independent) latent variables. Therefore and based on the results from table 3.2 on page 20 the proposition is made, that as little as ten latent variables explain the observed behaviour of students for the present dataset. With the above argumentation there may perfectly well be even less latent variables. And without more reliable analysis on more students there are possibly even more than ten. Further research could narrow down the real number of latent factors and new domain knowledge could become the result of an investigation of their true meaning.

Furthermore, different machine-learning techniques do communicate the relationship between students and students or observations and other observations. For example, frequent pattern mining might provide insights into the the aforementioned relationships. For that purpose, first, frequent items (such as labels or observations) and their frequent pairs, triples, ... are identified. Second, frequent patterns such as 'label A always with B' or 'tuple (A, B) always with (B, C)' are identified. Used on observations this could reveal patterns in the dataset.

In figure 2.1 on page 7, which just visualized a small part of the dataset, such patterns were already difficult to spot at a glance. Frequent-pattern mining would not suffer from this problem. Apart from that, frequent-pattern mining may predict like ALS-WR the assignment performance based on these patterns. However, in addition to that, it produces domain knowledge about relations between labels, which is possibly new. For example, a co-occurrence of the labels 'lacks understanding of automatons' and 'translation of the grammar into an automaton is incorrect' seems very obvious. But when frequent pattern mining instead identified the frequent but unexpected co-occurrence of 'lacks understanding of infinity' and 'wrong application of epsilon production', this would be worthwhile to investigate, as a connection between the two is neither implausible nor obvious. As a consequence for a lecture the order of topics may be rearranged or their focus shifted. For example, assuming that 'lacks understanding of infinity' leads to 'wrong application of epsilon production' and that relevant concepts of infinity have been identified, these concepts could be taught before epsilon productions or may be focused differently.

The great advantage of such explicit knowledge produced by some machine-learning techniques is, that not just the predictions but the knowledge itself becomes accessible. This allows to observe and interpret what the model actually learned. Eventually, there is even the possibility, that the application of the knowledge supersedes the prediction.

CHAPTER 6

Conclusion

Initially, three aims have been defined for this thesis. They are in a short summary the prediction model, the implementation and the greater aim of personalization in large blended learning environments.

First, the prediction model based on Collaborative Filtering has been successfully developed using the ALS-WR method. After a comprehensive evaluation of different hyperparameters the best performing ones ($k = 40$, $N = 10$, $\lambda = 0.065$) on the one available dataset were identified using a combination of different metrics (MRR, RMSE and AUC ROC). The resulting final model was evaluated in more detail, particularly when instead of continuous predicted ratings a predicted binary yes-/no-rating is desired. This can be implemented using a threshold, which divides the continuous predicted rating into two ranges, that respectively represent yes or no. The effect of choosing this threshold differently has been visualized by the Receiver Operating Characteristic and Precision-Recall curve. Particularly, they allow to identify a threshold setting that balances Recall, Fall-out and Precision depending on the individual needs of an application.

Overall this thesis executed fundamental research but with a practical orientation. For that purpose a reference implementation of the prediction model was implemented with the blended learning environment *Backstage 2* and used an existing prediction framework developed with the master thesis of Steven Dostert [13]. After defining a specification with the specific requirements of the new prediction model, parts of the framework with relevant limitations were identified. Consequently, several general enhancements to the framework were implemented in addition to a new prediction unit and the necessary data types for label prediction. Particularly, any part of *Backstage 2* may now use the cluster-computing engine Apache Spark designed to handle significantly larger datasets than the ones used so far. Herefore the necessary foundation has already been set by using Apache Spark with the reference implementation of the prediction model.

Finally, to use the predictions from the model first the differences to typical Collaborative Filtering were observed. Usually it is used with recommender systems where the predicted items are not observations but usually for example products, movies or news topics and articles. If labels could be just straightforward recommended, what is the meaning of "I recommend to you 'insufficient knowledge about implications"'? Chapter 5 tried to answer this question and essentially it is just a matter of rephrasing, e.g. to "I recommend to check your assignment for 'insufficient knowledge about implications"'. Ideas how to present this differently and more effectively were also discussed. Eventually this lead to the proposition of four criteria based on research about effective feedback and good recommendations. These criteria propose that feedback derived from predicted labels should guarantee constructiveness, justification, specificity and exclusivity.

To make these guarantees the underlying prediction model is of great relevance albeit not necessarily just its performance. The extensive evaluation of the proposed prediction model using ALS-WR clearly shows a non-marginal predictive value. It has been argued, that by providing 'justification' alongside the computed feedback, students are empowered to judge the relevance and quality of predictions themselves. This can limit the impact of wrong predictions. Since some amount of wrong predictions is unavoidable due to noise in the dataset, and otherwise the development of better prediction models may be expensive, effects of communicating the results differently (e.g. justification) can provide a viable alternative.

In any case the proposed prediction model makes some assumptions about the learning environment. First of all, the labels collected need to be available in some sort of database. Secondly, to make their collection worthwhile to correctors, they are equipped with a software solution such as *Backstage 2*, where for the returned assignments feedback is entered and available labels can be picked from a list. Thirdly, labels must be initially selected and later used with care. Their use should be consistent, as far as possible, and even with the best prediction model only their choice eventually decides the effectiveness of an application. Conclusively, personalized blended learning applications will particularly require further analysis of labels to effectively communicate their predictions to students. While the evaluation of different machine-learning techniques is possible and potentially useful, eventually and effectively the students should be reached.

Appendix

Source code

The source code of the implementation is available with the Git repository of *Backstage 2* at:
`http://gitlab.pms.ifi.lmu.de/niels/cwdl-projects`

Branch:	`feedback-prediction`
Final commit:	`6221c5ae`

Dataset and scripts used for evaluation and figures are available with the Git repository at:
`http://gitlab.pms.ifi.lmu.de/Abschlussarbeiten/zula-andreas-born`

Branch:	`master`
Final commit:	`a495c0f`

Both repositories require prior authentication by a valid PMS account.

Original German examples for labels

Implikationen nicht richtig verstanden

Wenn es eine Grammatik vom Typ n gibt ist die Sprache der Grammatik mindestens vom Typ n. Vielleicht gibt es ja noch eine *einfachere* Grammatik, mit der man die selbe Sprache erzeugen kann.

Merksatz: Der Typ der Sprache ist immer größer oder gleich dem Typ der Grammatik.

Insbesondere: Wenn man eine Typ 2 Grammatik für eine Sprache angeben kann, weiß man noch lange nicht, ob die Sprache nicht regulär ist.

Sprache unvollständig

Mit dieser Grammatik kann man nicht alle Wörter erzeugen.

Typisch: Das Wort '100' ist nicht in der Sprache.

Epsilon-produktionen übersehen

Wenn eine Grammatik eine Epsilon-produktion enthält, ist sie automatisch vom Typ 0.

Syntax von Grammatiken nicht eingehalten

Auf der rechten Seite einer Produktionsregel stehen ausschließlich Variablen der Sprache.

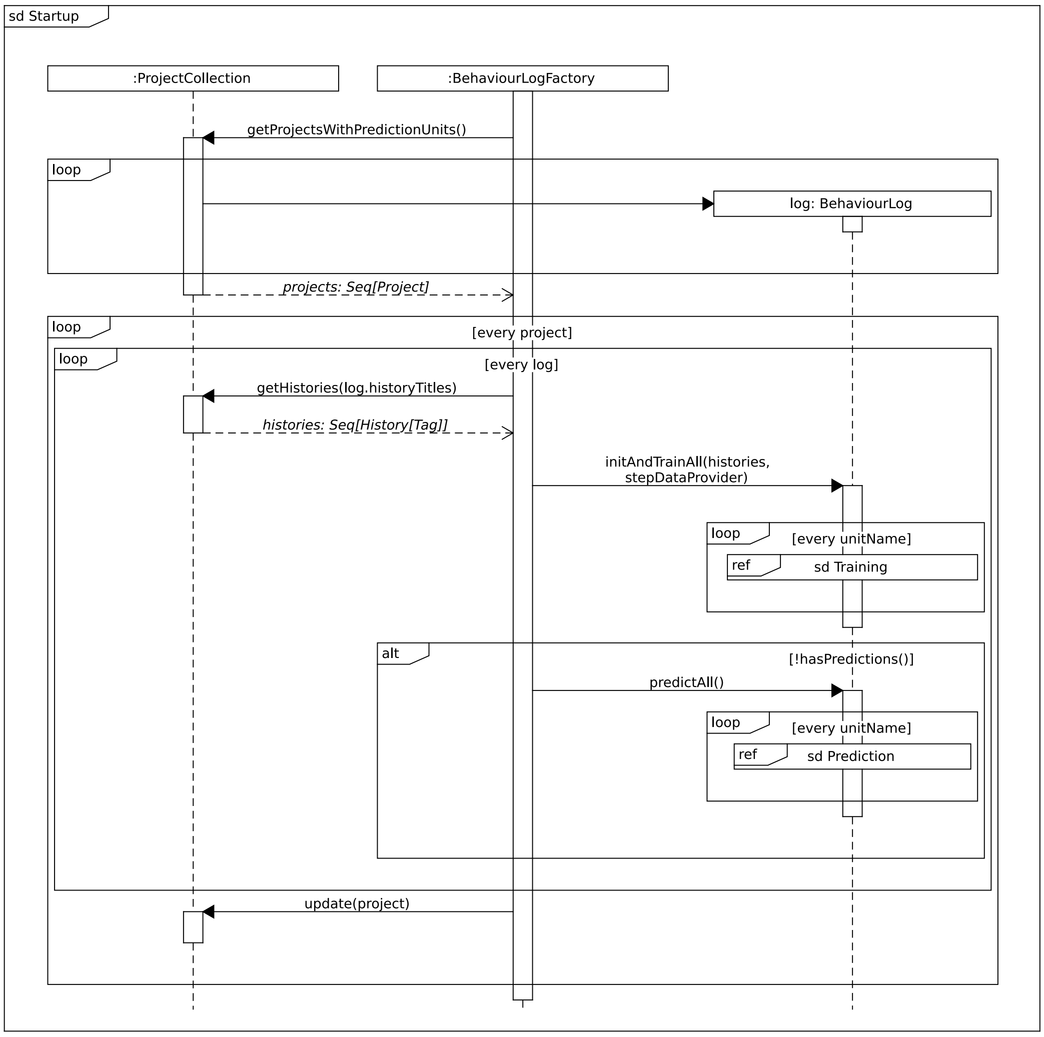
Nicht erlaubt sind: Reguläre Ausdrücke, ganze Sprachen, Automaten, . . .

Commented example configuration for Step Data

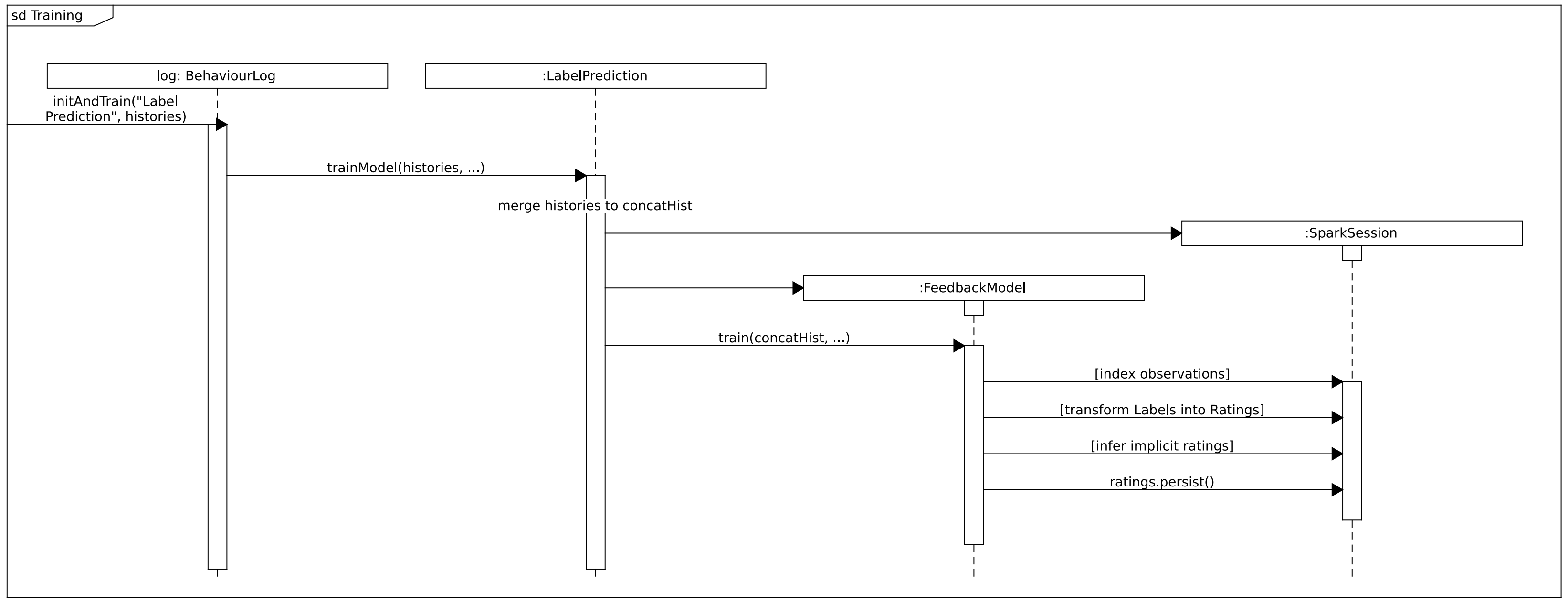
```
stepData.BehaviourLogName {
  // categories are provided by the corresponding behaviourLog
  // when requesting this data
  predictionDepth = 2 // number of future steps to make predictions for
  steps = [ // <-- list to preserve the order of steps
    {
      id = "u-0-2"
      // vvvv short notation for [ { val = "c-1" }, { val = "c-4" } ]
      observations = ["c-1", "c-4"]
      predictions = ["c-1", "c-4", "c-2"]
      // c-2 is not anymore used    ^^^
      // but if its predictions are interesting and some history
      // contains c-2 we can still receive predictions for c-2
    }
    {
      id = "u-0-1"
      observations = ["c-1", "c-2", "c-5"]  // BUT c-2 is still used for u-0-2
      // no predictions key => predictions = observations
    }
    {
      id = "u-1-1"
      observations = ["c-1", "c-24", "doesnotexistinhistory"]
      predictions = ["c-1", "c-24"] // ^ so no prediction possible
    }
    {
      id = "u-0-5"
      // Here we want predictions only based on previous steps
      // since what to observe for u-0-5 might be still unknown
      predictions = ["c-1", "c-3", "c-17", "alsodoesnotexistinhist"]
    }
    {
      id = "u-0-6"
      // no predictions, no observations, just a step
    }
    // vvvv throws an error due to duplicate id
    //{
    //  id = "u-1-1"
    //  predictions = ["c-1"]
    //}
  ]
}
```

Source: `conf/application.conf`

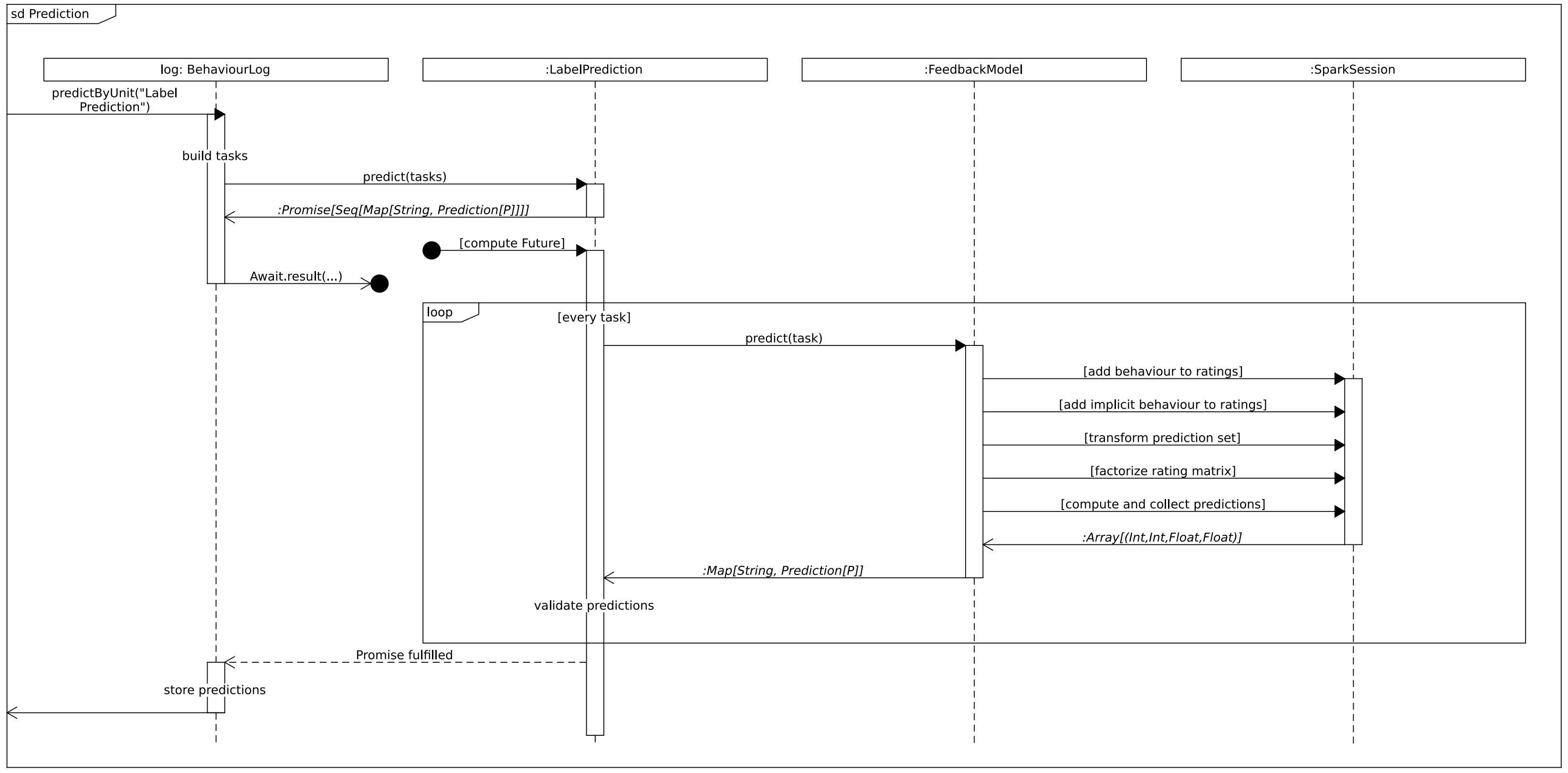
Control flows for the prediction use case



Control flow for the initialization of **`BehaviourLog`**s during prediction framework startup



Control flow for the training of the **`FeedbackPrediction`** prediction unit


sd Prediction
log: BehaviourLog
:LabelPrediction
:FeedbackModel
:SparkSession
predictByUnit("Label Prediction")
build tasks
predict(tasks)
:Promise[Seq[Map[String, Prediction[P]]]]
[compute Future]
Await.result(...)
loop
[every task]
predict(task)
[add behaviour to ratings]
[add implicit behaviour to ratings]
[transform prediction set]
[factorize rating matrix]
[compute and collect predictions]
:Array[(Int,Int,Float,Float)]
:Map[String, Prediction[P]]
validate predictions
Promise fulfilled
store predictions

Control flow for the prediction in the `FeedbackPrediction` prediction unit

Bibliography

[1] *Play framework Scala API*, `https://www.playframework.com/documentation/2.5.x/api/scala/index.html`, 2016.

[2] *Apache Spark API*, `https://spark.apache.org/docs/2.2.0/api/scala/index.html`, 2017.

[3] Charu C. Aggarwal, *Recommender Systems*, Springer International Publishing, Cham, 2016.

[4] Alan D. Baddeley and Graham Hitch, *Working memory*, Psychology of Learning and Motivation **8** (1974), 47–89.

[5] Robert M. Bell and Yehuda Koren, *Scalable Collaborative Filtering with jointly derived neighborhood interpolation weights*, Proceedings of the 7th IEEE International Conference on Data Mining, 2007, pp. 43–52.

[6] Ruth Boelens, Bram De Wever, and Michiel Voet, *Four key challenges to the design of blended learning: A systematic literature review*, Educational Research Review **22** (2017), 1–18.

[7] Denny Borsboom, Gideon J. Mellenbergh, and Jaap van Heerden, *The theoretical status of latent variables*, Psychological Review **110** (2003), no. 2, 203–219.

[8] Engin Bozdag and Jeroen van den Hoven, *Breaking the filter bubble: democracy and design*, Ethics and Information Technology **17** (2015), no. 4, 249–265.

[9] François Bry and Alexander Pohl, *Large-class teaching with Backstage*, Journal of Applied Research in Higher Education **9** (2016), no. 1, 105–128.

[10] Francesco Colace, Massimo De Santo, Vincenzo Moscato, Antonio Picariello, Fabio A. Schreiber, and Letizia Tanca (eds.), *Data management in Pervasive Systems*, Springer-Verlag GmbH, 2015.

[11] John Daniel, *Making sense of MOOCs: Musings in a maze of myth, paradox and possibility*, Journal of interactive Media in education (2012), no. 3.

[12] Jesse Davis and Mark Goadrich, *The relationship between Precision-Recall and ROC curves*, Proceedings of the 23rd international conference on Machine learning, ACM, 2006.

[13] Steven Dostert, *Predicting the learning behaviour of students from their weekly work assignments*, Master's thesis, Ludwig-Maximilans-Universität München, 2017.

[14] Bogdan Drăgulescu, Marian Bucos, and Radu Vasiu, *Predicting assignment submissions in a multiclass classification problem*, TEM Journal **4** (2015), no. 3, 244–254.

[15] David Dunning, Kerri Johnson, Joyce Ehrlinger, and Justin Kruger, *Why people fail to recognize their own incompetence*, Current Directions in Psychological Science **12** (2003), no. 3, 83–87.

[16] Karl Eilebrecht and Gernot Starke, *Patterns kompakt: Entwurfsmuster für effektive softwareentwicklung*, 3rd ed., ch. Strukturmuster, pp. 66–85, Spektrum Akademischer Verlag, Heidelberg, 2010.

[17] EPFL and Lightbend, *Scala reflection library API*, `http://www.scala-lang.org/api/2.11.7/scala-reflect/`, 2015.

[18] ______, *Scala standard library API*, `http://www.scala-lang.org/api/2.11.7/`, June 2015.

[19] Jerome Fan, Suneel Upadhye, and Andrew Worster, *Understanding receiver operating characteristic (ROC) curves*, Canadian Journal of Emergency Medicine **8** (2006), no. 1, 19–20.

[20] Simon Funk, *Netflix update: Try this at home*, `http://sifter.org/~simon/journal/20061211.html`, 2006, (last accessed: 28th September 2017).

[21] Lindsay D. Grace and Peter Jamieson, *Handbook of digital games, game design and development*, ch. Gaming with Purpose: Heuristic Understanding of Ubiquitous Game Development and Design for Human Computation, pp. 645–666, John Wiley & Sons, 2014.

[22] Mario Haim, Andreas Graefe, and Hans-Bernd Brosius, *Burst of the filter bubble?*, Digital Journalism (2017), 1–14.

[23] Sherif Halawa, Daniel Greene, and John Mitchell, *Dropout prediction in MOOCs using learner activity features*, Experiences and best practices in and around MOOCs **7** (2014), 3–12.

[24] Wolfgang Karl Härdle and Zdenek Hlávka, *Multivariate statistics*, 2nd ed., ch. Principal Component Analysis, pp. 183–203, Springer-Verlag GmbH, 2015.

[25] John Hattie, *Visible Learning*, Taylor & Francis Ltd., 2008.

[26] John Hattie and Helen Timperley, *The power of feedback*, Review of Educational Research **77** (2007), no. 1, 81–112.

[27] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl, *Explaining Collaborative Filtering recommendations*, Proceedings of the 2000 ACM conference on Computer supported cooperative work, ACM, 2000, pp. 241–250.

[28] K. F. Hew and W. S. Cheung, *Using blended learning: Evidence-based practices*, SpringerBriefs in Education, Springer Singapore, 2014.

[29] Geoffrey Hinton and Sam Roweis, *Stochastic neighbor embedding*, Advances in neural information processing systems (2003), 857–864.

[30] Yifan Hu, Yehuda Koren, and Chris Volinsky, *Collaborative Filtering for implicit feedback datasets*, Proceedings of the 8th IEEE International Conference on Data Mining, IEEE, 2008, pp. 263–272.

[31] Sui Huang, *When peers are not peers and don't know it: The Dunning-Kruger effect and self-fulfilling prophecy in peer-review*, BioEssays **35** (2013), no. 5, 414–416.

[32] Sanna Järvelä, *Personalised learning? New insights into fostering learning capacity*, Personalising Education, Organisation for Economic Co-operation and Development, 2006, pp. 31–46.

[33] André Klahold, *Empfehlungssysteme: Recommender Systems – Grundlagen, Konzepte und Lösungen*, Studium, Vieweg + Teubner, Wiesbaden, 2009.

[34] Janne V. Kujala, Ulla Richardson, and Heikki Lyytinen, *A Bayesian-optimal principle for learner-friendly adaptation in learning games*, Journal of Mathematical Psychology **54** (2010), no. 2, 247–255.

[35] David M. W. Powers, *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation*, Tech. report, Flinders University of South Australia, 2007.

[36] Miia Ronimus, Janne Kujala, Asko Tolvanen, and Heikki Lyytinen, *Children's engagement during digital game-based learning of reading: The effects of time, rewards, and challenge*, Computers & Education **71** (2014), 237–246.

[37] Jorge G. Ruiz, Michael J. Mintzer, and Rosanne M. Leipzig, *The impact of e-learning in medical education*, Academic Medicine **81** (2006), no. 3, 207–212.

[38] Takaya Saito and Marc Rehmsmeier, *The Precision-Recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets*, PLoS ONE **10** (2015), no. 3.

[39] Jonathan Richard Shewchuk, *An introduction to the conjugate gradient method without the agonizing pain*, `https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf` (1994).

[40] Artem Syromiatnikov and Danny Weyns, *A journey through the land of Model-View-Design patterns*, Master's thesis, Linnæus University, 2014, pp. 21–30.

[41] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown, *Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms*, Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2013, pp. 847–855.

[42] Laurens van der Maaten and Geoffrey Hinton, *Visualizing data using t-SNE*, Journal of Machine Learning Research **9** (2008), no. Nov, 2579–2605.

[43] Tuomo Virtanen, *Student engagement in Finnish lower secondary school*, Jyväskylä studies in education, psychology and social research **562** (2016).

[44] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan, *Large-scale parallel Collaborative Filtering for the Netflix Prize*, 4th Int'l Conf. Algorithmic Aspects in Information and Management, Springer, 2008, pp. 337–348.