UNIVERSITY OF TWENTE

MASTER THESIS

Predicting user loyalty in an education support web application based on usage data

Author:
Karim M. EL ASSAL

Supervisors:
Dr.ir. M. VAN KEULEN
Dr. K. SCHILDKAMP

External supervisors:
L.D.J. NIESINK
E.R.G. VAN DER VEEN

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Databases Research Group
Department of Computer Science

April 24, 2018

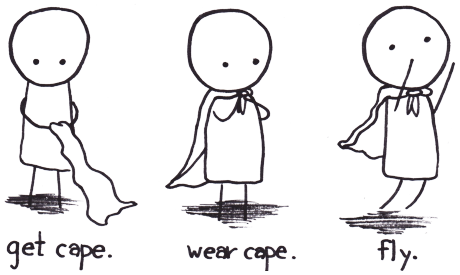
Declaration of Authorship

I, Karim M. EL ASSAL, declare that this thesis titled, "Predicting user loyalty in an education support web application based on usage data" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this university.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 24-04-2018


get cape.
wear cape.
fly.

University of Twente

Abstract

Faculty of Electrical Engineering, Mathematics and Computer Science
Department of Computer Science

Master of Science

Predicting user loyalty in an education support web application based on usage data

by Karim M. EL ASSAL

Data use in education has been increasing in the last 20 years. School management and teachers are moving towards a data-driven policy and improvement process due to the potential benefits. Studies show that data use can increase student achievement to some extent, and that the use of information systems that facilitate data use has a positive impact on educational management.

There are many barriers that limit the adoption of data use, one of which is the teachers and specifically their data (il)literacy and their attitude towards data use. The proper adoption of data systems by teachers is paramount in this context: without appropriate use of the means, data use can not be effectively utilized. A few factors that influence the adoption of data systems are data's availability, reliability, findability and interpretability.

Knowing users' opinions on these factors is essential to improving such a data system. Gathering these opinions, however, is time and resource intensive. That's why Reichheld's Net-Promoter Score (NPS) survey is so popular. It asks the question "How likely is it that you would recommend our system to a friend or colleague?" and expects an answer between 0 (extremely unlikely) and 10 (extremely likely). Users scoring 9 or 10 are called Promoters, users scoring 7 or 8 are Passives and users scoring between 0 and 6 are Detractors. It is considered a measure of loyalty: Reichheld argues that users who promote your system put their own reputation on the line.

This research focuses on predicting a user's NPS response based on their system usage data. With this loyalty prediction, system owners can better target their system evaluation and improvement efforts. The use case is Somtoday, an exemplary Dutch high school administration system developed by Topicus Education. The research question is: "How reliably can teachers' usage data, generated by an education support data system, be used to predict user loyalty towards that system?"

There are 1085 NPS responses available, consisting of a score and a reason for that score, with an average of 1408 relevant log entries per NPS response. Additionally, non-identifying profile data is available, such as a teacher's school and the education levels he or she is teaching.

Efforts to determine what behavior might be an influence on the NPS score have yielded data feature specifications about prevalence of functionality use, repetitive tasks, clickstreams, encountered downtimes, login frequencies, the amount of system usage and teacher profile data. Based on the expected impact on the NPS response, preparation time and generalizability to other systems, data features were selected about system usage, repetitive tasks and profile data. After applying a range of extraction, transformation and load (ETL) operations and applying the sequential pattern

mining algorithm Apriorirep1i we created, numerical or binary values for each data feature were extracted.

The actual NPS prediction was done using machine learning. A brute force approach was applied to account for differences in models, model parameters, and preprocessing methods such as normalization and outlier removal. Additionally, the data type of the predicted NPS score was treated in three different ways: numerical (0-10), eleven-value categorical (0-10), and three-value categorical (Promoter, Passive, Detractor). The best models performed with a mean absolute error (MAE) of 1.727 with numerical prediction, an accuracy of 27.92% with eleven-value categorical prediction, and 54.30% with three-value categorical prediction. If one always predicts the dominant class or value, i.e. 7 or Passive, the MAE with numerical prediction is 1.733, the accuracy with eleven-value categorical prediction is 27.63%, and the accuracy with three-value categorical prediction is 44.07%. Validation was provided by the utilized brute force approach and by looking at different performance metrics.

The small performance difference between the trained models and always predicting the dominant class or value shows that there is practically no predictive value in the dataset. The conclusion of this study is that the researched data features can not reliably be used with machine learning models to predict NPS scores.

This does not rule out the possibility that user loyalty towards a system can be predicted based on their behavior. Our main recommendation for Topicus is to focus on finding predictive value in other usage patterns. The recommendation from a more scientific perspective is the same, but with a preliminary step: to conduct a more in-depth feature discovery research project. A full-scale user experience study with the goal of usage data analysis gives the researcher quantitative and qualitative data about what users think of different aspects (e.g. navigation and layout) and functionalities and how their behavior is mapped onto the log entries dataset. Having better insight leads not only to validated feature selection, it also leads to refined knowledge about how data features can be measured and what the nuances are. Choosing this approach, one or multiple directed studies into specific usage patterns can be set up and the researcher has a better chance of finding predictive value in user behavior: the researcher is no longer looking in the proverbial dark.

Acknowledgements

This project has taught me many positive things about research, generating ideas, time management, prioritizing, motivation, and how much one can do when one's mind is set to it. I would like to thank Maurice van Keulen for the excellent guidance and sparring moments. Your tip that data preprocessing takes up most of the time in a data science project was no overstatement. You have always been positive and timely in your communications, even though I have consistently declined your requests for me to be a student assistant in your data science courses prior to this project. I would also like to thank Kim Schildkamp. You have also been a first-rate supervisor. Your points of feedback were paramount to the scientific quality of this thesis and were hugely appreciated. Your perspective as an expert of the field was essential. During your absence, Rilana Prenger took over your supervision wonderfully. Rilana, I thank you too for your feedback and contributions in our discussions.

I would also like to thank Luke Niesink and Egbert van der Veen from Topicus. You have been exceptionally constructive in your feedback, discussions and facilitating help. The working environment was pleasant and your enthusiasm was very motivating. On that note I would also like to thank Erik Dikkers for the opportunities and flexibility. Similarly, Thomas Markus deserves a big thank-you. Our discussions motivated me more than you think. Also, your help with the data infrastructure literally proved to be indispensable.

Furthermore, I would like to thank my roommates at Avion for their youthful enthusiasm and for reminding me that I was still a student during this project (albeit a hard working one). I also want to thank TNO, my current employer, for providing me in the last few months with ample opportunity to finish this project. Of course, my gratitude also flies out to my girlfriend, family and friends for their support before, during and after my medical leave of absence. I have reserved a special thanks for my girlfriend's Maine Coon kitten Woezel. The overload of cuteness always was and still is a welcome distraction.

Finally, I would like to thank you, the reader, for taking the time to read these words. Theses are written to be read, after all.

Contents

List of Figures

List of Tables

I dedicate this thesis to the Human Colossus[1], may it grow vast, contribution by contribution.

[1] https://waitbutwhy.com/2017/04/neuralink.html#part1

Chapter 1

Introduction

The improvement of our future relies immensely on the quality of our education. When we improve our education we not only better our future, but also the future of generations to come. To make sure that people focus their efforts on actual growth and not on mere futile attempts and approximations, the right decisions must be made based not on (gut) feelings, but on accurate data. One way of doing that is by using education support *data systems*[1]. Although there are other reasons that might hinder data-driven decision making, such as attitude and high workload, an effective and optimized data system comes a long way in stimulating data use. This study uses certain usage patterns within such a system to predict users' loyalty towards that system based on the NPS metric, so that system improvements can be better targeted and, in turn, to ultimately advance data use in education, education itself, and our future.

Data use in education has been increasing in the last 20 years. With the advent of the internet and the availability of enormous amounts of data, data use in schools has attained a prominent place in the educational system. School management and teachers are moving towards a data-driven policy and improvement process due to the potential benefits [7, 15, 16, 34, 60, 63]. Data use can be defined as "systematically analyzing existing data sources within the school, applying outcomes of analyses to innovate teaching, curricula, and school performance, and, implementing (e.g. genuine improvement actions) and evaluating these innovations" [52, p.482]. Data include not only assessment data, but anything about the school and students that might be relevant for decision making. Examples are student background data, process data (e.g. classroom observations and teacher interviews), school context data (e.g. information about the building), student assessment data and satisfaction data [3, 30, 53]. Several studies show that data use can increase student achievement to some extent [8, 35, 38, 45], for example by setting specific and measurable achievement goals [17]. Studies have also shown that the use of systems that facilitate data use[1] has a positive impact on educational management. Examples include better access to information, more efficient administration, higher utilization of school resources, reduction in workload, better time management, and improvement in the quality of reports [55]. These examples show that data use and the use of data systems improve several aspects of education, motivating a wider adoption of data use and its facilitating systems.

[1] The terminology in literature for these types of systems is unclear. Several studies use different terms to denote the same, such as "data use systems", "data management systems", and "information management systems". This thesis will use *"data systems"* as a term for systems that facilitate educational data use in the broadest sense: assessment data, administration data and any other type of data that are relevant to educational institutions.

1.1 Problem

There are, however, many barriers that limit the adoption of data use. One of the most influential factors is the teachers; specifically their data (il)literacy and their attitude towards data use [54]. Teachers that see the value in using data as evidence for the improvement of student achievement on an individual basis, tend to overcome barriers more easily and are more likely to use data systems [43]. The proper adoption of these tools by teachers is paramount: without appropriate use of the means, data use can not be effectively utilized.

Research has shown that a few key factors are important for the adoption of data systems, besides the adoption of data use as a concept. The most important factors are the data's availability, reliability (i.e. data being accurate and up to date) and the ease with which specific data can be found [6, 11, 17, 27, 52, 64]. Furthermore, teachers report they need functionalities to not only access and organize data, but also support the interpretation of data [34, 51]. This is necessary for users with limited data literacy, which can be assumed to be a very large part of all teachers. Functionalities supporting data interpretation may prevent or at least mitigate the chances of data misuse and abuse. Data misuse is basically a wrong interpretation of data. An example of data abuse is when teachers try to improve a class' test score average by practically giving up on the 'hopeless' students [39, 52].

The above mentioned studies have reported on teachers' opinions, which can ultimately improve the adoption of data use and data systems. Gathering these opinions, however, is time and resource intensive. One might think of composing focus groups and organizing constructive conversations with them, or constructing, administering and processing surveys. Companies developing data systems are very interested in the opinions of their end users. Yet, they might also have other reasons that make the method of obtaining feedback impractical. For example, the end users might not be in the management positions. At Topicus Education, the company of this thesis' use case (see section 1.2), they've run into a direct consequence of that distinction. They essentially have two paths of communication: one for business-to-business and one for end-user-to-business. The former is used for business-level subjects and is between a school's application manager and one of Topicus' customer relations employees. The latter is for gathering points of improvements to the system and consists mainly of talking to focus groups on a per-school basis, administering periodic surveys and making a feedback button available within the system (more on this in section 1.4). The problem is that with the business-to-business communications, an application manager acts as a spokesperson for the end users of their associated school. Topicus reports that more often than not, the application manager expresses the customer's positive opinion while in reality the end users are dissatisfied with several aspects of the system.

Like many other companies, Topicus Education uses the Net-Promoter Score (NPS) survey[49] to measure the performance of their system in terms of user loyalty. It focuses on asking users the question "how likely is it that you would recommend our system to a friend or colleague?", where the possible answers are in the discrete range of 0 to 10, inclusive. It is far too simple to include the nuances inherent to a user's opinion. However, for the goal of system improvement it is sufficient: users that would definitely not recommend the system are worth focusing improvement efforts on. Gathering responses to this question is less costly than is the case with extensive satisfaction surveys, but it still is resource intensive. Additionally, it only results in responses from users who take the little amount of time and effort to respond.

1.2 Objectives and use case

The above example of the application managers, combined with the general tediousness of feedback aggregation, illustrates the opportunities for improvement in the process of gathering feedback about a system. This study focuses on predicting user loyalty in a userbase-wide, non-invasive and fully automated way, i.e. by analyzing metrics (called *features*) derived from the usage data. For example, a feature could be the average amount of login actions per day and a prediction could be "if a user logs in 10 times per day on average, he is very likely to give a score of 4 on the NPS survey". With this loyalty prediction, system owners can identify potentially disloyal users and effectively target their system evaluation and improvement efforts. Additionally, this study will provide some insight into user behavior that qualifies users as loyal, neutral or disloyal.

The use case for this study is Somtoday[2], an exemplary high school administration system. It has a web-based user interface and is tailored for school staff, students and parents. It is owned and developed by Topicus Education[3] and has been first released in September 2006. As of October 2016, it had about 660,000 users [42] and for a few years now, a share of about 30% of the Dutch market in educational data systems in the higher education[4] segment [40]. In addition to the web-based user interface, Topicus has released a basic variant in 2015 called Somtoday Docent which is meant for mobile devices. Although it has less functionalities, it is more useful in class for administrative tasks such as absentee registration.

1.3 Research questions

The road to reach the goal described in section 1.2 is taken by finding an answer to the main research question:

> **How reliably can teachers' usage data, generated by an education support data system, be used to predict user loyalty towards that system?**

The following questions have been formulated to divide the main problem into distinct parts. Each question is answered by the respective chapters 3 to 7.

1. What might be an influence on the user loyalty towards the data system?
2. How can the raw data be put into a format usable for research?
3. How can values for each data feature be extracted from the data?
4. What is the predictive value of the dataset?
5. What is the predictive validity of the trained model?

Although an effort is being made in engaging students in utilizing their own data [31], the scope of this study has been limited by only looking at usage data of teachers. This is the group of users that is the most influential factor in the adoption of the system [27] and the most influential in a school's decision of data system.

[2] https://www.som.today/
[3] https://topicus.nl/onderwijs/
[4] This concerns the Dutch 'voorgezet onderwijs'.

1.4 Available data and technologies

This section describes the data and technologies that are available at Topicus. Some basic analyses have been made to show the size and distribution of the datasets.

Unfortunately, there are no reliable data available about encountered system bugs. This is due to a concurrency limitation within Somtoday, which makes it unusable for this study.

1.4.1 NPS responses

The individual NPS responses are the data points that are used to train the model in its predictive capabilities. Each day a few dozen users are randomly selected to receive the NPS survey. There is at least a time period of about three months between each survey to the same user to be certain that users aren't bothered too often. Additionally, the NPS survey is administered after notable client contact, such as with Topicus' service desk.

The raw data fields are described in table 1.1. Seeing as this research focuses primarily on teachers, the raw dataset was filtered on the role field: only NPS responses are considered where the user has the role 'teacher', 'mentor' or 'individual mentor'. Mentors can be assumed to also be teachers. The 'other' role includes parents and non-teacher types of school employees such as management and support staff.

TABLE 1.1: Raw NPS data fields. Note that the actual field names are slightly different in the actual dataset; either with different capitalization, translated, simplified or a combination of those.

Field	Description	Data format / possible values
date	The date and time the user responded to the NPS survey.	datetime string
INSTUUID	The institution (i.e. school) ID within Somtoday.	alphanumeric string of length 36
institution	The school of the user.	string
process	The process that was the motivation for administering the NPS survey.	'Active account Student|Employee|Caretaker' | 'Accountmanagement' | 'Servicedesk' | 'Training participant'
UUID	The user ID within Somtoday.	alphanumeric string of length 36
role	The role of the user. This is determined based on assignments within Somtoday. E.g. a user is a teacher if he is assigned to one or more classes.	'application manager' | 'teacher' | 'mentor' | 'individual mentor' | 'other'
NPS	The actual NPS response.	integer between 0 and 10 (inclusive)
root cause 1	The primary root cause for the NPS response.	any of the available categories, such as 'functionalities', 'availability', 'user-friendliness' and 'service'
root cause 2	The secondary, supplementary root cause for the NPS response.	any of the available categories, such as specific functionalities, the layout on specific devices and the length and frequency of system unavailability
open question	Any further clarification for the given response.	string

In the available dataset, there are 1085 NPS responses of teachers in the relevant time frame. Of those, there are 6 users that have two NPS responses. The time between responses is 4.4 months on average, with a minimum of 2.8 months. One anomalous case was removed. All NPS responses, including those of users with two responses, will be treated as separate and independent of each other. The response histogram is shown in figure 1.1 and the temporal distribution is shown in figure 1.2.

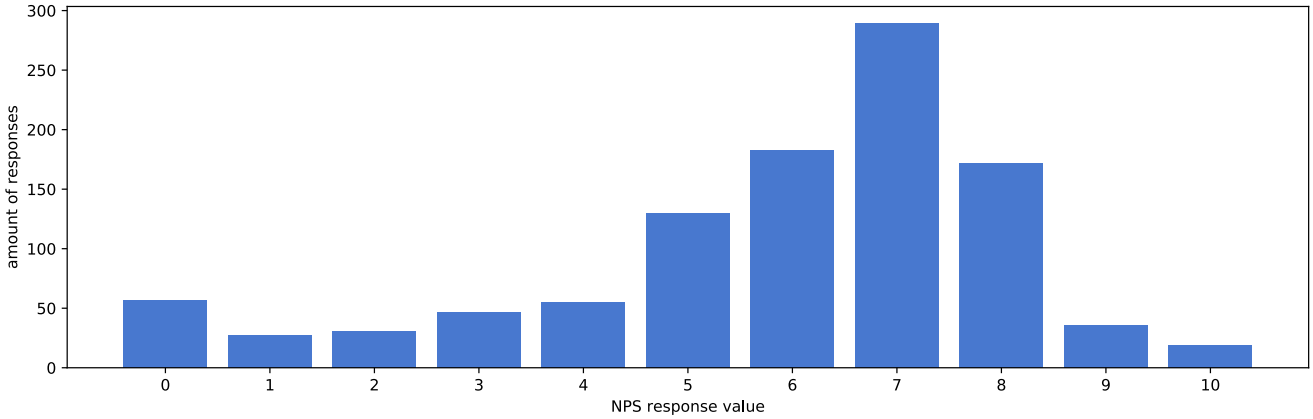


FIGURE 1.1: Response distribution of teacher NPS responses.

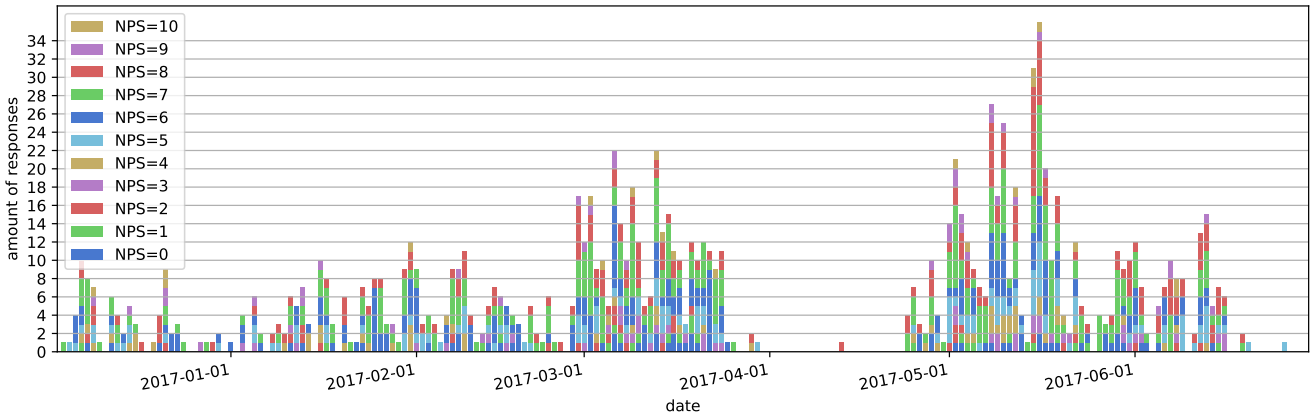


FIGURE 1.2: Temporal distribution of teacher NPS responses. Note that the different response values are shown solely to indicate that there is variation.

1.4.2 Usage data

The usage data is the main subject of this research. It consists of two types of web usage logs. The first is automatically collected by the framework in which the main Somtoday web application is developed: Apache Wicket[5]. This data is copied in a transformed format to the cluster for future analysis (more on the infrastructure in section 1.4.6). We'll call these the **non-API log entries**. Log entries of the second type are a result of the Somtoday Docent front-end application that generates API calls to the back-end and also end up on the same cluster. We'll call these the **API log entries**.

[5] https://wicket.apache.org/

The non-API and API logs have some intersecting fields. They are described in table 1.2.

TABLE 1.2: Raw usage data fields. Note that this table only contains fields that are possibly relevant for the rest of this research.

Field	non-API	API	Description	Data format / possible values
@timestamp	✓	✓	The timestamp of the request, accurate to the millisecond.	timestamp string
duration	✓	✓	The amount of milliseconds it took the server to process the request and build a response.	positive
requestedUrl	✓	✓	The URL.	string
userAgent_*	✓	✓	Several fields containing data about the user's user agent, such as the name, device type, operating system, browser and browser version.	integer in the case of versions, string in any other case
username	✓		The username of the user.	string
username		✓	The username of the user, including the organisation (i.e. school) abbreviation.	string: '<organisation-abbr>/<username>'
remote address	✓		The user's ip address.	IPv4 or IPv6 string
organisation	✓	✓	The full name of the user's school.	string
range		✓	Element used for pagination.	string, e.g. 'items=0-100'
restResource		✓	The URL to the resource, including parameter templates.	string
method		✓	The method with which the request was sent.	'GET' | 'POST' | 'OPTIONS' | 'PUT' | 'DELETE'
category		✓	The category of the resource.	string
status		✓	The HTTP status code sent back to the user.	integer
sessionId	✓		A unique session ID.	alphanumeric string of length 40
session_startDate	✓		The timestamp the session started, accurate to the millisecond.	timestamp string
session_numberOf-Requests	✓		The sequential number of the current request relative to the current session.	positive integer
session_totalTime-Taken	✓		The total number of milliseconds the current session is taking up until the current request.	(large) positive integer
eventTargetClass	✓		The Java class handling the request.	fully qualified Java class name
event_pageClass	✓		The Java page class that triggered the request.	fully qualified Java class name
componentClass	✓		The Java component (e.g. button) class that triggered the request.	fully qualified Java class name
componentPath	✓		The path to the component of 'componentClass'.	CSS selector-like string
behaviorClass	✓		The Java behavior class responding to the event. Examples are AJAX and timer events.	fully qualified Java class name
response_pageClass	✓		The Java page class responding to the request.	fully qualified Java class name

The are about 7 billion non-API log records and 2 billion API log records between March 1st, 2016 and October 1st, 2017. Not all of these are relevant: only the usage logs of the users that responded to the NPS survey are interesting. This is discussed in detail in chapter 4. The temporal distribution of the usage logs is shown in figure 1.3.

The weekend and holiday dips are clearly visible and the peak in June 2016 and the absence of entries in September 2017 are explained by irregularities in the process of log storage. The absence of API log entries before May 2016 is explained by the introduction of the API logging process around that date. These irregularities do not impact this research because they lie outside of the time frame of relevant entries, as chapter 4 also explains.

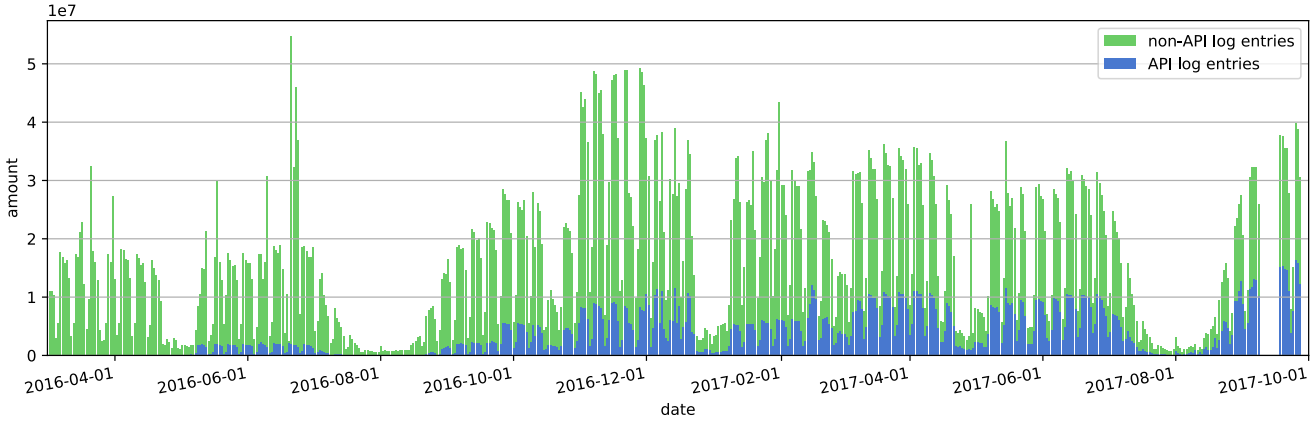


FIGURE 1.3: Temporal distribution of usage logs

1.4.3 Profile data

The Somtoday back-end has profile data stored of its users and schools in Oracle Database[6] divided over several servers. This data can be extracted with SQL queries that can become rather complex due to the normalized format of the relational storage. Note that the existence of this data does not mean it is used in this research.

Students Personal data (i.e. name, address and date of birth), familial relations, previous school, current school registration (including date of registration, education level[7] and education specialization[8]), internships, absence, study program, linked groups (i.e. classes), exam results, medical data, class schedule and homework (administration).

Employees Personal data (i.e. name, address and date of birth), linked groups (i.e. classes) and class schedule.

Schools General data (e.g. name, owner and data on all establishments), educational programs and settings within Somtoday (e.g. security rights, layout settings and templates).

1.4.4 (Unplanned) downtimes

The periods of unplanned downtime are tracked by the system monitoring company Uptrends[9]. They provide a web-based interface that includes the option to export data into XLSX file format. The available data consist of the moments at which a specific system has reported a failure or that it's available again. In the time period

[6] http://www.oracle.com/technetwork/database/
[7] In Dutch this is the opleidingsniveau, such as HAVO or VWO.
[8] In Dutch this is the opleidingsprofiel, such as *Natuurkunde en Techniek* or *Natuurkunde en Gezondheid*.
[9] https://www.uptrends.nl/

between July 1st, 2016 and June 30th, 2017, Uptrends reports 25 periods of unplanned downtime of Somtoday. The length of these periods range from mere minutes to half an hour, with a few exceptions reaching about 90 minutes.

There are no data available about planned downtime. However, this can be extracted from the logs by looking at periods of time in which the amount of log entries reaches zero. These maintenance periods are usually about every three weeks on Fridays between 5 and 6 PM, limiting the search scope considerably.

1.4.5 Auxiliary qualitative data

There are three auxiliary data sources available: the open question included in the NPS survey, the responses on the feedback button within Somtoday, and the service desk tickets. These have not been used for data analysis, but primarily to gain new insights into possibly relevant features. More on this is discussed in chapter 3. Note that the term *auxiliary* data is used here not as jargon, but exactly as the definition specifies it: supportive or supplementary data. This was chosen to distinguish data that is used for automated analysis and data that is used for manual analysis (i.e. feature discovery).

The NPS survey includes an open question asking for a reason behind the respondent's given response. Table 1.1 shows that respondents can classify and subclassify their main reason. This is supplemented with what the respondent wants to say. There are an even amount of data points as there are (relevant) NPS responses, although a small part of the open question responses is empty or otherwise useless.

Contact between Somtoday's end users and Topicus generally doesn't happen directly. Topicus handles contact with a school's application manager. This means that whenever an end user has a question or complaint, it is passed through the application manager to Topicus' service desk. Whenever answering a question takes more than a few seconds, a ticket is created in the issue tracking system JIRA[10]. There were 2314 tickets created between January 2nd, 2014 and April 25th, 2017.

Furthermore, Somtoday has got a feedback button integrated in its layout. This way, users can send feedback to the developers within a few seconds. These are meant to be ideas for new features, although users often send complaints, mistaking the feedback functionality for the service desk. All feedback sent via this button is collected by Freshdesk[11]. Topicus received 1953 responses between November 13th, 2015 and June 27th, 2017.

1.4.6 Infrastructure

Topicus Education has an infrastructure in place specifically designed for the analysis of usage logs. The relevant parts are explained here. All storage, applications and other processes reside in a cloud environment managed by Previder[12]. Authentication is not depicted here, but is embedded in the infrastructure and handled by Topicus' own Keyhub[13] service. Practically all services run in Docker[14] instances and load balancing on the cluster is managed by Kubernetes[15].

[10] https://www.atlassian.com/software/jira
[11] https://freshdesk.com/
[12] https://www.previder.com/
[13] https://www.topicus-keyhub.com/
[14] https://www.docker.com/
[15] https://kubernetes.io/

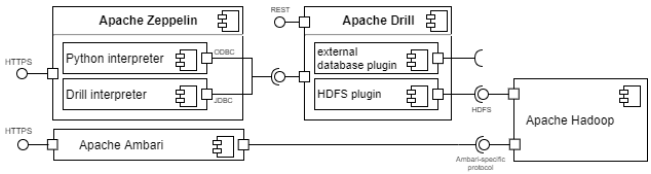
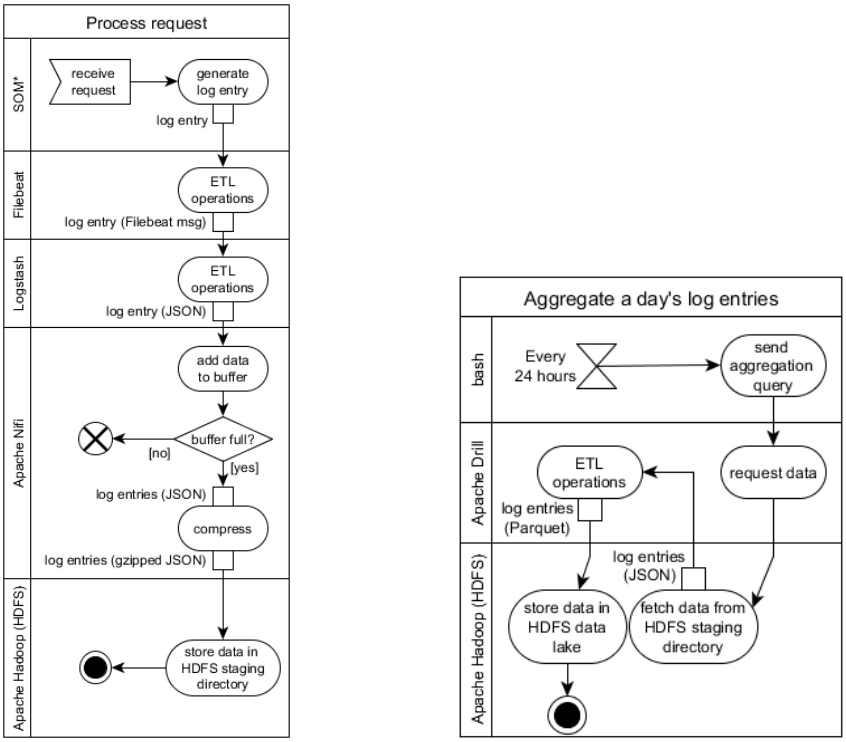


FIGURE 1.4: A UML component diagram that depicts Topicus Education's infrastructure relevant to data analysis. Note that the displayed interpreters and plugins are just practical examples: in reality, many more are implemented and used in their respective systems.



(A) The UML activity diagram of the process that stores real-time request log data

(B) The UML activity diagram of the transfer of log data to long-term storage

FIGURE 1.5

Real-time data collection Logging requests received by Somtoday and Somtoday Docent is handled in real-time. Figure 1.5a shows the order of operations upon receiving a new request. Elastic's Filebeat[16] and Logstash[17] pipelines handle several elementary extraction, transformation and loading (ETL) operations. A buffer handles any issues resulting from slow disk operations. This buffer's size is in 100 log entries. When the buffer is full it writes the compressed log entries to a staging directory on the Hadoop cluster, after which the log entries are available for analysis. This means this data collection process is nearly, but not entirely real-time.

Historical data collection Figure 1.5b shows the process that flushes the (near) real-time data to a *data lake* on the cluster: long-term storage that holds various types of data in (semi-)structured format. Each log type is stored in a different

[16] https://www.elastic.co/products/beats/filebeat
[17] https://www.elastic.co/products/logstash

subdirectory. This happens every 24 hours, at night. After this atomic operation completes, the real-time data from the staging directory are purged.

Data storage Topicus Education maintains its distributed storage using the Apache Hadoop[18] ecosystem. This cluster is managed with the graphical web-based user interface Apache Ambari[19] v2.4.2.0. The infrastructure is depicted in figure 1.4. All analysis operations and queries for stored data go through Drill and/or Hadoop.

Data storage format Apache Drill uses the Apache Parquet[20] format to store its tables. It uses a column-based storage structure, as opposed to conventional row-based structures. A big advantage is that it enables quick physical data retrieval for a few columns on a large dataset. For example, retrieving the timestamps of all requests of a specific user is fast, but retrieving all rows within a specific timespan is very slow.

Data querying (back-end) The center of action is Apache Drill[21] v1.7.0. Drill is a schema-free SQL query engine that Apache promotes as having far less overhead due to its flexibility and agility in easily working with multiple data sources. A disadvantage is that this proved false in our case. For example, when joining different tables, casting must take place to force a schema on each table so Drill can perform a proper hash join. An advantage is that it uses (ANSI) SQL for querying.

Data querying (front-end) Apache Zeppelin[22] v0.7.1 is a web-based notebook that interfaces with Drill. It enables execution of several languages, which is handled by *interpreters*, and incorporates rudimentary version control. The available languages are Angular (for interactive notebooks), Drill (for directly querying the back-end), Markdown (for simple text), Python (for handling data with a full-fledged programming language), and R (for more basic data analyses). It also offers a fairly extensive display system for easy visualization of (business intelligence) datasets.

Data processing Within Zeppelin, Python[23] v2.7 is used because of the advantage of having a complete programming language.

Other data, such as the NPS data and any data from the Somtoday database, are exported from their respective sources into CSV format and made available to Drill by uploading to the Hadoop cluster using Ambari. Seeing as the dataset does not get updated often, this is a manageable workflow.

1.5 Approach

This section summarizes the approach used in this research project. Each phase's challenges and results are discussed in more detail in their respective parts: chapters 3 to 7. The goal of each phase is to find an answer to the respective research

[18] https://hadoop.apache.org/
[19] https://ambari.apache.org/
[20] https://parquet.apache.org/
[21] https://drill.apache.org/
[22] https://zeppelin.apache.org/
[23] https://www.python.org/

questions described in section 1.3. The overall structure is a combination of the common approaches used in the fields of web usage mining and machine learning, seeing as this study is exactly that: a combination of those two fields. A basic version of this approach was used at the start of this project as a pilot to get familiar with the process, using raw login frequencies as a data feature.[24]

1.5.1 Feature discovery

The results of the first phase of this project determined the exact content of the subsequent phases: researching which data features might be an influence on a user's NPS response. Discovering which features might be relevant was done by speaking with several stakeholders that can be considered domain experts. These include not only teachers, but also Topicus' staff in the customer service, customer relations, development and data analysis departments. Several questions were prepared with the purpose of having a clear direction for each short interview. In addition to these questions the interview had a brainstorm approach, seeing as the purpose was to gain new insights and come up with new ideas for features. Additionally, the qualitative dataset was manually scanned for ideas. Suggestions were disregarded that were previously encountered, immeasurable (based on the available data), or otherwise had no practical value.

1.5.2 Data preprocessing

This phase is notorious for being underestimated in time spent on it. During this phase, the available data were transformed into a format that could be used in subsequent phases. For the NPS data this meant filtering on user type and time frame. For usage data this means only keeping entries that relate to the users of the NPS data, within a specific time window of the NPS response and discarding other irrelevant entries. While more features were researched, this phase was repeated. Note that validation of this phase proved to be important in ensuring that the dataset used in subsequent phases was correct. Validation was done by preprocessing the dataset at each atomic step in different ways, and comparing the resulting amount of records. Also, taking a random sample and zooming in on it was useful for detecting irregularities.

1.5.3 Pattern extraction

The purpose of this phase was to get meaningful, measurable data from the preprocessed, large dataset in the format of one or several values per data feature. The taken approach differs per feature. For some it was as simple as summing up values, for others it entailed coming up with a new algorithm.

1.5.4 Model training

At this point the working dataset was reduced to several features per user. This was used together with the known NPS responses to train a machine learning model, such that it could be used to predict NPS responses of other users based on new, unseen

[24] One should not consider the used data feature as potentially relevant. It was unprocessed: it simply counted how often the system's authenticator was called. This is not limited to users manually logging in, e.g. it also happens when the users are automatically logged in or external authenticators are used.

data. Several models, model parameters and preprocessing methods were investigated and evaluated. Examples of models include decision trees, linear regression, support vector machines, neural networks, and k-nearest neighbors. Training time wasn't much of an issue, provided that it's in the order of days, not weeks.

1.5.5 Validation

Validation was done by utilizing a brute force approach for the models, model parameters and preprocessing methods. Additionally, looking at the generated decision trees and confusion matrices of the best models adds to the comprehension of the model results and was thus also used for validation.

1.6 Ethical considerations

Data privacy issues are a hot topic lately. This is illustrated with the approaching date of the 25th of May, 2018, on which the General Data Protection Regulation will take effect [22] and the large impact this has on entities processing data [48]. The act of linking NPS responses to usage data on a per-user basis lies undoubtedly in a legal and ethical gray area. After having discussed this with their legal department, Topicus has given one-time permission for data processing as described in this thesis with the following arguments.

- The purpose of this study is to check the viability of predicting NPS responses based on usage data. It does not entail or build upon a practical application.
- The data is pseudonymised: any fields relating to individual users are scrambled, making it practically impossible to relate the data back to specific individuals.

Chapter 2

Background

This section gives background information on all relevant subjects discussed in this thesis, such that no external sources are needed to understand its content. This study focuses on predicting Net-Promoter Scores, explained in section 2.1, using most of the approach of the field of web mining, outlined in section 2.2. Machine learning is used to apply the predictions, discussed in section 2.3. One field of research specific for repetitive patterns is discussed in section 2.4.

2.1 Net-Promoter Score (NPS)

The Net-Promoter Score is a metric to measure customer loyalty and to some extent customer satisfaction [49]. Respondents are asked to answer one question: *"How likely is it that you would recommend our system to a friend or colleague?"*. The answer score is based on a scale of 0 to 10. Respondents giving a 9 or 10 are called *Promoters* and are considered loyal users that stimulate company growth by way of word-of-mouth advertising: they are likely to promote the system to others. Respondents giving a score of 0 to 6 are called *Detractors* and are considered users that are to some extent dissatisfied. Respondents giving a 7 or 8 are called *Passives*. They fall between the categories of Promoters and Detractors and could be labeled as moderately satisfied users who would easily switch to a cheaper system. The system's NPS value is calculated by using $NPS = \frac{Promoters - Detractors}{respondents} \times 100$. Or, put differently:

$$NPS = (\%Promoters - \%Detractors) \times 100$$

As such, the range is -100 to 100. Passives are added solely to the total respondents and shift the NPS value closer to zero.

The NPS questionnaire is supplemented with one or more requests for elaboration to the answer on the main question. From a corporate perspective, this is useful for a deeper analysis of user attitude towards the system.

The validity and reliability of this metric is much debated in scientific studies [21, 24, 33, 36, 37, 44]. Most of these studies focus on the inability to measure satisfaction and its inherent nuances: only a few such as [21, 36] claim the NPS does not actually measure loyalty. Irrespective of the extent of its validity, companies use the NPS metric to ascertain their overall performance which encourages more research into the subject.

2.2 Web usage mining

The internet is an important part of our lives these days. The Dutch government even went as far as saying that fast internet access is a primary need [32]. To offer good

services to their users, website owners constantly try to improve their websites. This is often done using explicit feedback from their users, but a less time and resource intensive way to collect this data is through the use of web mining [13]. This area focuses on analyzing the available data of a website to discover patterns that can lead to an optimization of the offered services. Web mining can be divided into three categories. *Web content mining* focuses on the information that is served on web pages, *web structure mining* focuses on the links between web pages, and *web usage mining* is all about the behavior of users.

Analyzing user behavior within an application can give interesting insights. A few of areas using this analysis are decision support in business, marketing and web design, personalization and recommendations [12, 47], and web caching. Many studies into web usage mining focus on the areas of e-commerce and search engines. Three main tasks are identified with web usage mining [58].

Preprocessing Often referred to as the most time consuming task in the process [29], preprocessing is about preparing the data for the next task. Difficulties lie (among other things) in identifying individual users, dividing a user's data in different sessions and handling missing data due to cached page views. Sometimes content preprocessing is also necessary. For example, pages can be classified according to their subject or in the case of e-commerce, product class. The same goes for structure preprocessing, e.g. classifying a web site into a hierarchical product class model in e-commerce.

Pattern discovery There are several techniques used in web usage mining for pattern discovery. *Statistical analysis* is the most common technique. It looks at descriptive statistics such as frequency, mean and median values about page visits, visit time, visit length, active users, user session length, etc. Another technique is *association rule discovery*. It looks at sets of pages that are often accessed together in a single session, irrespective of whether the pages are directly connected through hyperlinks. For example, in a web shop this might show that users shopping for electronics often also shop for sports clothing. This might indicate to the web shop owner that those categories could be placed closer together. The apriori algorithm is one of the most commonly used algorithms. *Clustering* tries to group pages or users together that have similar properties. This is especially useful when giving personalized recommendations to users. *Classification* attempts to categorize users, pages or sessions into predefined classes. Examples of classifiers are neural networks, decision trees, naive Bayesian and k-nearest neighbors. This can be used for targeted marketing. Finally, *sequential pattern mining* looks at temporal patterns in the scope of multiple sessions. A few examples are trend analysis, change point detection and similarity analysis. This task is called the pattern extraction phase in the current project.

Pattern analysis This task intends to draw conclusions from the discovered patterns. Commonly used mechanisms for analysis are SQL and OLAP operations with data cubes. This study uses machine learning to find predictive value in the patterns found in the previous step. The model training phase is aimed at this.

2.3 Machine learning

Naqa et al. describe machine learning as "an evolving branch of computational algorithms that are designed to emulate human intelligence by learning from the surrounding environment" [20]. Arthur Samuel coined machine learning as a term in

1959, defining it as "the field of study that gives computers the ability to learn without being explicitly programmed" [50]. Put differently, machine learning algorithms learn a mapping function from data, such that they can generalize that to new, unseen data.

The general process of building a machine learning model is as follows. One starts with a dataset in which each row is a data point called a *sample* and each column is a *feature* (sometimes also called an *attribute*). These features are values for each sample that are used by the model to make its predictions. This dataset is split into two subsets: a training and a test dataset. This training dataset is used to train the model, after which the model's performance is measured by applying the model on the test dataset. This performance can be increased by preprocessing the dataset, e.g. using normalization or outlier removal. A well-performing model is highly generalizable to new data.

The two largest directions of machine learning are supervised [56] and unsupervised learning [18, 46]. *Supervised* learning makes a predictive generalization based on a training dataset where the *target attribute* (or *label*) is known. If the outcome data type is discrete, it uses classification methods. If the outcome data type is continuous, it uses regression methods. *Unsupervised* learning is not based on training data. It tries to discover patterns based only on input data. There are also other directions. Examples are semi-supervised learning [9], in which only part of the classes of the training data is known, and reinforcement learning [61], in which a model learns based on continuous feedback. This study uses supervised learning since the focus is on training a model so that it can make predictions.

The dataset used for training has to be of certain quality. If the dataset is full of errors, missing data or otherwise of low quality, the model becomes useless. Several preprocessing methods can be used to deal with this.

Normalization can be applied to prevent dominance of certain features and prevent computing problems. This technique puts the features in the same range, for example by subtracting each value from that feature's mean and dividing by the feature's standard deviation.

A dataset might contain samples that are exceptions to the general rule and are thus not representable. There are various algorithms to detect such outliers, such as DIS, kNN and RNN [68].

Most models can not deal with missing values in the dataset. One solution is to simply discard a sample that does not have values for each feature. This has several drawbacks such as adding bias and decreasing representativeness. A better solution is data imputing. This is a concept similar to interpolation: it replaces missing data with estimated values.

Another factor that can impact a dataset's representativeness is class imbalance. If 90% of samples are of class A, a model can obtain 90% accuracy by always predicting class A. The effects of class imbalance can be mitigated in several ways. Collecting more data is an important one, although rarely feasible. Penalized models put more weight on misclassifying the underrepresented classes during training. Upsampling (or oversampling) copies samples from the underrepresented class, while downsampling (or undersampling) removes samples from the overrepresented class. Both resampling methods even out the amount of samples in the classes. Upsampling can also be applied by generating synthetic samples. The most commonly used algorithm is SMOTE [10], which uses an element of randomness to mitigate the chance of overfitting.

One pitfall that must be avoided with predictive modeling is being on one end of the bias-variance tradeoff. High variance means that a model is sensitive to small fluctuations in the training set. The model then fits the training data too well, but not any unseen data: the model is not generalizable. This is also known as overfitting. On the other hand, bias means too many assumptions are made and the model is thus not sensitive enough to the training set, resulting in underfitting.

One commonly used method of evaluating a model's performance is k-fold cross-validation [59]. The dataset is split into k equally sized subsets. For k times, $k-1$ subsets are used as the training dataset while the other one is used as the test dataset. The performances of the k evaluations are averaged. k is often chosen to be 10. This method eliminates the chance that a model's performance evaluation is an outlier.

Model performance can be measured with different metrics. The error in this context is the difference between a predicted value and the value of the target attribute of a sample. For regression models, the most commonly used metrics are R^2, mean squared error (MSE), root mean squared error (RMSE) and mean absolute error (MAE). R^2, also known as the coefficient of determination, is a measure of the variance explained by the model ranging from 0 to 1. If this value is 1 it means all variance in the data is explained by the model, but this most often means the model is a victim of overfitting. The MSE takes the mean of all squared errors, as the name suggests. The RMSE does the same and takes the root, changing the unit of this error metric back to the target attribute's unit. The MAE again does what the name suggests and has the added advantage that the metric is better comprehensible. Both the RMSE and MAE have the advantage that they are in the original unit of the predicted value. The RMSE is more influenced by large error outliers, whereas the MAE is more affected by the error variance.

For classification, commonly used metrics are the confusion matrix and a few of its derivatives: precision, recall and accuracy. Each row specifies the predicted classification, each column shows the true classification and each cell contains the amount of samples that have that combination of predicted and true value. The precision of each classification measures the proportion of accurate predictions to the predictions made for that classification. The recall of each classification measures the proportion of accurate predictions to the actual samples of that classification. The accuracy is the proportion of accurate predictions to the total amount of samples.

A few model concepts are outlined here. Note that each model has numerous variations, each with their own advantages and disadvantages. Several approaches can also be combined, either by joining the outputs [5, 66] or by integrating multiple algorithms [4]. Each one listed below is used in this study, be it the original algorithm or a variation based on the original.

Decision trees This represents a tree in which a decision is made at each node based on a feature (e.g. "is $x_1 \geq 2$?"). This way, the tree is traversed and ultimately a leaf node is selected, which represents a classification. Common algorithms are ID3, C4.5 and CART. A few advantages of a decision tree model are its simplicity, capability to handle both continuous and discrete data, capacity to process large datasets, and above all: its understandability to humans. Its simplicity also gives rise to its disadvantages: decision trees tend to be relatively inaccurate and are prone to failures if the dataset changes too much. Variations on this model used in this project are: decision stumps, which are decision trees of depth 1; random trees, which are decision trees that use random subsets of features; random forests, which are ensembles of decision trees that mitigate the

chance of overfitting; and Chi-square automatic interaction detection (CHAID), which prunes decision trees based on the chi-squared attribute relevance test.

Artificial neural networks These simulate the behavior of the human brain. It consists of many, simple units (neurons) that are activated by other neurons or input sensors connected to the environment. The connections between neurons are weighted. Learning occurs by changing the weights of connections until desired behavior results from the specific configuration. An advantage of neural nets is that they perform better at incremental learning. Their accuracy is similar to decision trees, although training time is much longer [19, 41]. One of the largest disadvantages is the lack of explanation. The logical steps of the process with which a neural network algorithm comes to a result is practically impossible to comprehend by humans. Neural nets can be used for both classification and regression tasks. Deep learning is a variation that uses multiple hidden layers of neurons.

Naive Bayes These classifiers are based on Bayes' theorem. They assume all features are independent, simplifying the probabilistic calculations. This assumption is rarely true, hence the 'naive' term. Despite the independence assumption, Naive Bayes classifiers often perform competitively. They are trained fast and scale well.

Support Vector Machines A Support Vector Machine places one or more separating hyperplanes between the data points in the training set. It tries to maximize the distance to all data points of the different classes. The side on which the data points are with respect to the hyperplane determines which class they belong to. An important advantage of this classifier is its independence on the amount of features. However, this technique has a fairly long training time and it needs the data to be in similar ranges. It can also work with regression problems.

Rule induction These iteratively add and remove rules to come up with an optimal set. Rule induction models are similar to decision trees, as they both use rules. An advantage is that rules are easily understandable by humans and computers (i.e. they can be expressed in programming language syntax without effort) and one's own rules can be manually added. A disadvantage is that the model does not scale.

Linear regression Regression works with numerical values and tries to predict a continuous value. Linear regression tries to fit a linear equation to the observed data. Polynomial regression tries to do the same, but with a higher order equation. The Generalized Linear Model (GLM) is an optimized variation on linear regression that is more flexible. Linear regression works well when the relationship between the features and the target attribute is linear (or polynomial), but can not properly handle situations where this is not the case.

k-Nearest Neighbors This model looks at a sample's k nearest neighbors. The distance to its neighbors is calculated in n-dimensional space based on its n features. An example distance metric is the Euclidian distance (e.g. in 1-dimensional space, the distance between point (3) and point (5) is 2). In the case of classification, it classifies the sample based on the majority of the neighbors. With regression, the predicted value is the neighbors' average. It can be useful to let closer neighbors weigh more than neighbors farther away. Advantages are that

this method is robust with non-linear relationships (as mentioned in the description for linear regression) and it is relatively simple. However, its training time is long and is sensitive to unbalanced datasets.

2.4 Sequential pattern mining

Agrawal and Srikant introduced the problem of sequential pattern mining in 1995 [1]. Their goal was to find patterns in sequences, specifically with the use case of customer transactions in retail. In the scenario of video rentals: if customers typically rent "Star Wars", then "Empire Strikes Back" and then "Return of the Jedi", a sequential pattern is identified. The input is a database of customer transactions, having three fields: customer identification, transaction order and the items contained in that transaction. The transaction order field is often a date and/or time value. Continuing the video rental example, table 2.1 shows a possible customer transaction database. Quantities are not taken into account.

TABLE 2.1: Example customer (video rental) transaction database used as input for sequence pattern mining

customer ID	transaction date	items	items (shortened)
1	2011-03-01	(Battlestar Galactica, Star Wars)	*(B, S)*
1	2011-03-03	(Empire Strikes Back)	*(E)*
2	2011-05-05	(Caprica, Star Wars)	*(C, S)*
2	2011-05-06	(Alien, Empire Strikes Back)	*(A, E)*
1	2011-05-07	(Minority Report, Return of the Jedi)	*(M, R)*
2	2013-06-07	(Killjoys, Return of the Jedi)	*(K, R)*

Each product (or video in this example) is defined as an *item* and each transaction as an **unordered** *itemset*, *event* or *element*. Note that the parentheses are usually omitted when denoting an itemset of length one. Items in the same itemset are written in alphabetic order, even though they are in an unordered set. To transform a customer transaction database into a format used for sequential pattern mining, each itemset is grouped per customer and ordered by transaction date. An ordered list of transactions is called a *sequence*. The keys of a sequence database are only used as sequence identifiers. Table 2.2 shows the mapping of the terminology used in literature onto the real-life example. The purpose of table 2.2 is to aid in understanding the original problem of Agrawal and Srikant.

Table 2.3 shows the sequence database of our example. This is the format that is analyzed in sequential pattern mining algorithms. SID is short for sequence ID.

Sequential patterns have to adhere to more requirements. The minimum *support* (or *minsupport*) is defined as the minimum amount of sequences a pattern occurs in. Patterns that occur at least in *minsupport* sequences are called *frequent patterns*. Additionally, frequent patterns must be *maximal*. This dictates that a pattern can not be a subpattern of another frequent pattern.

Applying this to our example and setting minsupport to 2, pattern SER is considered a frequent. Pattern SE is not: it is a subpattern of SER, which means it is not maximal.

TABLE 2.2: Sequential pattern mining terminology mapping

terminology of [1]	terminology in a real-life example	example value
item	product (or video)	S
itemset/element	transaction	(BS)
sequence	customer history	$<(BS)E(MR)>$
sequence database	transaction history	{1: $<(BS)E(MR)>$, 2: $<(CS)(AE)(KR)>$}

TABLE 2.3: Example sequence database

SID	sequence
1	$<(SB)E(RM)>$
2	$<(SC)(EA)(RK)>$

Countless algorithms have been developed since the initial problem was first proposed by Agrawal and Srikant. Each approach since then has had its optimizations. In addition, numerous variations of the original problem required specifically tailored solutions. Giving an introduction to several of the main approaches will support comprehension of the field of sequential pattern mining. All approaches use the apriori property defined by Agrawal and Srikant that says that a pattern is frequent only if all of its subpatterns are frequent. The AprioriAll algorithm [1] was the first to use this principle.

The Generalized Sequence Pattern (GSP) algorithm [57] was the first practical algorithm that built upon AprioriAll. Each item is counted during the first database pass and all non-frequent items are discarded. A loop is entered in which each iteration finds patterns of increasing length. In the candidate generation step, pairs found in the previous iteration are merged. If, for example, the previous iteration found abc and bcd, these are merged into $abcd$. In the candidate pruning step, candidates are pruned that contain an infrequent subsequence. $abcd$ would be pruned if acd was infrequent. Candidates are eliminated in the next step if their support proves to be insufficient, based on a traversal of the sequence database. This loop continues until no candidates or no frequent patterns can be found. Disadvantages of this approach are the amount of database passes and the amount of generated candidates. A strength compared to AprioriAll is the candidate pruning. It uses a horizontal format, illustrated in tables 2.3 and 2.4.

The algorithm Sequential Pattern Discovery using Equivalent Classes (SPADE) [67] uses a vertical format. Take the sequence database shown in table 2.4, with *minsupport=2*. SPADE first converts this to the vertical format shown in table 2.5a. EID is short for element ID (or event ID), of which the values are shown in factors of ten to emphasize the distinction between the SIDs. The conversion to vertical format requires only one database scan. Next, the *ID lists* of each item are generated, partially illustrated in table 2.5b. The ID lists of infrequent items are discarded, such as item e in our example. Candidates of length 2 can be found by joining rows of the ID lists of candidates of length 1: if they share the same SID and if their EID is sequential, join them. Table 2.5c shows the result of joining a with b and vice versa, and table 2.5d shows the result of joining ab with ba. This continues until no frequent patterns can

be found anymore. SPADE reduces the amount of database scans, but still requires large sets of candidates.

TABLE 2.4: Example sequence database for the SPADE algorithm

SID	sequence
1	<*a(abc)(ac)d(cf)*>
2	<*(ad)c(bc)(ae)*>
3	<*(ef)(ab)(df)cb*>
4	<*eg(af)cbc*>

TABLE 2.5: Example progression of SPADE

SID	EID	itemset
1	10	*a*
1	20	*abc*
1	30	*ac*
1	40	*d*
1	50	*cf*
2	10	*ad*
2	20	*c*
2	30	*bc*
2	40	*ac*
3	10	*ef*
3	20	*ab*
3	30	*df*
3	40	*c*
3	50	*b*
4	10	*e*
4	20	*g*
4	30	*af*
4	40	*c*
4	50	*b*
4	60	*c*

(A) Sequence database in vertical format

a		***b***		...
SID	EID	SID	EID	...
1	10	1	20	...
1	20	2	30	
1	30	3	20	
2	10	3	50	
2	40	4	50	
3	20			
4	30			

(B) ID lists of patterns length 1

ab			***ba***			...
SID	EID (a)	EID (b)	SID	EID (b)	EID (a)	...
1	10	20	1	20	30	...
2	10	30	2	30	40	
3	20	50				
4	30	50				

(C) ID lists of patterns length 2

aba				...
SID	EID (a)	EID (b)	EID (a)	...
1	10	20	30	...
2	10	30	40	

(D) ID lists of pattern length 3

An approach that greatly reduces the effort of candidate generation uses pattern growth. Prefix-projected sequential pattern mining, or PrefixSpan, uses this method [25]. It sees subsequences as prefixes and their corresponding suffixes as projected databases. Patterns are grown by examining frequent patterns in each projected database. We take same example example sequence database again, shown in table 2.6a. PrefixSpan first finds all length-1 sequential patterns, i.e. *a*, *b*, *c*, *d*, *e* and *f*. Each pattern is considered a prefix, where the suffix is its projected database. Only the first occurrence in each sequence are taken into account. Table 2.6b shows the projected database of pattern *a*, where underscores indicate that the prefix can also occur there. These projected databases are used to grow the pattern. In the case of

our pattern a, the frequent items are a, b, $\_b$, c, d and f. The length-2 patterns with prefix a are found to be aa, ab, (ab), ac, ad and af. Each is considered a prefix and their projected databases are generated. The projected database of prefix aa consists of <$(\_bc)(ac)d(cf)$> and <$(\_e)$>. As none of the items occur in both sequences, there are no frequent patterns. Table 2.6c shows the projected databases of prefix ac. This contains frequent patterns a, b and c, resulting in the length-3 patterns with prefix ac being aca, acb and acc. Doing this again for prefix acb results in the projected database in table 2.6d. This process is recursively repeated for the frequent patterns found in each projected database until no more frequent patterns can be found. Even though the projected databases keep shrinking, constructing them physically is costly. This can be minimized by using pseudo-projection: storing the starting index of the projected suffix instead of the whole sequence. This works if the database can be held in memory. If it can't, a combination of physical and pseudo projection can be made.

TABLE 2.6: Example progression of PrefixSpan

SID	original sequence	***a***-projected db	***ac***-projected db	***acb***-projected db
1	<$a(abc)(ac)d(cf)$>	<$(abc)(ac)d(cf)$>	<$(ac)d(cf)$>	
2	<$(ad)c(bc)(ae)$>	<$(\_d)c(bc)(ae)$>	<$(bc)(ae)$>	<$(\_c)(ae)$>
3	<$(ef)(ab)(df)cb$>	<$(\_b)(df)cb$>	<b>	
4	<$eg(af)cbc$>	<$(\_f)cbc$>	<bc>	<c>
	(A)	(B)	(C)	(D)

Chapter 3

Feature discovery

The first part of this project answers research question 1 by determining what might be an influence on the individual NPS responses. This is necessary for the subsequent phases, because performing arbitrary magic without a direction on a large dataset doesn't get viable results. This introduction first describes the format of data features and the reason for the general approach that was used. Section 3.1 describes the efforts and steps taken, including highlights of the results stemming from the different steps, while section 3.2 lists the actual results of the overall process of feature discovery. Section 3.3 concludes this chapter by listing a selection of data features that this project focuses on.

Each feature is meant to become an aggregation of a user's usage data into one or several metrics. An example of the result of such a data feature analysis could be "During the 90 days prior to his NPS response, user x has logged in 20 times per day on average". This has several components. The data feature in this case is the daily average login frequency. The result format is thus a floating point number. If the machine learning algorithm chosen later on requires discrete values, such a transformation will be applied. The time period of 90 days is implicit in all data feature definitions in this chapter and is an attempt at normalizing the available dataset per NPS response. This exact duration is selected to have a balance between having sufficient log data available, and potential issues resulting from improvements made to the system's structure.

The purpose of this part of the research is acquiring a prioritized list of data features to focus the usage data analysis efforts on. In an ideal situation, this would be accomplished by performing a full-fledged user experience study. Unfortunately, this is out of the scope of this project and that is why a heuristic approach was chosen instead.

3.1 Efforts

The following subsections describe the efforts taken to discover possibly relevant features in semi-chronological order. *Semi* because some efforts took place in parallel, which is stated explicitly if relevant. In addition to the different activities described here, several features were also discovered during undocumented, informal conversations with Topicus' developers, data analysts and other friends. The results of these chats have been included in section 3.2. Whenever a term such as "interesting" or "relevant" is used in this chapter, it means the construct in question is directly relevant for this research project.

Specific discussion topics were prepared for the conversations to give a clear direction in which to search for features. In addition to these topics, most conversations ended in a brainstorm-like discussion in which the main thread was basically the

first research question: *what might be an influence on the user loyalty towards Somtoday?* The prepared topics are outlined here, after which the different subsections specify how these were used.

Note that most of the discussed topics focus mainly on negative experiences with the system. This is because early conversations brought to light that proper use of the system might only be visible in efficient use and the absence of indicative patterns. Nevertheless, section 3.2 includes some suggestions that include these efficient 'power users'.

Integration into daily routine *"At which moments during a regular weekday do you use which functionalities of Somtoday?"* was the question formulated for this subject. The motivation for this is that the system is meant to be integrated into teachers' daily routine. Take for example absentee registration. If a teacher writes this down on paper during class and enters this into the system after school, s/he is essentially doing double work and that might influence the NPS response. The purpose of this subject was thus to see if something came up that was out of the ordinary. This was not the case. Except for the mentioned example, everyone described their routines as expected beforehand (which is based on personal preference): absentee registration during or right after class, homework planning during class or breaks, grade registration during breaks, in the evening or during the weekends, asessment planning during or right after class, etc. These results are not repeated in the subsections, seeing as there was nothing more worth mentioning.

Annoyances This is basically the research question of this chapter, only framed differently: *"What are your greatest annoyances about the use of Somtoday and which functionalities does this concern?"* This phrasing, as opposed to the literal research question, is easier to interpret for teachers. This way the interviewee does not have the opportunity to ascertain whether an answer is relevant or not.

User clustering This subject originated in the idea that some users might use the system efficiently (the previously mentioned 'power users') and some very differently. In turn, it might be that some features are specifically influential for certain types of users. For example, it might be that older teachers spend more time per page finding their way than younger teachers. Lengthy click paths might in that case be more significant than for younger teachers who quickly find their destination and who deprioritize the amount of clicks to some extent. Teachers were asked: *"How do you think that your colleagues use Somtoday differently than you and how would you differentiate different groups?"* If they could not give an answer, which was understandable due to the potential difficulty in interpreting the question[1], examples were given like the distinctions between young and old, and math and English teachers.

The interviews were recorded by writing down (during the interview) possibly relevant system behaviors and example situations. The same was done with the qualitative data: possible features were immediately extracted from the data without being subject to in-depth analyses.

[1] Note that much thought was put into phrasing this question correctly. This question was only asked in face-to-face interviews, so more explanation could be given about its purpose.

3.1.1 Service desk interview and qualitative data

Efforts concerning Topicus' service desk were twofold. On March 13th, 2017, the head of the service desk was interviewed. During this interview it was explained how primary (business-related) communication between Topicus and its customers (i.e. schools) works: each school has an application manager that acts as a spokesperson for all school employees. This has several advantages for Topicus, but is mainly a disadvantage for this study: practically all complaints and questions received by the service desk are from application managers. The only relevant result from the interview was this explanation and a warning that the associated qualitative data will probably not result in relevant features. The latter proved not to be true.

The second part, however, was more useful for feature discovery. As was explained in section 1.4.5, a ticket is created in the issue tracking system JIRA for each received complaint (i.e. bug report) or question that can not be answered immediately by the service desk. There were 2314 tickets created between January 2nd, 2014 and April 25th, 2017. A random sample of about 40 tickets was inspected from the 303 tickets created in the last 60 days (i.e. between February 25th and April 25th, 2017). Almost all of them were about highly specific, malfunctioning functionality from which no relevant features could be deduced. However, about 10% of the tickets was about faulty authentication services, i.e. failing log-ins or forced log-out situations. There are data available about these authentication scenarios, so this is a relevant feature.

3.1.2 Customer relations interviews

As opposed to Topicus' service desk, their customer relations department does have regular contact with teachers. The employees of this department are considered and called consultants, and each consultant is responsible for about 22 schools. Three consultants were interviewed, primarily about the subjects of *annoyances* and *user clustering*. These three were selected based on the explicit recommendation of their supervisor.

The first interview was on March 13th, 2017. The main theme was a specific school that had issues with Somtoday during the academic year 2015-2016. They held a negative opinion about the system and ultimately left as customer. The interviewee explained why this happened (in hindsight): the school's teachers complained that they were missing a great deal of functionalities, while unbeknownst to them, their application manager had disabled about 70% of Somtoday's functionalities. Metrics about this can be extracted from the data by looking at prevalence of functionality use and at which functionalities are disabled.

June 1st, 2017, was the date for the second interview. The key takeaway was that the usage of some functionalities can be quite complex initially, which teachers may find annoying. This can be gathered from the data by looking at long clickstreams, which might indicate that a user can't find their target page or simply needs too many clicks to complete an action. The latter can also be investigated by looking at repetitive tasks. Furthermore, ideas for user clustering were age, gender, years on the job, courses taught, whether the teacher teaches junior or senior classes, the class levels taught (e.g. HAVO or VWO) and the amount of system usage within or outside of scheduled hours.

The final interview was on June 19th, 2017. The interviewee gave a short list of annoyances that he had encountered in his experiences with teachers: repetitive tasks, long clickstreams (e.g. the amount of clicks between associated pages), login

frequencies, prevalence of functionality use, and how often users switch between Somtoday and Somtoday Docent. The latter was suggested because it might indicate missing functionalities or other reasons to favor one over the other. It was suggested that teachers of the higher class levels (i.e. HAVO/VWO/gymnasium) use more of the assessment registration functionalities (e.g. grade registration) than the teachers of the lower class levels (i.e. VMBO), which use the administration functionalities more often (e.g. absentee registration). This can be researched by looking at the class levels taught or the prevalence of functionality use. Another suggestion for user clustering is age. Specifically because intuitively, younger people are quicker in finding their way in a digital system.

3.1.3 Topicus' student convention

This event was held on June 7th, 2017, during which all Topicus' graduation interns were given the opportunity to present their research. This chance was taken to introduce this study and set up a discussion into the research question of feature discovery.

The time allotted for the presentation and discussion was only fifteen minutes, so the experience wasn't particularly extensive. Nevertheless, many good suggestions were given by the audience. These tips all boiled down to the analysis of clickstreams, prevalence of functionality use, or a combination of both. An audio recording was made of the presentation and subsequent suggestions.

3.1.4 Teacher survey

A digital survey was sent by e-mail to 27 teachers on June 13th, 2017. The last of the 12 responses was received 14 days later. These 27 teachers have explicitly indicated to Topicus that they want to cooperate with initiatives to improve Somtoday, hence the small subset. Consequently, this group is not representative, which was neither the intention nor the requirement: the purpose was to gain insights, not to poll the general opinion. Nonetheless, these 27 teachers are from 25 different schools throughout the Netherlands and together they teach almost all available subjects, from math and physics to German and economics. The survey results were collected and stored by SurveyMonkey[2].

The (open) survey questions were essentially those from the *integration into daily routine* and *annoyances* subjects, supplemented with an answering format that respondents could use multiple times per question. The format for the former subject was "*I do <task> most often during <time period>.*" where *<time period>* could be something like *during class*, *during breaks* or *in the evening*. The format for the latter subject was "*I find it annoying that <annoyance> with <functionality>.*". Both of these formats have been translated from Dutch, seeing as the survey was in Dutch. Additionally, the *user clustering* subject was potentially covered by asking if the respondents were prepared to further discuss during a phone conversation, on which half responded positively. The purpose of this was not only to get a more nuanced opinion, but also to ask about user clustering. This eventually turned out not to be necessary. In addition to these questions, the survey asked about two basic demographics: date of birth and courses taught.

Most responses were not directly relevant. Some were about the user interface being too cluttered, either in general or on mobile devices, missing information and a schedule component that should be placed differently. Others were about failing functionalities, such as planned homework that stays stationary when rescheduling

[2] https://www.surveymonkey.com/

the associated class, or missing functionalities, such as an auto-save function for notes and absentee registration for multiple days. There were also a few interesting annoyances. Several respondents mentioned that some functionalities require too many clicks, such as when switching to a next lesson or class, or with homework planning. Another annoyance was the (perceived) high occurrence of system downtime.

3.1.5 Teacher interviews

On June 22th, 2017, several Topicus employees had an appointment at the Etty Hillesum Lyceum in Deventer to introduce new improvements to Somtoday Docent. In parallel to this presentation, the teachers' break room was available for administering short interviews. Between 10:00 and 11:00 AM, five teachers that were on their break were interviewed. They were selected based on their availability in the break room and their willingness to spend some time on an interview. This is clearly not representative, as it concerns a very small subset of teachers and only from one school.

The subjects of annoyances and user clustering were discussed, but next to no new information came to light. Concerning the latter subject, there was one economics teacher suggesting that not age would be the distinctive property, but personality. This, however, is not a feasible feature to extract from the available data.

3.1.6 Qualitative NPS data

As is described in sections 1.4.1 and 1.4.5, the NPS data include answers to an open question in which respondents can elaborate upon the reason behind their NPS response. Of the 9160 NPS responses collected between August 25, 2016 and May 2nd, 2017, the latest 600 or so responses were browsed through. Over 50% were about the user interface not being sufficiently clear or intuitive or similar complexity issues.

Other recurring themes were the length and duration of clickstreams, login frequency, downtimes frequency, and page loading times; the latter especially in the context of repetitive tasks.

3.1.7 Qualitative feedback data

Between November 13th, 2015 and June 27th, 2017, Somtoday's integrated feedback button received 1953 submissions. The purpose of this specific data collection is to get new ideas and suggestions for improvements of Somtoday. About 40 responses submitted between February and June 2017 were sparsely browsed and checked for feedback that could be translated into data features.

There were only two points of interest encountered. The first is about fast switching between student groups, e.g. when a new class starts. Secondly, several instances were found where the responses effectively described annoyances related to repetitive tasks.

3.2 Results

The results of the efforts described in section 3.1 are aggregated here. These were selected based on whether they are measurable from the available data. The categories are listed in no particular order. They are explained with real-life examples and include specifications of one or multiple data features. This section purposefully uses generic specifications as much as possible: this way, they can be generalized to similar systems. Effort has been made to normalize these features over different

aggregation levels, such as individual NPS responses (the previously mentioned 90 day time period) and schools (see features DF6 to DF10).

Section 1.1 mentions the key factors of data system adoption: data availability, reliability, findability, and (supported) interpretability. Data mentioned in this context refers to the data contained in an educational data system. The data availability and reliability are difficult to accurately measure. Data findability can be measured by looking at the clickstreams and whether the user easily ends up on his target page. The data's (supported) interpretability is a bit more complex. In the case of Somtoday this can be measured by looking at the prevalence of functionality use, while treating the generation of reports as a distinct functionality.

Prevalence of functionality use [65]

This category focuses on the measured use of a system's distinct functionalities. In the first interview of section 3.1.2, an example of this category is described. If Topicus had looked at the prevalence of functionality use around that time, the whole situation could've been prevented. The hypothesis that this is an especially relevant feature category has two lines of reasoning. The first is that a positive opinion about a specific functionality results in its increased use. This is the simplest and most intuitive argument; if one likes something, one uses it more. However, the nature of educational data systems poses an important rebuttal: teachers are obligated by their employer to use certain functionalities such as grade registration and absentee registration. This can partially be taken into account by also looking at the distribution of their usage. The hypothesis for this line of reasoning is: the higher the prevalence of a functionality's use, the more positive the user's opinion.

The causality of the previous line of reasoning can also be reversed: someone that uses a functionality more often consequently develops a strong opinion about it. This can be increasingly positive on the one hand, for example when a user gets accustomed to utilizing a functionality. This has a scientific basis in the Mere Exposure Effect [23], which shows that over time, mere exposure results in a positive opinion. On the other hand, this opinion can become increasingly negative in the case that a user continuously encounters the same annoyances. The hypothesis resulting from this is: the higher the prevalence of a functionality's use, the more extreme the user's opinion (either negative or positive).

Both lines of reasoning indicate that the prevalence of functionality use is worth researching. Note that each data feature applies for each functionality of the system. This inherently means that during preprocessing, each log entry must be mapped to a functionality. Some log entries can be mapped to several functionalities. For example, generating reports is a functionality that is included in absentee registration, grade registration, and more. Additionally, note that a comparison should be made of prevalences of the same functionalities between users, not of prevalences between different functionalities per user.

Prevalence can be measured in different ways. The simplest is (per functionality) the amount of requests. A time period of a week should be good, seeing as this is also a common time unit used in education. This assumes these functionalities are used somewhat uniformly over the period of several weeks. For example, grade registration of exams at the end of a period is not taken into account. To deal with this to a limited extent, the average of monthly requests can be considered.

DF1: the average amount of weekly requests per functionality

The same can be done with a time unit. Seeing as the log data does not include time spent on a page, this would have to be estimated based on the timestamps of the requests. This can be done by calculating the time interval until the user's next request. If that interval is not too large, then it can be seen as time spent on the page. *Not too large* in this case would be an interval of which it would be reasonable that the user was still actively using the functionality. This would be something in the order of a few minutes, seeing as the system is built for data input and output, which both usually don't take long with the amounts of data involved here. A disadvantage of this method is that the time spent on the last page can't be estimated. This is the case for all data features that measure time.

DF2: the average amount of weekly time spent per functionality

As mentioned before, looking at the distribution of functionality usage might also prove useful. A relatively easy metric for this is the time spent on a functionality per session. Imagine a teacher spent a total of one hour on absentee registration in six different sessions, then the value of this metric would be 10 (minutes per session). This shows a clear distinction with a teacher who accumulates his work of absentee registration and inputs this into the system once a week for a duration of one hour. Feature DF2 would produce the same value, but DF3 would show the distinction.

DF3: the average amount of weekly time spent per session per functionality

Looking at the time of day of functionality use might prove useful. Interviews indicated that the exact time of day wouldn't be that relevant, but whether a functionality is used during or outside of class hours might be (recall from section 1.4 that class schedules are available in the dataset). This might indicate to what extent certain functionalities are integrated into the teacher's daily routine, especially concerning the absentee registration. This can, again, be measured in terms of number of requests and units of time. Additionally, seeing as this is a binary classification of requests, the measured feature is a ratio between number of requests or time spent during and outside of class hours.

DF4: the average ratio of weekly number of requests within vs. outside of scheduled class hours, per functionality

DF5: the average ratio of weekly time spent within vs. outside of scheduled class hours, per functionality

Scenarios were encountered where the absolute values of features DF1 to DF5 would be considered invalid, such as the example mentioned in section 3.1.2. In this case the school doesn't use a functionality at all. Whatever the underlying reason is, if the whole school doesn't use a functionality, a prevalence of zero has no meaning in the context of this research. The feature values should thus be normalized. This can be done by taking values relative to the school's average. Take for example a school where the per-teacher average time spent on absentee registration during the second week of 2017 is one hour. If a teacher spent half an hour that week on absentee registration, he has a relative value of 0.5. The feature value would then be an average of that value over multiple weeks.

DF6: the average amount of weekly requests, relative to the school average, per functionality

DF7: the average amount of weekly time spent, relative to the school average, per functionality

DF8: the average ratio of weekly number of requests within vs. outside of scheduled class hours, relative to the school average, per functionality

DF9: the average amount of weekly time spent per session, relative to the school average, per functionality

DF10: the average ratio of weekly time spent within vs. outside of scheduled class hours, relative to the school average, per functionality

Repetitive tasks

These are inevitable in administrative systems. Nonetheless, users might find them annoying, especially if the individual task requires many clicks or takes a long time to perform. A few requirements have been constructed to further specify what constitutes as a possibly influential repetitive task.

A repetitive task that happens in just one or two sessions probably has no real effect on the NPS response. Therefore the first requirement is that a repetitive task occurs in multiple sessions. This is the *inter-session occurrence* requirement. Note that if the partitioning into sessions would be omitted, no such distinction could be made. Subsequently, if a task is performed only once in a session, it might simply be a navigational pattern. However, if it concerns for example grade registration and the task requires three clicks per student, this has a far greater measure of repetitiveness. For that reason, a minimum of five occurrences within each session is required for a repetitive task to have an influence on the NPS response. The amount of occurrences is based on the fact that teachers often deal with whole classes within the system. This is the *intra-session occurrence* requirement. These are the requirements that apply to features DF11 to DF13:

$$\text{inter-session occurrence} \geq 3$$

$$\text{intra-session occurrence} \geq 5$$

By using the distinction of features DF11 to DF13, the several dimensions of the concept of repetitive tasks are taken into account. These different dimensions are explained in section 5.2.

DF11: the amount of distinct repetitive tasks

DF12: the amount of iterations of repetitive tasks

DF13: the weighted amount of task iterations This is calculated by multiplying each repetitive task length (i.e. the amount of actions) with its amount of iterations.

Clickstreams

Clickstreams are sequences of user clicks showing how the user traverses the application, often specifically from landing page to target page. The lengths and durations of these clickstreams were reported to be too long. A clickstream can be defined in

several ways. A simple example is to divide each session into subsessions with an expiry of 30 seconds, as navigational actions can be assumed to take less than 30 seconds and the last page would be the target page. Another example is to analyze clickstreams between pages that are connected based on association rules [26].

DF14: the average clickstream length

DF15: the average clickstream duration

DF16: the average amount of clickstreams per session

Miscellaneous

Some of these features can be considered profile data. They are listed nonetheless, because they can be added to the dataset with relatively minor preparation effort.

Encountered downtimes were reported to be quite an annoyance. As section 1.4.4 describes, two types of downtime exist: unplanned and planned downtimes. Taking into account the difference in frequency of both types, planned downtimes are the most relevant. These short periods of maintenance are always announced, which begs the question of causality: do the users experience hinder from the downtimes because they coincidentally try to use the system during those periods, or is this behavior fueled by the knowledge of the planned downtimes? Either way, encountered downtimes greatly impact user experience. From that point of view, it doesn't matter whether the system's unavailability was planned or not, so the unplanned moments should also be taken into account.

An inherent issue with analyzing encounters of unavailability is that the system does not generate log data. An approximation can be made by looking at the amount of log entries right before the relevant periods: if a user is using the system right before it becomes inaccessible, it is likely he experienced the transition and thus was hindered in his work. A first try for the 'right before' period is 10 minutes preceding the start of a downtime period.

DF17: the amount of downtime periods likely encountered based on activity 10 minutes preceding a downtime period

A convenient functionality of Somtoday is the (re-)authentication screensaver that is activated after 10 minutes of user inactivity. Seeing as the system is often used in class, and students are not always the most morally strong beings, this extra layer of security is certainly fitting. Security always comes with a trade-off in convenience. The screensaver is dismissed by entering user credentials, which can become annoying if this situation is encountered often. This is an example how **hindering login frequencies** might be influential. Hindering in this case means that the user experiences hinder to his workflow because of the required login action. A situation in which this is (possibly) not the case, is when a user manually logs out instead of being logged out due to a session timeout or screensaver. Because it is not possible to accurately know which login actions are hindering and which are not, feature DF18 attempts to approximate this by considering only *logout-decoupled* login actions: the amount of (manual) logout actions is subtracted from the amount of login actions.

DF18: the amount of average logout-decoupled login actions per week, only counting the weeks in which the system was used at all

The degree of competence in handling such a extensive system as Somtoday is likely to have an impact on the user experience. The hypothesis is that users having a low proficiency handling the system are less positive about it. More experience may indicate a higher proficiency, which is why feature DF19 looks at the time spent in the system. Another line of reasoning is the same as with prevalence of functionality use: one uses the things one likes more often. Also, the Mere Exposure effect is an argument for this feature. Features DF20 and DF21 are the result of breaking down feature DF19 into its separate interfaces.

DF19: the total time spent in the system This is normalized because the data analyzed in each data feature concerns only 90 days before each NPS response.

DF20: the total time spent in Somtoday

DF21: the total time spent in Somtoday Docent

Somtoday and Somtoday Docent are different interfaces for the same system. A risk of having two different interfaces is that users find it annoying that one has functionality the other doesn't. A practical example: one teacher uses the more basic interface of Somtoday Docent during class on his phone for homework planning. After school, he uses his laptop in combination with Somtoday for grade registration. This how the system is intended to be used. Another teacher does the same, but also often uses Somtoday during class for grade registration. He switches from Somtoday Docent to Somtoday, because grade registration is not possible in Somtoday Docent. The hypothesis is that this switch is annoying, irrespective of the reason (device switch, layout switch, finding functionality, etc.). This can be deduced from the data in a way similar to feature DF2, the difference being that only log entries are taken into account where the interface of the current log entry is different from the next, and it is measured in frequency instead of time.

DF22: the total amount of bidirectional switches between Somtoday and Somtoday Docent

The school where a user teaches might also be a predictor of the NPS response. There are several ways the school might be of impact. A school's policy is one: if all teachers are required to use iPads, they might dislike the interface of Somtoday and the limited functionality of Somtoday Docent. A school's geographic location might be another: if a school is surrounded by other schools that use another system, they might be inclined to like that system better because of word of mouth advertising. The school might also be using the system for a relatively short amount of time, giving the teachers less time to get accustomed with the system. Another example is the one mentioned in section 1.1 where teachers of one specific school were dissatisfied because of the few available functionalities (that their school management disabled). Either way, the school might be an indicator.

DF23: the school at which the user works

It is estimated that teachers educating different **class education levels** use the system differently. For example, teachers educating the higher levels[3] focus more on grades and graduation, whereas teachers educating the lower levels[4] focus more on

[3] HAVO/VWO in the Netherlands
[4] VMBO in the Netherlands

attendance and similar non-teaching activities. This data is readily available in the system.

DF24: the education levels a user teaches

To account for time-of-the-year differences, the time of the NPS response can be included. This has been formulated in this continuous value format:

DF25: the age of the NPS response measured from January 1st, 2016, in days

3.3 Selection

Researching all features listed in section 3.2 or determining feature relevance based on empirical evidence is out of the scope of this project. Features were selected if they have a high expected impact on the NPS response, require little time to prepare or are easily generalizable. The expected impact was assessed with the interviewed consultants mentioned in section 3.1.2 and one of Somtoday's product owners. The preparation time concerns the data preprocessing phase. A feature's generalizability implies how easily it can be used in similar systems (e.g. time spent on absentee registration is less generalizable than total time spent in the system).

With that in mind, the chosen features are as follows. They are put into categories for easy grouping in the rest of this thesis. The categories are put in chronological order of research efforts:

System usage

These were chosen first mainly because they are simple to measure. Additionally, DF19 is highly generalizable.

DF19: the total time spent in the system

DF20: the total time spent in Somtoday

DF21: the total time spent in Somtoday Docent

DF22: the total amount of bidirectional switches between Somtoday and Somtoday Docent

Repetitive tasks

This feature set is highly generalizable and was expected to have a high impact on the NPS response. People often do not like to reiterate the same task, especially if it is perceived to be inefficient (e.g. when a task requires too many clicks).

DF11: the amount of distinct repetitive tasks

DF12: the amount of iterations of repetitive tasks

DF13: the weighted amount of task iterations

Low-hanging fruit

These features were chosen because of their easy measurability.

DF23: the school at which the user works

DF24: the education levels a user teaches

DF25: the age of the NPS response measured from January 1st, 2016, in days

Chapter 4

Data preprocessing

This phase is said to take up a majority of the time spent on a data analysis project [29], as was indeed the case with the present study. The efforts put into the data preprocessing and its associated validation turned out to be essential. During validation, several bugs in the workflow and data infrastructure were uncovered. The steps taken, issues encountered and a basic analysis of the results are discussed in this chapter. Sections 4.1 and 4.2 cover the NPS and request log datasets, respectively, while sections 4.3 and 4.4 describe feature-specific efforts. Note that some efforts are similar and the preprocessed datasets could be combined to save storage space. However, seeing as storage space was no issue, a clear separation was preferred.

4.1 NPS

The desired dataset format is listed in table 4.1.

TABLE 4.1: Target format for the NPS dataset

Field	Data type	Description
ID	string	An identifier for the NPS response.
date	date	The date the NPS response was entered.
score	integer	The actual NPS response, being $0 \leqslant score \leqslant 10$.
user ID	string	An identifier for the user. This is used to join this dataset with the logs. This includes the username, organization and organization abbreviation, seeing as the non-API and API logs utilize distinct combinations of those.

The NPS dataset (discussed in section 1.4.1) was provided by Topicus in Microsoft Excel's XLSX format. The data integration suite Pentaho Kettle/Spoon[1] was used to apply some basic ETL operations: removal or renaming of columns, special character removal, string trimming and saving in the simple comma-separated value (CSV) format. The CSV file was then uploaded via Apache Ambari to the distributed storage, making it available to the Apache Zeppelin workspace.

Subsequently, Drill's SQL implementation was utilized to apply some more ETL operations and store the dataset in Drill's native Parquet format. The first priority was to only keep the NPS responses of teachers. This was done using two filters.

Prior process: *active account employee* Only the value *active account employee* was kept of the prior process field. Teachers are always employees, so students and

[1] http://community.pentaho.com/projects/data-integration/

parents are discarded. By disregarding non-'*active account*' values such as *service desk* and *training*, an attempt is made to discard NPS responses that were given in the context of a specific situation.

Roles: *teacher* and *mentor* The filter for the role of *teacher* is obvious. Nearly all mentors are also teachers, so this role was also taken into account. The few cases in which this is not the case (e.g. a dean might be a mentor, but not a teacher), are negligible according to domain experts at Topicus. Discarded roles included *application administrator* and *miscellaneous*.

Another filter is applied on the dates of the NPS responses. For the log dataset, the start date was chosen on which all schools in the country have started (seeing as this differs between regions): September 4th, 2016. This dictates the NPS date constraint of December 3rd, 2016, because 90 days of log data is analyzed for each NPS response. The maximum date is set on June 30th, 2017, as this is the end of the academic year. The system will be utilized after that date, but to normalize the data as much as possible this has not been taken into account.

The next step was making the join with the request log dataset possible. Individual log entries include a user's username and organization, while the NPS dataset includes user identification strings (*UUID*s) generated by Somtoday. Topicus has provided an XLSX file containing a mapping of UUIDs to usernames and organisations. This dataset was made available to Drill using the same workflow as used with the NPS dataset. Using this mapping, each NPS response was joined with the associated username and organization, enabling future joins with the log dataset. Each NPS response also received a unique ID.

There were still a few inconsistencies found during validation in the record counts between these datasets. In some cases, different capitalization was used in organization names or usernames. These issues were resolved by lowercasing all of these values after confirming with Topicus that the capitalization has nothing to do with user or organization uniqueness.

A different issue was encountered that is a result of merging schools. This resulted in some users having multiple organizations. Due to the nature of upcoming joins with the log data, it was possible to simply include multiple NPS response records per user and keep the NPS response ID the same, thus the only difference being the organization. When applying a left join between the NPS and log datasets later on, the correct log entries are associated with the correct NPS response. The same issue was identified and resolved in the context of usernames.

Finally, it also became apparent that some users had submitted a response to the NPS survey twice. Of the 1085 NPS responses, 12 (1.1%) were from users with multiple responses: 6 users had two responses. Statistics of this subset are shown in table 4.2. The time between responses and the variance in score is motivation to treat these NPS responses as distinct data points. Note that there was one anomalous instance in which a user had just 47 days between two responses. The first was discarded: the user has more experience with the system at the time of his second response, making a well-founded response more likely. To distinguish between responses from the same user, different IDs were assigned consisting of the response date concatenated with the first eight characters of the user's UUID. Those eight characters have been checked for uniqueness.

The response and temporal distributions are shown in figures 4.1 and 4.2 and are the exact same as figures 1.1 and 1.2 (respectively) from section 1.4.1. They are repeated here for convenience.

TABLE 4.2: Descriptive statistics of users with multiple NPS responses

	Time between subsequent responses		NPS score increase
Average	133.5 days	≈ 4.4 months	0.17
Sample standard deviation	45.1 days	≈ 1.5 months	0.98
Minimum	84.0 days	≈ 2.8 months	-1
Maximum	192.0 days	≈ 6.3 months	1

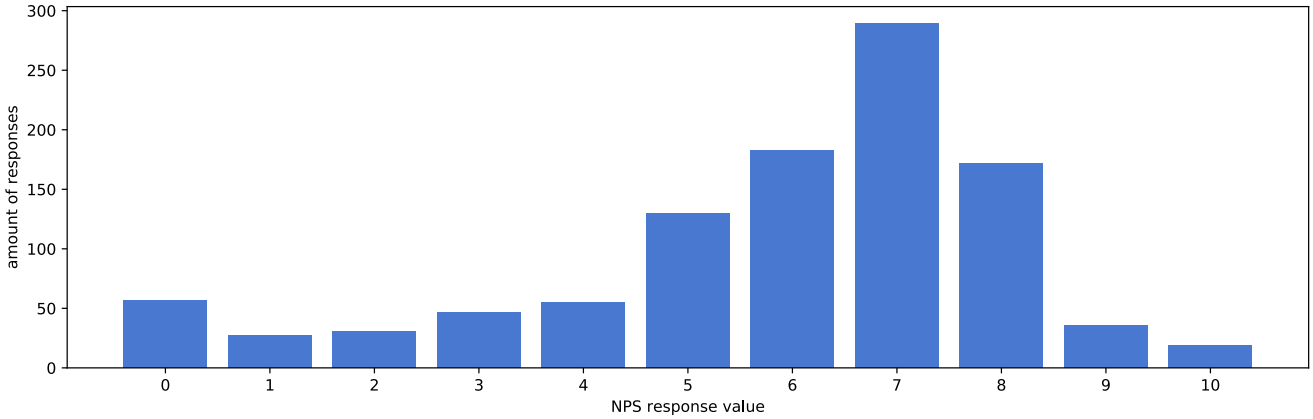


FIGURE 4.1: Response distribution of teacher NPS responses

After preprocessing the usage logs it was discovered that 39 NPS respondents had no usage logs that could be associated with them. These NPS respondents were discarded, as we could not reliably determine whether the absence of usage logs could be considered part of a usage pattern. To do that, research into (the data of) all 39 respondents would show whether they explicitly made a choice not to use Somtoday and Somtoday Docent or whether they switched accounts, only just started to use the system, or had some other reason for not generating log entries. After discarding these samples, the dataset consisted of 1046 NPS respondents.

4.2 Usage logs

The desired dataset format is listed in table 4.3. The non-API and API logs are treated separately due to the different level of detail included in their logs.

As explained in section 1.4.2, the request logs are accessible to Apache Drill via the Zeppelin web interface. Using SQL queries, a new dataset was created that only includes the relevant log entries and columns. To make efficient data analysis possible, several main filters had to be applied:

1. Only the logs relating to **NPS responses** are relevant. This filter was applied based on the user and organization associated with the log entry, and the users in the NPS dataset.

2. A **time constraint** is applied based on the date of the NPS response. This is an attempt to normalize the amount of log entries per NPS response, to keep the

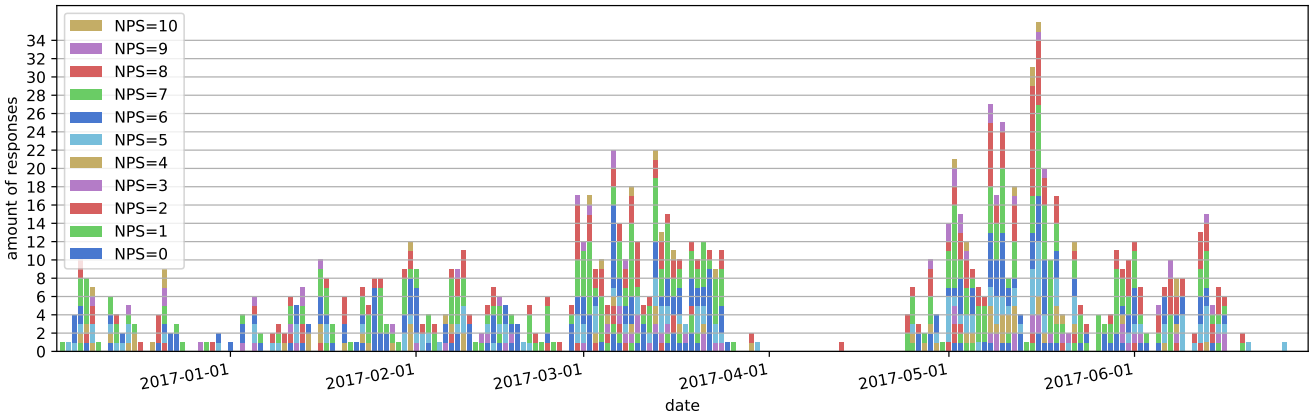


FIGURE 4.2: Temporal distribution of teacher NPS responses. Note that the different response values are shown solely to indicate that there is variation.

TABLE 4.3: Target format for the logs dataset.

Field	Data type	Description
NPS ID	string	A reference to the NPS response that the log entry is associated with.
timestamp	timestamp	The timestamp that Wicket received the request, accurate to the millisecond.
request details	strings and integers	Various details about the page requested and the page from which the request was made. This includes the response*, event*, *component* and behaviorClass fields for the non-API logs, the restResource, range, category and method fields for the API logs and the requestedUrl and duration fields for both log types. Specific field descriptions are available in table 1.2.

logs actually relevant to the NPS response (e.g. logs from two years prior an NPS response are expected to have next to no influence on the given NPS score) and to account for structural and functional improvements to Somtoday that accumulate over time. For this reason, a time period of 90 days prior to the day of the NPS response was chosen.

3. Duplicates should be removed. The process of data aggregation described in section 1.4.6 is not perfect. This has led to several instances in which data was aggregated multiple times, resulting in duplicate log entries.

The first issue encountered pertains to Drill's 'schema-free' capabilities. To apply filtering of any kind, conditional SQL statements must be possible. However, due to variation in the data formats and values, Drill often infers them inaccurately. Also, it does this for each value separately. This is useful in many situations, but in this case leads to numerous errors when applying comparisons and join operations. Drill might infer a numeric data type on a value that should be treated as a string, resulting in an error when a string data type and a 'numeric' data type are compared. This can be solved by explicitly casting each column and each value. In practice, this

was applied by creating a view that returns casted values for each column, thereby effectively enforcing a schema and abandoning Drill's schema-free advantages.

The next step was applying the first two filters mentioned above. In theory, this is fairly straight-forward: take the NPS dataset, use a left join with the logs on the user ID and apply a filter based on the NPS date. This method was invalidated by comparing the records counts of result sets generated by equivalent queries. By applying a step-wise approach, a solution was found. Each filter was implemented in different ways: using joins versus using conditionals with SQL's IN operator (filter 1), taking all original data as input at once versus taking subsets of the data based on their timestamp (filter 2), using SQL's SELECT DISTINCT operator versus grouping on all relevant fields (filter 3). In addition these filters were tried in different orders. Finally, the effectiveness of using a view-wrapper was validated by also trying the same without a view (but with casting). Both choices for filter 1 performed well, except that using a join has the advantage that the NPS ID can be included immediately instead of in a later stage. Applying filter 2 resulted in the conclusion that Drill somehow did not actually include all data when querying the complete dataset (and it was also slower, seeing as the complete dataset also includes irrelevant logs from 2015). The solution was to query the years 2016 and 2017 separately. The options for filter 3 had the same result and both gave rise to memory issues. This was resolved by applying filter 3 on the intermediate result set of filters 1 and 2. Eventually, this process of preprocessing consisted of eight queries. One extraction query that applied filters 1 and 2 (essentially extracting the relevant log entries from the original dataset) and one filter query that applied filter 3 on the result of the previous filter. These two queries are executed on each relevant year (2016 and 2017) and on both log types, which makes the sum of eight. The only non-semantical query difference between the log types is that some API logs have duplicates with different duration values. This was handled by taking only the log entry with the largest duration into account in the second query.

In a later stage, it became apparent that the desired dataset has another requirement: requests that are not an action of a user are irrelevant. Each of the following filters has been validated by selecting all matching requests and incrementally checking if they indeed are irrelevant. These have also been checked by data experts at Topicus.

Resources Resources are loaded together with user requests and are thus irrelevant for the analysis. All requests that have the value 'org.apache.wicket.request.handler.resource.ResourceReferenceRequestHandler' for the eventTargetClass field are requests for one of the following file types: jpg, png, gif, svg, js, css, pdf, csv, cls, doc, docx, txt, xml, ttf or woff. Additionally, requests with requestedUrls that contain the value 'nl.topicus.iridium.web.resource.ImageResourceReference' were removed. Image requests were also discarded by matching on values 'pagetitle:image', 'pagetitle:iconContainer:image' or 'title:iconContainer:image' values in the componentPath field.

Screensaver As mentioned before in section 3.2, Somtoday has a security measure implemented to protect against malicious students. After a short period of inactivity, a screensaver pops up, forcing a teacher to input his password. A request is made each time the screensaver is enabled or dismissed. These have the value 'screensaver' in the componentPath field.

Timed updates Some requests are caused by a client-side timer that means to update a component on the page, e.g. the calendar requesting any changes to the current view. These log entries have the value 'org.apache.wicket.ajax.AjaxSelfUpdatingTimerBehavior' for the behaviorClass field.

Semantically useless pages Non-API log entries that have a `NULL` value for their response_pageClass, event_pageClass and componentPath fields are useless for this project. Two examples of practical occurrences are with resource requests and when redirecting. In the latter, Wicket adds two log entries: one in which solely the URL contains all information and one in which the rest of the fields hold all the information. The former is discarded by this filter.

Lazily loaded content These requests always follow regular page request within mere milliseconds. They have the values 'org.apache.wicket.extensions.ajax.markup.html.AjaxLazyLoadPanel$1' or 'nl.topicus.iridium.web.components.panel.AjaxLazyLoadPanel$1' for the behaviorClass field and values 'lazyContentPanel', 'dossiers' or 'dataview' for the componentPath field.

The final and most unexpected issue encountered seems to be a bug in Drill: some queries returned results non-deterministically. There was a pattern to it, but instead of diving into the inner workings of Drill, the whole data preprocessing process was broken down into its most basic parts. Topicus' data storage workflow stores each batch of data in a separate directory. Drill's flexibility allows for easy aggregation of those directories, e.g. by using `'2016*'` to select every directory starting with `'2016'`. The solution was to *not* use that flexibility, query each separate directory in each step and only do one specific thing per step. Each step uses the dataset of the previous step. This temporarily consumes 482.6 Gigabytes of storage space, but ensures a validated and working process. The resulting process is:

Step CA This step copies all original log entries to a new directory, casting the relevant fields and discarding the irrelevant ones. Note that this is only applied for the relevant months, as it would be useless to make a casted copy of the complete historical dataset.

Step UN This step keeps the log entries that are relevant to the NPS dataset, based on username and organisation.

Step NP This step adds the NPS ID to each log entry.

Step TF This step applies the time frame filter, discarding log entries that are not within the 90 day period prior the relevant NPS response.

Step FT This step applies the action filter as described above. This step does not apply to API log entries, seeing as they do not need action filters.

Step DU This step removes duplicate entries using a SELECT DISTINCT statement.

Step FN This step simply copies all entries from the working directory to a final directory.

Each step is validated as described above. This process is applied to both non-API and API logs. The total amount of queries, taking into account the directories, steps and log types, is 5025. These took about 3.5 to 4 hours to complete and another 10

minutes to validate.

The time necessary for Drill to execute the preprocessing queries was optimized by discarding several columns. Which ones would be discarded was determined by looking at their usefulness for later analysis. For example, all fields relating to the user's device and browser were discarded due to their irrelevance in this study. The impact of this optimization was substantial because of the column-based nature of Drill's Parquet file storage format. In one instance during query construction, omitting one column resulted in an execution duration dropping from about 45 minutes to about 15 minutes.

Figure 4.3 shows the histogram of the relevant log entries per NPS response. The average is 1408.2 log entries with a standard deviation of 3296.9. Figure 4.4 shows the histogram of the relevant log entries per date. The weekends and holidays are clearly visible. Additionally, the tails at both boundaries are expected due to the overlap of relevant NPS response log periods.

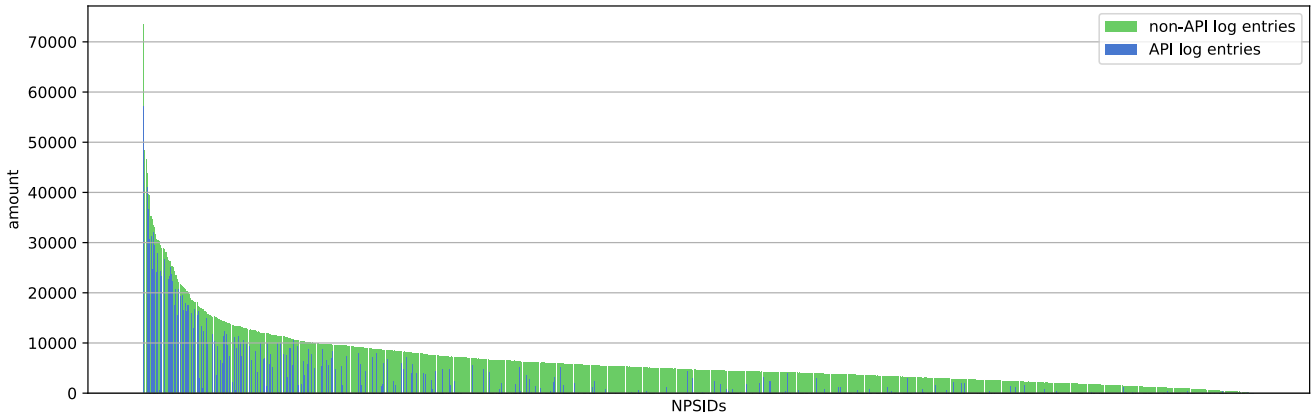


FIGURE 4.3: Response distribution of relevant log entries per NPS response

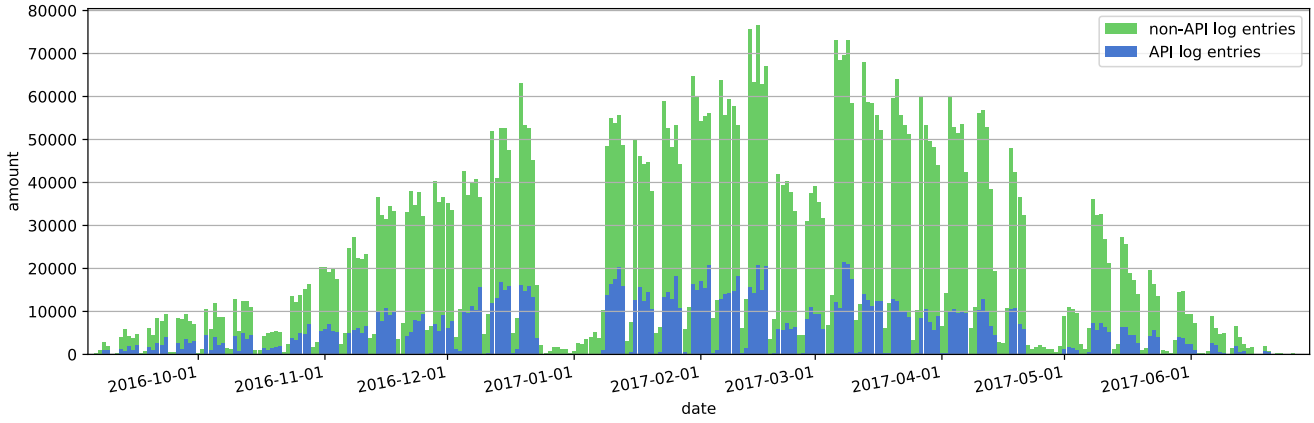


FIGURE 4.4: Temporal distribution of relevant log entries

A few of the common issues in the field of web usage mining are user identification, caching, and grouping requests into distinct sessions [13, 14, 58]. User identification is no issue in Somtoday and Somtoday Docent, seeing as user authentication is required to use the system. Wicket has built-in functionality to exert control over

client-side caching. It allows this only for resources such as images. Seeing as we ignore such resources, caching is no issue. Wicket also handles sessions. However, seeing as it does this in the context of security, it does not conform to what is required for our purposes. A quick analysis of sessions specified by Wicket shows that some sessions lasted for hours, which does not mean the user actually was active for hours. For example, the session length is increased significantly due to use of the screen-saver. To mitigate this, a custom session grouping was implemented. An expiration time commonly used in commercial applications is 30 minutes [14]. A time of 60 minutes was chosen for this project so that system activity at the start and end of a class is registered into the same session. Assigning session IDs is done in three stages. First, for log entries of both log types, the NPS ID and timestamp is extracted. Next, each log entry is assigned information about the timestamp of the previous and next timestamps having the same NPS ID, but only if the time difference between the current and previous or next timestamps is smaller than the session expiration time. In the final stage, each log entry without a previous timestamp is considered the start of a new session. A session starting request and all next requests (grouped per NPS ID) up until but excluding the next session starting request, receive the same session ID. The session ID consists of the NPS ID and an integer that equals the amount of preceding requests without a previous timestamp. The result is a table with the columns described in table 4.4 and that contains only the bare essentials necessary to identify user activity.

TABLE 4.4: Format for the sessions dataset

Field	Data type	Description
NPS ID	string	A reference to the associated NPS response.
timestamp	timestamp	The timestamp of the request.
previous timestamp	timestamp | `NULL`	The timestamp of the previous request, but only if $(current - previous) < session\,expiration$.
next timestamp	timestamp | `NULL`	The timestamp of the next request, but only if $(next - current) < session\,expiration$.
session ID	string	The generated session ID.

4.3 Feature-specific: system usage

To be able to extract data features DF19: the total time spent in the system, DF20: the total time spent in Somtoday, DF21: the total time spent in Somtoday Docent and DF22: the total amount of bidirectional switches between Somtoday and Somtoday Docent, first the data must be put into the format shown in table 4.5.

The first step was to combine the requests of the non-API and API datasets into one. Each entry received a value indicating its logtype. Only the NPS ID, timestamp and logtype fields were necessary. The next step made sure that each log entry received the timestamp and logtype of the subsequent log entry of the same NPS ID. The pattern extraction step aggregates this dataset and is discussed in section 5.1.

The reason this dataset only includes timestamps and logtypes is that these data features exclusively concern system use itself and nothing more.

TABLE 4.5: Target format for the system usage preprocessed dataset.

Field	Data type	Description
NPS ID	string	A reference to the associated NPS response.
timestamp	timestamp	The timestamp of the request, accurate to the millisecond.
logtype	enumeration	A value indicating the log type.
nextTimestamp	timestamp	The timestamp of the next request corresponding to the same NPS response.
nextLogtype	enumeration	A value indicating the log type of the next request corresponding to the same NPS response.

4.4 Feature-specific: repetitive tasks

To extract patterns of repetitive tasks for DF11: the amount of distinct repetitive tasks, DF12: the amount of iterations of repetitive tasks and DF13: the weighted amount of task iterations, the data has to be processed into the desired dataset format listed in table 4.6. The dataset contains both logtypes.

TABLE 4.6: Target format for the repetitive tasks preprocessed dataset

Field	Data type	Description
NPS ID	string	A reference to the associated NPS response.
timestamp	timestamp	The timestamp of the request, accurate to the millisecond.
session ID	string	The generated session ID.
action	string	A string representing the intended action of the request.

The session ID is introduced by including the sessions dataset described in section 4.2. The action string is a concatenation of a few fields. These are described in tables 4.7 and 4.8, respectively for the non-API and API logtypes. Note that table 4.7 does not include fields containing the target page. This is because a click on the same (type of) button is considered the same action and multiple clicks on similar buttons (e.g. going to the profile of student X and going to the profile of student Y) are the same task. Equivalently, the API action string does not include parameter values.

The resulting dataset (with the format described in table 4.6) is now a chronologically ordered list of tasks that can be grouped on session and corresponding NPS response. Analysis of repetitive tasks is the next step.

4.5 Feature-specific: low-hanging fruit

Data features DF23: the school at which the user works, DF24: the education levels a user teaches and DF25: the age of the NPS response measured from January 1st, 2016, in days consist mainly of profile data. Topicus supplied XLSX files with mappings from UUID to corresponding schools and UUID to education levels taught,

TABLE 4.7: Concatenated fields for the action string of the non-API logtype.

Field	Description	Transformations
event_pageClass	The Java (page) class that triggered the request.	Lowercased.
componentClass	The component that triggered the request, e.g. a button	Lowercased.
componentPath	The path to the component that triggered the request.	Lowercased and removed any IDs using regular expression ':[0-9]+'.

TABLE 4.8: Concatenated fields for the action string of the API logtype.

Field	Description	Transformations
restResource	The REST resource, with no parameters set.	Lowercased.
method	The method with which the request was sent, e.g. GET, POST, UPDATE.	Lowercased.

for features DF23 and DF24, respectively. These were preprocessed by Pentaho Kettle/Spoon, again only for simple ETL operations such as string trimming, field renaming and removal and saving into CSV format. Feature DF25 did not require any preprocessing as its data can be extracted directly from the NPS dataset.

Chapter 5

Pattern extraction

This chapter describes how values for each feature are extracted from the preprocessed dataset. These values are input for a machine learning algorithm in the next phase, which will be discussed in chapter 6. Pattern extraction entails only basic ETL operations for some features, as sections 5.1 and 5.3 describe for the system usage and low-hanging fruit features respectively, but uses more complicated algorithms for others, as section 5.2 describes for the repetitive tasks features.

5.1 System usage

Seeing as there is no data available about the time users spend on each page, this is measured indirectly by looking at the time the user spends browsing from page to page. This means a simple summation of intervals between a user's requests is sufficient. Recall from section 4.3 that each log entry in the preprocessed dataset includes the current and the next timestamp, as well as the current and the next type of log entry. A too large interval between requests indicates user inactivity. The example of a user reading or writing a lengthy report for several minutes is a reason to make the interval not too small. For that reason, a maximum of 10 minutes was chosen between subsequent requests. If two requests are more than 10 minutes apart, it is assumed that the user landed on its target page. This means it is essentially a partitioning into sessions with an expiration time of 10 minutes. These are the conditions considered per feature, for each log entry:

DF19: the total time spent in the system If the next timestamp minus the current timestamp is smaller than 10 minutes, add that interval to the user's total system usage.

DF20: the total time spent in Somtoday If the next timestamp minus the current timestamp is smaller than 10 minutes and the current entry's log type is non-API, add that interval to the user's total Somtoday usage.

DF21: the total time spent in Somtoday Docent If the next timestamp minus the current timestamp is smaller than 10 minutes and the current entry's log type is API, add that interval to the user's total Somtoday Docent usage.

DF22: the total amount of bidirectional switches between Somtoday and Somtoday Docent If the next timestamp minus the current timestamp is smaller than 10 minutes and the current entry's log type is different than the next entry's log type, increment the user's total interface switch count by one.

Note that, as mentioned in section 3.2, a disadvantage of this indirect measurement method is that the last visited page of a user's session is not taken into account for the user's system usage time, i.e. features DF19 to DF21.

5.2 Repetitive tasks

In a collection of sequences, sequential pattern[1] mining discovers patterns of elements that are often found in the same subsequent order. Scientific literature mentions three requirements that a pattern must meet to be eligible for further analysis: the minimum pattern length (*minpatternlength*) that specifies the minimum amount of elements in the pattern; the minimum support (*minsupport*) that says in at least how many of the sequences the pattern must occur and which relates directly to our *minimum inter-session occurrence* requirement mentioned in section 3.2; and the minimum repetition support (*minrepsupport*) that dictates the minimum amount of occurrences of the pattern in each sequence and which relates directly to our *minimum intra-session occurrence* requirement. The multitude of variations on each sequential pattern mining algorithm type adds and/or removes several pattern requirements. These ones mentioned here are the most basic and interesting, especially because they map directly onto our requirements for features DF11 to DF13. To clarify: the patterns that are found using these algorithms are the repetitive tasks that we are looking for.

In our initial literature study, no algorithm was identified that satisfied our requirements for features DF11 to DF13. A new algorithm was constructed to find the repetitive tasks we are looking for. The steps of our algorithm, Apriorirep1i, are based on the frequent items analysis algorithm Apriori and its sequential pattern mining extension AprioriAll [1]. This was chosen to support intuitive comprehension of the algorithm, especially in the maximal phase (explained later on). In hindsight, Apriorirep1i turned out to practically take a PrefixSpan approach, but breadth-first (finding all patterns of the same length) instead of depth-first (finding patterns of increasing length with the same prefix). Nearing the end of this project we found that important work was overlooked in our initial search that would have altered our approach to Apriorirep1i. Related work, including the overlooked work, is discussed in section 5.2.4. It should be noted that the implementation of a different algorithm (along with the validation of its correct execution) would have required a slightly less but similar amount of time than was spent on Apriorirep1i.

Semantic differences between the algorithms found in literature and the one required for repetitive system actions are shown in table 5.1. Especially notice the semantics of the itemset concept, because this is the most influential and practical difference. The difference in sequence database semantics means that the algorithm in our case is applied on each user instead of the complete dataset. Also note that an action is the same as a request.

The basic algorithm is discussed in section 5.2.1, an extension that makes it complete in section 5.2.2 and its application on the preprocessed dataset in section 5.2.3. The similarities and differences with regard to previous work are discussed in section 5.2.4.

5.2.1 Apriorirep1i (basic)

This algorithm was made while keeping in mind that results had priority over optimal performance. A few measurements are given in section 5.2.3 to indicate that this was not an issue. The Python 2.7 implementation that was used for this study can be

[1] In this section a pattern is a sequential pattern: an ordered series of items. In the rest of this thesis a pattern is a user's usage pattern.

TABLE 5.1: Semantic differences between use cases of the algorithms found in literature and our algorithm

concept	web shop records	web system usage logs
sequence database	complete history of bought products	one user
sequence	ordered history of bought products of one user	user session
itemset / element	set of products bought together	action[1]
item	bought product	action

[1] Due to the nature of web system usage logs, a user can not perform two actions at the same time. Consequently, itemsets are always of length 1.

found in appendix A and online[2].

The support is in literature defined as a relative value. In the case of the pattern support, for instance, it is the fraction of total sequences that support that pattern. However, in section 3.2 we defined the inter- and intra-session occurrences as absolute values. Since there are no practical advantages or disadvantages to using one or the other in our situation and since existing algorithms internally use the absolute value anyway, from now on the absolute value is meant when talking about the support.

Literature defines large itemsets or *litemsets* as itemsets that have the minimum support. Seeing as Apriorirep1i works with itemsets of length one, *litems* or large items are the equivalent of large itemsets of length one.

The algorithm uses the data structure *occurrence set* internally and as its output. This is a map of patterns mapped to their occurrences. The occurrences are maps, where the zero-based sequence indices of the sequence database are mapped to a set of zero-based indices pointing to the starting occurrence of the pattern within the sequence. To illustrate, the following is an occurrence set of two patterns occurring in the example sequence database of table 5.2: $(\text{"}abcd\text{"} \rightarrow (4 \rightarrow \{0, 4, 8\}, 5 \rightarrow \{0, 5, 9\}), \text{"}de\text{"} \rightarrow (5 \rightarrow \{3, 12\}))$. An advantage of this structure is that information about a pattern's occurrences is retained, while metrics can be obtained in a computationally cheap way by counting the elements in the lists of the occurrence set.

As a running example, we take the sequence database shown in table 5.2. The example includes a fair amount of self-succeeding cyclical occurrences. This was chosen to show that even though this algorithm finds both cyclical and non-cyclical repetitive patterns, it is not perfect. Note that this algorithm is independent of the lexical value of the element symbols: replacing element a with z yields the same results.

The patterns in the example that can be identified are $abcd$, abc, bca and cab. As one might notice, abc, bca and cab are the cyclically shifted versions of each other resulting from their self-succeeding occurrences. Should all of these patterns be identified as frequent patterns, or only one of them? This self-succeeding pattern origin ambiguity is addressed in section 5.2.2.

The input expected by Apriorirep1i is a sequence database and the parameters minsupport, minrepsupport and minpatternlen, having constraints minsupport>0,

[2] https://karim.elass.al/masterthesis/apriorirep1i

TABLE 5.2: The sequence database used as a running example to illustrate the workings of Apriorirep1i and has parameters minsupport=2, minrepsupport=2 and minpatternlength=2

seq. index	sequence
0	a b c a c
1	a c t y
2	a b c a b c a b
3	b c a b c a b c a b c
4	a b c d a b c d a b c d
5	a b c d e a b c d a b c d e
6	x y z

minrepsupport>1 and minpatternlen>1. Before the different phases are discussed, one step needs to be explained that is applied several times throughout the algorithm: the `purge_nonlpatterns` step. The term *lpatterns* (or *large patterns*) follows the equivalent definition of *litems*, but applied to repetitive patterns. The occurrence set of an lpattern adheres to the minrepsupport and minsupport requirements. This step works in this order:

1. Only occurrences are retained of sequences that adhere to the `minrepsupport` requirement, so this is evaluated per sequence.
2. Only patterns are retained that adhere to the `minsupport` requirement, so this is evaluated per pattern.

The algorithm overview is shown in listing 5.1. The differences with AprioriAll are discussed next, per phase. Note that the `purge_cyclic_shifts` and `purge_interconnecting_subpatterns` steps are discussed in section 5.2.2.

LISTING 5.1: Apriorirep1i algorithm

```
# IN: sequence database -> sequence_db
#                  int -> minsupport
#                  int -> minrepsupport
#                  int -> minpatternlen
# OUT: occurrence set

# litem phase
L_1 = get_litems(sequence_db, minsupport, minrepsupport)

# transformation phase
D_T = truncate_nonlitem_sequences(sequence_db, L_1)

for (k = 2; L_{k-1} ≠ ∅; k++):
    # sequence phase
    L_k = get_next_patterns(D_T, L_{k-1})
    L_k = purge_nonlpatterns(L_k, minsupport, minrepsupport)

    L_k = purge_subpattern_cycle_start(L_k, L_{k-1}, D_T)
    L_k = purge_cyclic_shifts(L_k, D_T)
    L_k = purge_nonlpatterns(L_k, minsupport, minrepsupport)

```

```
    # maximal phase
    if k - 1 < minpatternlen:
        delete L_{k-1}
    else:
        L_{k-1} = purge_interconnecting_subpatterns(L_{k-1}, L_k)

        L_{k-1} = purge_subpatterns(L_{k-1}, L_k)

L = purge_nonlpatterns(L, minsupport, minrepsupport)
return L
```

AprioriAll starts with a **sort phase** in which the original database with customer transactions is transformed into a sequence database. To increase generality, Apriorirep1i expects a sequence database and thus skips the sort phase.

AprioriAll's **litemset phase** focuses on finding all litemsets. Seeing as we work with items instead of itemsets, Apriorirep1i applies a litem phase. The sequence database is traversed, storing each item encounter. The `purge_nonlpatterns` step is applied and the result is the litem occurrence set L_1. In the case of our running example, L_1 is shown in table 5.3. An future optimization to reduce memory space is to apply occurrence purging based on the minrepsupport requirement during the litem phase.

TABLE 5.3: The result L_1 after Apriorirep1i's litem phase of our running example

seq.	occurrences of ***a***	occurrences of ***b***	occurrences of ***c***	occurrences of ***d***
s_0	$\{0, 3\}$		$\{2, 4\}$	
s_2	$\{0, 3, 6\}$	$\{1, 4, 7\}$	$\{2, 5\}$	
s_3	$\{2, 5, 8\}$	$\{0, 3, 6, 9\}$	$\{1, 4, 7, 10\}$	
s_4	$\{0, 4, 8\}$	$\{1, 5, 9\}$	$\{2, 6, 10\}$	$\{3, 7, 11\}$
s_5	$\{0, 5, 9\}$	$\{1, 6, 10\}$	$\{2, 7, 11\}$	$\{3, 8, 12\}$

The next phase for AprioriAll is the **transformation phase** in which all non-litems are dropped from the sequence database to speed up the subsequent sequence phase. This is not viable for Apriorirep1i because of its use of occurrence sets: if litems are omitted, the indices refering to occurrences don't match anymore. An approximation of this effort is made by truncating the sequences that do not contain litems. This way the sequence keys (i in s_i) in the occurrence set still refer to the correct sequences in the sequence database and those irrelevant sequences are removed from memory. The result of applying Apriorirep1i's transformation phase is that s_6 is truncated. The resulting sequence database is from here on out denoted as $\mathcal{D}_\mathcal{T}$.

The focus now shifts to finding the desired patterns. This is called the **sequence phase**. The algorithm enters into a loop, incrementing the pattern length k at each iteration and continuing while there are lpatterns found of length $k - 1$. Explaining the steps of this phase is easier when we skip two k. The occurrence set L_3 is shown in table 5.4. First all candidates of length $k = 4$ must be found. This step traverses all lpatterns in L_{k-1} and returns an occurrence set of patterns of length $k = 4$ found in $\mathcal{D}_\mathcal{T}$. This is done by looking at each pattern's occurrence and considering the element that directly follows the pattern. The candidates found in our running example are *abca*, *abcd*, *bcab*, *bcda*, *bcdb* and *cabc*. Again, the `purge_nonlpatterns` step is

TABLE 5.4: The intermediate result L_3 after two iterations of Apriorirep1i with our running example

seq.	occurrences of ***abc***	occurrences of ***bca***	occurrences of ***cab***	occurrences of ***bcd***
s_2	$\{0, 3\}$	$\{1, 4\}$	$\{2, 5\}$	
s_3	$\{2, 5, 8\}$	$\{0, 3, 6\}$	$\{1, 4, 7\}$	
s_4	$\{0, 4, 8\}$			$\{1, 5, 9\}$
s_5	$\{0, 5, 9\}$			$\{1, 6, 10\}$

applied to discard irrelevant occurrences. The result is shown in table 5.5. This does not include all candidates, since not all of them are lpatterns.

TABLE 5.5: The intermediate result L_4 of Apriorirep1i with our running example after the `get_next_patterns` step of sequence phase, with `purge_nonlpatterns` applied

seq.	occurrences of ***abca***	occurrences of ***abcd***	occurrences of ***bcab***
s_2	$\{0, 3\}$		$\{1, 4\}$
s_3	$\{2, 5\}$		$\{0, 3, 6\}$
s_4		$\{0, 4, 8\}$	
s_5		$\{0, 5, 9\}$	

As one might notice, *abca* is a superpattern of *abc*. Step `purge_subpattern_cycle_start` removes occurrences of *abca* if it links two occurrences of the subpattern *abc*. Index 0 in s_2 qualifies for the purge, but index 3 does not: subpattern *abc* has no occurrence at index 6 as the sequence ends after index 7. The result is shown in table 5.6. Only the occurrences of *abcd* remain after `purge_nonlpatterns` is applied again.

TABLE 5.6: The intermediate result L_4 of Apriorirep1i with our running example after the `purge_subpattern_cycle_start` step of sequence phase

seq.	occurrences of ***abca***	occurrences of ***abcd***	occurrences of ***bcab***
s_2	$\{3\}$		$\{4\}$
s_3	$\{\}$		$\{6\}$
s_4		$\{0, 4, 8\}$	
s_5		$\{0, 5, 9\}$	

Apriorirep1i places the next phase, the **maximal phase**, partially inside and partially outside of the loop, contrary to only outside as AprioriAll does. Since this phase focuses on patterns of length $k - 1$, this design choice allows for better comprehensibility. The maximal phase removes occurrences of patterns that are not maximal, i.e. that are found in other patterns. First it discards patterns of length $k - 1$ if they are smaller than the minimum pattern length.

If $k-1 \geq minpatlen$, it applies the `purge_subpatterns` step. Table 5.7 shows the input for this step limited to s_4 and s_5. It shows clearly that all occurrences of abc and bcd are part of the occurrences of $abcd$. These are found by looking at the two subpatterns obtained by removing the head (resulting in bcd)and tail (resulting in abc) elements, respectively, and comparing their occurrences with those of the superpattern. In this case, all occurrences of abc and bcd are discarded.

TABLE 5.7: The running example input for the maximal phase at $k = 4$ of Apriorirep1i reduced to s_4 and s_5

	L_4	L_3	
seq.	occurrences of $\boldsymbol{abcd}$	occurrences of $\boldsymbol{abc}$	occurrences of $\boldsymbol{bcd}$
s_4	$\{0, 4, 8\}$	$\{0, 4, 8\}$	$\{1, 5, 9\}$
s_5	$\{0, 5, 9\}$	$\{0, 5, 9\}$	$\{1, 6, 10\}$

Finally, the maximal phase concludes with applying `purge_nonlpatterns` again. Placing this inside the loop applies the step more often on smaller datasets, while placing it outside the loop applies it one time on the result. We've chosen the same as with AprioriAll: outside the loop.

The result of running the basic version of Apriorirep1i on our running example is shown in table 5.8. This shows that the occurrences of the individual items of abc, bca and cab have overlap and is discussed in section 5.2.2.

TABLE 5.8: The result of basic Apriorirep1i applied on the running example

seq.	occurrences of $\boldsymbol{abcd}$	occurrences of $\boldsymbol{abc}$	occurrences of $\boldsymbol{bca}$	occurrences of $\boldsymbol{cab}$
s_2		$\{0, 3\}$	$\{1, 4\}$	$\{2, 5\}$
s_3		$\{2, 5, 8\}$	$\{0, 3, 6\}$	$\{1, 4, 7\}$
s_4	$\{0, 4, 8\}$			
s_5	$\{0, 5, 9\}$			

5.2.2 Apriorirep1i (complete)

In the sequence <$abcabcabc$> it is clear that pattern abc is the main sequential pattern. Sequence <$bcabcabca$> clearly has the main sequential pattern bca. In a sequence database we should take them both into account, seeing as they both contain the other's main sequential pattern. This is what we've called **self-succeeding pattern origin ambiguity**: which pattern should be considered the main pattern? An imperfect solution was found using a sliding window approach.

This is mainly dealt with by the `purge_cyclic_shifts` step. We take the iteration $k = 3$ to illustrate. Traversing s_2, it starts the sliding window at index 0 and checks in L_k if there's a frequent pattern with an occurrence there. It finds abc and remembers it as the main pattern. Incrementing the index to 1, it checks if the cyclically shifted version bca is a frequent pattern occurring there. This is the case, so it purges that occurrence of bca. When the occurrence of the main pattern is traversed (after

index 2), it forgets the main pattern and starts looking for a new one. Continuing with s_3 the first main pattern found is bca and does the same. Table 5.9 shows the result of this step. It shows that it indeed discarded abc from s_3, bca from s_2 and cab from both s_2 and s_3.

TABLE 5.9: The intermediate result L_3 of Apriorep1i after `purge_cyclic_shifts` step of sequence phase with our running example

seq.	occurrences of $\boldsymbol{abc}$	occurrences of $\boldsymbol{bca}$	occurrences of $\boldsymbol{bcd}$	occurrences of $\boldsymbol{cab}$
s_2	$\{0, 3\}$	$\{\}$		$\{\}$
s_3	$\{\}$	$\{0, 3, 6\}$		$\{\}$
s_4	$\{0, 4, 8\}$		$\{1, 5, 9\}$	
s_5	$\{0, 5, 9\}$		$\{1, 6, 10\}$	

There is one additional step required as a consequence of the sliding window approach. Pattern ca is also recognized as frequent due to the fact that it connects self-succeeding occurrences of abc. In s_2, this results in occurrences at indices 2 and 5. The step `purge_interconnecting_subpatterns` identifies such cases by comparing occurrences of frequent subpatterns with the tail-less cyclically shifted versions of each pattern, e.g. subpatterns ab, bc and ca of abc. In our example, it retains index 5 of s_2 because no occurrence of abc is found at corresponding index 6.

Note that the result (shown in table 5.10) of running Apriorirep1i on our example ultimately does not include occurrences of abc and bca. This is because abc is a subpattern of $abcd$ in s_4 and s_5 and is thus discarded by the `purge_subpatterns` step. The remaining occurrences of abc in s_2 and bca in s_3 both do not adhere to the minsupport requirement. This interplay of side effects is less significant when dealing with large datasets.

TABLE 5.10: The result of complete Apriorirep1i applied on the running example

seq.	occurrences of $\boldsymbol{abcd}$
s_4	$\{0, 4, 8\}$
s_5	$\{0, 5, 9\}$

5.2.3 Application

Apriorirep1i expects a sequence database as input. This was done by simply mapping each action (as described in section 4.4) to a symbol. After the algorithm was applied on the log entries dataset for each NPS response, the results from the occurrence set were aggregated into single values corresponding to features DF11 to DF13 and stored for the next phase of this project. Additionally, a dataset with the found patterns was constructed to support Topicus with future analysis of the repetitive patterns.

The requirement in terms of efficiency was that it could run in an acceptable time frame. There were 1046 NPS entries, 107 355 sessions and 6 320 136 actions in

total. Averaging the average amount of actions per session over the amount of NPS responses, the result is 61.79 actions per session. Apriorirep1i took about 15 minutes to run on all of the log entry datasets of all NPS responses.

5.2.4 Related work

Toroslu introduced the notion of minimum repetition support [62]. He called this the field of *cyclic* pattern mining, a generalization of sequential pattern mining. It is a generalization because a minimum repetition support of 1 would yield the same results as sequential pattern mining algorithms. Toroslu presented an algorithm similar to Apriorirep1i. His algorithm is also based on Agrawal and Srikant's AprioriAll [1]. Contrary to sequential pattern mining algorithms and like Apriorirep1i, it finds only patterns with itemsets of length 1. Instead of our occurrence set structure, it uses a hash-tree for efficient storage of candidates. Where Apriorirep1i only finds patterns with consecutive elements, Toroslu's algorithm allows patterns to be interleaved with other elements. This is more appropriate for the use cases of his research: the stock market and customer transactions. To illustrate, consider the sequence <$cacbacacabbacabacaac$>. One of the cyclic patterns with repetition support 3 that Toroslu identifies is aba.

Another important difference is how Toroslu deals with his equivalent problem to the self-succeeding pattern origin ambiguity. In his case the patterns are not (necessarily) self-succeeding, as patterns can be interleaved with other elements. Terminology aside, the pattern origin ambiguity is still present and Toroslu takes a different approach. He views the shifted versions of a pattern as members of the same family. Consider the previously mentioned sequence again. The shifted patterns aab, aba and baa have repetition supports 2, 3 and 3, respectively. All three patterns are considered supported in the sequence (assuming a minimum repetition support of 3) because the family's highest repetition support (aba or baa) satisfies the minimum. The nature of these cyclic patterns ensure that the difference between the repetition support of family members is only 1. To illustrate both approaches in the context of this study, take the sequence <$cabcabcghabcabc$>. Apriorirep1i's basic version would recognize all occurrences of both the cab and abc patterns. Its sliding window approach of dealing with the ambiguity would identify cab as the main pattern of the first six indices and thus remove the first two found occurrences of abc. Its result would be repetition support 2 for both cab and abc patterns. Toroslu's approach would find repetition support 4 for pattern cab: the first two and the last one are obvious; the third occurrence is interleaved with the elements gh. Applying Toroslu's approach to Apriorirep1i can be easily done by joining occurrences of shifted patterns into one family. Table 5.8 illustrates that this would work: the shifts of each pattern's elements correspond to the shifts in their occurrence index, e.g. bca occurs one index later than abc.

Other related work focuses on approaches with more or different requirements. Hu and Chiang [28], who extended the work of Toroslu, state that a constant minimum repetition support is too coarse-grained to take all types of sequential events into account. They introduce multiple minimum repetition supports and implement this in their PrefixSpan-based algorithm rep-PrefixSpan. PrefixSpan only takes the first occurrence of a pattern as a prefix. Rep-PrefixSpan considers each occurrence of a pattern as a prefix and limits the projected database length with another parameter. It does not deal with the ambiguity mentioned above. Rep-PrefixSpan is less suitable for datasets where the elements are unknown beforehand, as the researcher has no

idea which element should have which minimum repetition support.

Barreto and Antunes [2] also use a PrefixSpan-based approach. They state that periodicity is relevant for cyclic patterns. They define a pattern as a (s, ρ, δ) tuple, where s is the (**contiguous**) sequence, ρ is the period (i.e. the space between two occurrences) and δ is the amount of repetitions. Although their algorithm, PrefixSpan4Cycles, aims to find periodic patterns, its first part finds cyclic patterns without considering their periodicity. It does this in a way highly similar to basic Apriorirep1i. The data structure used is almost identical to occurrence sets. The biggest difference is that PrefixSpan4Cycles is recursive (depth-first), while Apriorirep1i is iterative (breadth-first). It also supports itemsets of lengths greater than 1, which Apriorirep1i does not as this was not relevant for our use case.

In conclusion, Apriorirep1i's basic part should have been subject to Barreto and Antunes' approach of the first part of PrefixSpan4Cycles. Apriorirep1i's ambiguity issue can be solved by the familial approach provided by Toroslu. Combining these methods and doing measurements of effectiveness and efficiency is considered future work into this algorithm for finding repetitive patterns that consist of consecutive actions and without the requirement of periodicity.

5.3 Low-hanging fruit

The pattern extraction operations for features DF23 to DF25 are relatively simple. Each user can have multiple schools at which he's worked in the 90 days prior his NPS response. This most often occurs because schools fuse, according to Topicus experts. The data format for DF23: the school at which the user works should thus be binary, where each school is a field. The resulting (reduced) dataset format is shown in table 5.11. The source is the mapping data mentioned in section 4.5.

TABLE 5.11: The reduced dataset format for feature DF23

Field	Data type	Description
NPSID	string	The NPS ID.
school: *	boolean	Whether the user was associated with the school mentioned in the field ('*') or not.

A similar line of reasoning is used for DF24: the education levels a user teaches. Each user can teach multiple education levels, so the target data format should be binary. Table 5.12 shows the reduced format. Again, the source is the mapping data previously mentioned in section 4.5.

TABLE 5.12: The reduced dataset format for feature DF24

Field	Data type	Description
NPSID	string	The NPS ID.
edlvl: *	boolean	Whether the user taught the education level mentioned in the field ('*') or not.

Finally, feature DF25: the age of the NPS response measured from January 1st, 2016, in days has a numeric format and the format is shown in table 5.13. The source is the date of the NPS response, included in the NPS dataset.

TABLE 5.13: The dataset format for feature DF25

Field	Data type	Description
NPSID	string	The NPS ID.
npsage	integer	The age of an NPS response measured from Janaury 1st, 2016, in days.

Chapter 6

Model training

This chapter discusses how the preprocessed datasets are joined and applied to see if there is any predictive value in them. The format of the dataset resulting from the pattern extraction efforts discussed in chapter 5 is shown in table 6.1. Section 6.1 discusses the approach to applying machine learning and section 6.2 mentions what model-specific preprocessing is applied in an attempt to improve the results. Finally, section 6.3 presents the results.

TABLE 6.1: The dataset format for the machine learning model input

Field	Data type	Description
NPS	integer	The NPS score. This is the attribute that will be predicted.
total_time	integer	DF19: the total time spent in the system
total_napitime	integer	DF20: the total time spent in Somtoday
total_apitime	integer	DF21: the total time spent in Somtoday Docent
total_switches	integer	DF22: the total amount of bidirectional switches between Somtoday and Somtoday Docent
repetitive_patterns	integer	DF11: the amount of distinct repetitive tasks
accumulated_repetitions	integer	DF12: the amount of iterations of repetitive tasks
weighted_pattern_iterations	float	DF13: the weighted amount of task iterations
school: *	boolean	DF23: the school at which the user works
edlvl: *	boolean	DF24: the education levels a user teaches
NPSAGE	integer	DF25: the age of the NPS response measured from January 1st, 2016, in days

6.1 Approach

Table 6.1 shows that most data types of the dataset are numerical. Only two features are boolean, allowing for several possibilities in utilizing this dataset. The target attribute NPS can be treated as continuous (numerical) and discrete (categorical) due to the nature of the type of NPS scoring. The range 0 to 10 is numerical, but can be treated as categorical without loss of semantics. These categories can also be aggregated, again without loss of semantics from the perspective of NPS scoring. NPS scores 0 to 6 fall into the category 'Detractor', scores 7 and 8 in 'Passive', and 9

and 10 in 'Promoter'. The input attributes can also be treated in different ways. The boolean types can be discarded, allowing for a fully numeric dataset. This opens the door to models like linear regression. A different option is to discretize the numerical attributes to end up with only categorical and binary attributes. This discretization can be done using different methods, discussed in section 6.2. We end up with the combinations of data types shown in table 6.2.

TABLE 6.2: Datasets used with predictive modeling

input data types		output data types
numerical	binary	categorical (11 values)
numerical	binary	categorical (3 values)
categorical	binary	categorical (11 values)
categorical	binary	categorical (3 values)
numerical	binary	numerical
numerical		numerical

Seeing as all these different variations are not necessarily incremental in performance, a brute force approach was chosen. Multiple combinations were tried concerning the data types of the dataset, the models, the model parameters and the preprocessing methods used (more on those in section 6.2).

The main evaluation metric differs per data type of the target attribute. The accuracy was chosen for the categorical data type, which is a measure for the ratio of correctly classified samples. This metric would be high if the data has lots of predictive value. In each instance the confusion matrix and each class' recall and precision was examined to determine whether anything interesting was going on in addition to the accuracy.

For the numeric type the mean absolute error (MAE) was chosen. The error variance is more important than large outliers, making the MAE preferred over RMSE. Additionally, reasoning about the model's usefulness is more intuitive when the error is in the same unit as the predicted value. This makes the MAE preferred over R^2 and MSE. This value should be low if the data has lots of predictive value.

Model validation is done using 10-fold cross validation. The predictive value of a model will be determined by comparing the performance with a model that makes random or constant predictions.

Another metric to consider is the correlation between a feature and the target attribute (i.e. the NPS score). This measures their association and can thus be valuable for the results.

RapidMiner Studio[1] was chosen as tooling for training and evaluating the models. It is advertised as a visual environment that "streamlines transformation, development and validation" and includes lots of machine learning models. RapidMiner uses nested visualized data flows, so showing them here would not increase understanding. Instead, listing 6.1 shows the hierarchical flow in text format. Each line requires some elaboration. First the data is read from a CSV file which has the format described in table 6.1. Then the target attribute is preprocessed. This entails type

[1] https://rapidminer.com/products/studio/

casting the numerical values into the categorical values 0 to 10 or 'Detractor' to 'Promoter'. This step is skipped when the target attribute is treated as a continuous value. The model chooser applies its nested operators on different models, so it's basically a loop. The parameter chooser does the same but with changing model parameters. The cross validator again does the same, but with a different fold of the data. Within the cross validation, the attributes are preprocessed, followed by the training of the model. Note that the chosen model, model parameters and dataset are all passed down to this model training operation. Attribute preprocessing is applied again, but on the test subset. Any normalization or discretization model from the training phase is used at this point. The reason this attribute preprocessing is done within the cross validation instead of right after reading the data is to deal with knowledge leakage. The trained model is applied on the test dataset, the performance metric is extracted and passed back up the hierarchy together with the model parameters resulting in the best performance. These metrics are aggregated into a table for evaluation.

LISTING 6.1: RapidMiner workflow

```
1.1. read data
1.2. target attribute preprocessing
1.3. model chooser
    2.1. parameter chooser
        3.1. cross-validation
            4.1. training dataset: attribute preprocessing
            4.2. training dataset: model training
            4.3. testing  dataset: attribute preprocessing
            4.4. testing  dataset: model application
            4.5. testing  dataset: performance extraction
```

The models and parameters in tables 6.3 to 6.6 were chosen to be used with our brute force approach. Note that only limited effort was put in choosing the most optimal parameters: our brute force approach removes that need. These models were chosen based on the data type they can handle and availability within RapidMiner. The same goes for the model parameters. Each table shows the models used for a different input-output data type combination. Combinations of all variable parameters were tried. Note that several changes to both the static and variable parameters were heuristically tried, but with no significant effect on the results as they are presented in section 6.3.

TABLE 6.3: The selected machine learning models for **input types numerical and binary, output type categorical**

Model	Abbr.	Parameter	Value(s)
Decision tree	DT	criterion	{gain_ratio, information_gain, gini_index, accuracy}
		prepruning minimal gain	$[1 \cdot 10^{-3}, 1 \cdot 10^{-2}, \ldots, 1 \cdot 10^{1}]$
		maximal depth	20
		pruning confidence	0.25
		prepruning minimal leaf size	2

Continued on next page

Table 6.3 – continued from previous page

Model	Abbr.	Parameter	Value(s)
		prepruning minimal size for split	4
		number of prepruning alternatives	3
Naive Bayes	NB	laplace correction	{yes, no}
Naive Bayes (kernel)	NBk	laplace correction	{yes, no}
		estimation mode	full
		bandwidth selection	heuristic
k-Nearest Neighbors	kNN	k	[1, 2, . . . , 50]
		weighted vote	{yes, no}
		measure	MixedEuclideanDistance
Rule induction	RI	criterion	{information_gain, accuracy}
		sample ratio	0.9
		pureness	0.9
		minimal prune benefit	0.25
Random Forests	RF	criterion	{gain_ratio, information_gain, gini_index, accuracy}
		number of trees	[1, 3, . . . , 151]
		maximal depth	20
		apply pruning	yes
		confidence	0.25
		apply prepruning	yes
		minimal gain	0.1
		minimal leaf size	2
		minimal size for split	4
		number of prepruning alternatives	3
Random Tree	RT	criterion	{gain_ratio, information_gain, gini_index, accuracy}
		minimal gain	$[1 \cdot 10^{-3}, 1 \cdot 10^{-2}, \ldots, 1 \cdot 10^{1}]$
		minimal leaf size	2
		minimal size for split	4
		maximal depth	20
		confidence	0.25
		number of prepruning alternatives	3

Continued on next page

Table 6.3 – continued from previous page

Model	Abbr.	Parameter	Value(s)
Deep Learning	DL	activation	{Tanh, TanhWithDropout, Rectifier, RectifierWithDropout, Maxout, MaxoutWithDropout, ExpRectifier, ExpRectifierWithDropout}
		hidden dropout layers	0.5, 0.5
		hidden layer sizes	50, 50
		epochs	10
		train samples per iteration	automatic
		adaptive rate	yes
		epsilon	$1.00 \cdot 10^{-8}$
		rho	0.99
		standardize	yes
		L1	$1.00 \cdot 10^{-5}$
		L2	0
		max w2	10.0
Decision stump	DS	criterion	{gain_ratio, information_gain, gini_index, accuracy}
		minimal leaf size	1

TABLE 6.4: The selected machine learning models for **input types categorical and binary, output type categorical**

Model	Abbr.	Parameter	Value(s)
Everything the same as table 6.3 except for this replacement and addition:			
k-Nearest Neighbors	kNN	k	[1, 2, . . . , 50]
		weighted vote	{yes, no}
		measure	{NominalDistance, DiceSimilarity, JaccardSimilarity, KulczynskiSimilarity, RogersTanimotoSimilarity, RussellRaoSimilarity, SimpleMatchingSimilarity}
Chi-squared Automatic Interaction Detector	CHAID	minimal gain	$[1 \cdot 10^{-3}, 1 \cdot 10^{-2}, \ldots, 1 \cdot 10^{1}]$
		minimal leaf size	2
		minimal size for split	4
		maximal depth	20
		confidence	0.25

Continued on next page

Table 6.4 – continued from previous page

Model	Abbr.	Parameter	Value(s)
		number of prepruning alternatives	3

TABLE 6.5: The selected machine learning models for **input types numerical and binary, output type numerical**

Model	Abbr.	Parameter	Value(s)
k-Nearest Neighbors	kNN	k	[1, 2, . . . , 50]
		weighted vote	{yes, no}
		measure	MixedEuclideanDistance
Generalized Linear Model	GLM	solver	{IRLSM, L_BFGS, COORDINATE_DESCENT_NAIVE, COORDINATE_DESCENT}
		use regularization	{yes, no}
		family	automatic
		standardize	yes
		lambda search	yes
Deep Learning	DL	activation	{Tanh, TanhWithDropout, Rectifier, RectifierWithDropout, Maxout, MaxoutWithDropout, ExpRectifier, ExpRectifierWithDropout}
		hidden dropout layers	0.5, 0.5
		hidden layer sizes	50, 50
		epochs	10
		train samples per iteration	automatic
		adaptive rate	yes
		epsilon	$1.00 \cdot 10^{-8}$
		rho	0.99
		standardize	yes
		L1	$1.00 \cdot 10^{-5}$
		L2	0

TABLE 6.6: The selected machine learning models for **input type numerical, output type numerical**

Model	Abbr.	Parameter	Value(s)
k-Nearest Neighbors	kNN	k	[1, 2, . . . , 50]
		weighted vote	{yes, no}
		measure	{EuclideanDistance, CamberraDistance, ChebychevDistance, CorrelationSimilarity, CosineSimilarity, DiceSimilarity, DynamicTimeWarpingDistance, InnerProductSimilarity, JaccardSimilarity, KernelEuclideanDistance, ManhattanDistance, MaxProductSimilarity, OverlapSimilarity}
Neural net	NN	learning rate	{0.3, 0.03}
		momentum	{0.2, 0.02}
		training cycles	{500, 5000}
		decay	no
		error epsilon	$1.00 \cdot 10^{-5}$
Linear regression	LR	feature selection	{none, M5 prime, greedy, T-Test, Iterative T-Test}
		minimum tolerance	{0.5, 0.05, 0.005}
		eliminate colinear features	yes
		use bias	yes
		ridge	$1.00 \cdot 10^{-8}$
		alpha	0.05
		maximum iterations	10
		forward alpha	0.05
		backward alpha	0.05
Support vector machine	SVM	kernel type	{dot, radial, polynomial, anova, epachnenikov, gaussian combination, multiquadric}
		C	{-0.1, 0.01, 0.1, 1, 10, 100}
		convergence epsilon	0.001
		maximum iterations	100 000
		kernel gamma	1
		kernel degree	2
		kernel a	1
		kernel b	0

Continued on next page

Table 6.6 – continued from previous page

Model	Abbr.	Parameter	Value(s)
		kernel sigma1	1
		kernel sigma2	0
		kernel sigma3	2
		kernel shift	1
		L positive	1
		L negative	1
		epsilon	0
		epsilon plus	0
		epsilon minus	0
Polynomial regression	PR	replication factor	{1, 2, 3}
		maximum degree	{5, 6, 7, 8, 9, 10}
		maximum iterations	5000
		minimum coefficient	-100
		maximum coefficient	100

6.2 Model-specific preprocessing

Several preprocessing attempts were made to improve the results of the predictive models. Some models work better with data that is preprocessed in certain specific ways. Examples include normalization and outlier removal. To cover these cases, our brute force approach applied all preprocessing options to check which had any impact on the results.

The numeric attributes have different ranges, making normalization of the data a relevant choice. Several methods were tried:

Range transformation (range -1 to 1) This method downscales the values so that the minimum value becomes -1 and the maximum becomes 1.

Proportion transformation This method is based on the proportion of each attribute value on the whole attribute value set.

Z-transformation Also known as statistical normalization, this method makes sure the new dataset has a mean of zero and a variance of one. It reduces the influence of outliers and preserves the original distribution.

Discretizing was applied in the case where the numerical attributes were treated as categorical. Due to the skewness of the distribution of each attribute, the only feasible type of discretizing was binning. The amount of bins tried were 10 and 100.

Removal of ten outliers was tried based on the Euclidian distance function. The same was attempted with a cosine distance function.

Figure 4.1 shows the clear class imbalance of the target attribute. Different methods of dealing with this are:

Class upsampling[2] This multiplies random samples from the underrepresented classes so each class becomes a specified size.

Class downsampling[3] This cuts random samples from the dominant classes so that each class becomes a specified size.

SMOTE upsampling This method by Chawla et al., Synthetic Minority Over-sampling Technique [10], takes a sample from an underrepresented class, considers its k nearest neighbors and adds a random sample between itself and one of its neighbors to the dataset. This randomness makes sure the synthetic samples are similar to the existing, but not exact copies. This reduces the chance on overfitting.

Adding weights Weights can be added to each sample to increase or decrease its importance, e.g. for the underrepresented and dominant classes, respectively. This can also be used to give more priority to features with more predictive value.

6.3 Results

Table 6.7 shows the result of random predictions averaged over 1000 iterations. The values are calculated twice: with and without taking the NPS score distribution into account. Table 6.8 shows the results for the situation where the dominant classification 7 (or 'Passive') is always predicted. To place the MAE value in a bit more context: if 50% of the samples is predicted correctly and the rest randomly (without taking the distribution into account), the average MAE over 1000 iterations is 1.555. The same situation with 80% correctly predicted samples results in a MAE of 0.627.

TABLE 6.7: Results if prediction is done randomly, with and without taking the target attribute distribution into account, averaged over 1000 iterations

	with distribution		without distribution	
target attribute	accuracy	MAE	accuracy	MAE
categorical (11 values)	16.07%		9.093%	
categorical (3 values)	45.41%		41.24%	
numerical		2.457*		3.101

* This value is based on predicting integers instead of floats because of the known distribution. Adding random noise wouldn't have added any value because the prediction is already random.

The results shown in tables 6.9 to 6.11 are the best values that could be obtained using the model parameters from tables 6.3 to 6.6 and the preprocessing methods mentioned in section 6.2. Normalization, outlier removal and dealing with the class imbalance seemed to have no positive effect. Discretizing into 100 bins resulted in about a 0.1% accuracy increase compared to discretizing into 10 bins.

Most models performed similar with most of their model parameters. Notable mentions are the models based on decision trees (random forests, random trees, CHAID, decision stumps and decision trees), which performed consistently in the

[2] Also known as *oversampling*.
[3] Also known as *undersampling*.

TABLE 6.8: Results if the predicted value is always the dominant classification: 7 (or 'Passive')

target attribute	accuracy	MAE
categorical (11 values)	27.63%	
categorical (3 values)	44.07%	
numerical		1.7333

higher accuracy ranges, and the Naive Bayes models, which performed consistently on the low end. K-Nearest Neighbor model performance varied the most with the value of k, ranging from best to worst when compared to other models. There was no clear connection observed between the performance and the value of k.

TABLE 6.9: Prediction model results for output data type categorical (11 values)

Model	Optimal model parameters	Accuracy
Input data type: numerical and binary		
RF	number of trees: 21, criterion: gain ratio	27.92%
DS	criterion: gain ratio	27.73%
RT	criterion: gain ratio, minimal gain: 0.001	27.63%
DL	activation: RectifierWithDropout	27.63%
DT	criterion: accuracy, minimal gain: 0.01	27.62%
kNN	k: 50, weighted vote: false	25.24%
RI	criterion: information gain	21.32%
NBk	laplace correction: false	13.14%
NB	laplace correction: false	10.22%
Input data type: categorical and binary		
RF	number of trees: 9, criterion: gain ratio	27.47%
RT	criterion: gain ratio, minimal gain: 0.001	27.47%
DL	activation: RectifierWithDropout	27.38%
DS	criterion: gain ratio	27.38%
CHAID	minimal gain: 0.001	27.38%
DT	criterion: information gain, minimal gain: 0.001	27.38%
kNN	k: 49, weighted vote: false, nominal measure: DiceSimilarity	26.46%
RI	criterion: information gain	25.90%
NB	laplace correction: false	19.68%
NBk	laplace correction: false	19.68%

Finally, appendix B shows the correlations between the attributes and the NPS score in table B.1. The ten attributes with the highest correlations are repeated in table 6.12. Feature selection was tried by discarding the least relevant features based

TABLE 6.10: Prediction model results for output data type categorical (3 values)

Model	Optimal model parameters	Accuracy
Input data type: numerical and binary		
DL	activation: MaxoutWithDropout	52.64%
DT	criterion: accuracy, minimal gain: 0.001	51.53%
RF	number of trees: 49, criterion: gain ratio	51.43%
kNN	k: 47, weighted vote: false	51.16%
RT	criterion: gain ratio, minimal gain: 0.001	50.97%
RI	criterion: accuracy	50.69%
DS	criterion: gain ratio	50.60%
NBk	laplace correction: false	38.57%
NB	laplace correction: false	34.35%
Input data type: categorical and binary		
kNN	k: 43, weighted vote: true, nominal measure: NominalDistance	54.30%
DL	activation: Tanh	52.36%
RT	criterion: information gain, minimal gain: 0.001	52.08%
RF	number of trees: 5, criterion: information gain	51.53%
RI	criterion: information gain	51.52%
DT	criterion: information gain, minimal gain: 0.001	50.97%
CHAID	minimal gain: 0.001	50.69%
DS	criterion: gain ratio	50.60%
NB	laplace correction: false	49.07%
NBk	laplace correction: false	49.07%

TABLE 6.11: Prediction model results for output data type numerical

Model	Optimal model parameters	MAE
Input data type: numerical and binary		
DL	activation: MaxoutWithDropout	1.7613
GLM	solver: COORDINATE_DESCENT, use regularization: true	1.7732
kNN	k: 48, weighted vote: true	1.8102
Input data type: numerical		
SVM	kernel type: anova, C: 0	1.7267
kNN	k: 49, weighted vote: true, numerical measure: MaxProductSimilarity	1.7504
LR	feature selection: T-Test, minimum tolerance: 0.5	1.7866
NN	learning rate: 0.03, momentum: 0.02, training cycles: 500	1.8069
PR	replication factor: 1, maximum degrees: 6	4.5254

on correlation, chi squared, information gain and gini index. This did not lead to better results.

TABLE 6.12: Top 10 attributes having the highest correlation with the NPS score

Attribute	Correlation
school: #25	0.1108
total_napitime	0.1095
school: #123	0.0978
school: #84	0.0912
edlvl: VMBO BBL	0.0891
school: #116	0.0878
edlvl: VMBO KBL	0.0859
school: #114	0.0815
edlvl: VMBO TL/GL	0.0744
school: #19	0.0705

Chapter 7

Validation

The performance metrics suggest that there is almost no predictive value in the dataset, because the results are not significantly better than the scenario where the dominant class is always predicted. The brute force approach tells us that this is not due to our model, model parameter or preprocessing method choice. This line of reasoning shows that the brute force approach can be seen as a validation method, at least to show the predictive value of this dataset.

It was validated that the workflow as described in listing 6.1 avoids information leakage. The accuracy of the model increased when class upsampling was applied before the cross validation and kept growing with the class sample size.

Figure 7.1 shows that the decision tree model could find practically no predictive value in the dataset. Only two samples are directed to the 'true' branch (of which one is wrongly classified), hence the graphical differences of the two branches and two leaves. All other samples are predicted to have classification 7, which is in line with our calculations of always predicting the dominant class. The decision trees generated with other parameters and other datasets yielded similar results: only a few samples diverging from the main branch. Figure 7.2 supports this. Even though it has more branches, there is no feature that makes a clear cut between the data. Just a few of the samples are directed away from the main branch.

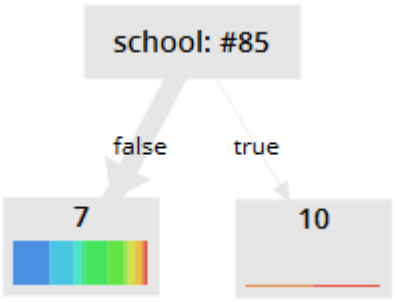


FIGURE 7.1: One of the decision trees generated for the dataset with input data types numerical and binary and output data type categorical. The accuracy of this model is 27.7247%. Colors orange and red signify true values 10 and 1, respectively.

Table 7.1 shows the confusion matrix of the best performing model of the dataset with input data types numerical and binary and output data type categorical (11 values). The underlined values on the diagonal are the amount of correctly predicted samples adding to the model's accuracy. This shows that this model also has almost no predictive value: there are relatively few correctly predicted samples. Additionally, the prediction of the dominant class does not have much value, seeing as 92.8% of all samples are predicted to belong to this class. Precision of some classes is high, but this does not work in favor of the model's predictability seeing as the recall of

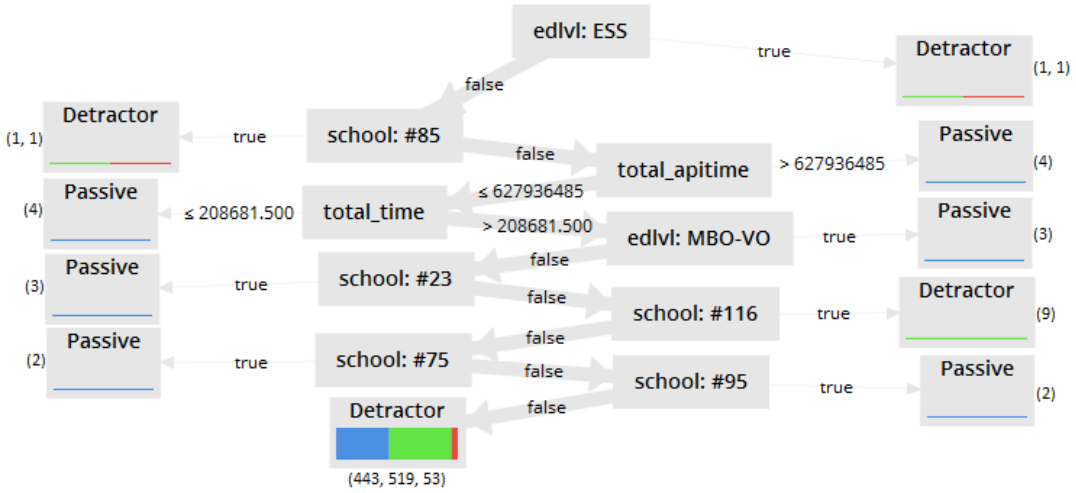


FIGURE 7.2: One of the decision trees generated for the dataset with input data types numerical and binary and output data type categorical. The numbers between parentheses denote the amount of samples in the respective true classes in that prediction branch. The accuracy of this model is 52.3901%. Colors blue, green and red signify true values Passive, Detractor and Promoter, respectively.

the corresponding classes is low. table 7.2 shows the confusion matrix for the best performing model of the dataset with input data types categorical and binary and output data type categorical (3 values). This shows that the performance is a bit better, although the underrepresented class (Promoter) isn't predicted at all. The dominant class (Detractor) is still the 'best' predicted class. Nevertheless, the increased accuracy can be attributed to the coarseness and imbalance of the classes. Other confusion matrices were similar: more samples predicted in the dominant class and sometimes more (incorrectly predicted) outliers.

TABLE 7.1: The confusion matrix of the best random forest model for the dataset with input data types numerical and binary and output data types categorical (11 values), with the optimal model parameters {number of trees: 77, criterion: gain ratio}

	true value											
predicted	10	9	8	7	6	5	4	3	2	1	0	precision
10	0	0	0	0	1	0	0	0	0	0	0	0.00%
9	1	0	0	0	1	0	0	0	0	0	0	0.00%
8	1	1	11	4	6	5	1	2	1	0	0	34.38%
7	15	34	153	275	170	122	52	41	28	26	55	28.32%
6	1	1	3	7	7	2	1	0	1	0	0	15.79%
5	0	0	4	2	2	1	1	1	1	0	1	7.69%
4	0	0	0	1	0	0	0	0	0	0	0	0.00%
3	1	0	1	0	0	0	0	1	0	1	0	25.00%
2	0	0	0	0	0	0	0	1	0	0	0	0.00%
1	0	0	0	0	0	0	0	1	0	0	0	0.00%
0	0	0	0	0	0	0	0	0	0	0	1	100.00%
recall	0.00%	0.00%	6.40%	95.16%	1.64%	0.77%	0.00%	2.13%	0.00%	0.00%	1.75%	27.92%
											accuracy ↗	

TABLE 7.2: The confusion matrix of the best k-Nearest Neighbors model for the dataset with input data types categorical and binary and output data types categorical (3 values), with the optimal model parameters {k: 43, weighted vote: true, nominal measure: NominalDistance}

	true value			
predicted	Promoter	Passive	Detractor	precision
Promoter	0	0	0	- [1]
Passive	22	126	89	53.16%
Detractor	33	335	441	54.51%
recall	0.00%	27.33%	83.21%	54.21% [2]
			accuracy ↗	

[1] The nature of the precision calculation ensures a division by zero when using only zero values.

[2] The accuracy differs from table 6.10 because it is based on a model applied on one fold instead of the average of a cross-validation.

For the regression models, other performance metrics are similar to the values obtained when always predicting the dominant value. Always predicting an NPS score of 7 results in a root mean square error (RMSE) of 2.583. The best performing regression models, DL (for input datatype numerical and binary) and SVM (for input datatype numerical), both have an RMSE of 2.301.

A model that always predicts a value of 7 has a mean squared error (MSE) of 6.667. The DL and SVM models perform with an MSE of 5.324 and 5.335, respectively. This seems like a large difference, but remember that the operation of squaring a value magnifies small differences. It tells us that the DL and SVM models have smaller errors compared to the 'constant' model, but are still not even close to accurate.

The other models had similar values for both performance metrics. These values support our results presented in section 6.3.

Chapter 8

Conclusion

With this study we've tried to find patterns in teachers' usage data with which NPS responses can be predicted regarding the education support application Somtoday. Section 8.1 presents our findings, and section 8.2 discusses those. Finally, section 8.3 points out this thesis' contributions and argues our recommendations.

8.1 Main research findings

Chapters 3 to 7 have answered the research questions and each one discusses a distinct phase of this project:

The feature discovery phase had the purpose of getting an indication of which usage patterns might be of influence on the NPS response. A list of 25 data features was created by talking to the system's users and domain experts and by looking at the qualitative data. Ten of those data features (DF11 to DF13 and DF19 to DF25), divided over the categories system usage, repetitive tasks and low-hanging fruit, were selected from this list to use in this project. Selection was done based on the features' expected impact on the NPS response, their preparation time and their generalizability to other systems.

The data preprocessing phase focused on transforming the raw data into a format that was usable for research. This was done by applying several filters (e.g. based on user role, time, relevance and duplicity) and ETL operations (e.g. grouping into sessions and including the timestamp of the next log entry).

The pattern extraction phase found the usage patterns for each user and compressed them into one value per data feature, resulting in ten values per NPS response. This was the most complex with the repetitive tasks category, as we initially could not find an algorithm in scientific literature that met our requirements. The algorithm Apriorirep1i was created to find repetitive tasks from which, in turn, values for features DF11 to DF13 were extracted. The system usage features (DF19 to DF22) were extracted by looking at the time between subsequent requests of the same user and the low-hanging fruit features were extracted from the NPS dataset (DF25) and Somtoday's database (DF23 and DF24).

The model training phase tried to find predictive value in the extracted patterns by applying a brute force approach. A multitude of combinations of models, model parameters an preprocessing methods was tried. The most significant findings are aggregated in table 8.1. This shows that even the best performing models are doing only slightly better than the scenario where the dominant class is always predicted. Applying a model with such performance in a production environment would yield results that are unreliable. Even though the three-value categorical case has a 10.23% accuracy improvement, the coarse granularity of the target attribute diminishes its significance. Thus, there is practically no predictive value in the dataset.

The validation phase had the same outcome. This was determined by validating that the model training workflow was correct and by looking at decision trees and other performance metrics. The low feature-to-NPS correlations shown in table B.1 support this.

TABLE 8.1: Top results from section 6.3 put together

target attribute	input attributes	model	model parameters	accuracy	MAE	accuracy*	MAE*
categorical (11 values)	numerical, binary	RF	number of trees: 21, criterion: gain ratio	27.92%		27.63%	
categorical (3 values)	categorical, binary	kNN	k: 43, weighted vote: true, nominal measure: NominalDistance	54.30%		44.07%	
numerical	numerical	SVM	kernel type: anova, C: 0		1.7267		1.7333

* If the dominant class is always predicted.

These answers to the research questions lead to the answer to the main research question: **teachers' usage data can not be reliably used to predict the user's loyalty towards the system.** At least, not based on the chosen data features and dataset.

8.2 Discussion

The answer to the main research question raises a new question: why can't the usage data be used to predict user loyalty? We expected to find at least *some* predictive value. Three themes were identified in which to look for an explanation.

The first is **the process** of going from the raw data to the NPS prediction. We skip the feature discovery phase for now, as this will be discussed in our recommendations.

In the context of data preprocessing more time could've been spent on getting to know the ins and outs of the (usage data) dataset, e.g. how each action or set of actions in the system is mapped in the dataset. With our approach we've done our best, for example, to discard all irrelevant log entries and only keep 'actionable' entries, but more irrelevant entries can probably be found if more time is spent on it.

In the pattern extraction phase, the Apriorirep1i algorithm can be improved with a better solution to the self-succeeding pattern origin ambiguity. One way is to extend the sliding window approach so that it also looks at the tail of a sequence of self-succeeding pattern occurrences, e.g. by detecting that sequence <$abcabcabc$> begins *and* ends with pattern abc. A different and preferred method is the approach used by Toroslu: considering a pattern's shifted versions (e.g. abc, bca and cab) as members of the same family and consequently dealing with pattern families instead of individual patterns. This would result in a better selection of repetitive tasks. Had our current methods found a hint of predictive value in the repetitive tasks data features, we would have reason to believe that these improvements to Apriorirep1i could result in better predictive performance. However, this is not the case, leading us to believe that such a scenario would not change the conclusions of this project.

Regarding the model training phase, we've tried to cover the possibility of using different models, model parameters and preprocessing methods with our brute force approach. As a result, we have no specific points of discussion for this phase.

The second theme is **the target data**: the NPS dataset. The class imbalance was evident during the model training phase. It is possible that a better distribution of NPS responses would've increased the model's predictive performance. This is unlikely, seeing as our efforts of class upsampling, downsampling and SMOTE upsampling did not yield better results.

A different and more Occam's razor-like explanation is that, in addition to the metric's debatable validity mentioned in section 2.1, NPS responses simply can't be predicted based on usage data. Because, for example, the NPS metric might be too coarse-grained and global to let behavioral patterns influence it. Perhaps NPS responses can only, or primarily, be predicted based on non-usage data, such as attitude, intrinsic motivations, general happiness or IT competence. The only way to be reasonably confident that this is the case is by researching more usage patterns.

The third theme is **the source data**: the log entries dataset. One way of looking at this is by applying the previous explanation to the dataset used in this project: there might indeed be no predictive value in *our chosen features*. This can also be tested by researching more usage patterns.

Another perspective shows that our familiarity with the dataset is limited to what was necessary for its analysis. It is possible that our specific knowledge hindered us from processing the dataset properly. We may have not taken into account nuances that we were unaware of. For example, why is there such a high variation in the amount of log entries per NPS response? And are the repetitive tasks found by Apriorirep1i indeed repetitive tasks, or can they be explained by something we haven't thought of? We have talked to domain experts, but this is merely an approximation an approximation to full familiarity with the data. If our knowledge had been all-encompassing, we'd be able to answer these kinds of questions. This understanding might have lead to better data preprocessing and pattern extraction, which in turn might have lead to better results. One way to deal with this is addressed in our recommendations.

8.3 Recommendations

Our contributions to Topicus and to science in general are largely similar. We looked into making first steps for creating a non-invasive, automated and userbase-wide way for system owners to discover which users to focus their improvement efforts on in the context of education support web applications. As our conclusion is that this is not feasible with our current research parameters, we contribute the notion to redirect focus to different data features. We provide Topicus with a list of data features that can be researched next. Likewise, the scientific community can use this list as a starting point for research with similar systems. We also provide Topicus with the process we used for data preprocessing. This validated workflow could save valuable amounts of time for future research efforts. In addition we have provided Topicus with the repetitive patterns found in the pattern extraction phase. These can be used for further analysis, for example to determine which tasks are repeated the most and thus can be considered for batching.

Although Apriorirep1i is not yet ready for practical use, it contributes to the field of data mining and specifically sequential pattern mining. Together with the suggestion of combining parts of the algorithms introduced by Toroslu in 2003 and Barreto and Antunes in 2014, it opens the door for efficient analysis into repetitive tasks consisting of consecutive actions and without the requirement of periodicity. Future

work includes developing this improvement and doing measurements into its effectiveness and efficiency.

The motivations for this research are well-founded and because the conclusion definitely does not rule out the possibility that user loyalty towards a system can be predicted based on their behavior, our main recommendation for Topicus is to focus on finding predictive value in other usage patterns. From the list compiled in our feature discovery phase, we'd recommend starting with the features about clickstreams (DF14 to DF16) and encountered downtimes (DF17). These can be researched in a cost-effective way, as they are fairly straightforward. Additionally, methods for analyzing clickstreams are abundandly researched by the scientific community.

Should Topicus indeed decide to search for predictive value in more usage patterns, then valuable time can be saved by improving their data infrastructure beforehand. To illustrate: one of Apache Drill's advertised advantages, its schema-free flexibility, considerably hindered our preprocessing efforts. Even though a strict schema might require a bit more effort to set up, it does allow for easier data analysis. The collection and storage of log data should facilitate in future data analysis. Our recommendation (to data-processing organizations in general) is to make sure the process of data analysis is actually assisted by the choice of collection and storage methods and not obstructed it.

Finding predictive value in more usage patterns should also be the goal for future work of the scientific community. There is, however, one concern with this project from a scientific standpoint: the indicative nature of the feature discovery results. Our recommendation is to conduct a more in-depth feature discovery research project. Although it is not primarily recommended to Topicus due to its extensiveness and the indirectness of its contribution to NPS predictions, it does provide value and is worth considering by Topicus. A full-scale user experience study with the goal of usage data analysis gives the researcher quantitative and qualitative data about what users think of different aspects (e.g. navigation and layout) and functionalities and how their behavior is mapped onto the log entries dataset. An extensive survey is one way, and should at least be supplemented by sit-in sessions where the researcher observes the user's behavior and the generated data in real-time. Having better insight leads not only to validated feature selection, it also leads to refined knowledge about how data features can be measured and what the nuances are. Choosing this approach, one or multiple directed studies into specific usage patterns can be set up and the researcher has a better chance of finding predictive value in user behavior: the researcher is no longer looking in the proverbial dark.

Data is a basis for research, so it holds that, especially in the context of data research: knowledge[1] is power.

[1] about the subtleties of the researched data

Appendices

Appendix A

Apriorirep1i implementation

This implementation is also available on https://karim.elass.al/masterthesis/apriorirep1i.

LISTING A.1: Python 2.7 implementation

```
from collections import defaultdict

def apriorirep1i(sequence_db, minsupport, minrepsupport, minpatternlen=2):
    '''
        A Python (2.7) implementation of the apriorirep1i algorithm, based on AprioriAll. More information and an online version can be found at https://karim.elass.al/masterthesis/.
        The return value is an occurrence set: a dictionary where the keys are tuples of frequent patterns, the values are dictionaries where, in turn, the keys are the sequence indices and the values are sets of integers. Those integers are the indices indicating where the frequent pattern starts in the sequence.
        The value of minsupport is considered to be absolute and minrepsupport must be larger than 1.
    '''

    # Sort phase is already done when receiving sequences.

    # Litemset phase.
    Lk = get_l1items(sequence_db, minsupport, minrepsupport)
    l1items = Lk.keys()

    # Transformation phase.
    # Truncate irrelevant sequences instead of removing all non-l1items: else the occurrence set data structure doesn't work anymore.
    Dt = truncate_nonlitem_sequences(sequence_db, l1items)

    # Sequence phase.
    k = 2
    # Lk is the occurrence data of k-1, L is the occurrence data of k.
    L      = defaultdict(lambda:defaultdict(set)) # occurrence data
    result = defaultdict(lambda:defaultdict(set)) # occurrence data
    while len(L) > 0 or k == 2:

        L = get_next_patterns(Lk, Dt)
        L = purge_nonlpatterns(L, minsupport, minrepsupport)

        # Purge occurrences of patterns that are a starting cycle of its subpattern (e.g. abca in abcabc with subpattern abc).
        L = purge_subpattern_cycle_start(L, Lk, Dt)

        # Purge occurrences of patterns that are a self-succeeding cyclically shifted variant (e.g. bca and cab in abcabcabc).
        L = purge_cyclic_shifts(L, Dt)
```

```
35
36          # Enforce minsupport and minrepsupport requirements.
37          # Necessary because irrelevant patterns shouldn't be taken into
            ↪  account in the next step.
38          L = purge_nonlpatterns(L, minsupport, minrepsupport)
39
40          # Maximal phase.
41          if k-1 >= minpatternlen:
42
43              # Purge subpatterns that connect two consecutive occurrences
                ↪  of a superpattern (e.g. ca in abcabc).
44              Lk = purge_interconnecting_subpatterns(Lk, L)
45
46              # Purge subpatterns that are actually part of a superpattern
                ↪  (e.g. ab in abcabc).
47              Lk = purge_subpatterns(Lk, L)
48
49              # Save Lk.
50              result.update(Lk)
51
52          # Continue.
53          Lk = L
54          k += 1
55
56      # Include the latest L in the result.
57      result.update(L)
58
59      # Enforce support requirements for patterns that had their occurrences
        ↪  decreased in the maximal phase.
60      result = purge_nonlpatterns(result, minsupport, minrepsupport)
61
62      return result
63
64  def get_l1items(sequences, minsupport, minrepsupport):
65      ''' Return an occurrence data structure of litems. '''
66      items = defaultdict(lambda: defaultdict(set))
67      for si,sequence in enumerate(sequences):
68          for i,item in enumerate(sequence):
69              items[(item,)][si].add(i)
70      return purge_nonlpatterns(items, minsupport, minrepsupport)
71
72  def truncate_nonlitem_sequences(sequences, litems):
73      ''' If there is not any litem in a sequence in L, truncate that
        ↪  sequence. '''
74      return [
75          []
76              if not any((item,) in litems for item in sequence)
77              else
78                  sequence
79          for sequence in sequences
80      ]
81
82  def get_next_patterns(Lk, Dt):
83      ''' Generate patterns of length k+1 based on the occurrences of
        ↪  patterns of length k. '''
84      L = defaultdict(lambda: defaultdict(set))
85      # For each pattern of length k...
86      for p, occurrences in Lk.iteritems():
87          # si : sequence id/index.
88          # psi: 'pattern in sequence' index (start of the pattern).
89          # For each pattern occurrence in each sequence...
90          for si, psis in occurrences.iteritems():
91              for psi in psis:
```

```
92                  # 'item in sequence' index of the litem subsequent to p.
93                  isi = psi+len(p)
94                  # Avoid out of bounds exception.
95                  if len(Dt[si]) <= isi:
96                      continue
97                  # Add new pattern to the result set.
98                  L[p + (Dt[si][isi],)][si].add(psi)
99      return L
100
101 def purge_nonlpatterns(L, minsupport, minrepsupport):
102     ''' Filter an occurrence data structure by minimum sequence and
        ↪ repetitions. '''
103     for c, occurrences in L.iteritems():
104         L[c] = defaultdict(set, {
105             si:s
106             for si, s in occurrences.iteritems()
107                 if len(s) >= minrepsupport
108         })
109
110     return defaultdict(lambda:defaultdict(set),
111         {
112             c: occurrences
113             for c, occurrences in L.iteritems()
114                 if len(occurrences) >= minsupport
115         })
116
117 def purge_subpattern_cycle_start(L, Lk, Dt):
118     ''' Purge the occurrences of candidates in L that are starting their
        ↪ self-succeeding cycle, based on occurrences of length k-1 (passed
        ↪ in Lk). '''
119     for c in L.keys():
120         if c[0] == c[-1] and Lk.has_key(c[0:-1]):
121             for si, s in L[c].iteritems():
122                 # Remove index if Lk[c[0:-1]] has the cyclical occurrence
                    ↪ (starting at i and i+length of c[0:-1]).
123                 L[c][si] = set([
124                     i
125                     for i in L[c][si]
126                         if not (i              in Lk[c[0:-1]][si] and
127                                 i+len(c[0:-1]) in Lk[c[0:-1]][si])
128                 ])
129     return L
130
131 def purge_cyclic_shifts(L, Dt):
132     ''' Purge the occurrences of self-succeeding cyclically shifted
        ↪ variants of a candidate. '''
133     # Use a sliding window to determine the first occurrence of a pattern.
134     if len(L) > 0:
135         l = len(L.keys()[0]) # pattern length
136         for si,s in enumerate(Dt):
137             cp  = None # current main pattern
138             cpi = None # current pattern index
139             for i in range(0, len(s)-l+1):
140                 p = tuple(s[i:i+l])
141
142                 # Within cyclical pattern range.
143                 if cp is not None and i < cpi+l:
144                     shift = i - cpi
145                     cp_shifted = cp[shift:] + cp[:shift]
146                     if p == cp_shifted and i in L[cp_shifted][si]:
147                         L[cp_shifted][si].remove(i)
148                 # Outside of cyclical pattern range or no main pattern set
                    ↪ yet, and pattern encountered.
```

```
149                 if (cp is None or i >= cpi+l) and i in L[p][si]:
150                     cp = p
151                     cpi = i
152     return L
153
154 def purge_interconnecting_subpatterns(Lk, L):
155     ''' Purge subcandidates that connect two consecutive occurrences of a
        ↪ candidate. '''
156     # For each candidate...
157     for c,_ in L.iteritems():
158         # For each cyclical variant of the candidate...
159         for shift in range(-1, -1*len(c), -1):
160             # Get the subcandidate (shifted variant of candidate, minus
                ↪ the last element).
161             subc_shifted = c[shift:] + c[:shift-1]
162
163             # For each sequence that the subcandidate occurs in...
164             for si, s in Lk[subc_shifted].iteritems():
165                 # Remove index if the subcandidate is part of two
                    ↪ consecutive occurrences of the main candidate.
166                 Lk[subc_shifted][si] = set([
167                     i
168                     for i in Lk[subc_shifted][si]
169                         if not (i-shift-len(c) in L[c][si] and
170                                 i-shift        in L[c][si])
171                 ])
172     return Lk
173
174 def purge_subpatterns(Lk, L):
175     ''' Purge non-cyclical subcandidates that are part of the larger
        ↪ candidate. This must happen after cyclical subcandidates are
        ↪ processed, not before. '''
176     # For each candidate...
177     for c,occurrences in L.iteritems():
178         j = len(c)-1 # Length of the candidates in Lk
179         # For each shift variation of the candidate...
180         for shift, subc in [(0, c[0:-1]), (1, c[1:])]:
181
182             # For each sequence of the candidate...
183             for si, s in L[c].iteritems(): # No list comprehension: we
                ↪ overwrite only the relevant sequence indices.
184                 # Only keep the occurrences that are not part an
                    ↪ occurrence of the main candidate.
185                 Lk[subc][si] = set([i for i in Lk[subc][si] if i-shift not
                    ↪ in L[c][si]])
186     return Lk
```

Appendix B

Correlations between attributes and NPS scores

The correlations of the binary *school* and *edlvl* attributes are calculated by transforming their values from false and true to 0 and 1, respectively. This should be taken into account when interpreting the correlation values.

Note that the school names have been removed as these are business sensitive for Topicus.

TABLE B.1

Attribute	correlation
school: #25	0.1108
total_napitime	0.1095
school: #123	0.0978
school: #84	0.0912
edlvl: VMBO BBL	0.0891
school: #116	0.0878
edlvl: VMBO KBL	0.0859
school: #114	0.0815
edlvl: VMBO TL/GL	0.0744
school: #19	0.0705
school: #4	0.0701
school: #49	0.0699
edlvl: VWO	0.0691
school: #63	0.0673
edlvl: VMBO BBL/KBL	0.0667
school: #46	0.0661
school: #21	0.0657
school: #13	0.0652
school: #72	0.0648
school: #20	0.0605
school: #68	0.0564
school: #34	0.0557
school: #74	0.0557
school: #126	0.0551
school: #134	0.0539
school: #94	0.0515
school: #32	0.0513
school: #71	0.0513

Continued on next page

Table B.1 – continued from previous page

Attribute	correlation
school: #127	0.0501
school: #115	0.0498
edlvl: PRO	0.0497
school: #28	0.0496
school: #59	0.0491
NPSAGE	0.0476
total_time	0.0476
school: #99	0.0445
school: #124	0.0444
school: #38	0.0442
school: #14	0.0436
school: #66	0.0436
school: #69	0.0436
school: #122	0.0425
school: #23	0.0424
school: #58	0.0423
school: #56	0.0423
school: #130	0.0413
school: #119	0.0412
school: #75	0.0409
school: #33	0.0408
school: #47	0.0408
edlvl: MYP	0.0401
edlvl: DP	0.0387
school: #35	0.0387
school: #6	0.0385
school: #129	0.0381
school: #76	0.0368
school: #92	0.0351
edlvl: OND	0.0348
school: #16	0.0348
edlvl: MBO-VO	0.0346
school: #45	0.0346
school: #102	0.0345
edlvl: HAVO	0.0339
school: #108	0.0333
edlvl: ESS	0.0314
school: #95	0.0314
school: #62	0.0310
school: #77	0.0298
school: #41	0.0284
school: #48	0.0280
school: #51	0.0271
school: #65	0.0269
school: #110	0.0253
total_apitime	0.0250
school: #11	0.0249
school: #9	0.0247

Continued on next page

Table B.1 – continued from previous page

Attribute	correlation
school: #54	0.0244
school: #82	0.0243
school: #61	0.0227
school: #52	0.0216
school: #101	0.0209
school: #22	0.0208
school: #106	0.0198
school: #31	0.0196
school: #96	0.0195
school: #86	0.0193
school: #40	0.0192
school: #89	0.0192
school: #29	0.0190
school: #55	0.0190
school: #121	0.0188
school: #2	0.0177
school: #17	0.0177
accumulated_repetitions	0.0176
repetitive_patterns	0.0166
school: #78	0.0161
school: #8	0.0159
school: #44	0.0158
school: #88	0.0155
school: #98	0.0155
school: #100	0.0155
school: #117	0.0155
school: #128	0.0155
school: #133	0.0155
school: #125	0.0155
school: #27	0.0151
school: #70	0.0151
school: #79	0.0151
edlvl: VMBO	0.0144
school: #97	0.0141
total_switches	0.0141
school: #10	0.0125
school: #107	0.0125
school: #118	0.0125
school: #120	0.0124
school: #91	0.0122
edlvl: HAVO/TL	0.0120
school: #109	0.0118
school: #105	0.0116
school: #26	0.0116
school: #15	0.0114
school: #42	0.0114
school: #1	0.0113
school: #12	0.0113

Continued on next page

Table B.1 – continued from previous page

Attribute	correlation
school: #43	0.0113
school: #64	0.0113
school: #132	0.0109
school: #39	0.0108
weighted_pattern_iterations	0.0095
school: #81	0.0086
school: #103	0.0080
school: #131	0.0077
school: #104	0.0074
school: #85	0.0065
school: #60	0.0061
school: #90	0.0061
edlvl: HAVO/VWO	0.0060
school: #24	0.0060
school: #67	0.0058
school: #73	0.0057
school: #111	0.0057
school: #3	0.0056
school: #5	0.0051
school: #37	0.0042
school: #87	0.0042
school: #18	0.0037
school: #7	0.0030
school: #50	0.0030
school: #53	0.0026
school: #80	0.0026
school: #83	0.0021
school: #112	0.0021
school: #113	0.0013
school: #93	0.0005
school: #36	0.0003
school: #57	0.0003
school: #30	0.0002

Bibliography

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, pages 3–14. IEEE, 1995.

[2] A. Barreto and C. Antunes. *Finding Cyclic Patterns on Sequential Data*, pages 110–121. Springer International Publishing, Cham, 2014.

[3] V. Bernhardt. *Data analysis for continuous school improvement*. Routledge, 2013.

[4] V. Biou, J. Gibrat, J. Levin, B. Robson, and J. Garnier. Secondary structure prediction: combination of three different methods. *Protein Engineering*, 2(3):185–191, 1988.

[5] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[6] A. Breiter and D. Light. Data for School Improvement: Factors for designing effective information systems to support decision-making in schools Support for Decision-Making. *Educational Technology & Society*, 9(3):206–217, 2006.

[7] C. Campbell and B. Levin. Using data to support educational improvement. *Educational Assessment, Evaluation and Accountability*, 21(1):47–65, 2009.

[8] D. Carlson, G. D. Borman, and M. Robinson. A Multistate District-Level Cluster Randomized Trial of the Impact of Data-Driven Reform on Reading and Mathematics Achievement. *Educational Evaluation and Policy Analysis*, 33(3):378–398, 2011.

[9] O. Chapelle, B. Scholkopf, and A. Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[10] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[11] E. Chen, M. Heritage, and J. Lee. Identifying and Monitoring Students' Learning Needs With Technology. *Journal of Education for Students Placed at Risk (JESPAR)*, 10(3):309–332, 2005.

[12] Y. H. Cho, J. K. Kim, and S. H. Kim. A personalized recommender system based on web usage mining and decision tree induction. *Expert systems with Applications*, 23(3):329–342, 2002.

[13] R. Cooley, B. Mobasher, and J. Srivastava. Web mining: Information and pattern discovery on the world wide web. In *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*, pages 558–567. IEEE, 1997.

[14] R. Cooley, B. Mobasher, and J. Srivastava. Data preparation for mining world wide web browsing patterns. *Knowledge and information systems*, 1(1):5–32, 1999.

[15] A. Cooper, B. Levin, and C. Campbell. The growing (but still limited) importance of evidence in education policy and practice. *Journal of Educational Change*, 10(2-3):159–171, 2009.

[16] A. Datnow and V. Park. *Data-driven leadership*. John Wiley & Sons, 2014.

[17] A. Datnow, V. Park, and P. Wohlstetter. Achieving with Data: How high-performing school systems use data to improve instruction for elementary students. Technical report, Center of Educational Governance. Rossier School of Education. University of Southern California, 2007.

[18] J. G. Dy and C. E. Brodley. Feature selection for unsupervised learning. *Journal of Machine Learning Research*, 5:845–889, Aug. 2004.

[19] P. W. Eklund and A. Hoang. A performance survey of public domain supervised machine learning algorithms. *Australian Journal of Intelligent Information Systems. v9 i1*, pages 1–47, 2002.

[20] I. El Naqa and M. J. Murphy. What is machine learning? In *Machine Learning in Radiation Oncology*, pages 3–11. Springer, 2015.

[21] J. K. Eskildsen and K. Kristensen. The accuracy of the net promoter score under different distributional assumptions. In *Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), 2011 International Conference on*, pages 964–969. IEEE, 2011.

[22] European Parliament and European Council. Regulation on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation), Apr. 2016. http://eur-lex.europa.eu/eli/reg/2016/679/oj.

[23] P. C. Gordon and K. J. Holyoak. Implicit learning and generalization of the "mere exposure" effect. *Journal of Personality and Social Psychology*, 45(3):492, 1983.

[24] D. B. Grisaffe. Questions about the ultimate question: conceptual considerations in evaluating reichheld's net promoter score (nps). *Journal of Consumer Satisfaction, Dissatisfaction and Complaining Behavior*, 20:36, 2007.

[25] J. Han, J. Pei, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *proceedings of the 17th international conference on data engineering*, pages 215–224, 2001.

[26] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining—a general survey and comparison. *ACM sigkdd explorations newsletter*, 2(1):58–64, 2000.

[27] I. Hoogland, K. Schildkamp, F. van der Kleij, M. Heitink, W. Kippers, B. Veldkamp, and A. M. Dijkstra. Prerequisites for data-based decision making in the classroom: Research evidence and practical illustrations. *Teaching and Teacher Education*, 60:377–386, 2016.

[28] Y.-H. Hu and I.-C. Chiang. Mining cyclic patterns with multiple minimum repetition supports. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, volume 3, pages 1545–1549. IEEE, 2011.

[29] T. Hussain, S. Asghar, and N. Masood. Web usage mining: A survey on preprocessing of web log file. In *Information and Emerging Technologies (ICIET), 2010 International Conference on*, pages 1–6. IEEE, 2010.

[30] G. S. Ikemoto and J. A. Marsh. chapter 5: Cutting through the "data-driven" mantra: Different conceptions of data-driven decision making. *Yearbook of the National Society for the Study of Education*, 106(1):105–131, Apr. 2007.

[31] J. B. Jimerson, V. Cho, and J. C. Wayman. Student-involved data use: Teacher practices and considerations for professional learning. *Teaching and Teacher Education*, 60:413–424, Nov. 2016.

[32] Kamp: supersnel internet is basisbehoefte. http://nos.nl/artikel/2146969-kamp-supersnel-internet-is-basisbehoefte.html, 2016. Accessed: 2017-03-05.

[33] T. L. Keiningham, B. Cooil, L. Aksoy, T. W. Andreassen, and J. Weiner. The value of different customer satisfaction and loyalty metrics in predicting customer retention, recommendation, and share-of-wallet. *Managing Service Quality: An International Journal*, 17(4):361–384, 2007.

[34] K. A. Kerr, J. A. Marsh, G. S. Ikemoto, H. Darilek, and H. Barney. Strategies to Promote Data Use for Instructional Improvement: Actions, Outcomes, and Lessons fromThree Urban Districts. *American Journal of Education*, 112(4):496–520, Aug. 2006.

[35] S. Konstantopoulos, S. R. Miller, and A. van der Ploeg. The Impact of Indiana's System of Interim Assessments on Mathematics and Reading Achievement. *Educational Evaluation and Policy Analysis*, 35(4):481–499, 2013.

[36] K. Kristensen and J. Eskildsen. Is the NPS a trustworthy performance measure? *The TQM Journal*, 26(2):202–214, Mar. 2014.

[37] M. W. Krol, D. Boer, D. M. Delnoij, and J. J. Rademakers. The net promoter score–an asset to patient experience surveys? *Health Expectations*, 18(6):3099–3109, 2015.

[38] M. K. Lai, S. McNaughton, H. Timperley, and S. Hsiao. Sustaining continued acceleration in reading comprehension achievement following an intervention. *Educational Assessment, Evaluation and Accountability*, 21(1):81–100, 2009.

[39] M. K. Lai and K. Schildkamp. *Data-based Decision Making: An Overview*, chapter 2, pages 9–21. Springer Netherlands, Dordrecht, 2013.

[40] H. Lauckhart-Hassig. "Wij willen geen tevreden klanten, wij willen fans". https://www.som.today/nieuws/wij-willen-geen-tevreden-klanten-wij-willen-fans, 2015. Accessed: 2017-06-19.

[41] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine learning*, 40(3):203–228, 2000.

[42] A. Lohuis. Statistisch is 20 oktober de leukste dag van het jaar! https://www.som.today/nieuws/dag-van-de-statistiek, 2016. Accessed: 2017-06-19.

[43] M. Luo. Structural Equation Modeling for High School Principals' Data-Driven Decision Making: An Analysis of Information Use Environments. *Educational Administration Quarterly*, 44(5):603–634, 2008.

[44] P. C. Mandal. Net promoter score: a conceptual analysis. *International Journal of Management Concepts and Philosophy*, 8(4):209–219, 2014.

[45] S. McNaughton, M. K. Lai, and S. Hsiao. Testing the effectiveness of an intervention model based on data use: a replication series across clusters of schools. *School Effectiveness and School Improvement*, 23(2):203–228, 2012.

[46] P. Mitra, C. Murthy, and S. K. Pal. Unsupervised feature selection using feature similarity. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI*, 24(3):301–312, 2002.

[47] B. Mobasher, R. Cooley, and J. Srivastava. Automatic personalization based on web usage mining. *Communications of the ACM*, 43(8):142–151, 2000.

[48] PwC. Pulse Survey: US Companies ramping up General Data Protection Regulation (GDPR) budgets, Dec. 2016. https://www.pwc.com/us/en/increasing-it-effectiveness/publications/assets/pwc-gdpr-series-pulse-survey.pdf.

[49] F. F. Reichheld. The One Number You Need to Grow. *Harvard Business Review*, 81(12), 2003.

[50] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[51] K. Schildkamp, L. Karbautzki, and J. Vanhoof. Exploring data use practices around Europe: Identifying enablers and barriers. *Studies in Educational Evaluation*, 42:15–24, 2014.

[52] K. Schildkamp and W. Kuiper. Data-informed curriculum reform: Which data, what purposes, and promoting and hindering factors. *Teaching and Teacher Education*, 26(3):482–496, 2010.

[53] K. Schildkamp, M. K. Lai, and L. Earl. *Data-based decision making in education: Challenges and opportunities*, volume 17. Springer Science & Business Media, 2012.

[54] K. Schildkamp, C. Poortman, H. Luyten, and J. Ebbeler. Factors promoting and hindering data-based decision making in schools. *School Effectiveness and School Improvement*, pages 1–17, 2016.

[55] M. Shah. Impact of Management Information Systems (MIS) on School Administration: What the Literature Says. *Procedia - Social and Behavioral Sciences*, 116:2799–2804, 2014.

[56] L. Song, A. Smola, A. Gretton, K. Borgwardt, and J. Bedo. Supervised feature selection via dependence estimation. In *Proceedings of the 24th international conference on Machine learning*, pages 823–830. ACM, 2007.

[57] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology*, pages 1–17. Springer, 1996.

[58] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations Newsletter*, 1(2):12–23, Jan. 2000.

[59] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society. Series B (Methodological)*, pages 111–147, 1974.

[60] J. A. Supovitz and V. Klein. Mapping a Course for Improved Student Learning: How Innovative Schools Use Student Performance Data to Guide Improvement. *Consortium for Policy Research in Education*, Nov. 2003.

[61] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[62] I. H. Toroslu. Repetition support and mining cyclic patterns. *Expert Systems with Applications*, 25(3):303–311, 2003.

[63] J. C. Wayman, V. Cho, J. B. Jimerson, and D. D. Spikes. District-Wide Effects on Data Use in the Classroom. *Education Policy Analysis Archives*, 20(25):1–28, 2012.

[64] J. C. Wayman, V. Cho, and M. T. Johnston. *The data-informed district: A district-wide evaluation of data use in the Natrona County School District*. PhD thesis, Austin: The University of Texas, Aug. 2007.

[65] J. C. Wayman, S. Shaw, and V. Cho. *Second-Year Results From an Efficacy Study of the Acuity Data System*. PhD thesis, Austin: The University of Texas, 2011.

[66] D. H. Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[67] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1-2):31–60, 2001.

[68] K. Zhang, M. Hutter, and H. Jin. A new local distance-based outlier detection approach for scattered real-world data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 813–822. Springer, 2009.