Educational Data Mining for Analysis of Students' Solutions

Karel Vaculík, Leona Nezvalová, and Luboš Popelínský

KD Lab, FI MU Brno
{xvaculi4,popel}@fi.muni.cz, xnezva36@mail.muni.cz

Abstract. We introduce a novel method for analysis of logical proofs constructed by undergraduate students that employs sequence mining for manipulation with temporal information about all actions that a student performed, and also graph mining for finding frequent subgraphs on different levels of generalisation. We show that this representation allows one to find interesting subgroups of similar solutions and also to detect outlying solutions. Specifically, distribution of errors is not independent of behavioural patterns and we are able to find clusters of erroneous solutions. We also observed significant dependence between time duration and an appearance of the most serious error.

Keywords: educational data mining, logical proofs, clustering, outlier detection, sequence mining.

1 Introduction

Teaching constructive tasks, i.e. tasks that a student has to build in several steps, like tasks in descriptive geometry or logical and math proofs, requires advanced evaluation techniques. For example, in the case of resolution proofs in logic, see examples in Fig. 1, it is not sufficient to assign the mark based only on the conclusion that the student reached. To evaluate a student solution properly, a teacher needs not only to check the final result of a solution (the set of clauses is or is not contradictory) but also to analyse the sequence of steps that a student performed, with respect to correctness of each step and with respect to correctness of that sequence. We show that novel machine learning methods, such as graph mining and sequence mining, can be very helpful in that situation because of their capability to process structural and temporal information. Specifically, graph mining methods work with data represented as graphs, in our case one graph for each instance, and take into account the structural information of the graphs. An overview of graph mining methods can be found in [5]. Sequence mining is another topic of data mining oriented to structured data. In comparison to graph mining, data are arranged in sequences, usually ordered by time, and they are assumed to be discrete. More information on sequence mining can be found in [6].

Up to our knowledge, there is no work on analysis of student solutions of logical proofs by means of graph mining. Definitely, solving logical proofs, especially

G. Agre et al. (Eds.): AIMSA 2014, LNAI 8722, pp. 150–161, 2014.

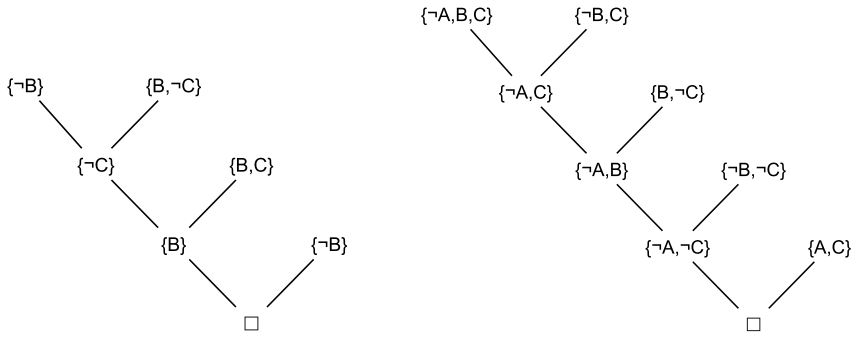

Fig. 1. An example of a correct and an incorrect resolution proof

by means of resolution principle, is one of basic graph-based models of problem solving in logic. In problem-solving processes, graph mining has been used in [21] for mining concept maps, i.e. structures that model knowledge and behaviour patterns of a student, for finding commonly observed subconcept structures. The combination of multivariate pattern analysis and hidden Markov models for the discovery of major phases that students go through in solving complex problems in algebra is introduced in [1]. Markov decision processes for generating hints to students in logical proof tutoring from historical data has been solved in [2,3,18]. Authors of [11] analyzed students' ordinary handwritten coursework with a digital pen by means of sequence mining techniques to identify patterns of actions that are more frequently exhibited by either good- or poor-performing students. In [14] sequential pattern mining was used for analysis of activity around an interactive tabletop and finding frequent sequences that differentiate high achieving from low achieving groups.

In our previous work [19] we presented a method for analysis of students' solutions of resolution proofs that employed graph mining. It used frequent subgraph mining algorithm Sleuth [23] for finding frequently occurring subgraphs. These subgraphs were then generalized and used as new features for representing the data. In [20] we extended this procedure and left out the frequent subgraph mining algorithm. More information can be found in Section 5.1. Although the method displayed very high accuracy in classification, it was unable to exploit temporal information about how the students solve the task, neither was appropriate for finding outliers – anomalous solutions.

In this paper we propose a novel method that is much more robust and is actually independent of a particular student task. In addition, it processes also information about the sequence of steps that a student performed, like adding/deleting a node or edge in the proof and also about text (i.e. a formula) modification. It also exploits information about the time a particular operation was performed and uses temporal information for finding outlying solutions. We use three different feature extraction methods and show that by using those new temporal and structural features we are able to find clusters of erroneous

solutions. We also show that there is a significant dependence between time duration and appearance of errors in a student solution. Moreover, by means of class outlier detection we are able to find solutions that are anomalous and for that reason difficult to detect automatically.

It has to be stressed at the very beginning that we do not aim at building a tool for automatic classification of students' solutions. The goal that we cope with here is different: to find typical and abnormal patterns in data that are strongly correlated with wrong solutions and with particular errors. We show that with the proposed feature extraction methods we are able to find students that are at risk of producing wrong solutions. We are also able to detect outliers, i.e. solutions that are hard to detect as incorrect.

The paper is structured as follows. Section 2 contains a description of data and two representations of frequent subsequences (episodes). In Section 3 we use clustering on those two representations and show our first results about the distribution of errors in the clustered data. In Section 4 we analyse explicit time stamps and show that there is a significant dependence between time intervals of particular operations (as well as total time duration) and an appearance of the most serious error. Outlier detection in resolution proofs is solved in Section 5. A discussion and conclusion are presented in the last two sections.

2 Data and Data Pre-Processing

The data, 873 solutions altogether, was obtained in the course on Introduction to logic. Via a web-based tool, each of the 351 students solved at least three tasks randomly chosen from 19 exercises. The data set contained the resolution tree and also dynamics of the solutions, i.e. all the actions performed together with temporal information. Among these 873 different students' solutions of resolution proofs in propositional calculus, 101 of them were classified as incorrect and 772 as correct.

The most serious error in resolution is resolving on two literals. In this text we denote this error as E3. Other common errors in resolution proofs are the following: repetition of the same literal in the clause, a literal is missing in the resolved clause, resolving on the same literals (not on one positive and one negative), resolving within one clause, resolved literal is not removed, the clause is incorrectly copied, switching the order of literals in the clause, proof is not finished, intentional negation of literals in a clause. Information about the error that appeared in the logical proof is also part of the data.

All actions that a student performed, such as adding/deleting a node, drawing/removing an edge, writing/deleting a text into a node, were saved into a database. The collected data contains also a timestamp for each action performed by a student. The timestamps also allow us to get the order of actions and use techniques for sequence mining as discussed in the following sections.

2.1 Types of Sequences

For processing temporal information, we used two sequence representations of the data. Each resolution proof can be represented as a sequence in either

representation. The first one is composed of two events: addition of a node (clause) into a proof and addition of an edge. An example of such sequence is CCCCCEEEE, where C denotes node addition and E denotes edge addition. The second representation uses the same events as the first one, but it also contains events of text modification[1]. An example of a sequence in the second representation is CTCTCTCTCTEEEE, where elements C, T and E denote node addition, text modification and edge addition, respectively. From now on, we will use CE and CET abbreviations for the representations.

2.2 Frequent Subsequences

As sequences cannot be processed by commonly used machine learning algorithms, such as classification, clustering or outlier detection algorithms, we need to transform the data. Simple and common practice is to use subsequences as features [6]. We considered only subsequences consisting of elements that are consecutive in original sequences, i.e. without *gaps*, because subsequences with gaps are not descriptive in case of our sequences. Formally, we say that a sequence $\alpha = \alpha_1\alpha_2...\alpha_n$ is a *subsequence* of another sequence $\beta = \beta_1\beta_2...\beta_m$ with $m \geq n$, if there exists an integer $1 \leq k \leq m - n + 1$ such that $\alpha_j = \beta_{k+j-1}$ for each $1 \leq j \leq n$.

To find all potentially useful subsequences, we employed cSpade [22] algorithm for frequent sequence mining. For a given value $min\_support \in [0, 1]$, this algorithm finds all subsequences whose $support \geq min\_support$. Support of a subsequence α is a fraction of input sequences which contain α as a subsequence. Specifically, we set $min\_support = 0.1$ to get only subsequences that occur at least in 10% of all input sequences. We obtained 121 frequent subsequences from sequences in CE representation, and 242 subsequences in case of CET representation. Each frequent subsequence is used as a new feature with value equal to 1 if the subsequence appears in the given sequence, and 0 otherwise.

3 Sequence Clustering

3.1 Method

Having the resolution proofs represented by features constructed from the two representations of sequences, we performed clustering on features of each representation. For the purpose of clustering, we set the values of features as follows: if the sequence contains the corresponding subsequence, we set the value as the squared length of the subsequence. Otherwise we set the value to 0. The rationale for this is that long subsequences should be more explanatory so they carry more weight.

On this representation of data we performed cluster analysis using the AGNES (AGglomerative NESting) hierarchical clustering and PAM (Partitioning Around

[1] We also tried sequences with node- and edge-deletion events, but these events did not affect the results due to their sparse occurrence.

Table 1. Internal evaluation of CE-2, CE-8, CET-2, and CET-8 clusterings by Dunn index (DI) and avg. silhouette width (SIL)

	AGNES		PAM	
clustering	DI	SIL	DI	SIL
CE-2	0.14	0.60	0.01	0.60
CE-8	0.35	0.78	0.05	0.76
CET-2	0.09	0.53	0.14	0.57
CET-8	0.16	0.64	0.02	0.61

Medoids) algorithms, description of both algorithms can be found in [13]. In case of AGNES we used average linkage method and for both algorithms we used Manhattan distance metric.

To evaluate different numbers of clusters, i.e. different cuts in AGNES dendrogram and different number of PAM medoids, we utilized two metrics, Dunn index [7] and average silhouette width [17]. Higher value of either metric indicates better clustering. For each algorithm we performed clustering with different numbers of clusters, specifically we used all integers from the interval [2, 12]. To select the most appropriate number of clusters, we ranked the values of the two metrics for each algorithm. The higher the value of a metric, the lower the rank. Then we calculated the total rank by summing over all four ranks. For the CE representation, the value of total rank decreased with larger number of clusters, but from 8 clusters onward, the change was not substantial, so we selected 8 clusters as sufficient. Specifically, the values of total rank from 2 to 12 clusters were following: 60, 56, 50, 43, 46, 31, 25, 31, 24, 21, and 21. In case of CET representation, the lowest value of total rank was calculated for 8 clusters. Results for both cases are depicted in Table 1, clustering is encoded as [*sequence representation*]–[*# of clusters*]. We also included results for CE-2 and CET-2, cases with the smallest number of clusters, for comparison. These two clustering divisions are also considered in statistical tests described later. In Table 1 we can see that the Dunn index was generally quite low, especially for PAM algorithm.

For each cluster we also looked for the most representative sequence. In case of PAM algorithm, it was enough to take the medoids. However, hierarchical clustering algorithms does not use medoids, so we designed and used the following procedure. First, we computed the average value for each feature on the set of sequences from a specific cluster. The features were the same as for clustering. Then we used the same distance metric, Manhattan distance, to find a sequence most similar to the average.

3.2 Clustering Results

Resulting representative sequences for the above mentioned clusterings are shown in Table 2. By comparing both algorithms, we can see that they share a lot of similar representatives. Simple division of the proofs can be seen in case of the

CE-2 clustering for both algorithms, see the first two rows of the table. The first cluster groups proofs solved in a step-by-step fashion, where a step means an application of the resolution rule and relevant edges are added immediately after clauses (nodes) in each step. The second cluster groups proofs solved in such a way that the nodes are added first and all the edges afterwards.

Table 2. Cluster representatives of CE-2, CE-8, CET-2, and CET-8 clusterings

Clustering	AGNES	PAM
CE-2	CCCEECCCCEECEECEE	CCCEECCEECCCCEEEE
	CCCCCCCCCEEEEEEEE	CCCCCCCCCEEEEEEEE
CE-8	CCCEECCEECCEECCEE	CCCEECCEECCEECCEE
	CCCCEECEECCCCEEEE	CCCCCEEEECCCCEEEE
	CCCCCCCEEEEEE	CCCCCCCEEEEEE
	CCCEECCEECCEE	CCCEECCEECCEE
	CCCCCCCCCEEEEEEEE	CCCCCCCCCCCCEEEEEEEEE
	CCCEECCCECEEE	CCCEECCEECCEECCEECCEE
	CCCCCEEEE	CCCCCEEEE
	CCCEECCEE	CCCCCCCCCEEEEEEEE
CET-2	CTTCTCTCTCTCTCTCTEEEEEECTEE	CTCTCTCTEECEETCTCEECTTCTEE
	CCCCCCTTTTTTCTEEEEEE	CTCTCTCTCTCTCTCTCTEEEEEEE
CET-8	CTCTCTCTCTCTCTCTCTEEEEEEEE	CTCTCTCTCTCTCTCTCTEEEEEEE
	CCCCCCTTTTTTCCCTTTEEEEEEEE	CCCCCTTTTTCCCTTTCTEEEEEEE
	CCCCCCCTTTTTTTEEEEEE	CCCCCCCTTTTTTTEEEEEE
	CTCTCTCTCTEEEE	CTCTCTCTCTEEEE
	CTTCTCTEETCTCEETCTCTEECTCTEE	CTTCTCTEECTCEETCTECTE
	CTCTCTEECTCTEE	CTCTTCTCTCTCTCTEEEEEE
	CCCCTTEETTTCCCEETTEETCTCTEET	CTCTCTCTEECEETCTCEECTTCTEE
	CTCTCTEE	CCCCCCCCTTTTTTTTTTTCTEEEEEEE

On the other hand, CET-8 clustering of AGNES, for example, is slightly more difficult to analyse than CE-2. Nevertheless, several distinctive characteristics can be seen from the representatives. In the first half of sequences, all edges are added last, and in the second half, they are added approximately after each step. Similar phenomenon can be observed for the T events with respect to the C events – in some cases, most of the C events are added first, and in some cases, these events are alternating with the T events.

In addition, the first and the fifth clusters from CET-8 of AGNES contained most of the instances, precisely 696 out of 873 instances. This means that for most of the proofs, it should hold that they are not short, node and text addition is alternating, and there is no prevailing way of edge addition. The last cluster in the table, represented by the CTCTCTEE sequence, was also interesting, as its 12 out of 16 instances contained E3 error. Let us remind that there were only 53 instances with E3 error in total. It means that almost 1/4 of all erroneous solutions appeared in that cluster. We can exploit the information from the representative sequence for detecting potential error and maybe warn a student or offer them a hint even before finishing their solution.

3.3 Analysis of Sequence Clusters

As we want to find whether some behavioural patterns of students are connected with errors in solutions, we analysed our sequential data with respect to solution errors. From the set of common errors, we considered the most serious error – the error of resolving on two or more literals at the same time, i.e. the E3 error. This error is the most common and also the only one with occurrence rate greater than 5%. We performed Fisher's exact test [9] to compare the occurrence of the E3 error and each of the four sequence clusterings, taken for both clustering algorithms. Considering the 5% significance level, we can conclude from the test results that the data provides convincing evidence that the occurrence of E3 error is not independent of any of those four clusterings of any of the two algorithms. Moreover, by analysing clusters as in the previous section, we can discover more useful patterns, such that for CET-8 there was a cluster with majority of wrong solutions. Such information may help early detect students that are at risk of performing the E3 error.

4 Time Duration

4.1 Time Features

We also analysed explicit time information. For each resolution proof, we computed several time characteristics. They are expressed in seconds and can be divided into two groups. The first group reflects time duration between simple events such as node addition and deletion, edge addition and deletion, and text modification. As there was no time limit for solving a resolution proof, students could keep the web tool opened for a long time without actually solving the exercise. Therefore we marked 5% of the longest durations as outliers and replaced each such outlier by a mean value of non-outlying durations of the corresponding proof. Specifically, the threshold for outliers was 15 seconds in this case. From these new values of durations we calculated the mean and the maximum value[2] and also the sum of durations, which represents the total duration of proof solving.

The second group was derived from durations of resolution rule application. In particular, for each application of resolution rule, we calculated time interval between text modification of parents and text modification of the corresponding resolvent. We omitted cases in which a parent was modified after its resolvent and, again, 5% of the longest durations were replaced by mean values. In this case, the outlier threshold was 60 seconds exactly. At the end, the maximum, the minimum and the mean values were calculated as well as the sum of the values.

4.2 Analysis of Time Duration

Then we investigated the relation between errors and time duration of a resolution proof solution. Although the duration is a continuous variable, we did

[2] We omitted the minimum value as it was almost always equal to zero because some actions were saved simultaneously.

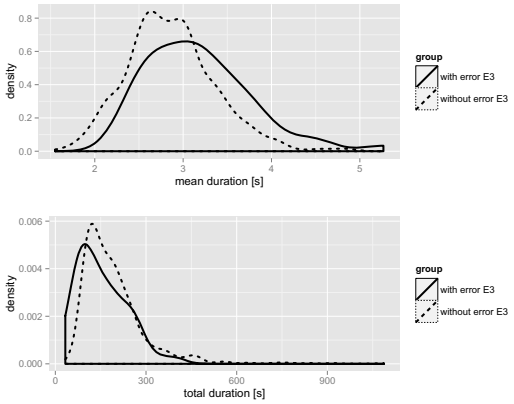

Fig. 2. Density plots for mean duration (top) and total duration (bottom) with respect to E3 error

not use the Kolmogorov-Smirnov test because the data contains mostly integer values with many ties between groups. Instead, we performed a permutation test which is based on a comparison of means. More precisely, we used two-sample exact permutation test estimated by Monte Carlo [8] on data consisting of duration summary statistics of simple events. For each summary statistic, this test compared distributions corresponding to erroneous and correct solutions for a given type of error.

Significant results were obtained only for the E3 error and for mean values of durations and sums of durations, i.e. total durations of resolution proofs. The significance level was 0.05. In both cases, the test even yielded the p-value of 0.002. This means that the distribution of mean values of durations for solutions without the E3 error is different from the distribution for solutions with the error. The same holds for the two distributions of sums of durations. Distributions of mean values can be found in the density plot at the top of Fig. 2. At the bottom of the same figure, the density plot of the distributions of total durations is depicted. We can conclude that solutions built in short time but on the basis of slow actions are more likely to contain the E3 error.

5 Outlier Detection in Resolution Graphs

5.1 Generalized Subgraphs

As in the case of sequences, graph features may be constructed from graph substructures found by an algorithm for frequent subgraph mining [5]. The task of frequent subgraph mining is very similar to the frequent sequence mining, but now the frequently occurring subgraphs are looked up in a set of graphs. Next, boolean features are used and the value of a feature depends on whether the corresponding substructure occurs in the given instance or not. Because the

node labels of incorrect solutions have very small support and frequent subgraph mining can be inefficient for small values of minimum support, we created a new method for finding interesting patterns. The other reason was that similar subgraphs differed only in alphabet letters and in the order of literals in text labels or in the order of parent nodes. The main idea is to generalize and unify subgraphs consisting of two parent nodes and a resolvent. Such generalized subgraphs may be expressed shortly in the following form: $parent1; parent2 - - > resolvent$. An example of generalized subgraph is $\{\neg Y, Z\}; \{\neg Y, \neg Z\} - - > \{\neg Y\}$, where Y, Z are variables that can match any propositional letter. We also created a new, higher-level, generalization [20], which further generalizes the patterns found earlier. As a result, new patterns in *added*; *dropped* form are created. The *added* component simply denotes literals which were added erroneously to the resolvent and the *dropped* component denotes literals from parents which participated in the resolution process. Continuing with the example, the higher-level pattern will be $\{\}; \{\neg Z, Z\}$. The detailed description can be found in [20].

5.2 Method

The data we process has been labeled. We focused only on the E3 error as it was the most common and the most serious error. Specifically, there were two values of the class attribute, corresponding to the occurrence or nonoccurrence of the error. Unlike in common outlier detection, where we look for outliers that differ from the rest of "normal" data, we need to exploit information about a class. That is why we used weka-peka [16] that looks for class outliers [15] using Random Forests (RF) [4]. It extends the outlier detection method in RF implemented in Weka [10] that actually works for classical settings – normal vs. anomalous data to manage class information. The main idea lies in different computation of the proximity matrix that exploits also information about a class label [16].

5.3 Results

When analysing the strongest outliers that weka-peka discovered, we found three groups according to the outlier score. The two most outlying examples, with outlier factor overcoming 130, significantly differ from the others. The second cluster consists of four examples with the outlier score between 50 and 100, and the last group is comprised of instances with the lowest score of 15.91. Analysis of individual outliers let us draw several conclusions. Two most outlying instances contain one specific pattern, *looping*. This pattern represents the ellipsis in a resolution tree, which is used for tree termination if the tree cannot lead to a refutation. Both instances contain this pattern, but neither of them contains the pattern of correct usage of the resolution rule. The important thing is that these two instances contain neither the E3 error nor other errors. This shows that it is not sufficient to find all errors and check the termination of proofs, but we should also check whether the student performed at least few steps by using the resolution rule. Otherwise we are not able to evaluate the student's

skills. Instances with the outlier score less than 100 are less different from other instances.

6 Discussion

The other method that we used for outlier detection was CODB [12]. However, when compared with weka-peka, CODB returned much worse results mainly because of using density and distances (to nearest neighbours and to all members of the class) for outlier detection. Such poor results may be caused by the fact that those metrics are too rough for our task. Moreover, it is much more difficult to obtain a comprehensive explanation of why a particular solution is an outlier.

As we stressed in the introduction, this method has not been developed for recognition of correct or incorrect solutions. However, to verify that the feature construction is appropriate, we also learned various classifiers of that kind. Best result was achieved by SMO (SVM implementation in Weka) – 96.9% accuracy[3]. Similar results were obtained when only the higher level of subgraph generalization was used, again with SMO.

7 Conclusion

We proposed three different feature extraction methods for temporal as well as structural information in logical proof solutions. We showed that by using those new features we are able to find clusters of erroneous solutions, dependence between time duration and errors in a student solution, and also solutions that are anomalous and for that reason difficult to detect automatically.

We believe that this method can be used even by non-expert in machine learning. There are only few parameters to be set: the minimum support for frequent subsequences and also frequent subgraphs, maximum number of clusters in sequence clustering, and the length of delay that indicates outliers in time duration analysis.

This method is general. It can be used also for other logical proofs, such as tableaux proofs, and for any construction tasks that are based on graphs. The main idea, frequent subgraph mining and sequential frequent mining, can be used without a change. However, a new way of subgraph pattern generalization must be developed for a specific construction task.

There is a big potential of the results displayed above in practical education. By using the detection and the analysis of clusters with higher frequency of erroneous solutions a teacher can detect potential reasons of errors and find shortcomings in tutoring. Even in the process of solving the task, it is possible to detect behavioural patterns before completing the proof and warn the student. Outlier detection particularly helps to discover picturesque students' solutions and fix drawbacks in automatic evaluation.

[3] The rate of correct predictions made by the model.

In future we will use this method for resolution in predicate logic and also for tableaux proofs. This may require an extraction of different structural features (frequent subgraphs) but the rest can be used without changes. A challenge is also the use of inductive logic programming that can better cope with domain knowledge.

Acknowledgments. This work has been supported by Faculty of Informatics, Masaryk University and the grant CZ.1.07/2.2.00/28.0209 Computer-aided-teaching for computational and constructional exercises. We would like to thank Alex Popa and members of KD lab for their help.

References

1. Anderson, J.R.: Discovering the Structure of Mathematical Problem Solving. In: Proceedings of EDM (2013)
2. Barnes, T., Stamper, J.: Toward automatic hint generation for logic proof tutoring using historical student data. In: Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (eds.) ITS 2008. LNCS, vol. 5091, pp. 373–382. Springer, Heidelberg (2008)
3. Barnes, T., Stamper, J.: Automatic Hint Generation for Logic Proof Tutoring Using Historical Data. Educational Technology and Society 13(1), 3–12 (2010)
4. Breiman, L.: Random Forests. Machine Learning 45(1), 5–32 (2001)
5. Cook, D.J., Holder, L.B.: Mining graph data. Wiley-Interscience, Hoboken (2007)
6. Dong, G., Pei, J.: Sequence data mining. Springer, New York (2007)
7. Dunn, J.C.: A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. Journal of Cybernetics 3(3), 32–57 (1973)
8. Fay, M.P., Shaw, P.A.: Exact and Asymptotic Weighted Logrank Tests for Interval Censored Data: The interval R package. Journal of Statistical Software 36(2), 1–34 (2010), http://www.jstatsoft.org/v36/i02/
9. Fisher, R.A.: Statistical Methods for Research Workers. Oliver & Boyd (1970)
10. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explor. Newsl. 11(1), 10–18 (2009)
11. Herold, J., Zundal, A., Stahovich, T.F.: Mining Meaningful Patterns from Students' Handwritten Coursework. In: Proceedings of EDM (2013)
12. Hewahi, N., Saad, M.: Class outliers mining: Distance-based approach. International Journal of Intelligent Technology 2(1), 55–68 (2007)
13. Kaufman, L., Rousseeuw, P.J.: Finding groups in data: an introduction to cluster analysis. Wiley, Hoboken (2005)
14. Martinez, R., Yacef, K., Kay, J., Al-Qaraghuli, A., Kharrufa, A.: Analysing frequent sequential patterns of collaborative learning activity around an interactive tabletop. In: Proceedings of EDM (2011)
15. Papadimitriou, S., Faloutsos, C.: Cross-outlier detection. In: Hadzilacos, T., Manolopoulos, Y., Roddick, J., Theodoridis, Y. (eds.) SSTD 2003. LNCS, vol. 2750, pp. 199–213. Springer, Heidelberg (2003)
16. Pekarčíková, Z.: Supervised outlier detection. Master's thesis (in Czech). Masaryk University (2013), http://is.muni.cz/th/207719/fi_m/diplomova_praca_pekarcikova.pdf

17. Rousseeuw, P.J.: Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. Computational and Applied Mathematics 20, 53–65 (1987)
18. Stamper, J.C., Eagle, M., Barnes, T., Croy, M.J.: Experimental Evaluation of Automatic Hint Generation for a Logic Tutor. I. J. Artificial Intelligence in Education 22(1-2), 3–17 (2013)
19. Vaculík, K., Popelínský, L.: Graph Mining for Automatic Classification of Logical Proofs. In: CSEDU (2014)
20. Vaculík, K., Popelínský, L., Nezvalová, L.: Graph mining and outlier detection meet logic proof tutoring. Submitted to Graph-based Educational Datamining 2014 (2014)
21. Yoo, J.S., Cho, M.H.: Mining Concept Maps to Understand University Students' Learning. In: Proceedings of EDM (2012)
22. Zaki, M.J.: Sequences Mining in Categorical Domains: Incorporating Constraints. In: 9th ACM CIKM, pp. 422–429 (2000)
23. Zaki, M.J.: Efficiently Mining Frequent Embedded Unordered Trees. Fundamenta Informaticae 66(1-2), 33–52 (2005)