Domain-Independent Proximity Measures in Intelligent Tutoring Systems

Bassam Mokbel
CITEC Center of Excellence
Bielefeld, Germany
bmokbel@techfak.uni-bielefeld.de

Sebastian Gross
TU Clausthal
Clausthal-Zellerfeld, Germany
sebastian.gross@tu-clausthal.de

Benjamin Paassen
CITEC Center of Excellence
Bielefeld, Germany
bpaassen@techfak.uni-bielefeld.de

Niels Pinkwart
TU Clausthal
Clausthal-Zellerfeld, Germany
niels.pinkwart@tu-clausthal.de

Barbara Hammer
CITEC Center of Excellence
Bielefeld, Germany
bhammer@techfak.uni-bielefeld.de

ABSTRACT

Intelligent tutoring systems (ITSs) typically analyze student solutions to provide feedback to students for a given learning task. Machine learning (ML) tools can help to reduce the necessary effort of tailoring ITSs to a specific task or domain. For example, training a classification model can facilitate feedback provision by revealing discriminative characteristics in the solutions. In many ML methods, the notion of proximity in the investigated data plays an important role, e.g. to evaluate classification boundaries. For this purpose, solutions need to be represented in an appropriate form, so their (dis-)similarity can be calculated. We discuss options for domain- and task-independent proximity measures in the context of ITSs, which are based on the ample premise that solutions can be represented as formal graphs. We propose to identify and match meaningful contextual components in the solutions, and present first evaluation results for artificial as well as real student solutions.

1. INTRODUCTION

Intelligent tutoring usually relies on knowledge about the domain being taught and adaptation of pedagogical strategies regarding learners' individual needs. Therefore, ITSs typically use formalized domain knowledge to provide intelligent one-on-one computer-based support to students. Often, even the specific learning task must be modeled explicitly, which requires significant effort by human experts. Hence, among several directions of research, one major idea regards ITSs which are adaptive, based on examples and using ML or data mining techniques, rather than an explicit modeling of the background information. This direction also opens a way towards the application of ITSs in domains where a formalization of the underlying knowledge is hardly possible, such as ill-defined domains in which there may exist a wide variety of strategies for solving a given task [4]. Example-based learning has shown to be an effective tutoring approach in supporting learning, see [1], which can also be applied without formalizing domain knowledge. In [2], the authors propose ways how feedback provision can be realized in example-based learning environments.

Acknowledgments: This work was supported by the German Research Foundation (DFG) under the grant "FIT - Learning Feedback in Intelligent Tutoring Systems." (PI 767/6 and HA 2719/6).

Assuming that effective feedback-strategies can be established based on appropriate examples, let us restrict to a scenario where a set $\widetilde{X}$ of examples $\tilde{\mathbf{x}}^j$ is explicitly given, and a student solution $\hat{\mathbf{x}}^i$ from the set $\widehat{X}$ needs to be associated to the most suited example. We further assume that examples are themselves solutions (or are represented in the same form) and we can process them in the same manner. Let $d(\mathbf{x}^i, \mathbf{x}^j)$ be a meaningful proximity measure which indicates the dissimilarity of any two solutions $\mathbf{x}^i, \mathbf{x}^j \in X = \widehat{X} \cup \widetilde{X}$ by a positive value. Then, a student solution $\hat{\mathbf{x}}^i \in \widehat{X}$ can simply be associated to the most similar (and thus most suited) example by choosing: $\arg\min_j \; d(\hat{\mathbf{x}}^i, \tilde{\mathbf{x}}^j),\; j \in \{1, \ldots, |\widetilde{X}|\}$. In the following, we will present general approaches to calculate this dissimilarity, if solutions are represented as formal graphs with annotations. The overall calculation is not tailored to a specific learning task or domain, if general data representations are used. To explain the details of the approach, and show first experimental results, we will refer to our example application scenario: an ITS to support programming courses for the Java language.

2. THE PROXIMITY OF SOLUTIONS

The basic requirement is that solutions can be represented as graphs, with different kinds of meta information annotated on the nodes and edges. Each node represents a (syntactic or semantic) element of the solution, and edges establish relationships between them. Solutions $\mathbf{x}^i \in X$ are thus graphs $G_i = (V_i, E_i)$. Considering a very simple annotation, we require that all vertices are attributed to a certain *node type*, a symbol from the finite alphabet $\Sigma = \{l_1, \ldots, l_T\}$. In our application scenario, we consider *syntax trees* of Java programs, where the nodes represent syntactic elements and the corresponding node types indicate their functionality, e.g. the declaration of a variable, a logical expression, a variable assignment, etc. Additionally, a parser adds edges denoting relationships between the syntactic elements, like the call to a function, the usage of a variable, etc.

Using classical data mining approaches, there are several ways to define a proximity measure for these annotated graphs. For example, to represent a solution by a *feature vector*, one can extract frequencies of syntax elements within a solution vs. all solutions to gain a representation analogous to popular *tf-idf weights* [5]. By $d_{\texttt{tfidf}}(\mathbf{x}^i, \mathbf{x}^j)$ we refer to the

Euclidean distance between the tf-idf weight vectors of two Java syntax graphs. This kind of feature encoding captures statistics about the symbols, however their relations are not considered. Measures for *symbolic sequences* respect the ordering of symbols, e.g. alignment measures [3]. For this purpose, we can encode the syntax trees as sequences $\mathbf{s}^i \in \Sigma^*$ by visiting vertices in a depth-first-search order, concatenating their node types. This ordering corresponds to the original sequence of statements in the Java source code. Let $d_{\texttt{align}}(\mathbf{x}^i, \mathbf{x}^j)$ be the dissimilarity score of a Smith-Waterman *local alignment* of the respective sequences $\mathbf{s}^i$ and $\mathbf{s}^j$, with fixed costs for edit operations, see [3].

Both, tf-idf weights and alignment do not explicitly consider contextual relationships within the syntax. To take these structural characteristics into account, we propose to identify densely connected subgraphs and calculate a piecewise proximity between parts of the solutions. Our basic assumption is that solutions consist of semantic building blocks, in which the elements are in close relation to each other, and are less related to other elements. For example, in a computer program the variables and expressions within a for-loop are likely to be connected to each other, but would be less connected to elements outside the loop. In the following, we call these dense subgraphs *fragments*. To identify such fragments, we use the graph clustering algorithm *Spectral Clustering* (SC) [6]. SC separates the graph into a fixed number of subgraphs, and as a result, we get an assignment of every node to one out of m identified fragments $\{F_1^i, \ldots, F_m^i\}$ of the graph G_i for solution $\mathbf{x}^i$, where $\bigcup_{s=1}^m F_s^i = G_i$ and $F_s^i \cap F_t^i = \emptyset \;\; \forall s,t \in \{1,\ldots,m\},\; s \neq t$.

The goal of fragmenting the graph is to compare distinctive parts of the solutions independently. To evaluate the dissimilarity of a single pair of fragments (F_s^i, F_t^j) with $i \neq j$ and $s,t \in \{1,\ldots,m\}$, one can rely on established proximity measures from the literature, as exemplified by $d_{\texttt{tfidf}}$ or $d_{\texttt{align}}$. This requires only that each fragment can be represented individually to apply the respective measure, e.g. as a string, a numeric vector, etc. We call this the *signature* of the fragment. Let $d(F_s^i, F_t^j)$ be the dissimilarity of the fragment pair. To compare the two underlying solutions $\mathbf{x}^i$ and $\mathbf{x}^j$ as a whole, m suitable pairs of fragments (F_s^i, F_t^j), $(s,t) \in M \subset \{1,\ldots,m\}^2$ have to be established for comparison. Since we want those pairs to yield the best overall match, i.e. the minimum sum of dissimilarities, we arrive at an optimal matching problem. For now, we use a simple greedy heuristic to gain an approximate matching M of all fragments. We then compute the overall dissimilarity as the mean $\delta(F^i, F^j) = \frac{1}{m} \sum_{(s,t)\in M} d(F_s^i, F_t^j)$.

3. EVALUATION AND CONCLUSION

We use three datasets consisting of Java programs, where a semantically meaningful class separation is given in each set. To quantify the discriminative quality of the different proximity measures, we report the results of a simple *k-nearest-neighbor* (k-NN) classifier w.r.t. this class structure, see Tab. 1. The *Artificial* dataset consists of 48 programs created by a human expert and serves as a basic testbed. The 48 programs all solve the simple task to decide whether all words in a given input sentence are palindromes. The programs are deliberately designed to form 8 groups of 6 solutions each. Each group represents a distinct approach to solve the task, resulting from a combination of 3 simple (binary) design choices: using Java utility functions or not, using String objects or arrays of characters, splitting the sentence into words or iterating over the whole input sequence. Within each group, the 6 programs are only slightly different, with altered syntactic details like variable names and the sequence of operations. The *'Tasks'* dataset consists of 438 real student solutions, collected during 3 different programming exams for business students. Each solution was provided by an individual student, and the data is class-labeled according to the 3 different tasks assigned in the respective exam: [I] implementing Newton's method to find zeros in 2nd order polynomials (144 solutions); [II] calculating income tax for a given income profile (155); and [III] checking if a given sentence contains a palindrome, and if the sentence is a pangram (139). The *'TextCheck'* dataset consists of 68 student solutions which solve the above-mentioned task [III]. Here, class labels were provided by tutoring experts who were asked to determine meaningful groups in the solution set. The experts distinguished them according to 3 design choices very similar to the ones used in the *Artificial* set (which was subsequently created). This resulted in 8 classes corresponding to distinct strategies to solve the task. In general, solutions are very heterogeneous and classes are highly imbalanced. Therefore, this dataset represents a state-of-the-art challenge for a real ITS.

After preprocessing as described, we applied four variants of proximity measures, evaluating the accuracy of a 3-NN classifier, see Tab. 1. The results show that with all measures the solutions from the *Artificial* and *Tasks* dataset are classified rather reliably, which indicates that the measures are semantically meaningful. Accuracies for the *TextCheck* data are generally low, as expected from the challenging scenario. However, the measures based on fragmentation, $\delta_{\texttt{tfidf}}$ and $\delta_{\texttt{align}}$, showed a performance increase as compared to their simpler counterparts $d_{\texttt{tfidf}}$ and $d_{\texttt{align}}$. A rigorous evaluation of the approach is the subject of ongoing work.

Dataset	#fragments	$d_{\texttt{tfidf}}$	$\delta_{\texttt{tfidf}}$	$d_{\texttt{align}}$	$\delta_{\texttt{align}}$
Artificial	$m = 4$	0.94	0.92	0.94	0.94
Tasks	$m = 6$	0.98	0.83	0.98	0.98
TextCheck	$m = 6$	0.34	0.32	0.41	0.54

Table 1: Classification accuracies of a 3-NN classifier with different proximity measures on the experimental datasets. The number of fragments m in the measures $\delta_{\texttt{tfidf}}$ and $\delta_{\texttt{align}}$ was chosen with regard to the average size of graphs in the respective dataset.

4. REFERENCES

[1] T. Gog and N. Rummel. Example-based learning: Integrating cognitive and social-cognitive research perspectives. *Edu. Psych. Rev.*, 22:155–174, 2010.

[2] S. Gross, B. Mokbel, B. Hammer, and N. Pinkwart. Feedback provision strategies in intelligent tutoring systems based on clustered solution spaces. In *DeLFI 2012*, pages 27–38. Köllen, 2012.

[3] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[4] C. Lynch, K. D. Ashley, N. Pinkwart, and V. Aleven. Concepts, structures, and goals: Redefining ill-definedness. *Int. J. of A.I. in Edu.*, 19(3):253 – 266, 2010.

[5] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, Aug. 1988.

[6] U. von Luxburg. A tutorial on spectral clustering. *Stat. Comput.*, 17(4):395–416, 2007.