ESANN 2015 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 22-24 April 2015, i6doc.com publ., ISBN 978-287587014-8.
Available from http://www.i6doc.com/en/.

Adaptive structure metrics for automated feedback provision in Java programming

Benjamin Paassen, Bassam Mokbel and Barbara Hammer *

CITEC centre of excellence
Bielefeld University - Germany

Abstract. Today's learning supporting systems for programming mostly rely on pre-coded feedback provision, such that their applicability is restricted to modelled tasks. In this contribution, we investigate the suitability of machine learning techniques to automate this process by means of a presentation of similar solution strategies from a set of stored examples. To this end we apply structure metric learning methods in local and global alignment which can be used to compare Java programs. We demonstrate that automatically adapted metrics better identify the underlying programming strategy as compared to their default counterparts in a benchmark example from programming.

1 Introduction

Intelligent tutoring systems (ITSs) have made great strides in recent years; they offer the promise of individual one-on-one computer based support in the context of scarce human resources such as common e.g. for MOOCs [14]. However, researchers have reported 100 - 1,000 hours of authoring time for one hour of instructions in ITSs [12]; in addition, ITSs usually require an underlying domain theory such that their applicability is limited in areas where problems and their solution strategies are not easy to formalise [9]. Besides ill-defined domains such as argumentation or writing, this also applies to more classical scenarios such as programming due to different programming styles or algorithmic strategies [4].

In the domain of programming, many ITSs generate feedback based on explicit modelling such as compiler instructions or constraints [7, 8]. This restricts their applicability to settings where declarative knowledge about the underlying programming strategies is available, but it rules out appropriate feedback if the student follows a different strategy as compared to the pre-coded model. It has been shown that, as an alternative, example based feedback provision, by highlighting or contrasting a student solution to a known exemplar from a list of examples gathered over time, can offer valuable feedback in such cases [5].

Such techniques, besides data, require a suitable metric based on which to compare solutions. In this contribution, we will investigate how to efficiently obtain a metric based on which to compare programs and distinguish their underlying solution strategy. Thereby, we will face two problems. (1) Solutions are typically non-vectorial, such that structure metrics have to be used. We will rely on the approach [10], which compares programs by an alignment of the basic ingredients as present in their syntax trees. Unlike previous work, we will allow alignment strategies which enable a skip of irrelevant parts at low costs. (2) A structure metric crucially depends on the metric parameters; we will investigate

*Funding by the DFG under grant numbers HA 2719/6-1 and HA 2719/6-2 and the CITEC center of excellence is gratefully acknowledged.

efficient strategies how to autonomously learn these parameters. Based on the work [11] which transfers the powerful concept of metric learning from vectorial data such as summarised in [1] to the structural domain, we will investigate the effect of relevance learning for different alignment metrics.

2 Data representation of programming tasks

In this contribution, we will rely on the following empirical observation investigated e.g. in [5, 4]: in the absent of explicit models, feedback strategies which contrast a learner solution by a structurally similar known solution are efficient. Hence we face the following machine learning problem: how to compare two programs such that structurally similar solutions are identified, while different programming strategies are judged as dissimilar? We will focus on the domain of Java programming. The similarity of programs cannot be decided based on its computed function only, since the same functionality can be realised based on different programming principles (e.g. iterative versus recursive programming). Therefore, we will rely on a comparison of syntactical and structural differences.

In [10], an efficient technique to compare two programs based on their syntax only has been proposed. Program code can be transferred to an abstract syntax tree (e.g. using the Oracle Java Compiler API). Every vertex of this parse tree is characterised by a feature vector encoding characteristic properties. For Java code, nine features are provided: the vertex type, the scope, the parent vertex, code position, name, class name, return type, external references, and internal references (number of edges). We represent this tree as a string by using its prefix notation. This way, the ordering of vertices corresponds to their natural sequential order when executing the program. More importantly, a comparison of programs can then be based on string alignment as suggested in [11, 10].

3 Alignment metric

Assume data (in our case programs) are encoded as sequences $\bar{a} = (a^1, \ldots, a^{|\bar{a}|})$ with entries $a^i = (a^i_1, \ldots, a^i_n) \in X$, the direct product of the $n = 9$ feature sets. We assume that comparators $d_j(a_j, b_j) \in [0, 1]$ enable the comparison of single features, inducing a local distance of sequence entries $a, b \in X$

$$d_\lambda(a, b) = \sum_{j=1}^{n} \lambda_j d_j(a_j, b_j)$$

with *relevance weights* $\lambda_j \geq 0$ which sum up to 1. For sequences, $\bar{a}$ and $\bar{b}$, due to their possibly different length, a direct vectorial comparison is not possible, rather dissimilarities are induced by *alignment*. An alignment of two sequences $\bar{a}$ and $\bar{b}$ consists of extensions of the sequences by the symbol *gap* $-_g$ such that the resulting sequences $\bar{A}$, $\bar{B}$ have equal length, and not both entries A^i and B^i equal $-_g$. For such an alignment, fixing gap scores $d_\lambda(a, -_g) = d_\lambda(-_g, b) = c_g$, alignment costs are defined as

$$D_\lambda(\bar{A}, \bar{B}) = \sum_{i=1}^{|\bar{A}|} d_\lambda(A^i, B^i)$$

An alignment with minimum costs $d_\lambda(\bar{a}, \bar{b}) = \min_{\bar{A},\bar{B}} D_\lambda(\bar{A}, \bar{B})$ can efficiently be computed using dynamic programming (DP) based on the recursion

$$\begin{aligned} d_\lambda(\bar{a}(I+1), \bar{b}(J+1)) \quad = \quad & \min\{d_\lambda(\bar{a}(I), \bar{b}(J)) + d_\lambda(a^{I+1}, b^{J+1}), \\ & d_\lambda(\bar{a}(I+1), \bar{b}(J)) + d_\lambda(-_g, b^{J+1}), \\ & d_\lambda(\bar{a}(I), \bar{b}(J+1)) + d_\lambda(a^{I+1}, -_g)\} \end{aligned}$$

where $a(I)$ refers to the first I entries of the sequence only [13].

This alignment has the drawback that it is *global*, i.e. gaps have the same costs no matter whether they are distributed or occur as block. For programs, it is much more likely to exchange entire blocks rather than single lines. Therefore, we consider an extension, relying on the idea of [3], where we allow cheaper skips $-_s$ in addition to deletes. We refer to an alignment with skips as *local alignment*; formally, this refers to an extension of strings $\bar{a}$ and $\bar{b}$ by the symbols $-_g$ and $-_s$, such that the resulting sequences have the same length, every position contains not only $-_g$ and $-_s$, and $-_s$ occurs in blocks of length at least $k = 3$ only. Provided $d_\lambda(a, -_s) = d_\lambda(-_s, b) = c_s < c_g$ is fixed, this notion induces an arrangement of inserts and deletes in blocks, if possible. An optimum local alignment can efficiently be computed similar to a global one using DP, where, due to the requirement of skip blocks having a minimum length, several tables have to be maintained, see [3] or the more general framework [2] for its realisation.

4 Adaptive alignment

Both, global and local alignment crucially depend on the metric parameters λ which weight the relevance of the features associated to the nodes of a Java program. Following the approach [11], we propose to automatically adjust these parameters based on auxiliary information. Note that metric learning constitutes an advanced and very powerful paradigm for vectorial machine learning tasks, while only few methods have been proposed in the context of structure metrics so far [1, 11, 15]. We assume that label information is available for an example set of programs, whereby the label refers to the underlying programming style or any other class of interest. We adapt metric parameters together with an efficient, metric-based classification algorithm as provided by relational generalised learning vector quantisation (RGLVQ), which aims for a minimisation of the mislabeled examples by the model [6]. This procedure has the advantage that metric parameters and class model can be adjusted simultaneously, whereby the classifier relies on a discrete representation of structured data in term of pairwise dissimilarities only, aiming for a robust solution separation which is characterised by a large hypothesis margin, see [6]. At the same time, metric parameters are adapted in order to maximise the discriminative power of the resulting representation, whereby these adjusted metric parameters can be used independently of the underlying classification model, afterwards.

Formally, metric learning as well as model optimisation is based on the objective function of RGLVQ. Model updates as reported in [6] result. For the metric parameters, a gradient technique essentially relies on the gradient of the underlying costs with respect to the involved dissimilarity measure (see e.g. [11]), and the gradient of the dissimilarity, i.e. the alignment $d_\lambda(\bar{a}, \bar{b})$, with respect to metric parameters λ. For the latter, we approximate the discontinuous function min which occurs in the metric DP scheme by softmin, and compute derivatives based on the same recursive DP scheme as the dissimilarities itself. See [11] for the formulas in case of a global alignment, for a local alignment similar formulas can be derived depending on its slightly more complex DP scheme.

5 Experiments

We evaluate the technique in a benchmark example from [11]: 64 Java programs from 37 different web sites are gathered, which comprise two different sorting paradigms (35 bubble sort, 29 insertion sort, 4 resp. 2 of which are implemented recursively, the remainder uses loops). Local comparators $d_j(a_j, b_j)$ are normalised to the interval $[0, 1]$, and gap and skip costs are fixed as $c_s = 0.67 < c_g = 1$. We train a RGLVQ classifier with one prototype per class, whereby the task is to predict the underlying sorting strategy (bubble sort versus insertion sort). The initialisation of the relevance weights is uniform 1/9 with small noise added for symmetry breaking. We evaluate the results of a repeated five fold cross-validation with five repeats and the following different metric choices: no adaptation of metric parameters versus metric learning, and local versus global alignment. Besides the classification accuracy of RGLVQ, the result of an 5-NN error and an SVM is reported as well. The latter two are based on the same metrics, whereby an SVM kernel matrix is obtained using double centering of the dissimilarities and flip-correction of negative eigenvalues.

The resulting classification accuracy (larger is better) as well as the separation ratio indicating class separability induced by the respective dissimilarity (smaller is better) are reported in Tab. 1. Obviously, a local versus global alignment as well as adapted versus default metric parameter allow for an improvement of the classification accuracy by about 10%. Interestingly, the separation of the classes becomes much more pronounced, as can also be seen from a visual inspection of the data: Fig. 1 displays a t-SNE projection of the data and prototypes when using the adapted local alignment measure. This display clearly shows a comparably good separation of the two classes with a large margin in between the classes. Interestingly, the class of recursive programs clearly sticks out as a separate cluster (marked R), albeit this information has not been used while training. Thus, it can be expected that the resulting metric parameters constitute a good choice based on which example solutions can be selected as feedback mechanism in corresponding learning scenarios.

The metric learning scheme allows us to inspect the semantics of the resulting

Method	Train (Std.)	Test (Std.)	SVM (# SV)	5-NN	Sep.
global	0.75 (0.03)	0.74 (0.10)	0.65 (47)	0.77	0.88
global adapted	0.80 (0.02)	0.80 (0.09)	0.63 (46)	0.92	0.75
local	0.77 (0.03)	0.74 (0.12)	0.74 (48)	0.62	0.9
local adapted	0.85 (0.03)	0.85 (0.10)	0.78 (47)	1.00	0.74

Table 1: Mean training and test set accuracy with different adapted metrics and resulting separation ratio

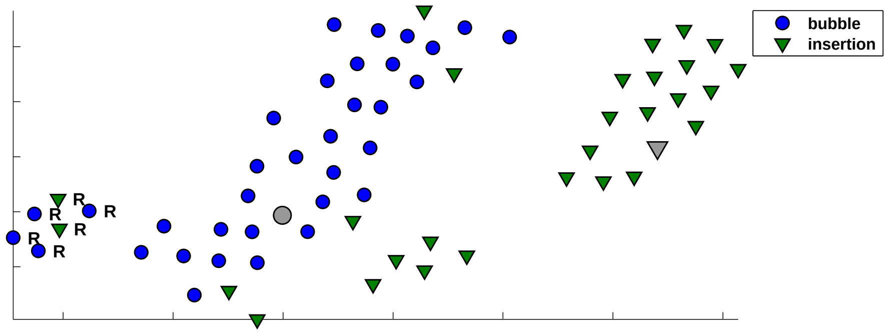

Fig. 1: t-SNE projection of the programs using the adapted local alignment. Prototypes are marked in grey. R refers to a cluster of recursive programs.

metric since we can project the relevance profile of features λ_j (normalised by their usage in the data), see Fig. 2. Interestingly, a semantically meaningful profile results, which marks the type and scope as most prominent structuring elements to distinguish programming styles. In contrast exact code position, as well as several other features play only a very minor role. This relevance profile carries the promise to transfer to different scenarios beyond the considered programming task for sorting programs, such that it can induce a suitable metric based on which to determine an exemplar for automated feedback.

6 Conclusion

We have investigated the suitability of structure metric learning for local and global alignment strategies in the context of the classification of programming strategies. Metric adaptation can be done efficiently based on a distance-based classification model, which enables direct gradient techniques for its optimisation. Metric parameters itself can be adapted following the same DP scheme as the underlying alignment. Based on a cross-validation, it is apparent that an adapted metric allows a better separation of the strategies; whereby this result is independent from the subsequent classifier used (LVQ, kNN, and SVM). This observation can be substantiated by a better visualisation of the solution space, in which not only the different strategies but also the (priorly not marked) recursive strategies form well separated clusters. In addition, metric parameters allow a semantic insight into the meaningful descriptors for program nodes. This result lays the foundation for a semantically meaningful distance measure based on which to select possible feedback-partners. It is subject of ongoing work in how far the results transfer to different tasks and domains.

References

[1] A. Bellet, A. Habrard, and M. Sebban. A Survey on Metric Learning for Feature Vectors and Structured Data. *ArXiv e-prints*, (1306.6709), 2013.

[2] R. Giegerich, C. Meyer, and P. Steffen. A discipline of dynamic programming over sequence data. *Science of Computer Programming*, 51(3):215 – 263, 2004.

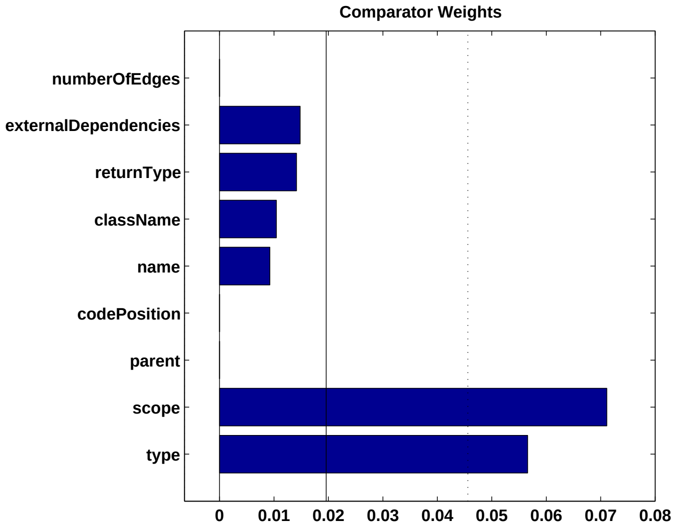

Fig. 2: Weight profile of the adapted metric indicating the relevance of the node characteristics for distinguishing programming styles.

[3] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

[4] S. Gross, B. Mokbel, B. Hammer, and N. Pinkwart. Example-bases feedback provision using structured solution spaces. *Int. Journ. Learning Technologies*, 9(3):248–280, 2014.

[5] S. Gross, B. Mokbel, B. Hammer, and N. Pinkwart. How to select an example? A comparison of selection strategies in example-based learning. In *ITS*, 340–347, 2014.

[6] B. Hammer, D. Hofmann, F. Schleif, and X. Zhu. Learning vector quantization for (dis-)similarities. *Neurocomputing*, 131:43–51, 2014.

[7] J. Holland. *A constraint based ITS for the Java Programming Language.* PhD thesis, University of Canterbury, U.K., 2009.

[8] J. Jeuring, A. Gerdes, and B. Heeren. Ask elle: A haskell tutor – demonstration –. Technical Report UU-CS-2012-010, Utrecht University, The Netherlands, 2010.

[9] C. Lynch, K. D. Ashley, N. Pinkwart, and V. Aleven. Concepts, structures, and goals: Redefining ill-definedness. *Int. Journ. AI in Education*, 19(3):253–266, 2009.

[10] B. Mokbel, S. Gross, B. Paaßen, N. Pinkwart, and B. Hammer. Domain-independent proximity measures in intelligent tutoring systems. In *EDM*, pages 334–335, 2013.

[11] B. Mokbel, B. Paaßen, F.-M. Schleif, and B. Hammer. Metric learning for sequences in relational LVQ. *Neurocomputing*, accepted, 2015.

[12] T. Murray, S. Blessing, and S. Ainsworth. *Authoring tools for advanced technology learning environments: Toward cost-effective adaptive, interactive and intelligent educational software.* Springer, 2003.

[13] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970.

[14] R. Nkambou, R. Mizoguchi, and J. Bourdeau. *Advances in Intelligent Tutoring Systems.* Springer, 2010.

[15] P. Schneider, M. Biehl, and B. Hammer. Adaptive relevance matrices in learning vector quantization. *Neural Computation*, 21(12):3532–3561, 2009.