

---

# Transformation-Based Regression

---

**Björn Bringmann**

**Stefan Kramer**

Institute for Computer Science, University of Freiburg, Georges-Köhler-Allee Geb. 079, D-79110 Freiburg i.Br., Germany

BBRINGMA@INFORMATIK.UNI-FREIBURG.DE

SKRAMER@INFORMATIK.UNI-FREIBURG.DE

**Friedrich Neubarth**

**Hannes Pirker**

**Gerhard Widmer**

Austrian Research Institute for Artificial Intelligence, Schotteng. 3, A-1010 Vienna, Austria

FRIEDRICH@AI.UNIVIE.AC.AT

HANNES@AI.UNIVIE.AC.AT

GERHARD@AI.UNIVIE.AC.AT

## Abstract

In this paper, we introduce Transformation-Based Regression (TBR), a novel rule-based, symbolic regression technique based on Transformation-Based Learning (TBL). Although Transformation-Based Learning has been introduced already a couple of years ago, it has not yet been considered for regression-type tasks. The proposed method should be particularly useful for learning from examples with a given neighborhood relation, where the dependent variable of one example also depends on neighboring examples. Thus, the method should have a potential for learning from sequence and spatial data. In the paper, we demonstrate the capabilities and limitations of the approach in two highly complex real-world domains, musicology and speech synthesis.

## 1. Introduction

This paper introduces a new technique for rule-based, symbolic regression that is based on Transformation-Based Learning (TBL) (Brill 1995). Transformation-Based Learning has been devised for classification-type tasks such as part-of-speech tagging or spelling correction in natural language processing (Mangu & Brill 1997). Surprisingly, since the invention of TBL, no one has considered the potential of this technique for regression problems yet (Brill 2002).

Transformation-based learning is particularly appealing in domains with neighborhood relations between

examples, where the dependent variable of one example may depend on the one of neighboring examples. So, it should in general be interesting for learning from sequence or spatial data.

This paper is organized as follows. In the next section, we will briefly review Transformation-Based Learning. Section 3 will introduce Transformation-Based Regression (TBR), a novel regression technique based on TBL. Subsequently, we will present experiments with two complex real-world datasets, where the capabilities and limitations of the TBR algorithm are demonstrated. The final section of this paper touches upon related work and concludes.

## 2. Transformation-Based Learning

Transformation-Based Learning (TBL) is a technique originating from the field of Natural Language Processing. TBL has been shown to work successfully in wide range of applications, such as part-of-speech tagging and spelling correction (Brill 1995; Mangu & Brill 1997).

Essential to transformation-based learning is that predictions are not performed in a single pass, but in several passes. The algorithm starts off with assigning baseline predictions to all examples in the training set. In each subsequent iteration of the algorithm, the current predictions are transformed. In other words, the algorithm *attempts to improve the predictions based on the current predictions*. The power of the learned transformation lists stems from *intermediate results*, which are reflected in the current label of an example and available to be used for the prediction of other examples.

Transformation-Based Learning differs from conventional symbolic Machine Learning techniques in several respects:

- First of all, TBL performs multiple passes of predictions, and not “one-shot” learning. In conventional symbolic Machine Learning, exactly *one* prediction is made *once* for each example, and this prediction is not further refined.
- Secondly, TBL makes use of intermediate prediction results. Intermediate results are stored and can be used subsequently.
- Thirdly, TBL starts off with some plausible initial baseline prediction and then refines it. The nice thing about this approach is that one can input “educated guesses” regarding the class or number to be predicted (or even take the prediction from some other system), and then attempt to get rid of particularly hard cases (by transforming them closer to the correct output). If the input from another system is used, one can find out where the majority of its errors is made. Many other interesting things may be envisaged: for instance, attempt to learn the difference between two sets of classifications or numbers.

Table 1 shows the pseudo-code of TBL. The input to the algorithm is the training set *Examples*. In the first step, the algorithm assigns a baseline prediction to each example in the training set. For classification, this could be, e.g., the majority class, the majority class of some subgroup of examples, or some more sophisticated choice. Subsequently, the algorithm sequentially constructs a transformation list *Rules* consisting of transformation rules. A transformation rule is a rule with conditions on the examples in the left-hand side and a transformation of the class label in the right-hand side (see below). In each iteration, *FindBestRule* searches for the rule that transforms the current prediction of the covered examples closest to the actual values according to some scoring function. The best transformation rule *BestRule* is then appended to the transformation list. Finally, the training examples are transformed according to *BestRule*. In this step, the current prediction of each example covered by the left-hand side of the transformation rule is updated using the transformation in the right-hand side. This process continues until no rule can be found or no improvement can be achieved according to the scoring function.

Table 1. Pseudo-code of transformation-based learning

```

TRANSFORMATIONBASEDLEARNING(Examples)
1  for each example  $e_i \in Examples$ 
2  do assign baseline prediction to  $e_i$ 
3   $Rules \leftarrow []$ 
4  while true
5  do
6     $BestRule \leftarrow FindBestRule(Examples)$ 
7    if ( $BestRule = none$ )  $\vee$  no improvement
8      then return  $Rules$ 
9    else  $Rules \leftarrow append(BestRule,$ 
10                                      $Rules)$ 
11                                      $Examples \leftarrow apply(BestRule,$ 
12                                      $Examples)$ 

```

### 3. Transformation-Based Regression

In the area of natural language processing, transformation-based learning has predominantly been used for classification-type learning, mainly in the context of part-of-speech tagging. Here, we adapt the general TBL framework and present a new *regression rule learning* algorithm based on continuous transformations. We call it *Transformation-Based Regression (TBR)*.

In the following, we will explain the basic concepts involved, describe an algorithm for learning regression rules in this setting, and comment briefly on the expressiveness of the corresponding class of models.

#### 3.1 Basic Concepts

In the following, an example  $e_i$  is a pair  $(\mathbf{x}_i, y_i)$ , where  $\mathbf{x}_i$  is a vector of attribute values, and  $y_i$  is the numerical value to be predicted. Since we transform the current prediction from iteration to iteration, we need a notation for the prediction of example  $e_i$  in iteration  $t$ . In the following,  $y_i(t)$  denotes the prediction for example  $e_i$  in iteration  $t$ . The next transformation of value  $y_i(t)$  is obviously denoted  $y_i(t+1)$ .

The representation of transformation rules for regression looks as follows:

$$\text{if } c(e_i) \text{ then } y_i(t+1) \leftarrow a + b * y_i(t) \quad (1)$$

In the left-hand side (LHS) of these rules (also called the rule bodies in the following),  $c(e_i)$  denotes some conditions on the examples  $e_i$ , and in the right-hand side (RHS, also called head in the following), the new prediction for example  $e_i$ ,  $y_i(t+1)$  is defined as a linear transformation of the current prediction for the example,  $y_i(t)$ . For all examples  $e_i$  covered by  $c(e_i)$ ,

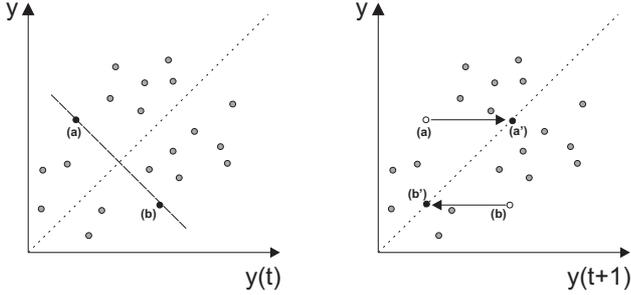


Figure 1. Example of effect of transformation: two erroneous points are swapped

the current prediction at step  $t$  is transformed linearly into the new prediction at step  $t + 1$ ,  $y_i(t + 1)$ .

**Example:** Figures 1 and 2 illustrate the effect of transformation-based regression. In the toy example in Figure 1, points (a) and (b) are covered by the conditions on the left-hand side, and mapped to the correct values of  $y$  by the linear transformation. The other example in Figure 2 shows how points near a steeply ascending line are picked by some rule and mapped to the diagonal, that is, the correct value for  $y$ .  $\square$

Parameters  $a$  and  $b$  obviously should be chosen such that the resulting error after the transformation is minimized. If the chosen error function is the mean squared error, then we would like to find parameters  $a$  and  $b$  that minimize

$$\sum_{i=1}^n (y_i - (a + b * y_i(t)))^2 \quad (2)$$

where index  $i$  runs over the examples covered by the rule body. That is, the task is simply to perform linear regression of the target value  $y$  on  $y(t)$ . Since  $y(t)$  takes the role of  $x$  in linear regression, parameter  $b$  is simply the covariance  $s_{y(t)y}$  divided by  $s_{y(t)}^2$ , the variance of  $y(t)$ . The optimal value for  $a$  can be found analogously.

The conditions in the body may contain tests on attributes of the example itself, and even the current prediction of the example. Since we aim at applications where each example has a defined set of neighboring examples (such as in spatial or sequence data), the conditions may also refer to the neighbors' attributes or even their current predictions. Since we only deal with sequence data in this paper, we just have to define  $succ.x_j$  as the attribute  $x_j$  of the successor example, and  $pred.x_j$  as the attribute of the predecessor example. This schema can easily be extended to arbitrary neighborhood relations. If we want to refer to a neighbor's current prediction, we have to use tests

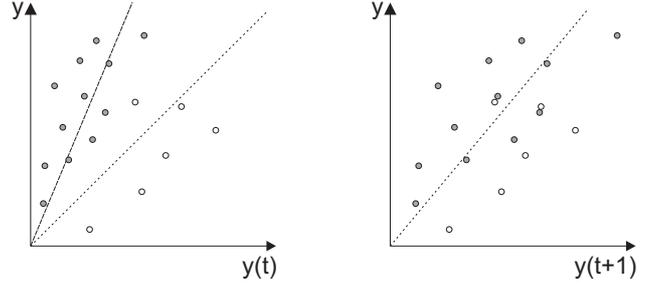


Figure 2. Example of effect of transformation: points near a steeply ascending line are transformed towards the (correct) diagonal

on  $succ.y_i(t)$  and  $pred.y_i(t)$ . In the section on experimental results, we investigate whether the use of  $y_i(t)$ ,  $succ.y(i)$  or  $pred.y(i)$  in the LHS of transformation rules improves the performance of the learner. Please note that even if the current prediction is not used in the LHS, it is always used in the linear transformation in the RHS. The “specialty” of Transformation-Based Learning is that the current prediction is transformed into a new prediction in the RHS of the rules.

### 3.2 Algorithm

The top-level algorithm of TBR differs from the general one in Table 1 in minor details. Actually, the only changes concern the stopping criterion and some additional effort for selecting the “right” transformation list based on some validation set.

Since we are using TBL for predictive purposes, and not, for instance, part-of-speech tagging, we need a way to decide when further transformations actually harm the generalization performance of the learner. Our approach to this problem is rather straightforward: We use one third of the training examples for validating the results and two thirds for learning.

The algorithm adds transformation rules to the transformation list for a predefined number of iterations. While the transformation list is built, we record the mean squared error on the validation set. After the construction of the full transformation list, we choose a part of it (actually, the first  $m$  rules) based on the error on the validation set. Here, we have two options: either we choose the transformation list with the minimum error on the validation set, or the smallest one within one standard error of the best rule on the validation set.

The algorithm for finding the best transformation rule is described in Table 2. The algorithm starts with

Table 2. Pseudo-code of greedy search for best transformation rule in transformation-based regression

```

FINDBESTRULE(Examples)
1  BestRule ← 'if true then 0 + 1 *  $y_i(t)$ '
2  BestError ← RMSE(BestRule, Examples)
3  while true
4  do
5      (Rule, Error) ←
6          FindBestRefinement(BestRule,
7                               Examples)
8      if Error ≤ BestError
9          then
10         BestRule ← Rule
11         BestError ← Error
12     else return BestRule

```

Table 3. Pseudo-code of finding the best refined rule, given some rule, in transformation-based regression

```

FINDBESTREFINEMENT(Rule, Examples)
1  LHS ← LHS of Rule
2  BestError ← +inf
3  for Cond ∈ PossibleConditions
4  do NewLHS ← Refine(LHS, Cond)
5      CoveredExamples ←
6          Covered(NewLHS, Examples)
7      Calculate optimal a and b
8          for CoveredExamples
9      NewRule ← NewRule(NewLHS, a, b)
10     NewError ← RMSE(NewRule, Examples)
11     if NewError < BestError
12         then BestRule ← NewRule
13         BestError ← NewError
14 return (BestRule, BestError)

```

the trivial transformation rule (the one that does not change anything), and continually refines it by greedy search. As such, the algorithm is reminiscent of rule building algorithms in regular classification rule learning. One detail deserves special attention: The algorithm stops growing a rule as soon as the error would increase after the refinement of the LHS. The evaluation function is  $1/N \sum (y_i(t+1) - y_i)^2$ , the mean squared error of *all* examples after the transformation. The evaluation of the rule (list) before the refinement is obviously  $1/N \sum (y_i(t) - y_i)^2$ . If we want to know whether a refinement pays or not, we just have to consider those examples that are covered by the rule so far. The chosen stopping criterion makes sense for transformation rules, because the error on the training set can increase after refining a rule.

**Example:** As an illustration of this, let us consider that the LHS of a rule at one point covers the four examples  $e_1, e_2, e_3$  and  $e_4$ . The best refinement of the rule only covers examples  $e_1$  and  $e_2$ . The pairs  $(y(t), y)$  of these examples are as follows:  $e_1 = (3, 6)$ ,  $e_2 = (9, 12)$ ,  $e_3 = (10, 110)$ ,  $e_4 = (15, 115)$ . In the example, the instances with the biggest discrepancy between the current prediction and the actual value are not transformed after the refinement. In contrast, the two remaining examples covered by the rule after the refinement can be “perfectly” transformed (i.e, without error).<sup>1</sup> Thus, the sum of the squared error is  $100^2 + 100^2 + 0^2 + 0^2 = 20,000$ .

Performing linear regression for all four points (before the refinement) gives a sum of the squared error of only approximately 3,942. Thus, the refinement does not pay and we choose to stop at this point of the rule growing process. □

The pseudo-code for the refinement of a given rule is shown in Table 3. The algorithm simply takes all possible refinements of the rule (in other words, the additional conditions for the LHS), and then determines the optimal parameters  $a$  and  $b$  for the covered examples. From all candidate refinements, we choose the one with the minimum error on all examples (see above).

### 3.3 Expressiveness

It is easy to see that TBR rules can in principle represent the same functions as regression trees. (This can be constructed in the same way as for the classification variant of transformation-based learning, where all decision trees can in theory be emulated by transformation rules (Brill 1995).) To do this, one has to set value  $b$  to zero (that is, the current prediction does not matter), and choose the value of  $a$  in the head appropriately. For each split, we obtain two transformation rules of this kind.

## 4. Experimental Results

The TBR algorithm was experimentally tested on two large, real-world datasets representing sequential phenomena. In the experiments, we compared it with simple linear regression as implemented in WEKA (Witten & Frank 1999) (as a benchmark) and CUBIST (RuleQuest Research 2002) (as a state-of-the-art regression rule learner). In addition, a comparison to these regression methods was performed on a number of standard regression datasets.

<sup>1</sup>Two points can always be perfectly transformed using linear regression (see also the example above).

#### 4.1 Expressive Music Performance: Predicting a Concert Pianist

The data used in the first study stems from a large research project that studies the fundamentals of *expressive music performance* (Widmer 2001). Expressive performance is the art of shaping a piece of music, by continuously varying parameters like tempo, loudness, etc. while playing a piece (beyond what is prescribed in the written music score). Expressive performance is what makes music sound musical, and what makes certain musicians famous.

Can an ‘artistic’ behavior like this be predicted at all? Recent studies show that it can, to a surprising extent. This raises a number of interesting questions, which are beyond the scope of this paper. Here, we simply use expressive performance as a source of test data.

The data sets used in the following experiments are based on performances of 13 complete piano sonatas by W.A. Mozart (K.279–284, 330–333, 457, 475, and 533) by a Viennese concert pianist. The pieces were played on a Bösendorfer SE290 computer-monitored concert grand piano, which measures every key and pedal movement with utmost precision. From these recordings, the ‘melodies’ (mostly the soprano parts) were extracted, which gives an effective training set of 46,378 notes. Each note is described by a number of attributes that represent both intrinsic properties (such as scale degree, duration, metrical position) and some aspects of the local context (e.g., melodic properties like the size and direction of the intervals between the note and its predecessor and successor notes). The total number of attributes per note is 20.

There are three numeric target attributes, corresponding to three different dimensions of expressive variation: *timing* (the actual time between onsets of successively played notes, versus the time prescribed by the score and the current tempo), *dynamics* (the relative loudness of individual notes), and *articulation* (the actual sounding duration of a played note, i.e., whether it is held for the entire time interval dedicated to it (*legato*) or shortened and followed by a period of silence (*staccato*)). Thus, we have 3 learning problems.

As the examples are based on melodies, they naturally form a sequence where it makes sense to speak of successors and predecessors. Moreover, it is intuitively clear that there should be dependencies between neighboring examples: for instance, the loudness with which a particular note is going to be played should depend on musical properties of the note and its neighbors as well as on the loudness of these neighbors.

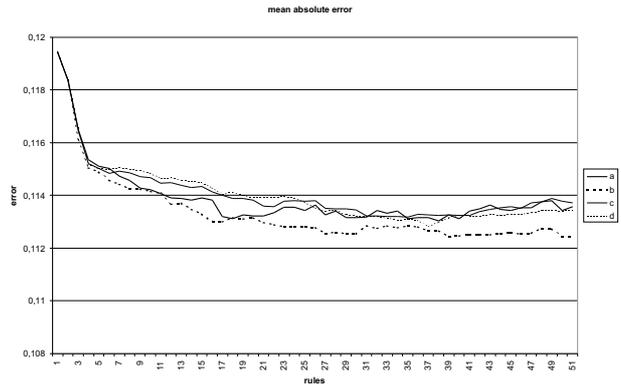


Figure 3. Dynamics: Mean Absolute Error

##### 4.1.1 RESULTS

Due to the size of the datasets (46,378 examples), a one-time evaluation on a randomly generated training (approx. 2/3) and testing set (1/3) was considered sufficient.<sup>2</sup> The training set was further divided into a learning (2/3) and a validation set (1/3), where the validation set was used by TBR to pick the ‘right’ transformation list (see above).

Figure 3 shows the evolution of the mean absolute error on the test set for the target variable *dynamics* over the application of rules in a transformation list. TBR was made to learn a fixed number (50) of rules, in order to see how the error evolves over a fixed range of cumulatively applied rules. From all possible transformation lists, TBR selects either the one with the minimum error on the validation set, or, alternatively, the smallest within one standard error of the best transformation list. Four variants of TBR with different capabilities are compared: TBR-a has access both to (the attribute values of) an example’s neighbors ( $succ.x_j$ ,  $pred.x_j$ ) and the currently predicted values  $y_i(t)$ ,  $pred.y_i(t)$ , and  $succ.y_i(t)$ . TBR-b has access to the neighbors, but none of the current predictions  $y_i(t)$ , TBR-c can look at  $y_i(t)$ , but not the neighbors, and TBR-d has neither.

The plot shows the expected shape of learning curves, with the error starting to rise again in some cases as too many rules are used — the usual overfitting scenario. Among the four TBR variants, the two that have access to an example’s neighbors (TBR-a/b) are

<sup>2</sup>The splitting of the datasets was done randomly on a piece-by-piece basis. That is, entire sonata sections were assigned in their entirety to either the training or the test set. Thus, the successor relation between consecutive notes was preserved. The analogous procedure was followed for sentences in the speech domain (see below).

Table 4. Experimental results in musical domain.

	correl.	MAE	rel.AE	# rules
<i>timing</i>				
TBR-a	.38	0.09	0.94	33
TBR-b	.22	0.09	0.98	2
TBR-c	.28	0.09	0.96	7
TBR-d	.29	0.09	0.96	40
CUBIST-b	.34	0.09	0.94	51
CUBIST-d	.28	0.09	0.96	44
Lin.Reg.-b	.31	0.09	0.98	-
Lin.Reg.-d	.27	0.09	0.96	-
<i>dynamics</i>				
TBR-a	.30	0.11	0.95	22
TBR-b	.31	0.11	0.95	15
TBR-c	.26	0.11	0.96	11
TBR-d	.25	0.11	0.96	8
CUBIST-b	.33	0.11	0.93	54
CUBIST-d	.32	0.11	0.94	48
Lin.Reg.-b	.31	0.11	0.95	-
Lin.Reg.-d	.27	0.11	0.96	-
<i>articulation</i>				
TBR-a	.02	0.32	0.81	11
TBR-b	.02	0.32	0.82	9
TBR-c	.01	0.33	0.83	11
TBR-d	.02	0.32	0.82	14
CUBIST-b	.03	0.30	0.77	47
CUBIST-d	.03	0.31	0.79	49
Lin.Reg.-b	.03	0.31	0.80	-
Lin.Reg.-d	.03	0.32	0.81	-

usually better than those that do not. Surprisingly, also having access to the current predictions of the target values ( $y_i(t)$ ) does not necessarily help. The plots for other error measures and domains look quite similar.

Table 4 gives the correlation coefficient, mean absolute error, and relative absolute error<sup>3</sup> for TBR, CUBIST and linear regression over all three target variables. TBR used the validation set to decide when to stop learning rules, according to the heuristic proposed in (Breiman et al. 1984) (i.e., TBR chooses the smallest rule list within one standard error of the best one on the validation set). In the table, CUBIST-b refers to CUBIST with access to the example’s attributes as well as the neighbors’ attributes. So, in principle, the same conditions may occur in the bodies of TBR-b and CUBIST-b rules. CUBIST-d is CUBIST with only the example’s own attributes available. Note that to be perfectly fair to CUBIST, which does not require a validation set, it was given the full training set (learning + validation) for learning, so it had more examples to learn from than TBR. We also included the figures for linear regression with both sets of features (denoted as Lin.Reg.-b and Lin.Reg.-d).

A statistical evaluation of the results in Table 4 (ac-

<sup>3</sup>These are also the measures that CUBIST reports.

ording to a paired t-test and a sign test – both on the basis of individual predictions) shows that TBR-a performs equally to CUBIST-b for *timing* (with a better correlation coefficient), but worse for *dynamics* and *articulation*. (In the speech domain presented below, TBR significantly outperforms linear regression and CUBIST.) Comparing TBR with our benchmark of linear regression, significant differences can be found in favor of TBR for *timing* and *dynamics*, but against TBR for *articulation*.

Having access to an example’s neighbors leads to improved results in this sequential domain: TBR-a/b tend to produce better results than TBR-c/d. With the stopping criterion used in this experiment, TBR-a improves over TBR-b in one domain (*timing*) and is practically indistinguishable in the other two.

As the quantitative results indicate, having access to the currently predicted target values ( $y_i(t)$ ,  $pred.y_i(t)$ ,  $succ.y_i(t)$ ) does not necessarily improve prediction accuracy. But it may improve other qualities. Looking at the learned musical rules that do explicitly refer to their neighbor’s expression values, we observe some musically sensible uses of this device. For instance, in the *dynamics* (loudness) domain, we find rules that raise a note’s loudness value if both the note’s predecessor and successor already have a high predicted loudness. That presumably improves the smoothness of the dynamics contour. This kind of effect would be very difficult, if not impossible, to achieve in other learning schemes.

## 4.2 Improving Naturalness of Synthetic Speech: Predicting Durations of Sounds

The speech related data used in this study was collected within a project that is concerned with the modeling of the durational variation of spoken sounds (Neubarth et al. 2000). From a practical point of view this work is motivated by the fact that the ‘proper’ prediction of durations is indispensable in order to improve the naturalness and thus acceptability of synthetic speech. The duration of a given sound within a language is influenced by a number of factors, e.g. its ‘intrinsic’ length, the identity of neighboring sounds, accentual status, position within a phrase etc. Thus the length of a single sound can well be varying in the range between 40 to 220 ms.

The data for the experiment was extracted from a corpus which comprises approximately 50,000 phones (2 hours of speech) of read Standard Austrian German, recorded by a single speaker. The speech data was semi-automatically segmented into sounds and annotated with a number of linguistic attributes. As in the

Table 5. Experimental results with speech data.

	correl.	MAE	rel.AE	# rules
TBR-a	.76	15.89	0.64	103
TBR-b	.79	15.09	0.61	150
TBR-c	.74	16.79	0.68	180
TBR-d	.72	17.12	0.69	84
CUBIST-b	.72	16.50	0.66	57
CUBIST-d	.74	16.08	0.65	60
Lin.Reg.-b	.71	17.07	0.69	-
Lin.Reg.-d	.69	17.61	0.71	-

musical domain above, these data have a sequential interpretation, and it is also known that there are strong influences of the neighboring sounds.

The feature set employed was taken from (Riedi 1998). Each example is labeled with attributes that convey the following information: *Segmental properties* (e.g. for consonants: place and manner of articulation) of the sound itself as well as for its two neighbors to the left and right, *positional* information (e.g. its position within a syllable, the syllable’s position within a word), information on the *size* of embedding constituents (e.g. number of syllables in the word) and its *accentuation level*. The target attribute is simply the sound’s measured duration expressed in [ms].

Table 5 compares the four TBR variants to CUBIST and linear regression on this dataset. Here, TBR-b takes the lead, followed by TBR-a, CUBIST-d, and CUBIST-b. TBR-b is significantly better than CUBIST-b, indicating that being able to transform predicted values can be a real advantage in this domain.

Again, somewhat surprisingly, being able to refer to the currently predicted target values  $y_i(t)$  in rule conditions can hurt as it can help: TBR-b performs better than TBR-a, but TBR-c performs better than TBR-d.

### 4.3 Evaluation on Standard Test Data

Just to check whether TBR is broadly competitive with other standard regression methods, we also tested it on a number of benchmark databases (which do not include meaningful neighbor relation that could be exploited by TBR). We chose the intersection of the datasets from (Torgo 2000) and (Indurkha & Weiss 2001) in our experiments (<http://www.ncc.up.pt/~ltorgo/Regression/DataSets.html>). The results are summarized in Table 6.

## 5. Conclusion

This paper has introduced Transformation-based Learning (TBL) as a general learning technique, and

has presented a new algorithm called Transformation-based Regression (TBR) for learning regression rules in this paradigm. Experiments with two large, complex data sets have shown that TBR seems competitive with the state-of-the-art regression rule algorithm CUBIST. The results show clearly that the ability to look at an example’s neighbors, as implemented in TBR, leads to improved results. Also, in a number of cases, although not always, the transformation approach itself pays in terms of numerical error measures, as can be seen by the comparison with CUBIST.

The transformation approach has other potential advantages. As it starts from a given baseline, which it tries to improve step by step, we can input educated guesses as baseline (or even take predictions from some other system as starting point), and then use TBR to refine these predictions or fix errors in these. One might also try to use TBR to model the difference between one set of predictions and another, by using the former as the starting set of baseline predictions, and the latter as the target values.

Transformation-based regression is related to a large number of Machine Learning techniques. First of all, it is related to recent regression methods in Machine Learning (Torgo 2000), in particular to other rule-based regression learners (Karalic & Bratko 1997; Indurkha & Weiss 2001). Furthermore, it is related to (Breiman 2001), where the outputs are also recoded in iterations, but in the context of bagging, in order to debias regression models. It would be interesting to analyze TBR in terms of bias and variance. (Blokkeel & Dehaspe 2000) seem to have anticipated the idea of transformations used for regression, but did not exploit the idea of iterated, linear transformations. Finally, we believe there exists a connection between the presented research and meta-learning or ensemble learning schemes, especially methods like Stacking (Wolpert 1992) or Cascading (Gama 1998), where the predictions made by one classifier are fed as new or additional attributes into a meta-learner.

Transformation-based learning originated in the field of natural language processing. With this paper, we hope to have brought this interesting approach to the attention of a wider machine learning audience, and to have encouraged further research on this methodology.

## Acknowledgments

Parts of this work were supported by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung (FWF)* under grants no. P12645, P13224 and Y99-INF. The Austrian Research Institute for AI

Table 6. Performance from 10-fold cross-validation of linear regression, CUBIST and TBR in standard benchmark domains. The same fold partitionings were used for all methods. TBR vs. linear regression gave a 4/0/2 (wins/draws/losses) result according to a paired t-test, and a 4/1/1 according to the sign test. The result of TBR vs. CUBIST was 2/0/4 according to both tests. We used a very simple discretization method (*equal width* with 25 intervals) in all experiments.

	<i>Lin.Reg.</i>			<i>CUBIST</i>			<i>TBR</i>		
	correl.	MAE	rel.AE	correl.	MAE	rel.AE	correl.	MAE	rel.AE
Ailerons	.90	0.0001	0.41	.00	0.0003	1.00	.89	0.0001	0.43
Census	.51	25447	0.78	.74	16974	0.52	.67	20909	0.64
CompAct	.85	6.08	0.57	.99	1.67	0.16	.98	2.15	0.20
Elevators	.90	0.002	0.44	.87	0.002	0.50	.87	0.002	0.49
Kinematics	.64	0.16	0.75	.82	0.12	0.55	.74	0.14	0.65
PoleTelecomm	.68	26.6	0.71	.99	2.34	0.06	.96	4.14	0.11

is supported by the Austrian Federal Ministry of Education, Science and Culture. Special thanks go to Luc De Raedt for interesting and fruitful discussions on the topic. We also wish to thank the anonymous reviewers for helping to improve the paper.

## References

- Blockeel, H. and Dehaspe, L. (2000). Cumulativity as Inductive Bias. in *Working Notes of the PKDD-2000 Workshop on Data Mining, Decision Support, Meta-Learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions*, (P. Brazdil and A. Jorge, eds.), 61–70.
- Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*, The Wadsworth Statistics/Probability Series. Belmont, CA, Wadsworth International Group.
- Breiman, L. (2001). Using Iterated Bagging to Debias Regressions. *Machine Learning*, 45(3):261–277.
- Brill, E. (1995). Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging *Computational Linguistics* 21(4):543–565.
- Brill E. (2002). Personal communication.
- Gama, J. (1998). Combining Classifiers by Constructive Induction. In *Proceedings of the Ninth European Conference on Machine Learning (ECML’98)*. Berlin: Springer Verlag.
- Indurkha, N. and Weiss, S.M. (2001) Solving Regression Problems with Rule-Based Ensemble Classifiers. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 287–292.
- Karalic, A. and Bratko, I. (1997) First Order Regression. *Machine Learning*, 26(2-3):147–176.
- Mangu, L. and Brill, E. (1997) Automatic Rule Acquisition For Spelling Correction. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML-97)*.
- Neubarth, F., Alter, K., Pirker, H., Rieder, E., Trost, H. (2000). The Vienna Prosodic Speech Corpus: Purpose, Content and Encoding, in Zuehlke W., Schukat-Talamazzini E.G. (eds.), *Konvens 2000 - Sprachkommunikation*, VDE Verlag, Berlin, 191–196.
- Riedi, M. (1998). *Controlling Segmental Duration in Speech Synthesis Systems*, Institut für Technische Informatik und Kommunikationsnetze, ETH Zürich, TIK-Schriftenreihe Nr. 26, PhD thesis, No. 12487.
- RuleQuest Research (2002). Cubist commercial regression tool. <http://www.rulequest.com/cubist-info.html>
- Torgo, L. (2000). Partial Linear Trees. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, 1007–1014.
- Widmer, G. (2001). Using AI and Machine Learning to Study Expressive Music Performance: Project Survey and First Report. *AI Communications* 14(3):149–162.
- Witten, I. and Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.
- Wolpert, D.H. (1992). Stacked Generalization. *Neural Networks* 5(2):241–260.