When Errors Become the Rule

A Survey of Transformation-Based Learning

Marcus Uneson

Thesis for a diploma in computer science, 30 ECTS credits,
Department of Computer Science, Faculty of Science, Lund University

Examensarbete för 30hp,
Institutionen för datavetenskap, Naturvetenskapliga fakulteten, Lunds universitet

When Errors Become the Rule: A Survey of Transformation-Based Learning

Abstract

Transformation-based learning (TBL) is a machine learning method for sequential classification, invented by Eric Brill (Brill, 1993c, 1995a). It is widely used within natural language processing (but surprisingly little in other areas).

TBL is a simple yet flexible paradigm, which achieves competitive or even state-of-the-art performance in several areas and does not overtrain easily. It is especially successful at catching local, fixed-distance dependencies. The learned representation – an ordered list of transformation rules – is compact and efficient, with clear, declarative semantics. Individual rules are interpretable and often meaningful to humans.

The present thesis has two main parts. First and foremost, we offer a survey of the most important theoretical work on TBL. It is intended to be informal but relatively comprehensive, addressing a perceived gap in the literature. Second, in a more practical part, we describe a recursive, parallelizable rephrasing, well suited for declarative languages, of a fast imperative learning algorithm proposed by Ngai and Florian (2001b). We implement and test this algorithm in the functional language Haskell.

När fel blir regel: En översikt över transformationsbaserad inlärning

Sammanfattning

Transformationsbaserad inlärning (Transformation-based learning, TBL) är en maskininlärningsmetod för sekventiell klassificering, uppfunnen av Eric Brill (Brill, 1993c, 1995a). Den används regelmässigt för många uppgifter inom automatisk processning av naturligt språk (men förvånansvärt sällan på andra områden).

TBL är en enkel men flexibel metod som når konkurrenskraftiga resultat på många områden, utan att vara sårbar för överträning. Den kan särskilt framgångsrikt fånga lokala beroenden inom sekvensintervall av förbestämd storlek. Den representation som lärs in – en ordnad lista av transformationsregler – är kompakt och effektiv, med deklarativ semantik. Enskilda regler är tolkningsbara och ofta meningsbärande för människor.

Föreliggande uppsats har två huvuddelar. I den första ger vi en översikt av de viktigaste teoretiska arbetena rörande TBL. Översikten är informellt hållen men förhållandevis omfattande; den avses därmed fylla en lucka i den befintliga litteraturen.

I den andra delen, mer praktiskt inriktad, beskriver vi en rekursiv, parallelliserbar formulering, väl lämpad för deklarativa programspråk, av en effektiv imperativ algoritm för inlärningsfasen, föreslagen av Ngai and Florian (2001b). Vi implementerar och testar denna algoritm i det funktionella programmeringsspråket Haskell.

Preface

The present work provides a survey of Transformation-Based Learning (TBL), a supervised machine learning algorithm for sequential classification invented by Eric Brill (Brill, 1993a, 1995a). It also presents an efficient training algorithm for TBL in a declarative paradigm. In our view, both TBL and declarative programming deserve wider attention. On the whole, the work presented here concentrates on the former, but declarativity resurfaces also in our discussion of how a well-designed domain-specific language may extend the expressivity and usefulness of TBL.

This is a thesis in Computer Science, rather than, say, Computational Linguistics. It is intended to be readable without much acquaintance with neither specialized linguistic terminology nor the toolbox of computational linguistics. Linguistic terminology cannot be entirely avoided, however, and some familiarity with the concepts and methods will certainly do no harm. To date, TBL has been applied almost exclusively to natural language data, and most citations must necessarily be drawn from that area. Where deemed necessary, we have tried to explain non-elementary concepts in a phrase or two in the body text; sometimes we also provide more extensive but less crucial comments in endnotes. This is hopefully enough to illustrate the inputs and outputs of a certain problem, but it is almost certainly not enough to convey the rationales behind posing it in the first place. Furthermore, in some cases, where exact understanding of the terminology might not be needed for the understanding of the algorithmic aspects, we found that further detours added more clutter than clarity. When explanations given here are insufficient, we refer to some dedicated textbook in Computational Linguistics, for instance the excellent Jurafsky and Martin (2008). Similarly, for machine learning terminology, we refer to Mitchell (1997) (which, however, has little to say specifically about classification of sequences).

This thesis began life as a chapter of a yet-to-be-finished PhD thesis; but later it ran away, got a life of its own and did not want to fit in (apparently this may happen to children of the brain as well as children of the flesh). I'd like to thank Torbjörn Lager for valuable encouragement and feedback on short notice, Mats-Eeg Olofsson for meticulous proofreading, Radu Florian for graciously providing tex sources of the FnTBL algorithm, and Karin Palm-Lindén for generous hospitality in critical moments.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Four aspects of a restaurant conversation

Consider the following hypothetical fragment of a dialogue, perhaps between a head waiter and a nervous, newly employed colleague in an overworked restaurant kitchen.

– Replace the fork on table four.
– OK. Should I apologize for the wait?
– For now it's enough to light the candle on the table.

It is a perfectly ordinary piece of language, English, in this particular case – indeed, it may be ordinary enough to be uninteresting to most people. But let us assume that we have some valid reason to study this sample – maybe we are engineers and want to build some technical application which might receive it as input, or maybe we are linguists and would like to investigate the language phenomena it exemplifies.

In any case, we probably have many more samples like it. We would like to describe them all in some abstracted way, which highlights their similarities and differences with regard to the aspect we currently happen to be interested in. For instance, in Example 1 our domain of interest is the sequence of words, and to each element of this sequence we wish to assign a *part-of-speech*, or *POS* – classes such as verb, noun, preposition, etc. We will use the notation $w_1/\textsc{pos}_1\ w_2/\textsc{pos}_2 \ldots$ to indicate such a classification (and generalize as needed).[i] [1]

(1) – Replace/VB the/DT fork/NN on/IN table/NN four/CD ./.
– OK/JJ ./. Should/MD I/PN apologize/VBP for/IN the/DT wait/NN ?/.
– For/IN now/RB it/PP 's/VBZ enough/JJ to/TO light/VB the/DET candle/NN on/IN the/DT table/NN ./.

In another scenario (Example 2), we are more interested in the turns of the dialogue itself than in the exact wordings of the utterances. In *dialogue act tagging*, we try to label entire utterances by an abstracted representation of the speaker's intentions: GREET, INFORM, REQUEST, SUGGEST, REJECT, APOLOGIZE ...

(2) – Replace the fork on table four./REQUEST
– OK./ACCEPT
Should I apologize for the wait?/YES-NO-QUESTION
– For now it's enough/REJECT
to light the candle on the table./REQUEST

Going from larger elements to very small ones, in Example 3 we instead want to study how letters correspond to speech sounds (or, with a posh term, their *graphophonemic relationships*). Here, the data consists of an alignment of each letter to its corresponding pronunciation (bottom row, in IPA[2]), with "_" as a placeholder when one-to-one-alignment is inappropriate. Similar

[1] Roman numbers refer to the notes at the end of the paper, mostly elaborating on specifically linguistic issues – here, for instance, the cryptic VB, DT, etc.
[2] IPA is the International Phonetic Alphabet, http://www.langsci.ucl.ac.uk/ipa/ipachart.html.

subtasks often appear in speech processing systems, but may also be useful for things like spelling correction or normalization of names in search queries.[ii]

(3) – *r e p l a c e t h e f o r k o n t a b l e f o u r*
ɹ i p l eɪ s _ ð _ ə f ɔ ɹ k ɑ n t eɪ b əl _ f ɔ _ ɹ

– *o k s h o u l d I a p o l o g i z e f o r t h e w a i t*
oʊ keɪ ʃ _ ʊ _ d _ aɪ ə p a l ə ʤ aɪ z _ f ɔ ɹ ð _ ə w eɪ _ t

– *f o r n o w i t s e n o u g h t o l i g h t t h e*
f ɔ ɹ n aʊ _ ɪ t s ɪ n ʌ _ f _ t u l aɪ _ _ t ð _ ə

c a n d l e o n t h e t a b l e
k æ n d əl _ ɑ n ð _ ə t eɪ b əl _

In yet another setting (Example 4), we are interested in finding exactly what part of a sentence is modified by some prepositional phrase (PP). For instance, in the example, we would like to decide whether "on the table" says something about the activity of lighting, or about the candle which is being lighted. This is the problem of *PP attachment*: in this case, attachment to the verb, $[\text{light}]_{\text{V}}$, or to its associated object noun phrase, $[\text{the candle}]_{\text{NP}}$. In the example, the latter choice turned out to be the correct one; with for instance "for now it's enough to dance the rumba on the table", it would have been the former.[iii]

(4) – $[\text{Replace}]_{\text{V}}$ $[\text{the fork}]_{\text{NP}}$ $[\text{on table four}]_{\text{PP}-\text{NP}}$.
– OK. Should I apologize for the wait?
– For now it's enough to $[\text{light}]_{\text{V}}$ $[\text{the candle}]_{\text{NP}}$ $[\text{on the table}]_{\text{PP}-\text{NP}}$.

1.2 Classification of elements and sequences

All of the tasks described are common, often needed (as preprocessing steps) in real-world applications. They all involve *classification*: given a set of observations, each one describable by some predefined characteristics (its *features*), the job is to assign each observation to one out of a likewise predefined set of discrete *classes*. A more demanding variant is *probabilistic* classification, where we need to return a probability distribution over the entire set of classes (or, less ambitiously, a ranked list of the k most probable ones).

What kind of knowledge sources do we have at our disposal, to inform such a classification? Well, if the observations are taken from a predefined set, then we might have some *a priori* knowledge, irrespective of the data set at hand. We might know what the most common class for each element is, or we might even have a probability distribution over all possibilities. If there is no such predefined domain (or if there is one, but it is not closed and thus not guaranteed to contain all new data), then we will sooner or later encounter elements that we have never seen before. However, we might still make an educated guess from a dynamic analysis of the features, which thus constitute a second knowledge source.

Actually, in Examples 1 – 3, what we are given is not a set of observations, but a set of *sequences* of observations. The classification of each element depends on its local context: its neighbours (within some not-too-wide window), and their classifications. *Sequential* classification tasks often appear when we deal with symbols ordered in time or space, such as those present in human language. In such tasks, an additional, third knowledge source – by definition – is the sequential context: which are the neighbours of the sample we are trying to classify, what are their features, and what is our (current) idea of their classification?

The example applications illustrate the varying importance of these knowledge sources:

- In part-of-speech tagging, the domain is semi-closed: most words are likely to be known beforehand, and we might well have them specified in a lexicon. Still, previously unseen words are certain to occur now and then in any real-world application, and we are much helped by being able to make intelligent guesses from dynamic feature analysis – for instance, guessing that *staycation* is a noun and *defriend* is a verb.[3] Generally, ambiguous words cannot be resolved without sequential context.

- In dialogue act tagging the domain is truly infinite, and only seldom will we listen to utterances which we have heard in their entirety before (when it does happen, it is usually short phrases: single words, or word-like groups of words: *yes*, *what's up*, *I don't know*). Thus, appropriate feature extraction is crucial. Sequential context is clearly important – the answer to a SUGGEST is much more likely to be an instance of ACCEPT or REJECT than GREET, no matter the phrasing.

- In finding letter-to-sound correspondences, we are very unlikely to encounter any previously unseen letters, Thus, feature extraction is pointless – whatever we might wish to use features for would better have been included elsewhere, as a priori knowledge. The background knowledge specifies default correspondences and sequential context can (crucially, for many languages) be used to emend these.

- In PP attachment, the domain is again infinite, but in contrast to the other examples, sequential context has no influence: the fact that a PP was attached to the verb in the previous sentence tells us nothing about the current one. Thus, intelligent feature extraction is the single source of information.

Another interesting dimension along which these examples vary is the well-definedness of the classifier range. In PP attachment, we generally have two answers to choose from, and if we look at a wide enough context, exactly one of them is correct. In finding letter-to-sound correspondences, we may argue about the best alignment, but there is usually reasonable agreement on the lexical pronunciation (at least if we consider some reference variety of the target language). POS tagging is trickier: it is only meaningful with respect to some stipulatively defined tagset, specific to a language and sometimes also to a certain data set.[4] Dialogue act tagging, finally, is less studied and

[3] New entries in the Oxford English Dictionary 2010.

[4] Of course, such tagsets are not created in a vacuum; they build on each other and for a given language, differences between them partly reflect the number of subdivisions made. Thus, a larger set can often be converted to a smaller with relative ease.

understood; thus, it has the characteristics of POS tagging to an even higher extent, with tagsets depending also on domain or setting. As can be expected, human interannotator agreement for the four tasks decreases in the order given.

1.3 The present work: Classification by transformation

The sequences we have encountered so far are all finite and not very long. On the other hand, we may have many of them – thousands, millions, or billions – and we certainly want a computer to help us. One road to automate the chore is to implement a classifier as a set of manually specified rules. For some combinations of task and data, this is actually the best solution. Restricting the data type to natural language for the sake of discussion, it is easy enough to write a letter-to-sound converter for Finnish, Spanish, or Turkish by enumerating the few necessary rules in a page or two. Much more substantial human effort was invested into the thousands of rules of the EngCG POS tagger (Karlsson et al., 1995), for a long time one of the best part-of-speech taggers for English.[5]

For most instances of sequential classification, including those illustrated in Section 1.1, this approach is simply infeasible: there are too many and too weak dependencies, and it is far too laborious to try to specify them by hand. Instead, we may choose among many reasonable machine learning approaches: decision trees, hidden Markov models, neural networks, maximum entropy models, memory-based learning, to mention just a few. These are well-known techniques in the machine learning community, and certainly good choices in many situations. Their main drawback for the tasks we are interested in is the opacity of the learned representation. For the mentioned techniques, learning amounts to filling a black, inscrutable box with estimated parameters. The exception is decision trees, which do slightly better: they give us a somewhat interpretable tree with `if...else` questions at every node. These, however, tend to be overwhelmingly many for any real-world task.

The focus of the present thesis is on yet another machine learning method: *Transformation-based learning* (TBL). It was invented by Eric Brill (Brill, 1993c, 1995a) and has been refined by him and many others since. In terms of the techniques mentioned, TBL is a hybrid: its representation involves rules, or *transformations*, but these are learned automatically from the training data. Rules are iteratively created and evaluated based on how well they deal with the current set of errors in the data; hence, the approach is often termed *error-driven*.

TBL is typically used as a supervised machine learning technique for classification of sequences, where each element is represented as a symbolic feature vector and assigned a single symbolic value in the classification. Interpretability of representation is but one out of several properties which make the method appetizing for applications involving natural language; some others are the natural ease with which it handles local (especially fixed-width) dependencies; its resistance to overtraining; and its general flexibility and adaptability to different tasks. In the next section, we will look further into these. The method itself, however, does not (and good implementations should not) make any assumptions which are valid only for natural language classification tasks. To be sure, natural language abounds with ambiguities to pit machine learning methods against, TBL or others; and it also abounds with weak, mostly local, and incompletely understood depen-

[5] EngCG later got augmented by automatically derived rules. See also Section 3.2.2.

dencies which these methods may exploit. But Life offers many examples of symbols ordered in time or space – just to mention a few, DNA and protein sequences, musical notes, suns and clouds in weather forecasts.

Furthermore, the problem characteristics described are typical, not mandatory – with the appropriate problem encoding, it is perfectly possible to apply TBL to most any classification task. In addition, many extensions have been suggested, some of which are specifically aimed at widening the method's expressive power. For instance, TBL can be rebuilt into a regressor (with real-valued output), or a probabilistic classifier (with a probability distribution over the entire set of classes as output); see further Section 3.

There are two immediate aims to the present thesis. The first and most important one is theoretical: to provide a self-contained but still relatively comprehensive introduction to an interesting machine learning technique, without much formal detail and reasonably readable also without linguistic training. Thus, we give an overview of vanilla TBL as per Brill (1993c, 1995a), and a brief survey of the most important later developments.[6]

The second aim is practical: to propose, implement, and test a declarative and parallelizable phrasing of a fast algorithm for learning TBL.

In a slightly broader sense, however, this thesis hopes to promote TBL as a general machine learning method, useful for many kinds of linguistic tasks but potentially also for other types of supervised sequential classification problems. For some reason, despite its many strong points and several successful use cases, many Computational Linguists still associate TBL with the rather restricted task of part-of-speech tagging. More significantly, outside that relatively small community, TBL is practically unheard of (cf. Table 8). In our view, TBL could well be tried on a wider array of problems, posed by Computational Linguists or others.

This thesis is organized as follows. Section 2 and Section 3, together making up the main part of the work, provide a survey of original TBL and later improvements. In Section 2, we review the original algorithm proposed by Brill (1993c, 1995a), its inherent strong points, and the relation to its distant cousin decision trees. This section also contains a brief overview of TBL uses in practical applications – many and varied, but, as mentioned, almost exclusively within the realm of Computational Linguistics and its closest neighbours. Section 3 provides an overview of the most important developments which have appeared later, aimed at augmenting the original paradigm in different directions. We review attempts to extend the range of possible inputs and the expressivity of the output; to relax the amount of supervision needed; to improve efficiency; and to ease the problem description.

Section 4 exhibits the algorithmic content of the present work: a recursive phrasing of a fast training algorithm, parallelizable and well suited for declarative languages. Section 5 concludes and hints at some possible future directions.

[6] A by-product of this work is a TBL bibliography, at the time of writing comprising around 180 entries. It is an update and slight extension of a similarly scoped bibliography (with around 75 entries) collected up until 2002 by Torbjörn Lager, `http://www.ling.gu.se/~lager/Mutbl/bibliography.html`; this is likely where the updated bibliography will end up as well.

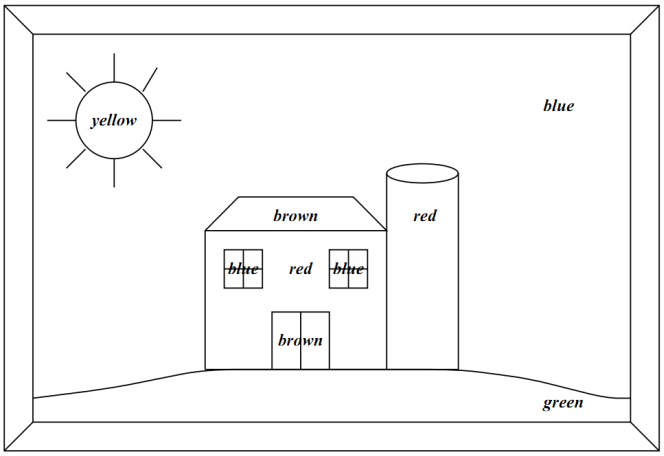


Figure 1: A barnyard scene, for transformation-based painters. From Samuel (1998b)

2 Plain vanilla Transformation-Based Learning

2.1 A painting analogy

A useful TBL picture-painting analogy is offered by Samuel (1998b), who attributes it to Terry Harvey. It generates several of the right intuitions, so we will retell it here (slightly adapted).

Consider the barnyard picture in Figure 1 (where colours have been named rather than rendered for ease of exposition). A painter comes by. As it happens, he is actually a transformation-based painter, which is mostly like any other painter, except he does not ever want to change from a smaller brush to a larger. He is also more-than-average cavalier about making mistakes, claiming that they can always be fixed later.

Our painter finds the barnyard picture and decides to reproduce it on his own canvas, as follows. First, he looks at the current state of his painting (a blank canvas) and compares it to the target, or *truth*, represented by the figure. He notes that the most efficient way to reduce the difference between his painting and the truth is to take the largest brush he has and paint the entire canvas blue. When the paint has dried, he again compares the current state of his painting to the truth. This time he finds that the easiest way to increase the similarity to the target is to take a slightly smaller brush and paint the filled outlines of a red barn. There is no need to worry about non-red details, such as windows, doors, and roof, as these will be taken care of in later stages, by smaller brushes.

And so our painter goes on. With each change of colour, he picks a finer brush and uses it with increasingly thin and precise strokes. Coarser brushes, used early on, cover a large part of the picture – they add a lot of paint, but they also make many mistakes. With later, thinner brushes less paint will be added, but also fewer errors. The final step might be to fill in the fine black lines with a very fine brush.

The main point of the analogy is that the painter uses a sequence of colour-brush pairs, in de-

scending order according to how much paint they add to the canvas. Each point of the canvas may be repainted several times; although we can be convinced that the overall result looks increasingly like the target with each application of a brush, we cannot be sure about the colour of a specific point until all brushes have been applied.

2.2 Algorithmic overview

Transformation-based learning works in much the same way as transformation-based painting. The method produces a sequence of rules, or *transformations*, ordered after impact. Early rules are very general and may change classifications on large fractions of the data, usually committing errors in so doing. Subsequent rules are more specific and may correct errors introduced by earlier ones. A single transformation rule has the general form

```
if CONDITION(x) then do ACTION(x)
```

where `x` is a data sample; `CONDITION`, sometimes referred to as "context", is a predicate[7] on attributes of `x` and/or its local context; and `ACTION` changes some attribute of `x`. The rules are induced automatically from the training data. The actual structure of `CONDITION` and `ACTION` is specified by patterns known as *templates*; thus, templates define the transformation space.

The main data structures and the data flow of TBL training (as presented in the seminal works Brill (1993c) and Brill (1995a)) are shown in Figure 2.[8] The output of the learning algorithm is an ordered sequence of transformation rules. New data (once it has been initialized in the same way as the training data, see below) can now be classified by applying this sequence of learned transformations to it. In the following, we look at the data sets; at the templates, and at the flow of control. Finally, we return to some critical points where design choices lurk.

2.2.1 Corpora

Like most machine learning methods, TBL takes as point of departure a data set of a certain size. For applications dealing with natural language, such a data set is usually referred as a *corpus* – indeed, corpora are the bread and butter of Computational Linguistics.[9]

In the TBL case, this data set is assumed to come with reference classifications, making it a *reference corpus*. Thus, TBL is a *supervised* method: it depends on the existence of some annotations which can be taken as *truth*. The annotations are usually provided (or at least proof-read) by humans. Manually annotated corpora thus represent large investments, sometimes enormous, and creating them from scratch for a single project is rarely an option.

[7] A *predicate* is a boolean-valued function, i.e., it returns true or false.

[8] Somewhat unconventionally, we avoid traditional pseudocode, focusing on flow of data rather than flow of control. This view fits better with the declarative phrasing we will use later (Section 4). For more traditional descriptions, we refer to Brill (1995a) (or almost any other paper which introduces the algorithm).

[9] A *corpus*, pl. *corpora*, is basically a sizable collection of real-world natural language data – text or speech or something more exotic, like video recordings of sign language. We use that term because it is the most common for the tasks TBL has been applied to. However, it was not created by any transformation-based deities, and the reader should feel free to replace it with "big data set" at any time. For non-linguistic applications, this may roll more smoothly off the tongue.

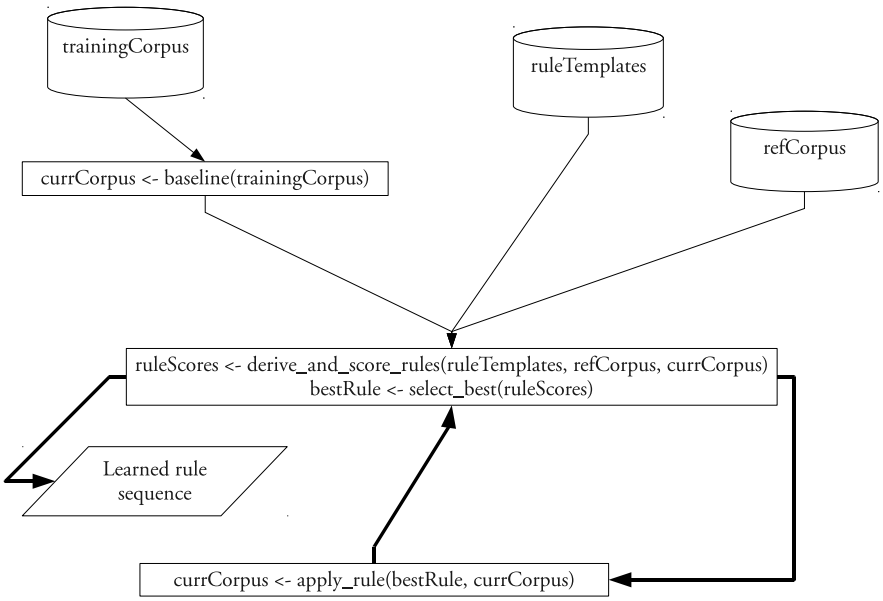


Figure 2: Data flow of TBL training, Brill's original algorithm. Main loop in heavier stroke.

A corpus can generally be thought of as a set of items which are independent of each other, by nature or by assumption, with respect to the task at hand. For instance, in POS tagging (Example 1), we can be reasonably sure that the POS of the words in the current sentence does not depend on those in the previous one. In dialogue act tagging, each utterance (Example 2) is clearly dependent on the previous one, but the dialogue acts in one conversation are likely independent of those in another. Partitioning the corpus into a set of subsets for which we can assume mutual independence is highly beneficial, for the quality of the learned representation as well as the efficiency with which it can be learned. To tag POS, we will want our corpus split into sentence-sized chunks. For the case of dialogue acts, the corpus will hopefully contain some delimiters distinguishing individual dialogues (this case also illustrates that the need of relevant and reliable annotations may go beyond individual samples). The individual classifications are often called *tags* (irrespective of their being true or not, and irrespective of the task at hand).[10] In the TBL case, we will often speak of the *training corpus*, which is essentially the reference corpus with the annotations deleted.

Template	Change A to B whenever ...
pos:A>B <- pos:C@[-1]	...the preceding word has pos C
pos:A>B <- pos:C@[2]	...the word two after has pos C
pos:A>B <- pos:C@[1,2]	...one of the two following words has pos C
pos:A>B <- wd:C@[0]	...the current word is C
pos:A>B <- wd:C@[-1] & pos:D@[1]	...the preceding word is C and the following has pos D

Table 1: Sample templates for POS tagging by TBL, original phrasing, in μ-TBL syntax and as prose.

2.2.2 Templates

A TBL problem specification uses *templates* to describe the allowable space of transformations, and this is the main way of encoding any a priori ideas we may have – domain and expert knowledge, constraints and assumptions. For many tasks, the templates are a natural, transparent, and compact way of specifying such assumptions.

A few sample templates for POS tagging are shown in Table 1.[11] As an example, if we rephrase the top one in prose we might get "change value of feature 'pos' from A to B, whenever the feature 'pos' one step to the left has value C". A, B and C are variables implicitly ranging over the domain of their respective features – over all parts-of-speech, in this case.

Templates are a core component of TBL systems. From the perspective of machine-learning theory, the template specification carries (a large part of) the *inductive bias* (Mitchell, 1997) of TBL. For instance, if we only use the single template just described, we are actually disregarding all dependencies of neighbours except the one immediately to the left (which clearly is a strong and simplistic assumption). Similarly, if we wish to enforce that all decisions be made only from left context, perhaps because the system is to be used in real-time word-for-word processing, then this assumption can be enforced by choosing the appropriate templates.

The templates in Table 1 are a subset of the 26 proposed in Brill (1995a)). In real-world scenarios, depending on the task, this number may be a magnitude less (e.g., (Brill, 1995b)) or greater (e.g., (Carberry et al., 2001)). A large number of templates may be difficult (or just tedious) to specify by hand, especially for domains we may not understand completely. We will later see several developments addressing this and other template-related problems (see Sections 2.3.4, 3.1.2, 3.1.3, 3.5).

[10]Somewhat confusingly, the term tag may sometimes be short for part-of-speech. However, in this paper we will treat the latter as a special case of the former.

[11]The syntax used for example templates and rules, here and throughout this thesis, is borrowed from the μ-TBL system (Lager, 1999b). The templates with prose descriptions in Table 1 and the derived rules in Table 6 should be self-explaining and varied enough to exemplify all constructions we will meet, but otherwise manuals are available at `http://www.ling.gu.se/~lager/Mutbl/manuals.html`

2.2.3 Flow of control

With these pieces of declarative knowledge in place, the control flows as follows (Figure 2). In an initialization step, all samples of the training corpus are classified (annotated, tagged) according to some simple baseline algorithm, perhaps giving each word its most common tag according to a lexicon or database we have, or which we extract from the reference corpus. The result is preliminarily annotated data, the *current* corpus $c_i = c_0$.

In the main loop of the algorithm (heavier stroke in Figure 2), the current corpus c_i is compared to the truth, probably uncovering some errors. For each error, we use the templates to derive rules which will correct it. Conceptually, each of the rules derived is then *scored*: the rule, call it r, is tentatively applied to (a fresh copy of) the current corpus. The result is compared to the truth, which yields some number of corrected errors (good applications, g_r) and some number of newly introduced errors (bad applications, b_r). The score of the rule f_r is normally a function of g_r and b_r – often simply $f(g_r, b_r) = g_r - b_r$.

One of the rules receives the highest score (with respect to f). It is selected, added to the list of learned rules, and applied to c_i, returning the next current corpus c_{i+1}. The main loop repeats until some termination criterion is fulfilled – for instance, when there are no more rules with positive scores.

2.2.4 Fork and wait

We return to Example 1 for a toy-sized illustration of POS tagging[iv]. Selecting a fragment of the example as a minimal training corpus and the top line of Table 1 (`tag:A>B <- tag:C@[-1]`) as a minimal template set, a trace of the learning phase is given in Table 2.

The top line contains the words wd and the second line the true parts-of-speech tags c_r, together making up the reference corpus. From the reference corpus, we get the initial current corpus c_0 by replacing c_r with the baseline. Here, we follow tradition and take as baseline annotation the most common POS for each word when context is disregarded (we probably have a lexicon for that), perhaps resulting in the c_0 shown in the third row of the table. Next, we find all the errors in c_0 by comparing it to c_r (it turns out that there is only one, for the word *wait*), and we consult the templates to derive rule candidates which might correct the errors (in this case, only one, `tag:VB>NN <- tag:DT@[-1]`). Following that, we score all our candidates by counting the errors they correct or induce, at all their sites of application (for the single rule candidate in this case, one correction and zero new errors committed). Finally, we identify the highest-scoring rule candidate with a positive score, add it to the list of learned transformations, apply it to the current corpus, and repeat. In this case, applying our single candidate from above to c_0 yields c_1. It turns out that c_1 has no more errors left to correct. Thus, there will be no rules with positive scores, and training terminates.

If we now want to use the learned rule sequence to classify new data, we just apply the same baseline and the rules after that. For instance, with another fragment of Example 1 as test data, we might get the classification in Table 3 – the rule we learned could correct the erroneously tagged *wait*.

In application, we usually won't know what the truth is. In this case, we do, and we may use this knowledge to evaluate the performance of the classifier. To emphasize that truth now only is

wd	Should	I	apologize	for	the	wait	?
c_r	MD	PN	VB	IN	DT	NN	.
c_0	MD	PN	VB	IN	DT	VB	.
c_1	MD	PN	VB	IN	DT	NN	.

Table 2: Trace of TBL learning of the rule sequence `tag:VB>NN <- tag:DT@[-1]` (see text)

wd	Replace	the	fork	on	table	four	.
c_e	VB	DT	NN	IN	NN	CD	.
c_0	VB	DT	VB	IN	NN	CD	.
c_1	VB	DT	NN	IN	NN	CD	.

Table 3: Trace of application of the rule sequence `tag:VB>NN <- tag:DT@[-1]` (see text)

used for evaluation and no longer can influence the workings of our classifier, we have renamed c_r to c_e (for evaluation). In the example, our minimal test data had a single error after the baseline annotation, and our learned rule sequence could successfully correct it.

2.2.5 Notes on design choices

In this section we take the same stroll again, but with more attention to details previously left out. Unfortunately, these are often incompletely specified in system descriptions.

- **The baseline annotation** can actually be even simpler than suggested: TBL does not really care where the first current corpus comes from. Thus, we could assign random tags, or the most common tag overall to all the samples, or even just a placeholder.

  If we do, we deliberately avoid incorporating some useful information, and we may have to pay with somewhat lower performance (at the very least, we will need more rules, and they will take longer to learn). However, as we will illustrate later, our main interest sometimes is knowledge rather than performance: the rules themselves. If so, we might prefer that rules encode everything we are able to induce from data, not just what we can add to some task-and-language-specific-performing baseline, however simple.[12] Dumber baselines may form a background against which the learned knowledge more clearly stands out.

  On the other end of the scale, the initial annotation might well be sophisticated, perhaps the output of another classifier. In this case, TBL only acts as a postprocessing step, specialized in correcting the errors of others.

[12]For a comparison, the very simple baseline for POS tagging previously described – just assign each word its most common tag – often reaches 90% correctness for inflection-poor languages such as English.

Template (`ACTION <- CONDITION`)	Change the current pos into B whenever ...
pos:>B <- pos:A@[0] & pos:C@[-1]	...the current word has pos A and the preceding has pos C
pos:>B <- pos:A@[0] & pos:C@[2]	...the current word has pos A and the word two after has pos C
pos:>B <- pos:A@[0] & pos:C@[1,2]	...the current word has pos A and one of the two following has pos C
pos:>B <- pos:A@[-1]	...the preceding word has pos A
pos:>B <- wd:W@[0]	...the current word is W

Table 4: Sample templates for POS tagging by TBL, as in Table 1 but generalized: the precondition of the current tag is moved from `ACTION` to `CONDITION`

- **The templates** suggested by Brill and repeated in most descriptions of TBL have the form "change A to B whenever condition C holds", as exemplified in Table 1. For some tasks, this phrasing facilitates a certain optimization of the training process (most useful for POS tagging of English and in any case obsoleted by later developments; see Sections 4 and 3.3.1). A strictly more expressive formulation of the templates (Samuel, 1998a; Ngai and Florian, 2001a) moves the `tag=A` part from the `ACTION` to the `CONDITION`. Table 4 rephrases the first three templates of Table 1 in this way. In addition, it gives two templates to learn useful rules such as "change any tag into B when preceded by tag A" or "change any tag into B for word W", which were not possible to express in the original formalism.

- **The generation of rule candidates** is generally best driven by examining the E existing errors in the N-sized corpus and using the T templates to derive rules which correct them. The $O(ET)$ rules thus derived are known to be at least somewhat helpful: no time will be wasted with rules which will never correct any errors, or (worse) rules which will never be triggered by the training data at all.[13]

  We will return to training efficiency issues (Section 3.3.1); here, we only note that most rules which were candidates for the current corpus c_i also will be so for the next c_{i+1}. Thus, much of what we record about the rules could be recorded once and then cached and minimally updated between iterations. This observation underlies several of the faster techniques we will see later.

  A point of ambiguity in deriving rules, unfortunately not often specified in the description of practical implementations, is how to handle templates which refer to non-existing positions in the sequence. For instance, consider the following two-sample corpus, while again learning from the `tag:A>B <- tag:C@[-1]` template:

  (5)

	truth	*a*	*b*
current	corpus	x	b

[13] Conceptually, though, we note that we could arrive at the same set of good candidates and uncountably many more useless ones without looking at the data, by blindly instantiating all templates with all possible values of the (discrete) feature domains.

...	-2	-1	0	1	...	n	n+1	n+2	...	Position / Strategy
...	$\emptyset$	$\emptyset$	$\emptyset$	w_1	...	w_n	$\emptyset$	$\emptyset$	...	a) Template not applicable OOB
...	ϵ	ϵ	ϵ	w_1	...	w_n	ϵ	ϵ	...	b) Single OOB token
...	$\emptyset$	$\emptyset$	$\triangleright$	w_1	...	w_n	$\triangleleft$	$\emptyset$	...	c) Boundary markers only
...	ϵ	ϵ	$\triangleright$	w_1	...	w_n	$\triangleleft$	ϵ	...	d) Boundary markers and OOB token
...	$\triangleright\triangleright\triangleright$	$\triangleright\triangleright$	$\triangleright$	w_1	...	w_n	$\triangleleft$	$\triangleleft\triangleleft$	...	e) Position-unique token

Table 5: Handling templates with out-of-bound (OOB) accesses, five example strategies. From the perspective of a template, a sequence $w_1 \dots w_n$ may behave a) as if surrounded by nothing; or b) as if preceded and followed by an infinite number of OOB tokens (here ϵ); or c) as if it had a special left boundary marker (here $\triangleright$) at position 0 and a special right boundary marker (here $\triangleleft$) at position $n + 1$; or d) a combination of b) and c); or e) as if surrounded by an infinite number of position-unique tokens (here $\triangleright$, $\triangleright\triangleright$, etc.; $\triangleleft$, $\triangleleft\triangleleft$, etc.) in both directions. See also text.

One answer is to stipulate that the template does not apply if it refers to any position outside the sequence at hand; in the example, learning would thus immediately terminate. Another way, suggested by Curran and Wong (1999) but probably used by many, is to extend the vocabulary with special tokens – perhaps a single special symbol for any access outside the bounds, or just one marker for the left boundary and one for the right, or a combination of the two, or a unique token for each position where access was attempted. Table 5 spells out these possibilities; there are many variations. Any of them except the first would allow us to learn a rule which corrects the last error in Example 5, for instance as `tag:x>a <- ◁@[-1]`. For a corpus of mostly long sequences – or for a corpus where we only have a single sequence, perhaps because we haven't bothered to split the data into mutually independent sequences in the first place – the difference is negligible. With many short ones, as is common in natural language processing, it may be of importance.

For certain applications, we might have a particular interest in high-accuracy rules. An often used approach is to compute the accuracy of a rule candidate, $acc(g_r, b_r) = g_r/(g_r + b_r)$, and provide an *accuracy threshold* a, $0.5 < a \leq 1.0$, which the rule must surpass to be further considered.

- **The scoring** of a rule candidate r employs some user-defined idea of goodness f, normally a function of the of the number of good g_r and bad b_r changes that r will bring about if applied. The straightforward $f(g, b) = g - b$, as used above, is an obvious candidate for f. We note, however, that, as far as the basic TBL algorithm is concerned, any f which fulfills $f(g, b) > 0$ iff $g > b$ is good enough. Put into words, the only requirement is that any rule with a positive score will decrease the total error count (and thus the algorithm must terminate); and any rule that decreases the total error count will have a positive score (and thus all positive rules may be learned). With this observation, it is conceivable for f to

introduce a bias – for instance, we might simply generalize the previous scoring function to $f_\alpha(g,b) = g^\alpha - b^\alpha$. With $\alpha = 1$, we retrieve the original function. With $\alpha < 1$, the scoring function will reward high-accuracy rules (for instance, $f_{0.9}(100,10) > f(200,100)$). Similarly (but probably less useful), with $\alpha > 1$, it will reward rules with large impact on the corpus (for instance, $f_{1.1}(200,100) > f(120,10)$).

So far we have ignored the number of *neutral* applications n_r, where r just changes one error into another. It is also mostly ignored in the literature, or found to be of little importance when mentioned. For example, in experiments described by Lager (1999b), the rules learned by the two different scoring functions $f_{\alpha=1} = g_r - b_r$ and $f'_{\alpha=1} = g_r - b_r - n_r$ were not significantly different. Nevertheless, the task at hand may dictate valid reasons for letting f depend also on n_r – perhaps the main interest lies with the rules learned, rather than in the number of reduced errors, and we prefer low-impact rules to high-impact ones with the same or even higher $g_r - b_r$. One should also note that most reports on the empirical behaviour of TBL describe the special case of POS tagging on English, which has several properties not necessarily shared by other tasks (such as initial accuracy on the order of 90%, a tagset size on the order of 100, few dependencies on larger distance than 2 or 3). Other questions, perhaps yet unasked, may have 0% or 50% as initial accuracy, tagset sizes of 2 or 2000, and much wider sequential dependencies.[14]

Of course, more generally speaking, any scoring function which reflects our ideas of the task at hand by quantifying the gain of applying a candidate rule could be used. Such a function could well take other inputs than (g_r, b_r, n_r) – say, sequence length, current correctness, or estimated classification probability (cf. Section 3.2.1). For instance, we might be more interested in maximizing the number of correctly tagged *sequences* rather than the number of correctly tagged sequence *elements*. If so, we will weight rule applications in almost-correct sequences, whether they correct or introduce errors, differently from those in sequences with more errors.

More elaborate scoring schemes seem to be little explored (but see Section 3.2.3). One should note that some of the faster training algorithms we will meet later (Section 3.3.1) assume simple scoring rules and will not work with more complex variants. In addition, with exotic scorings, the user assumes all responsibility of defining a terminating process. If a user-defined scoring function cannot be used directly, this could be done by predefining a maximum number of rules learned.

See also the discussion on scoring in a multidimensional learning setting, Section 3.1.1.

- **The selection of the best rule** uses *greedy search*: whenever a choice needs to be made, the locally optimal one (here, the highest-scoring rule) is picked, without worrying about its impact on future choices. This simplification represents a major pruning of the search space, which is indeed immense. Somewhat simplifying an example from Curran and Wong (2000), if we consider only rules where conditions and actions read the same attribute and conditions all refer to all positions in a fixed window of width C, then with a tag vocabulary of size $|V_t|$ and templates of the type exemplified in Table 4, we get $|V_t|^{C+1}$ different

[14] See also Endnote v.

transformation candidates for each rule. Learning P of these in the optimal order involves $P!\,|V_t|^{(C+1)P}$ possibilities. Pruning is clearly necessary for other than toy-sized examples.

Greedy algorithms, however, will produce the globally optimal solution to a problem only if it exhibits certain properties, in particular *optimal substructure*: an optimal solution to the problem contains optimal solutions to its sub-problems. TBL does not have this property and it is not difficult to find problem instances where greedy rule selection will fail to produce the globally best solution. For instance, consider the following five-sample corpus (for clarity, with only attribute 'tag' shown), and learning with the same single template as before (`tag:A>B <- tag:C@[-1]`):

(6) *truth* *a b a c a*
current corpus a b d b d

In this case, the greedy algorithm will learn the single rule `d>a <- tag:b@[-1]`, which will correct two out of the three errors, and then terminate. The optimal solution, however, is to start with `b>c <- tag:d@[-1]`, which only corrects a single error but allows two other rules, `d>a <- tag:b@[-1]` and `d>a <- tag:c@[-1]` (in any order), to take care of the remaining two errors.

In practice, however, the greedy approach seems to work well over a wide range of applications, and apparently (probably due to the already problematic training times) nothing else has been tried. We also note that, although a complete scrambling of the learned rule list certainly will hurt, TBL is not generally very sensitive to minor reorderings (for the common case study of English POS tagging, see Curran and Wong (2000)). Clearly, rules which do not alter each others' context are independent and can be reordered without consequence. Generally speaking, later rules are more often independent, if nothing else because they have fewer application sites and apply in more specific conditions. Similarly, a better baseline will produce fewer and more independent rules.

- **The application of the best rule**, once it is selected, could happen in several ways, and as pointed out in Brill (1995a) the difference is crucial for rules where the action happens to influence the condition. Either we can first identify all the places where the rule is to apply and then apply it to them all at once (delayed application); or we may allow earlier changes to influence later: first check the first position to see if the condition holds, apply the rule if it does, and repeat at the next position (immediate application). In the latter case, there is no particular reason why first and next should be taken in left-to-right order: we could just as well start from the right (or, to be sure, in any other of the $N!$ ways, like left-to-right but odd positions before evens; but let us restrict discussion to the less far-fetched options).

  Brill (1995a) points out the differences but takes no obvious stand. In our view the intelligibility and declarativity of the rule representation may suffer badly with immediate application. For instance, modifying an example of Brill's, suppose we have the rule `tag:A>B <- tag:A@[-1] & tag:A@[1]` and a nine-sample corpus, a sequence of nine 'A' tags. After applying the rule in the three different ways, we get

```
A A A A A A A A A (current corpus)
```

```
A B B B B B B B A (delayed application)
A B A B A B A B A (immediate application, left-to-right)
A B A B A B A B A (immediate application, right-to-left)
```

Now consider the application of the same rule, but to a ten-sample corpus instead:

```
A A A A A A A A A A (current corpus)
A B B B B B B B B A (delayed application)
A B A B A B A B A A (immediate application, left-to-right)
A A B A B A B A B A (immediate application, right-to-left)
```

That is, if our corpus contain an odd number of samples, we get 100% coincidence between left-to-right and right-to-left application of the same rule; but if it contains an even number, we get 0%. This is hardly the way to achieve clear, declarative rule semantics. In our view, immediate application is a bad idea for much the same reason that self-modifying code is, and delayed application is the only option if we are interested in interpretability of the induced knowledge.

- **The stopping condition** is normally a *score threshold*; when there are no more rules reaching this threshold, training is terminated. However, since the rules are ordered in terms of impact, we will get a meaningful result also if training is interrupted after, say, k rules or h hours. A well-chosen score threshold will minimize learning of spurious rules (and training time) without compromising performance. See also the discussion in Section 2.3.4.

2.3 Free with vanilla: TBL strong points

Albeit simple, the variant of TBL as we have seen so far exhibits several desirable properties. Below we expand on these. The shortcomings of standard TBL and some attempts to remedy them will be the topic of Section 3, as well as extensions to ease its restrictions on input and output.

2.3.1 Interpretability of learned representation

Sometimes our main interest may lie in the learned representation itself: the declarative knowledge that a classifier induces, rather than its actual performance when this knowledge is applied. Statistical representations, essentially black boxes of numbers, are generally not very informative in this respect.

By contrast, the interpretability of the representation is high for rule learning algorithms, and even more so if they can provide some kind of relevance ranking of the learned rules. TBL does this very efficiently, by outputting its rules ordered after expected impact.

We are not, of course, claiming that interpretability amounts to cognitive or psychological validity – whatever human processes are employed in sequential classification, they are unlikely to employ hundreds of rules (or, even more unlikely, millions of conditional probabilities or other statistical parameters). But the sequence of rules is understandable enough that it might encourage inspection, modification, experimentation, and occasionally give a new insight.

To illustrate, Table 6 gives the first few learned transformation rules from a widely used English corpus and provides examples from the same text. All of the rules cited are general enough that they could be suggested as rules of thumb for human part-of-speech taggers (at least inexperienced ones, still trying to internalize and abstract the definitions). The accuracy figures simultaneously hints at the reliability of the rules thus learned (however, the actual scores of the rules are unimportant, as they depend on the size of the training corpus and the specific tagset chosen). For instance, rules 4-5 may in such a setting be paraphrased

> "Uncertain if you look at a noun or a verb? Well, here is some help. If the previous word is a modal, such as *can, should, may*, then what you see is almost certainly a verb. If one of the two previous words is a determiner (of which the most common cases are articles, such as *a, an, the*), then you probably look at a noun."

The two things to note here is that these rules make perfect sense, and that they were extracted automatically.

For ease of exposition, we have used the same 26 templates as in Brill (1995a). Note, however, that the alternative templates mentioned in Section 2.2.5 might permit even stronger generalizations: rules 1 and 4 may possibly be merged, and the same goes for rules 2 and 8.

Although of great practical value, the main point of POS tagging rules is seldom to provide insights we did not have before. An example with different priorities is lexical stress and word accent prediction for Swedish. Somewhat simplified, each Swedish non-compound, non-inflected word has a particular syllable which bears (main) *stress*. The stressed syllable is associated with one out of two possible *word accents*, corresponding to pitch contours of the voice. Precisely on what syllable stress is placed and which of the two accents (`Accent I` or `Accent II`) the syllable will have is mostly predictable from orthography, but not trivially so. These are good circumstances for automatic detection of interesting rules. Due to reasons of space we cannot be very detailed, but the main setup and results of a minimal such study are summarized in Table 7. Note the high value of the accuracy threshold, typical where the contents of rules are more interesting than their scores.

The words in the study were all non-compound and non-inflected, and several rules would have looked different otherwise (for instance, rule 4 is not true for common inflected forms such as 'pulled', *talat* 'spoken', *huset* 'the house'). However, the results are already enough to reject the popular misunderstanding[15] that Swedish basically has stress on the first syllable. As can be seen, almost all of the rules are conditioned on suffix, not on prefix, and stress placement is more reliably done from the end (this fact is reflected as negative indices in the accent/stressed syllable column).

2.3.2 Compactness of learned representation

A major aspect of interpretability, but also of great help in practical implementations (cf. Section 3.3.2), is the fact that the learned representation is very compact. As an extreme example, Brill (1994) presents a TBL system for unknown-word-guessing (i.e., assigning the most likely part-of-speech to words not in the system lexicon – for instance, reviving the examples from Section 1.2,

[15] Just to be clear: this is a popular misunderstanding, not a professional one.

ID	Score	Acc	Rule	Example
1	98	0.99	pos:'VBP'>'VB' <- pos:'MD'@[-1,-2,-3]	government will/MD decide/VBP>VB on rates
2	51	1.00	pos:'VBP'>'VB' <- pos:'TO'@[-1]	compelled to/TO serve/VBP>VB the interests
3	42	0.82	pos:'VB'>'VBP' <- pos:'NNS'@[-1]	interest rates/NNS continue/VB>VBP to undermine
4	42	1.00	pos:'NN'>'VB' <- pos:'MD'@[-1]	would/MD cost/NN>VB the Treasury far more
5	41	0.81	pos:'VB'>'NN' <- pos:'DT'@[-1,-2]	a/DT leading force/VB>NN in the field
6	41	0.67	pos:'IN'>'WDT' <- wd:that@[0] & pos:'NNS'@[-1]	alternative fuels/NNS that/IN>WDT don't pollute
7	38	0.97	pos:'VBN'>'VBD' <- pos:'NP'@[-1]	Dexter/NP reduced/VBN>VBD its interest in 1987
8	28	0.60	pos:'NN'>'VB' <- pos:'TO'@[-1]	a contract to/TO supply/NN>VB equipment

Table 6: Sample POS tagging transformations, the first few learned from English text (60,000 words of financial news from Wall Street Journal), with example phrases from the same source. Templates as in Brill (1995a). For clarity, the examples only show the tags matching the corresponding rule. Tags appearing: VBP: verb, present, not 3rd person singular; VB: verb base form (infinitive); MD: modal verb (e.g, *can*, *will*, *should*, *may*); TO: the word *to*; NNS: plural noun; NN: singular or mass noun; DT: determiner; NP: proper noun; IN: preposition or subordinating conjunction; WDT: wh-determiner (e.g., relative *which*).

ID	Score	Acc	Accent@Syll	<-	Condition	Example
1	1453	1.00	accI@[-1]	<-	len:mono	bil, jobb
2	891	0.99	accII@[-2]	<-	suff:a & len:bi	väska, springa
3	707	0.99	accI@[-2]	<-	suff:isk	mystisk, arabisk
4	551	0.96	accI@[-1]	<-	suff:t & len:tri+	desperat, kolorit
5	482	0.99	accI@[-2]	<-	suff:ra	parera, konstruera
6	373	0.99	accI@[-2]	<-	suff:ing	etablering, parkering
7	351	1.00	accII@[-2]	<-	suff:ig & len:bi	konstig, stenig
8	320	0.98	accII@[-3]	<-	suff:are	hammare, visare
9	234	0.96	accI@[-1]	<-	suff:on & len:tri+	konvention, reklamation
10	197	0.99	accII@[1]	<-	suff:de & sylls:3	yttrande, leende
11	195	1.00	accI@[-1]	<-	suff:sm	kubism, kataklysm
12	185	0.97	accI@[-2]	<-	suff:er & len:bi	vacker, smicker
13	176	0.97	accI@[-1]	<-	suff:i & sylls:4	pedanteri, fotografi
14	166	0.98	accI@[2]	<-	pref:för	förvisa, förmå

Table 7: Sample stress and word accent rules for Swedish, the first few transformations learned from 33390 syllables in 12396 non-compound, non-inflected entries of a Swedish pronunciation dictionary (Hedelin et al., 1987). 35 templates with the following features: relative syllable position (left/right); word length (redundantly) both in numeric and syllabic representation (#syll[able]s, mono/bi/tri+); pref[ix]/suff[ix] of length 1..6. Accuracy threshold = 0.96. Positive indices count syllables from the beginning of the word, negative from the end. For instance, rule 7 says that bisyllabic words ending in *-ig*, such as *konstig, stenig*, will have accent II on the next-to-last syllable. See also text.

guessing that *staycation* is a noun and *defriend* a verb). He quotes comparable performance for his 148-rule system and an existing statistical unknown-word-guesser with 100, 000, 000 parameters.

2.3.3 Competitive performance

The rules induced by TBL may be interesting or at least interpretable reading to humans. Sometimes, however, we don't really care about such fringe benefits, but only about classification performance. As a stand-alone classifier, TBL generally reaches competitive results on a wide range of tasks, and state-of-the art for some. For other tasks, it lags somewhat behind the best statistical classifiers (at the time of writing often Support Vector Machines). However, the trend in recent years is that the best overall results are reached not by single, stand-alone classifiers, but by combining several of these into *ensemble learners*. Such systems generally gain from diversity in their constituents, and indeed TBL often contributes diversity. We return briefly to classifier combination in Section 2.5.2.

2.3.4 Resistance to overtraining

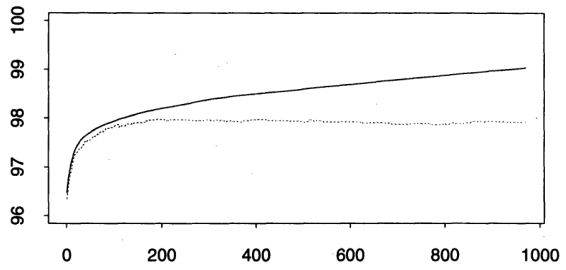
For most machine learning algorithms, a major problem is *overtraining*: the learned representation describes random error or noise and thus fails to generalize outside the training data. For instance, it is very easy to detrimentally overfit decision trees, and careful measures must be taken to avoid it (e.g., by growing the trees to completion and then back-prune; or by performing some statistical analysis before deciding that a node should be further split (Mitchell, 1997)).

TBL, by contrast, comes with an implicit *ranking* of the learned rules – they are automatically ordered after expected impact. This fact is the main reason for the method's remarkable insensitivity to overtraining.

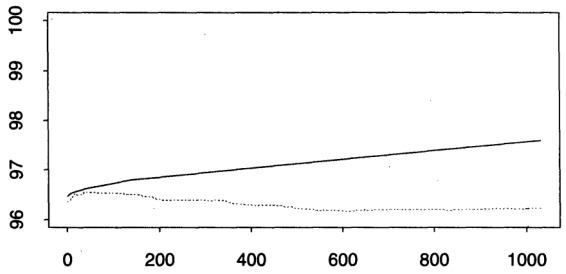
To be clear, TBL does overtrain – that is, if left to train until conclusion with a very low score threshold, it will learn a large number of spurious, low-impact rules with no prospects of generalization (most of which will apply to a single site in the training corpus). But this trail of irrelevant rules does not significantly influence overall performance (in either direction). In case we prefer not seeing them anyway (perhaps because our main interest are the relevant rules only, and not overall classifier performance), an efficient filter is just to raise the score threshold.[16] More sophisticated approaches are also conceivable, for instance by combining several TBL classifiers; we return to this topic in Section 2.5.2.

Perhaps more disturbingly, irrelevant rules may conceivably emanate from unfortunate choices of templates. Ramshaw and Marcus (1994) briefly investigate this issue. They report on experiments with training in the presence of a template which can safely be assumed to be irrelevant (such as "the POS of the word 37 positions to the left of the current"). When used in isolation, such a template naturally yields a large number of spurious rules; but when combined with relevant templates its influence is largely neutralized (Figure 3). Their conclusion is that the presence of irrelevant templates will have little impact, if only they are mixed with relevant ones. This is

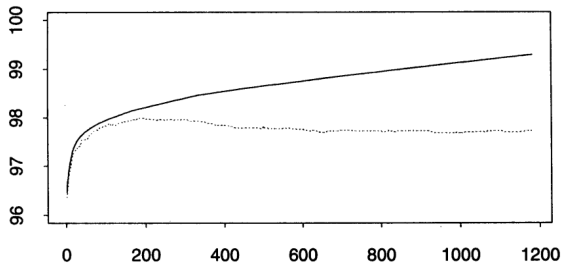
[16] Exactly where to put it will depend on task, intention, corpus, tagset size etc. and may need some experimentation, but it need not be very high. As a comparison, Brill recommends a score threshold of 2 for his POS tagger designed for English (typical tagset sizes $50 - 150$), on the corpus sizes of the mid-90's (10^5 words).



(a) 7 relevant templates



(b) 1 irrelevant template



(c) 7 relevant and 1 irrelevant templates

Figure 3: Learning curve in the presence of relevant and irrelevant templates. POS tagging of Greek, 120kW. Solid line shows performance on training set, dotted line performance on test set, as a function of the number of learned rules. From Ramshaw and Marcus (1994).

useful knowledge in particular for cases where we are uncertain on what templates best catches the dependencies of the problem – except training time, there is little risk in specifying all possible templates we can think of (see also Section 3.1.2).

We note that Ramshaw and Marcus (1994) are brief on their results, and in any case, more investigation into TBL overtraining behaviour would be welcome, for differently sized data and tag sets and for other templates and tasks.[17] In the words of Manning and Schütze (2001), it appears to be more of an empirical result than a theoretical one, and this judgment still seems valid in 2011.

[17] As stated in several places in this thesis, there is a preponderance for certain tasks and languages in the literature of

2.3.5 Real-world objective function

TBL is an example of *error-driven* learning: the objective function[18] that we wish to minimize is the number of errors; and the evaluation function we use to choose between different solution candidates is also typically (a monotonic function of) the current number of errors. Thus, our way of evaluating competing rules or rule sets directly optimizes a measure in which we have a practical, real-world interest, and differences in this measure correspond to differences in performance.

By contrast, many classifiers use some less straightforward evaluation function (e.g., information gain for decision trees) which is only indirectly related to the classifier performance. While the correlation of course is designed to be strong, it is not necessarily be perfect.

The real-world relevance of the objective function also allows TBL training to be recast as an optimization problem. This view allows the application of typical optimization techniques to the task. For instance, Wilson and Heywood (2005) use genetic algorithms to minimize the error function between reference and current corpus (see further Section 3.1.3).

2.4 TBL vs. Decision Trees

As several authors have noted (Ramshaw and Marcus, 1994; Brill, 1995a; Manning and Schütze, 2001), TBL and decision trees (Breiman et al., 1984; Quinlan, 1993) have several commonalities. A prototypical decision tree (DT) outputs a set of yes/no-questions which can be asked about a sample to get at its classification, very similar to the context part of a transformation rule (even more so with templates of the form described in Section 2.2.5). The questions may refer to attributes of the sample being classified, but also to those of its neighbours. The TBL baseline simply corresponds to a default classification.

The theoretical and practical differences between the two are important, however. For one thing, DTs synthesize complex questions on any subset of the available attributes, whereas vanilla TBL requires the format of the rules to be specified by the templates.[19] For another, arguably more important, DTs have no (easy) way of saving away current hypotheses, and thus, the questions cannot refer to intermediate predictions. Thus, in DTs (as in most other ML classification schemes) classification is performed once and never changed. By contrast, TBL makes several passes through the data and later predictions may be improved based on earlier. As Ramshaw and Marcus (1994) put it: "decision trees are applied to a population of non-interacting problems that are solved independently, while rule sequence learning is applied to a sequence of interrelated problems that are solved in parallel, by applying rules to the entire corpus". In fact, Brill (1995a) proves by induction that ordinary TBL rules are strictly more expressive than DTs: precisely due to the possibility of leveraging intermediate results, there are classification tasks which can be solved by TBL but not by DTs. These tasks may be of marginal importance, but it is not difficult to find real-world cases where a solution with transformations is much more concise and natural than an equivalent DT. Brill exemplifies with tagging a word whose left neighbour is *to*,

Computational Linguistics, and sometimes "we have answered this question" is an inappropriate abbreviation for "we have answered this question for English". See also Endnote v.

[18] Sometimes the terms "objective function" and "evaluation function" are used as synonyms. Here, we take the former to describe the purpose to be fulfilled by any learner (generally minimizing or maximizing something), whereas the latter refers to the quality of any particular solution, with respect to some representation.

[19] However, see also Section 3.1.3 and 3.1.2.

which may be an infinitival marker (*to eat*) and a preposition (*to Scotland*). In the first case, *to* is an excellent cue for verbs, in the second a good one for nouns. However, it may well be that the tagging of *to* itself into these two classes is unreliable. If so, TBL can automatically delay the exploitation of this cue until it has been more reliably established by other, intermediate rules, working on current predictions. By contrast, a DT which exploits the same information is quite complicated and will likely contain duplicate nodes.

The difference between DTs and TBL can also be viewed as one between *stateless* and *stateful* classification. State is a mixed blessing, the discussions on which fill up many a book in an average computer science library.[20] Here, we will only note that a DT at work has no notion of order or time, whereas TBL rule sets are strictly ordered,[21] with current predictions representing state.

Brill (1995a) mentions other practical advantages of TBL over DTs: better resistance to problems with sparse data and overtraining (a TBL rule has access to the entire training data when being evaluated; by contrast, a DT recursively splits its training data in smaller subsets at each node); the possibility to postprocess other output, and the transparency of the objective function. All of these have been treated in more detail in previous sections.

As a final point, Hepple (2000) notes that for POS tagging, rule learning generally starts from a very good baseline. This means that relatively few samples are retagged by rules, and fewer still are retagged more than once. On these observations he bases two major simplifying assumptions: *independence* (rule interaction is ignored, so that early rules are not allowed to change context for later ones – in other words, all rules are learned in the context of the initial annotation); and *commitment* (any particular sample is changed by at most one rule). The rationale for the assumptions is the massive gains in training time that they allow, and so we will revisit them in that context (Section 3.3.1). They are mentioned here because with these assumptions, transformation rules actually become a form of decision trees.

2.5 TBL in practice

2.5.1 TBL as a standalone classifier

TBL is a flexible and adaptable method, as witnessed by the large number of tasks it has been applied to. Table 8 lists a selection of tasks and associated central references. It is intended to be suggestive rather than exhaustive (see the bibliography mentioned on p. 10 for a somewhat more serious attempt at completeness) but should cover the most diverse cases. The task names listed in the table may not be very informative for readers who are not familiar with linguistic terminology. Indeed, some may be cryptic even to those who are: not all of the tasks listed are urgent or even relevant for all languages, due to factors such as cross-linguistic variation, differences in writing systems, and availability of appropriate data.[v] We refer to standard textbooks such as Jurafsky and Martin (2008) for the interested;[vi] however, the details aren't very important. The main point is that practically any level of language, spoken or written, is replete with symbolic classification tasks based on sequential information in the local context; and with clever encoding, many problems can be made to fit this mould, some of them perhaps not obviously sequential. Indeed, in recent years the idea that *all* useful linguistic mappings can be cast as classification

[20] The classic Abelson and Sussman (1996) is but one of them.
[21] Brill (1995a) describes a transformation list as a processor, not a classifier.

Task	Ref
Part-of-speech (POS) tagging	Brill (1993c, 1995a)
Unknown word guessing	Brill (1994); Mikheev (1997)
Text chunking	Ramshaw and Marcus (1995)
Prepositional phrase attachment	Brill and Resnik (1994)
Parsing/grammar induction	Brill (1996)
Morphological disambiguation	Oflazer and Tür (1996)
Spelling correction	Mangu and Brill (1997)
Word segmentation	Palmer (1997)
Message understanding	Day et al. (1997)
Dialogue act tagging	Samuel et al. (1998)
Prosody prediction	Fordyce (1998)
Ellipsis resolution	Hardt (1998)
Word sense disambiguation	Dini et al. (1998)
Document format processing	Curran and Wong (1999)
Grapheme-phoneme conversion	Bouma (2000)
Grammar correction	Hardt (2001)
Handwritten character segmentation	Kavallieratou et al. (2000)
Regression	Bringmann et al. (2002)
Hyphenation	Bouma (2003)
Named entity recognition	Florian et al. (2003)
Compound segmentation	Park et al. (2004)
Disfluency detection	Kim et al. (2004)
Semantic role labeling	Williams et al. (2004)
Word alignment	Ayan et al. (2005)
Information extraction	Nahm (2005)
Biomedical term normalization	Tsuruoka et al. (2008)
Human activity recognition	Landwehr et al. (2008)

Table 8: Sample TBL applications

[light the candle] on [the table]

(a) light/I the/I candle/I on/O the/I table/I

light [the candle] on [the table]

(b) light/O the/I candle/I on/O the/I table/I

Figure 4: NP chunks with encodings for *light the candle on the table* after Ramshaw and Marcus (1995), before and after applying the transformation `I>O <- wd:the@[1]`. I and O mark inside and outside an NP, respectively. A third tag, B, is used for the first word in a new chunk directly following another one, as in *give*/O *the*/I *king*/I *his*/B *throne*/I.

.

tasks, either as disambiguation or as segmentation, has gained support (Roth, 1998; Daelemans, 1995). This assumption has opened several new fields to machine learning algorithms – TBL is just one out of many.

We will only give two examples here, chosen to illustrate diversity in problem encodings. A few more can be found elsewhere in this text (e.g., Section 3.5) and, of course, many others in the references of Table 8.

In *NP chunking*, a string of words is to be divided into non-overlapping, non-recursive subsequences corresponding to noun phrases (NPs).[22] Figure 4 shows an example and a way (due to Ramshaw and Marcus, 1995) of casting the task as a classification problem very much like POS tagging, with the same type of rules learned.

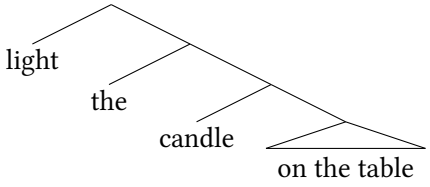
The more intricate problem of *parsing* requires the encoding of recursive structures. The straightforward although somewhat un-linguistic solution proposed in Brill (1993b) is shown in Figure 5.

2.5.2 TBL in company: Ensemble learning

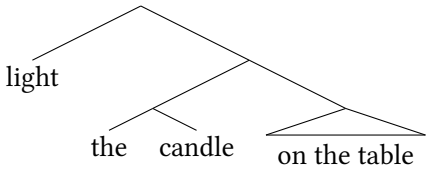
Like any other classifier, in addition to its stand-alone use exemplified in the previous section, TBL may be used in combination with other classifiers. We have noted (Section 2.2.5) that a particularly simple way of doing this is to use TBL as a post-processing, error-correcting step: since the algorithm assumes an initial-state annotation, but cares little where it comes from, it could just as well take the output of any other classifier as this initial baseline (e.g., Ruland, 2000; Wu et al., 2004). Besides (hopefully) boosting overall performance, TBL rules learned in this way may provide some error analysis of the underlying classifier, summarizing the systematic errors it makes.

There are more sophisticated approaches than this to *ensemble learning*, i.e., combining multiple classifiers into one in the hope of reaching better predictive performance than could be obtained from any of the constituent models alone. For instance, a typical *committee classifier* might combine multiple ground-level classifiers ("committee members") with a higher-level classifier

[22] There is no single "correct" solution to this problem; practical needs will decide how to treat modifiers such as possessives, prepositional phrases, or relative clauses. Of course, whatever the decision, it should be consistently held to.



(a) (light (the (candle (on (the table)))))



(b) (light ((the candle) (on (the table))))

Figure 5: Parse trees with encodings for *light the candle on the table* after Brill (1993b), before and after applying the transformation `delete a left paren to the right of a determiner`. Not shown are some behind-the-scene machinations to cater for parenthesis balancing (see Brill (1993b) for details).

.

("president"), with the idea that the different committee members have complementary strengths and weaknesses, and the job of the president is to learn when to trust whom. Ensemble learners generally gain from being composed by classifiers with complementary strengths (Kuncheva and Whitaker, 2003). To put it another way, ensemble learning is pointless if all component classifiers make the same errors.

Classifier combination largely falls outside the scope of this thesis; we will here only note that TBL is different enough from most statistical sequential classifiers to make it a worthy member of many committees (e.g., Brill and Wu, 1998; Florian et al., 2003; Xin et al., 2006).

2.5.3 TBL in TBL company

The aim of classifier combination of the type described in Section 2.5.2 is chiefly to boost performance: combining the complementary strengths of several classifiers into a new and better. This can be done also with TBL as the single base learner. For instance, a number of slightly different classifiers can be induced from repeated resamplings of equally-sized subsets of the training data (bootstrap aggregating, or *bagging* (Breiman, 1996)). At classification time, each TBL classifier is applied independently, and its output taken as one vote for that class. After counting votes, the majority wins. Santos et al. (2010) claim to be the first to try this approach, for their ETL algorithm (Section 3.1.2). The resulting combined classifier is tried on text chunking, named entity recognition, and semantic role labeling. It shows substantial improvement on ETL alone, especially for the semantic role labeling task, and achieves competitive or close to state-of-the-art results for all of them.

Another idea for combining the knowledge of several TBL classifiers, apparently unexplored, is to merge the learned rule sequences, rather than weighing together the classifications they emit. Basically, rules which are ranked high by all of the classifiers are highly likely to be relevant; but rules ranked low or (in particular) learned only by a few are likely not. The rule sequences might be combined by some rank combination measure and appropriate thresholds.[23]

As pointed out previously, TBL may overtrain in the sense that many of its late learned rules are spurious and irrelevant; its resistance to overtraining lies in the fact that such rules are automatically ranked as low-impact – they won't influence performance much in either direction. Thus, we might not expect a great performance boost from weeding out irrelevant rules. However, if our main interest is the knowledge distilled, that is beside the point.

[23] Finding combinations of ranked lists is a common problem in fields such as Information Retrieval and Computational Biology.

3 Adding flavour: TBL extensions

We have showcased several desirable properties of the basic TBL method, but there is certainly room enough for development, improvement, and extension. In the following, we briefly survey the most important new ideas for TBL: extending the domain of learned hypothesis (Section 3.1: to multidimensional learning, learning templates jointly and automatically, and learning without any templates at all); extending its range – its predictions – (Section 3.2: to probabilities, sets, and regressors); increasing efficiency (Section 3.3: in training and application); decreasing the need of supervision (Section 3.4); and facilitating declarative problem specification by domain-specific languages (Section 3.5).

3.1 Extending the hypothesis domain

3.1.1 Multidimensional learning

Many real-world applications involve more than one subtask – perhaps one classifier to assign parts-of-speech to each word and another for finding binary branching trees. A simple way of doing such combined tasks is to put the appropriate classifiers in a pipeline, resulting in a strictly feed-forward system, where the outcome of task k cannot influence the outcome of task $k-1$. For very different and/or truly independent tasks, this may be the best way. A variation is *beam search*, where we keep $n > 1$ hypotheses from early steps and defer complete disambiguation until later.

However, when two tasks A and B are somewhat dependent we may benefit more from *multitask learning* (Caruana, 1997). Intuitively, if we can expect that solutions to A may provide useful information for solving B and vice versa, then it would be better not having to impose an ordering of these tasks on the system, but rather solve them in parallel. In that way, the easy cases (of both A and B) may provide information for the more difficult ones (of both A and B). Two such interrelated tasks are POS tagging and *chunking* – dividing sentences into larger, non-overlapping units (somewhat like parsing without tree structures; we saw a simpler special case of chunking in Figure 4). Here, easy cases are for instance POS and chunks which can be reliably predicted directly from word forms.

Indeed, multitask learning on well-chosen representations may be worthwhile even when we are not really interested in solving all of the tasks – learning POS by jointly learning chunks is conceivably easier than learning POS alone.

One way to set up such a joint classifier is to simply create a new derived feature as the concatenation of the features we wish to combine. However (in analogy with computing joint probability distributions) this may cause problems of data sparseness with many types of data. Unusual joint values will have unreliable estimates, and joint values which do not occur in the training data will never be predicted.

A better way for many purposes is to let the tasks share a common representation and let each classifier work on it simultaneously. TBL is well suited for joint learning in this way. Florian and Ngai (2001) point out the few changes needed to the main algorithm, mainly small modifications to the scoring function f. As stated before (Section 2.2.5), to guarantee termination, f needs to assign positive values only to rules which actually decrease the current error count. This requirement

is easily met also in a multidimensional setting, by just letting f taking more than one field into account:

$$f(r) = \sum_{s \in C} \sum_{i=1}^{n} w_i \cdot (S_i(r(s)) - S_i(s))$$

Here,

- C is the set of training samples (the training corpus);
- r is a candidate transformation rule to be scored;
- $r(s)$ is the sample which is the result of applying r to sample s;
- n is the number of fields (tasks);
- w_i is an optional weight or priority (see below) for task i;
- $S_i(s)$ is an indicator of the current classification of sample s for task i: 1 if correct, 0 if not.

The weights w_i could be used to manually assign priorities to each subtask. They might also be initialized from training data based on the counts of correct and incorrect sample classifications figures for each field – say, as constants for the entire training session, or as per-iteration values to be recalculated and reassigned after each rule application. Differently weighted attributes are apparently an unexplored option.

To conclude, in the TBL case, multidimensional learning can be seen as a generalization of one of the previous key arguments for the method: we can use intermediate results to guide later predictions. The multidimensional addition is that such intermediate results may refer to more than one feature.

3.1.2 Automatic template learning

Templates are the main tool for embedding and encoding domain knowledge in TBL. On the other hand, they can be tedious or difficult to produce. The tedium may to a large extent be alleviated by automation (e.g., the template compiler described in Lager (1999b)). As we have seen, overtraining is seldom a problem in TBL, so unnecessary templates will mostly only affect training time. A greater concern is insufficient domain knowledge: for less well-understood problems, it may be difficult to know where to start. Learning templates automatically then becomes an attractive option.

Curran and Wong (2000) envision "evolving templates that change in size, shape and number as the learning algorithm continues", starting from few and simple templates which are gradually refined into (or replaced by) more complex ones in a data-driven fashion. They present no implementation, but they show empirically that the number of conditions in the templates and their specificity (e.g., words rather than tags) increase during learning – simple templates with few conditions are most efficient early on, but later more complex templates tend to pay off better. However, the task providing their data is again POS tagging for English; it would be desirable to see their claims corroborated for other tasks and data sets.

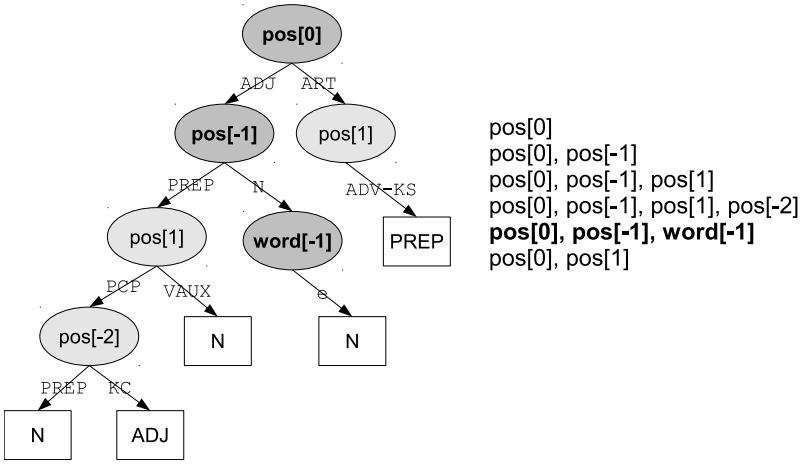


Figure 6: Template extraction from decision tree. First, the leaves and the labels of the learned tree are discarded. In the resulting subtree, each path from the root to an internal node corresponds to a template (right). Redrawn from Santos (2009).

Less abstractly, Milidiú et al. (2007) implement a genetic algorithm (Mitchell, 1997) for automatic template learning. Their system shows impressive performance but their reported setup suffers from slow training; with the TBL algorithm as the fitness function, training must happen for each individual of the population, for each generation. For anything but the smallest feature sets, this rapidly becomes intractable.

A recent approach (Santos and Milidiú, 2009; Santos, 2009), dubbed "Entropy-based Transformation Learning" (ETL), instead constructs templates from a decision tree (DT) trained on the task at hand. A DT can be thought of as a series of yes-no questions asked about an object to be classified, with the questions ordered in terms of descending Information Gain (IG) (see Mitchell (1997); Quinlan (1993), cf. also Section 2.4). The main idea behind ETL is that the features which are addressed by the DT-induced questions on task X are likely to make up a good set of TBL templates for X. Each path from the root to any internal node of the learned DT corresponds to a specific series of questions and thus a specific set of features (Figure 6) – i.e., a template.

Some care needs to be taken to avoid typical DT training problems. The standard algorithms will strongly favour high-dimensional features – for instance, word identity, rather than part-of-speech. The version of ETL described in Santos (2009) tries to control this by sorting the values of a high-dimensional feature in decreasing IG order and replace all except the z top-scoring ones with a dummy value, where z is a parameter of the algorithm. Furthermore, as discussed in Section 2.4, decision trees are inherently stateless and has no notion of a "current" classification. For the purposes of template generation, ETL solves this by introducing the true value of the classifications in the context (but not for the current object).

Santos and Milidiú (2009) report excellent results for ETL on a number of tasks (Fernandes et al.,

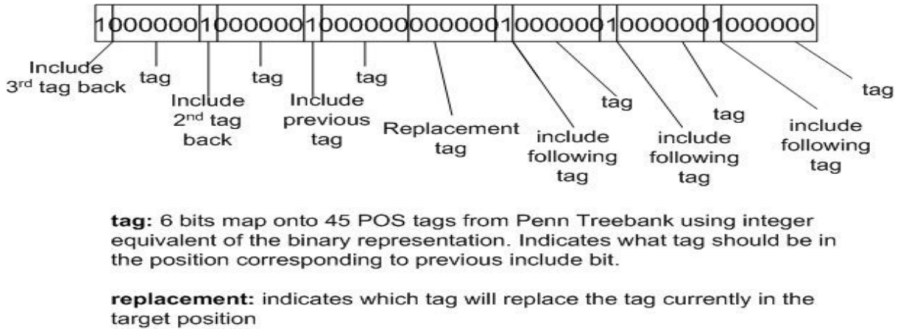


Figure 7: Bit string encoding of one particular rule in TBL by genetic algorithm. Any subset of tags within a window of current position ± 3 can be encoded, replacing a manually specified template set. From Wilson and Heywood (2005).

2010; Milidiú et al., 2008, 2010; Santos et al., 2008, 2010). The approach has apparently so far only been used by this single group of researchers; a more thorough evaluation will have to wait.

3.1.3 Template-free TBL

An even more radical approach than automatic template learning is TBL without any templates at all. Wilson and Heywood (2005) suggest a genetic algorithm (Mitchell, 1997), which (in contrast to the previous genetic approach by Milidiú et al. (2007)) does away with templates entirely. Instead, an entire sequence of rules corresponds to one individual in the population; and one individual is a collection (384, in the experiment) of rules represented as a fixed-length bit strings (Figure 7).

Although an undeniably novel approach, several critical choices are tuned to the specifics of POS tagging (in particular, for English). Thus, the rather small tagset size is part of the encoding assumptions: it is not clear how the method would handle a larger target set or many-valued features (as well may be crucial in other tasks). It is also unclear how it would respond to a lower-scoring baseline. The paper does not exemplify any rules learned; that would otherwise have made interesting comparisons with standard TBL.

TBL without manually specified templates seems in particular well-motivated when the choice of templates is problematic, perhaps due to poor understanding of the domain. This is hardly the case for POS tagging in English, so template-less genetic algorithms may have more potential for other tasks. Performance is also quite a bit lower (the authors report 89.8% from a baseline of 80.3%; Brill's original algorithm scores 97.0% on the same corpus).

Bach et al. (2008) suggest template-free learning by an exhaustive search through the power set of a user-defined feature space, where each subset of features is allowed to generate one rule for each error. The paper leaves quite a few questions unanswered and we won't discuss it further.[24]

[24]For one thing, it seems contradictory on how output rule lists are actually compiled. There is also no information

3.2 Extending the hypothesis range

3.2.1 Predicting probabilities: Soft classification

The vanilla output of a TBL classifier is one class per sample, with no information as to the uncertainty of the class chosen. These are *hard decisions*, committing the system to a single possibility with no information as to the certainty of the choice.

Probabilistic systems, by contrast, make *soft decisions*, providing confidence measures for each classification. Confidence measures are useful for many purposes, and indispensable for some. For instance, in *ensemble systems* (Section 2.5.2), the member classifiers may disagree; it is then very useful to know how much they are willing to insist. In *active learning*, the idea is to minimize manual effort; such systems use confidence measures to identify the most uncertain (and thus, to the system, most informative) samples and ask the annotator for their classification. For some applications hard decisions are simply inappropriate. Larger, multi-component systems (e.g., speech recognizers) are generally built from many smaller modules which all deal with probabilistic input and output (say, as probability distributions, or as ranked candidate lists).

Two notable attempts have been made to add enhance TBL with probabilistic classification. They share the basic idea of splitting the training data into equivalence classes: all the samples which have been tagged X for reasons Y are considered together. Then probabilities can be estimated for each equivalence class by standard means (e.g., maximum likelihood estimation, probably with some smoothing).[vii]

Florian et al. (2000) gives an algorithm for transforming a learned rule list into an equivalent decision tree, and then taking the leaves of that tree as equivalence classes. They note that the equivalence classes thus constructed will tend to vary a lot in size. In particular, with a good baseline, the equivalence class of samples to which no rules apply at all can easily make up most of the corpus, and the probability estimates of that class will be close to those arrived at without any learning at all. To remedy that, the learned tree is treated as a (highly accurate) prefix, whose paths are grown further with standard decision tree learning methods (Quinlan, 1993).

Santos and Milidiú (2007) instead construct equivalence classes from the baseline classification and rule traces – for instance, all samples that had initial classification NN and were later touched by rules 11, 57 and 88 form an equivalence class of their own. The problem of unevenly sized classes is solved by subdividing on manually specified auxiliary features. The authors claim a significant improvement over Florian et al. (2000) on comparable tasks. Their method arguably adheres better to the inherently stateful TBL paradigm (cf. Section 2.4), but seems to involve more task-specific hand-coding (the auxiliary features to specify) as well as more assumptions on the data (the relevance of the initial classification).

In any case, probabilistic TBL is an interesting subject, with several unexplored paths. For instance, it is not clear what influence a dumber initial classification (forcing more samples to be touched by some rule) or a higher accuracy threshold (inducing a bias for more accurate rules) would have on the quality of the estimates learned with one method or the other.

on running time, nor any performance comparisons with ordinary-templated TBL – only modest error correcting scores on good or excellent baseline systems are listed.

3.2.2 Predicting sets: Constraint grammar

We have already mentioned (Section 1.3) that one of the most successful POS taggers for English is the originally hand-written rule system EngCG (Karlsson et al., 1995). CG stands for *Constraint Grammar*. Very briefly (and somewhat simplified), such a system starts with initializing a candidate set of tags for each word to the set of all possible tags for that word (as found in a lexical lookup, or in a reference corpus). Then it traverses a list of rules, each formalizing some requirement, or *constraint*, on the solution.[25] Tags not fulfilling a certain constraint can be removed from the candidate set (the last tag should not be removed, however).

Ideally, only one correct tag will remain when all rules have been applied. In reality, of course, some samples will have had the correct tag removed in the process and others will still have more than one possibility left. Thus, the evaluation measure of a CG-like system needs to be adapted accordingly. It is usually calculated on precision and recall instead of percentage correct – typically as the F1 score borrowed from Information Retrieval.[26]

A great advantage of CG-like systems is that they can exploit negative information, say "a modal verb is never followed by an adjective" as easily as positive. On the other hand, a major drawback is the reliance on every single rule being correct: in contrast to other variations of TBL, it is not possible to let later rules correct the mistakes of earlier ones. If the correct tag is erroneously removed from a sample, there is no magic black hat from which it could be reproduced. Thus, this style of learning will strongly emphasize rules with perfect or almost perfect accuracy.

CG rules were originally specified by hand, but naturally, several attempts have been made to extract them automatically from a corpus (e.g., Samuelsson et al., 1996). The question of interest here is how to do so by TBL. Lager (1999b) shows that this can be achieved by conceiving of transformations not as replacement rules but rather as set operations. He incorporates this style of learning in his μ-TBL system. Especially important are *reduction* rules, which remove an element from a set if it is not the last (otherwise, they do nothing). An example of such a reduction rule is "reduce the current tag set of a word with tag VB if the word immediately to the left is uniquely tagged as DT". In the μ-TBL formalism this rule would read `pos:red vb <- unique pos:dt@[-1]` and it would probably be derived from the template `pos:red A <- unique pos:B@[-1]`. Lager (2001) explores this learning style for the case of POS tagging. The results are promising, but prohibitively slow for scaling up: from a modest-sized 240kW corpus with 15 templates, the TBL Constraint Grammar learner described in Lager (2001) learns 4866 rules in three weeks (!). A more efficient algorithm for CG-style TBL is clearly crucial if the method is to gain wider usage.

A final disadvantage of CG-style systems is that their output is made up of sets, and using it as part of a larger system thus presupposes that interfaces of later links are prepared to handle unranked sets as input. In an increasingly probabilistic view on knowledge induction and transfer, this assumption may be problematic.

[25] The "Grammar" part of the term is less informative and reflects the early uses of the approach – it is certainly conceivable to use it on problems outside grammar or POS tagging.

[26] http://en.wikipedia.org/wiki/F1_score

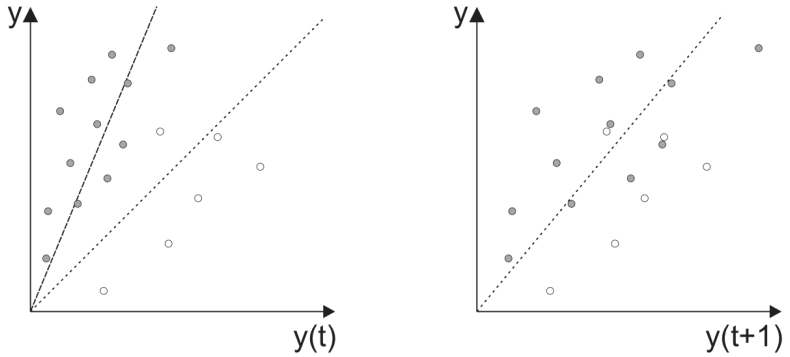


Figure 8: Applying a TBR rule r = if $c(s_i)$ then $y_i(t+1) \leftarrow a + b * y_i(t)$. The subset of the data (grey) for which the condition part c of r is true undergoes a linear transformation $y(t+1) = a + b * y(t)$ towards the (correct) diagonal. From Bringmann et al. (2002).

3.2.3 Predicting numbers: Transformation-based Regression

An innovative extension of the prediction range from discrete classes to real numbers is suggested by Bringmann et al. (2002): TBR, short for Transformation-Based Regression. The end product looks somewhat like a regression tree (Breiman et al., 1984; Quinlan, 1993), but, as we have seen (and the authors point out), TBL gains additional expressivity by making multiple passes through the data. Thus, intermediate results can be leveraged, and predictions can be improved based on current predictions, rather than set once and for all.

TBR deals with *continuous* transformations. A TBR rule has the following form:

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

Here,

- $s_i = (x_i, y_i)$ is the i^{th} sample in the data set;
- x_i is a vector of attribute values describing the i^{th} sample;
- y_i is a numerical value to be predicted for the i^{th} sample;
- $c(s_i)$ is a predicate on the sample s_i and/or its neighbours (just like before);
- $y_i(t)$ denotes the value y_i at iteration t;
- a, b are parameters chosen to minimize the error after the transformation.

A rule only applies to (i.e., transforms linearly) the subset of the data for which c holds (Figure 8). The algorithm for rule instantiation and scoring needs some modification for TBR and the stopping criteria is handled somewhat differently; we refer to Bringmann et al. (2002) for details.

Two drawbacks are difficult to avoid when leaving the world of discrete classifiers and entering the continuous regression domain. First, overtraining becomes a much more pressing issue. This is a standard problem, with several common solutions. The one proposed by Bringmann et al. (2002) is similar to that often used in pruning decision trees: partition the available training data into a development and a validation set, and then prune the tree/truncate the list to the smallest one within a certain error (e.g, one standard error) from the best one on the validation set. Second, arguably a greater loss, TBR sacrifices a large part of rule interpretability (cf. Section 2.3.1): the meaning of rules in the form above is difficult to grasp. Although the top scoring rules may make some intuitive sense, rule effect is generally cumulative, and later rules which linearly transform the output of earlier ones make no more sense to humans than does any other black-box method.

It should be noted, however, that these downsides are no worse for TBR than for other regressors. Performance-wise, the authors report TBR to be competitive with state-of-the-art regression learning algorithms on difficult sequential regression tasks (predicting segment durations from phonological features, and predicting three main musical parameters of expression in piano playing).

3.3 Improving efficiency

3.3.1 Efficiency in training

One major disadvantage of the TBL algorithm of Section 2.2 is the very slow training. Brill's original implementation (in C) was generously made available on the web early on and was often used for POS tagging as a preprocessing step in other applications. However, although it contained clever optimizations, it commonly needed days of training time. For instance, Ngai and Florian (2001b) report 38 hours training for the simple task of POS tagging on a 1 million word corpus, which still is small by the standards of 2011.[27]

The time complexity of the TBL training phase is difficult to specify. The worst-case analysis is clear, but rather absurd: a corpus of size N where all initial tags are incorrect and the correct tags are all unique. Thus, no possible rule will ever correct more than a single error. Given T templates, we get $O(NT)$ rules to choose from in each iteration,[28] and since there are N errors to correct and no rule corrects more than one of them, we will have to learn $O(N)$ rules. Thus, training takes $O(N^2T)$ time, and no indexing schemes will improve the situation. Of course nobody would want to try machine learning on such a data set, but it at least establishes an upper bound. We could also consider $O(N)$ a lower bound – we won't need the templates to find out that there are no errors to correct, but we clearly need to look at the entire corpus.

The average case (which is of course what we are really interested in: a corpus with some but not complete regularity and a meaningful baseline which does not get everything wrong) is more difficult to pinpoint. The most informative statistic is the number of learned rules R, which, however, will vary a lot with the task at hand: it will grow with the size of the corpus N, with

[27] More complex tasks are rather worse; and the situation is further exacerbated when a TBL system is used as one of several base learners in iterating ensemble learners.

[28] Templates may contain alternative positions, such as `pos:A@[1,2]` meaning some pos tag A at position 1 or 2. Given a specific sample with erroneous classification, such a template will usually generate more than one rule that corrects it. We will disregard that in the discussion here. At any rate, considering the maximum context width a constant, $O(T)$ templates will still generate $O(T)$ rules per application site.

the initial number of errors E, and with the number of templates T, but not independently and not very predictably with either.

Roughly, with $O(E)$ errors in the current corpus, learning a single rule with the greedy but otherwise naive algorithm of Section 2.2 will involve identifying $O(ET)$ candidates in time $O(N + ET)$, scoring them in $O(ENT)$, picking the best in $O(ET)$, and applying it in $O(N)$. The expensive scoring step can be optimized by sorting the candidates according to their descending good application counts g_i, keeping a best score s_b, and interrupting the scoring as soon as the g_i falls below s_b. This optimization, present in most implementations, is very efficient for the first few rules, when scoring can be interrupted early, but grows increasingly inefficient as rule scores drop, when a large fraction of the rules will have to be investigated. As a result, this method typically finds the initial rules in $O(E_0T)$ but then slows down to $O(E_RNT)$ as training progresses (where R is the number of rules learned and E_i is error count after applying i rules).

An early attempt to speed up training is represented by Ramshaw and Marcus (1994). They proposed an elaborate indexing scheme and minimal state updates between iterations. In an initial phase, the corpus is traversed linearly twice, once to identify $O(ET)$ positive rules for each error and each template; once to identify the negative application sites of those rules. As a result, each rule in effect has a list of (pointers to) the samples it may be applied to, and each sample a list of (pointers to) the rules that can be applied to it. The list of rules is scored and sorted and the top scoring rule is selected. When applied, each of its sites of application can be accessed in constant time, and all the rules affected by the change of corpus state can in turn be efficiently updated. The initial phase takes $O(ENT)$, then the application of each of the R rules starts at $O(ET)$ for the first one and approaches $O(T)$ as the number of samples changed C_r (at most $2g_r$) approaches the constant score threshold. Thus, once the index is built, no linear searches need to be performed in the corpus.[29] Unfortunately, this method consumes $O(NT)$ memory (or possibly $O(ET)$, with judicious filtering of rules which have higher negative than positive score). This makes it infeasible for most real-world tasks with more than a handful of templates.

Ngai and Florian (2001b) propose what is essentially a much more memory-friendly simplification of the algorithm of Ramshaw and Marcus (1994). Instead of storing $O(E)$ pointers for each rule in $O(ET)$ space, they only store the good/bad count (g_r, b_r). At the cost of a linear search per iteration, the rule-to-score list then becomes much smaller and the sample-to-rule list superfluous. We defer the details of this algorithm to the next section, where we rephrase it for a declarative paradigm; here, we only note that after building and indexing the initial rule list the top scoring rule can be found in constant time, applied in $O(N)$, and the state updated in $O(N + CTw)$, where w is the width of the widest template context and C is the number of application sites. Towards end of training, this approaches $O(N + STw)$, where S is the score threshold.

An entirely different approach to improving training efficiency is Monte Carlo (MC) sampling of the rule space (Samuel, 1998a; Carberry et al., 2001). In each iteration, for each error discovered in the current corpus, a fixed number of templates K are chosen randomly and used to generate correcting rules. In effect, we get an unbiased sample of (the currently interesting part of) the rule

[29] Previously unseen rules may occasionally be created in the neighbourhood of application sites, to correct errors introduced by the application. Scoring such rules will need linear corpus scans. In our experience, creation of new rules can usually be disabled without performance loss.

space. Usually the highest-scoring rule will be the best one, and almost always one of the best. As we have mentioned (p. 20), small reorderings of good rules are usually not crucial: when several rules have about the same efficiency, either they correct the same error, in which case we could pick any of them; or they correct different errors, in which case they are usually independent and their local ordering relative to each other is of little consequence. Running time does not depend on the number of templates, and thus MC sampling is particularly useful where the number of templates is high – for instance, when the feature-space is inherently high-dimensional and/or the domain is not (yet) completely understood.

For POS tagging, Carberry et al. (2001) show impressive speedups (two orders of magnitude) with no concomitant drop in performance. There might be tasks where performance is more impacted. For instance, low-scoring rules yield smaller samples and thus are scored with less certainty than high-scoring, for well-known statistical reasons; also, if there are very many rules with approximately the same score, true ordering requirements among them are more likely to be violated. Thus, the approach may be more problematic when the bulk of the error correction is catered for by a large number of low-scoring rules. At any rate, MC sampling often deserves consideration when non-determinism is acceptable.

Yet another approach are the explicitly stated independence assumptions proposed by Hepple (2000), that we met in Section 2.4: *independence* (no rule interaction with respect to the condition part of a rule – all rules are learned in the context of the initial annotation); and *commitment* (at most one change per sample). Hepple describes two algorithms based on them (ICA and ICP: Independence, Commitment, and either Append or Prepend, depending on whether the rules are applied in the learned order or in reverse). The simplified learning shows very impressive speedups (several orders of magnitude).

Clearly, the validity of the IC assumptions depend very much on the problem. Some problems are not sequential; some are sequential, but can reasonably be treated as if they weren't; some cannot be handled as sequential without major information loss. Furthermore, the error rate of the baseline is crucial. Fewer errors of course mean fewer and lower-scoring rules to learn; thus, it is generally far between rule application sites and rules will not interfere. Poorer baselines mean more rule interaction. For POS tagging, the performance loss is slight (and probably outweighed by the possibility of training on much larger data sets). For several other tasks, IC is significantly worse (Ngai and Florian, 2001b).

3.3.2 Efficiency in application

The naive application of a set of learned rules to new data is by sequential substitution, one rule at a time. Thus, it takes $O(RKn)$ time to apply R rules requiring context of size K to a input of size n. This may be good enough for some purposes, but when the rules for instance describe a preprocessing step for data-intensive applications (say, information extraction), it is too slow.

Roche and Schabes (1995) apply finite-state algebra to the rule sequences. They show that a single learned TBL rule r_i may be regarded as a non-deterministic transducer t_i. It follows that the entire set of rules corresponds to the composition of such transducers, which in itself is a non-deterministic transducer T. Not all non-deterministic transducers can be determinized. However, the authors show that T, resulting from composing fixed-width context TBL rule sequences, actually can (except perhaps for practical limitations). This yields a transducer with $O(n)$ perfor-

mance in the size of the input, with very low constants – in fact, the run time of their POS tagger is dominated by the data transfer time from disk.

3.4 Widening the bottleneck: Unsupervised learning

Classification tasks are by definition supervised – they depend on annotations provided by humans beforehand. For instance, the original TBL algorithm presupposes for the learning step a reference corpus where each element in each of the sequences, whatever they might represent, has been tagged with its true class. Such data sets are costly to construct. If one can get away with less expensive resources, it is of course useful. Brill (1995a) proposes an extension to TBL POS-tagging, where the baseline annotation of a word is created by simply listing all of its possible tags in some fixed but arbitrary order, disregarding context. For instance, the baseline annotation of *the* will likely be the singleton `DT`; but *export* will perhaps list `VP|VBP|NN` (present-tense, non-3rd person singular verb, as in *Cubans export sugar*; infinitive verb, as in *Cubans want to export more sugar*; or noun, as in *Cuban export of sugar will increase*). The system treats this three-item set as a single tag like any other except that it can be decomposed for scoring (see below). The templates look just like before, for instance `A > B <- tag:@[-1]`.

Rule scoring has to be adapted: since we do not have access to truth, we cannot simply count errors corrected. We can, however, score rules according to how efficiently we expect them to reduce uncertainty. This can be done in many ways. Brill (1995a) uses the following formula[30] to score rules derived from the template `A > a <- C` (change A to a in context C, where A is an ambiguous tag and a is one of its possibilities):

$$\text{score}(A, a, C) = U(a)\left[\frac{U_C(a)}{U(a)} - \max\frac{U_C(Y)}{U(Y)}\right]$$

Here, $U(X)$ is the total number of words in the corpus uniquely tagged X, $U_C(X)$ is the number of words uniquely tagged X in the context C, and Y is a variable ranging over the ambiguous tags of A except a.

In the example above, with the specific context c = `dt@[-1]` (determiner one step to the left), this amounts to

$$\begin{aligned}
\text{score}(\text{VB|VBP|NN}, \text{VB}, c) &= U(\text{VB})\left[\frac{U_c(\text{VB})}{U(\text{VB})} - \max\left(\frac{U_c(\text{VBP})}{U(\text{VBP})}, \frac{U_c(\text{NN})}{U(\text{NN})}\right)\right] \\
\text{score}(\text{VB|VBP|NN}, \text{VBP}, c) &= U(\text{VBP})\left[\frac{U_c(\text{VBP})}{U(\text{VBP})} - \max\left(\frac{U_c(\text{VB})}{U(\text{VB})}, \frac{U_c(\text{NN})}{U(\text{NN})}\right)\right] \\
\text{score}(\text{VB|VBP|NN}, \text{NN}, c) &= U(\text{NN})\left[\frac{U_c(\text{NN})}{U(\text{NN})} - \max\left(\frac{U_c(\text{VB})}{U(\text{VB})}, \frac{U_c(\text{VBP})}{U(\text{VBP})}\right)\right]
\end{aligned}$$

Terms of the form $\frac{U_C(X)}{U(X)}$ range from 0 to 1, and thus the second factor ranges from -1 to 1. Given the context of the example, any real-world corpus will find it close to -1 for the first two cases and close to 1 for the third.

[30] although differently presented, in our opinion less clearly

Word	POS	True chunk	Initial chunk
O	art	i	i
aluno	noun	i	i
esqueceu	verb	o	o
o	art	i	i
caderno	noun	i	i
de	prep	i	i
caligrafia	noun	i	i
amarelo	adj	i	o
em	prep	o	i
casa	noun	i	i

Figure 9: Chunking the Portuguese sentence *O aluno esqueceu o caderno de caligrafia amarelo em casa* 'The student left the yellow calligraphy notebook at home'. Example from Santos and Oliveira (2005).

With only four templates, Brill (1995a) reports rather impressive 96% (from a baseline of 90%). However, he simplistically assumes a complete lexicon. Aone and Hausman (1996) extend the method for Spanish to cope with unseen words, and Becker (1998) generalizes and parameterizes the approach, allowing compositional templates and different scoring schemes. For instance, the scoring scheme above disregards all ambiguous contexts, which presupposes that there are enough of the non-ambiguous ones; but one might also want to include ambiguous tagging in the scoring (and then probably weighted according to degree of uncertainty).

We may extrapolate this idea to Constraint Grammar-style rules (Section 3.2.2) by considering *all* tags as sets, possibly singletons, and have the transformations implement set operations rather than replacements (e.g., reduction: remove a tag from the possibilities). This allows much easier exploitation of negative information and might be a way to explore unsupervised transformation-based learning also for the Constraint Grammar paradigm. The search space is very large, though, and the problematic training time of TBL-CG needs to addressed first.

3.5 Abstracting the problem: Template compositionality and DSLs

Santos and Oliveira (2005) apply TBL to chunking (p. 29) in Portuguese, but find that traditional templates are too inflexible to catch important lexical relations. As example, they give the relation between preposition *em* 'on' and the verb *esqueceu* 'left' (Figure 9). With the verb *esquecer* 'leave', a prepositional phrase headed by *em* is generally an adverb, does not belong to the object noun phrase, and should be tagged o. As a remedy, they propose a widened "constraint atomic term", where templates can express things like "condition on value x of feature X of the current word's closest neighbour which has value y for feature Y". By way of example, they give a template of the form (where [x;y] indicates a closed interval from x to y, inclusive):

`word[0;0](pos=prep) word[-2;-10]`[31] `(pos=verb)`

From the data in Figure 9, such a template would allow the system to induce the rule

`word[0;0](pos=prep)=em word[-2;-10](pos=verb)=esqueceu -> chunk=O`.

This rule should be read "if pos[0]=`prep` and word[0]=*em*, and, for the first item in the closed interval [-2;-10] where pos=`verb`, word=*esqueceu*; then change the value of feature `chunk` to `O` for the target item". The authors then suggest algorithms to implement their new extension.

We mention this idea not because we find it a good one, but to illustrate a point. In our view, the suggested extension is at best an acceptable solution to the wrong problem, a problem which was phrased too specifically to begin with.[32] Much better than adding one ad-hoc feature or another to the syntax of TBL templates is to make the templates themselves form a little language, with values, abstractions, and combinators as befit the domain. Such a language is known as a *domain-specific language* (DSL); in the words of van Deursen et al. (2000), a DSL is a "small, usually declarative, language that offers expressive power focused on a particular problem domain." An appetizing alternative to implementing a DSL from scratch is to extend some suitable base language with domain-specific constructs. In this way, all features of the base language can be reused. More importantly, so can the users' knowledge of it. Such an extension can range in complexity from some preprocessing step up to a complete embedding (an *embedded* DSL (Hudak, 1996, 1998)).

To our knowledge, there is only one attempt at a DSL for TBL: the template language of the µ-TBL system (Lager, 1999b). A moderately complex template description for a particular task (dialogue act tagging) in this system is given in Figure 10.

µ-TBL is implemented in the logic programming language Prolog, and its template language is just a Prolog extension (by preprocessing). Figure 10 (as well as the examples below) may not be very informative for readers unfamiliar with that language,[33] and we must anyway omit some details on data format, argument passing to the auxiliary predicates, etc.[34] The point should be clear, however: some features are given directly in the data representation, while others are computed dynamically by auxiliary predicates of arbitrary complexity, but the templates use both kinds and can't tell the difference between them. If we were to trade space for time by including some of the computed features in the data representation instead, the templates wouldn't notice. Furthermore, just as important, the templates themselves are compositional; they are ordinary Prolog terms and can be assembled and disassembled by the standard mechanisms of the language.

We will return to µ-TBL in Section 5. Here, we will only take the Portuguese example as a minimal coding exercise in a DSL (fictive, but perfectly conceivable and not much beyond that of Figure 10).

[31] Starting two to the left (-2) rather than with the left neighbour (-1) leaves a spot for the direct object, as in *don't leave keys on the table.*

[32] We unfortunately find this a common pattern. Non-compositional problem encodings (often with too much focus on performance) tend to create monolithic systems with hard-coded choices: hard to extend, hard to experiment with, hard to understand.

[33] Good books on Prolog, in suggested reading order, are Clocksin and Mellish (1994); Covington et al. (1997); Bratko (2001); Sterling and Shapiro (1994); O'Keefe (1990)

[34] see `http://www.ling.gu.se/~lager/Mutbl/mutbl_system.html`

```
%%% DATA REPRESENTATION
s(1,speakerA).
da(1,acknowledge).
da(acknowledge,ready,1).
u(1,[ehm, right]).

s(2,speakerA).
da(2,acknowledge).
da(acknowledge,instruct,2).
u(2,[you,start,at,the,caravan,park]).

s(3,speakerB).
da(3,acknowledge).
da(acknowledge,acknowledge,3).
u(3,[ok]).
%...


%%% TEMPLATES
da:A>B <- da:C@[-1].
da:A>B <- da:C@[-1,-2].
da:A>B <- da:C@[-1] & da:D@[-2].
da:A>B <- u_first:W@[0].
da:A>B <- u_mem:W@[0].
da:A>B <- u_bigram:W@[0].
da:A>B <- s:C@[0].
da:A>B <- s:C@[0] & u_mem:W@[0].
da:A>B <- s_change:C@[0] & u_mem:W@[0].
```

```
%%% AUXILIARY PREDICATES
u_mem(Position, Word) :-
    u(Position, Words),
    member(Word, Words).

u_first(Position, Word) :-
    u(Position, [Word|_]).

u_bigram(Position,(Word_i, Word_j)) :-
    u(Position, Words),
    nextto(Word_i, Word_j, Words).

s_change(Position, Change) :-
    (   s(Position, Speaker),
        Position1 is Position-1,
        s(Position1, Speaker)
    ->  Change = false
    ;   Change = true
    ).


%Condition the transformation rule on...
%  the previous dialogue act
%  any of the two previous dialogue acts
%  both of the two previous dialogue acts
%  the first word in current utterance
%  any word in current utterance
%  any two adjacent words in current utt.
%  the current speaker
%  any combination of word and speaker
%  any word in current utterance, if a
%    speaker change has just occurred
```

Figure 10: Dialogue act tagging in μ-TBL: data representation, auxiliary predicates, annotated templates. Adapted excerpt from Lager and Zinovjeva (1999). Abbreviations: s:speaker, u:utterance, da:dialogue act, mem:member.

First, assuming a data representation similar to what we have seen (Figure 11, top), we find that the example template can be expressed with our existing toolkit (Template 1a in Figure 11, middle).

Of course, if our DSL allows partial application, there is no reason to hardcode the direction, the bounds, or the POS we look for. Instead, we could parameterize them in the generally useful predicate `closest/6` (Figure 11, bottom). This allows Template 1a to be rephrased as Template 1b. More interesting, with `closest/6` in place, our DSL may now let us replace the direction to search in (`left`) and the POS tags to search for (`verb` and `prep`) with variables, as usual to be filled in with the values which make the strongest predictions. For instance, Template 2 in Figure 11 expresses something like "condition the chunk tag change on the current word, its part-of-speech

```
%%% DATA REPRESENTATION
wd(1, o       ). pos(1, art). chunk(1, i). chunk(i, i, 1).
wd(2, aluno   ). pos(2, n  ). chunk(2, i). chunk(i, i, 2).
wd(3, esqueceu). pos(3, v  ). chunk(3, o). chunk(o, o, 3).
%...
```

```
%%% TEMPLATE 1a
chunk:A>B <-word:W@[0] & pos:prep@[0] & closest_left_verb_2_10:VerbWord@[0].

%%% AUXILIARY PREDICATE
closest_left_verb_2_10(Position, Word) :-
   between(2, 10, I),
   Position1 is Position - I,
   pos(Position1, verb),
   !,
   word(Position1, Word).
```

```
%%% TEMPLATE 1b
chunk:A>B <-word:W[0] & pos:prep@[0] & closest(left, verb, 2, 10):VerbWord@[0].

%%% TEMPLATE 2
chunk:A>B <-word:W[0] & pos:POS1@[0] & closest(Dir, POS2, 2, 10):POS2Word@[0].

%%% AUXILIARY PREDICATE
closest(left, POS, ClosestBound, FarthestBound, Position,Word) :-
   between(ClosestBound, FarthestBound, I),
   Position1 is Position - I,
   pos(Position1,POS),
   !,
   word(Position1, Word).
closest(right, ...)               %identical, except Position1 is Position + I
```

Figure 11: DSL templates for Portuguese chunking. Data representation (top) and auxiliary predicates in different degree of abstraction (see text).

P1, and the word (within a window to either left or right) which belongs to a part-of-speech that P1 usually has informative associations with."

To be sure, Template 2 in Figure 11 is not necessarily a good one. If nothing else, it is almost certainly inefficient: depending on the size of the corpus and the tagsets, the combinatorial explosion may render it unusable in practice. However, decisions on which searches to perform and which to prune are better left to the user than to the designers of the template language.

We could go on and parameterize the window edges, or, even better, replace the window entirely with some predicate "earlier in the sentence". We will stop here, however, in the belief that we have demonstrated the expressivity and conciseness a DSL can gain by reusing well-known concepts and constructs of the base language.

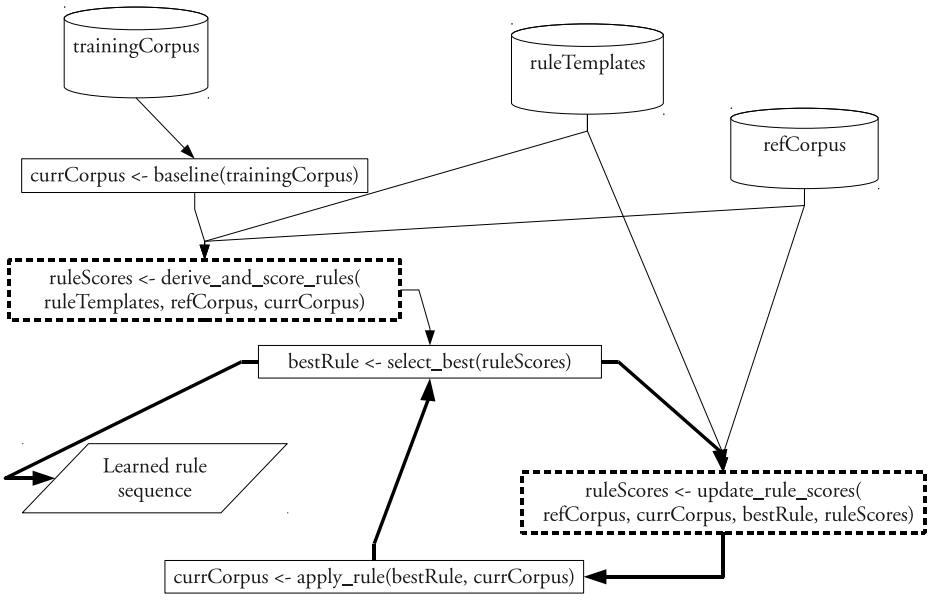


Figure 12: Data flow of TBL training, extended with state (dashed boxes) `ruleScores`

4 Fast, declarative TBL

As pointed out, one of the major problems of the standard TBL training algorithm is the very slow learning phase. Section 3.3.1 discussed some proposed improvements. Here, we look closer at one of those, due to Ngai and Florian (2001b). Like many other attempts at improving efficiency we have seen in TBL (and, to be sure, in Computer Science in general), it is based on the idea of using state to avoid recalculation. In the TBL case, this involves generating all potentially helpful rules once, computing some amount of information for each, and saving that information as part of global state. Then, after each rule application, it is enough to update state minimally, instead of regenerating new rule candidates from scratch. Figure 12 shows the system architecture augmented with state (cf. Figure 2).

The algorithm by Ngai and Florian (2001b) resembles the one described by Ramshaw and Marcus (1994). In that case, however, we saw (Section 3.3.1) that the amount of information kept track of for each rule turned out to be prohibitively memory-expensive for most practical uses. By contrast, in the algorithm described here, each rule r is associated only with two integers: the counts for good and bad applications.

In the following, we give an algorithmic overview, and we present a novel, more declarative rephrasing of the originally strongly imperative algorithm. Our version is slightly slower than the

original[35] (replacing hash-based maps with functional, tree-based counterparts will often come with a $O(\log(N))$ cost, although there might be alternatives in specific cases), but it is better adapted for declarative languages and (in our opinion, anyway) easier to understand. In addition, in contrast to the imperative variation, ours is parallelizable (although, admittedly, the speedups measured so far fall quite a bit below their theoretical ceiling, see Section 4.6).

4.1 Notation

The following notation and definitions are used throughout this section (from Ngai and Florian (2001b), slightly adapted):

- S is the sample space – a set of mutually independent sequences of positive length;
- C is the set of possible classifications of a sample;
- $C[s]$ is the current classification of sample s;
- $T[s]$ is the true classification of sample s;
- p is a predicate on S;
- r is a rule: a pair (p, t) of a predicate p and a target t (a class label);[36] given a rule r, we write p_r and t_r for its predicate and target, respectively;
- $good(r)$ is the count of good applications (successful error corrections) of rule r;
- $bad(r)$ is the count of bad applications (newly introduced errors) of rule r;
- R is the set of all rules that we keep track of (a subset of the total rule space);
- $GB^i[r]$ denotes the current counts $(good(r), bad(r))$ of rule r in iteration i;
- GB^i is used for the current good/bad counts for all rules in iteration i;
- A rule r *applies* to a sample s if $p_r(s)$ and $t_r \neq C[s]$.

Any other notation will be introduced as needed.

4.2 Algorithmic overview

Clearly, if we have GB^i in some appropriate data structure, it is easy to efficiently find the best-scoring rule b_i in each iteration i. The question then reduces to two subtasks (dashed boxes in Figure 12): how do we initialize the state and how do we update it; or, more declaratively spoken, how do we get GB^0; and how do we get to GB^{i+1} from GB^i, given b_i.

The first subtask is very similar to what we have seen many times before.

[35] For time complexity estimations, we refer to Section 3.3.1.

[36] Note that rules in this formalism correspond to the templates of Table 4, not to those of Table 1.

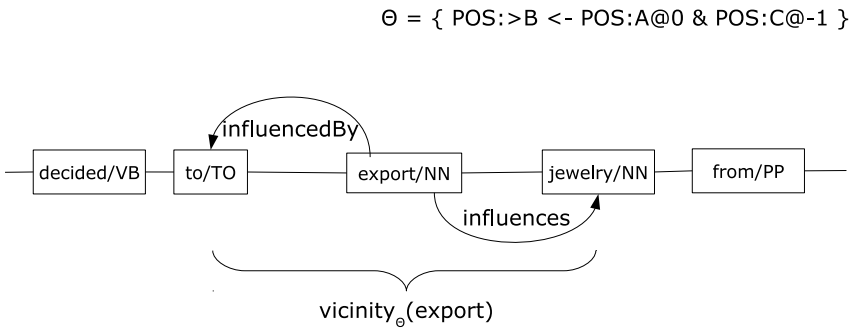


Figure 13: Vicinity of a node. The figures shows the fragment *...decided to export jewelry from ...* just before applying the rule `>VB <- TO@-1 & NN@0`, which will change the tagging of *export*. The vicinity of this node with respect to the given template set Θ is simply the *export* node itself and its left and right neighbour. Arrows showing reflexivity omitted for clarity.

- **1. calculate-good-scores-for-all-rules** For all samples s that satisfy $C[s] \neq T[s]$, generate all rules r that correct the classification of s; increase $good(r)$.
- **2a. calculate-bad-scores-for-all-rules** For all samples s that satisfy $C[s] = T[s]$, generate all rules r that introduce an error; increase $bad(r)$.

Of course, we are not really interested in rules with no positive scores at all, so the second step can be optimized:

- **2b. calculate-bad-scores-for-good-rules** For all samples s that satisfy $C[s] = T[s]$, generate all predicates p such that $p(s) = true$; for each rule r resulting from step 1 such that $p_r = p$ and $t_r \neq C[s]$, increase $bad(r)$.

We can thus get GB^0 simply by `calculate-good-scores-for-all-rules` followed by `calculate-bad-scores-for-good-rules`.

The second subtask, the updating step, is more challenging. The key observation behind the efficiency of the algorithm is that when performing this minimal update, any nodes which are not in the neighbourhood of a changed node (usually the vast majority) can be disregarded. This idea can be formalized (Ngai and Florian, 2001b) by defining the *vicinity* V_Θ of a node n with respect to a template set Θ as

$$V_\Theta(n) = \{n\} \cup \{x \mid x \texttt{ influences } n\} \cup \{x \mid x \texttt{ influencedBy } n\},$$

where x influences y if there is any template $\theta \in \Theta$ such that x occurs in the `CONDITION` of θ and y in the `ACTION`. The converse relation, of course, is `influencedBy`. Vicinity is illustrated in Figure 13.

4.3 A low-level view: Updating GB

The update step for iteration i as proposed by Ngai and Florian (2001b), given a rule b_r to apply, involves identifying all samples which are in the vicinity of any node changed by b_r, and to distinguish all the different ways these changes influence the to-be-finished rule counts GB^i. This operation is complicated and we will not delve into the details here. Imperative pseudocode for their entire algorithm is given in Figure 14 (lines 1-5 correspond to the initialization of GB^0; lines 9 and below perform updating of GB^i).

4.4 A high-level view: Computing GB_Δ

Although very useful, in our experience the amount of book-keeping detail makes the implementation of the algorithm of Figure 14 difficult to get right. Furthermore, it is geared towards data structures which use memory destructively;[37] thus, it is difficult or impossible to parallelize.

It turns out that the algorithm can be made quite a bit easier to understand when viewed at a higher level of abstraction. In addition, if we stick to purely functional data structures, we can make it parallelizable[38] Even in an imperative paradigm, with destructive memory access and thus lesser prospects on parallelism, the gained clarity might be reason enough to prefer the alternative view.

Crucially, we prefer to think of the updating step as performing exactly the same operation as the initial rule scoring, except on a set of independent (hence the potential for parallelization) subcorpora – namely, the union of all vicinities of all application sites of the best-scoring rule b.

To see how this view is useful, we first define an *addition* operation for rule counts. Given two rule counts GB_a and GB_b we define their sum to be the multiset sum $GB_{a+b} = GB_a \uplus GB_b$. That is, GB_{a+b} is the mapping such that

$$GB_{a+b}[r] = GB_a[r] + GB_b[r]$$

where r ranges over the rules in either GB_a or GB_b; and $GB[x]$ is taken to be 0 whenever x is not in the map. Rule count *subtraction* is defined analogously.

Second, we define the *partitioning* and the *filtering* of a data set D (a set of independent sequences of positive length) with respect to a rule r and a template set Θ:

$$\text{partition}_{r,\Theta}(D) = (D_{V_\Theta(r)}, D_{\neg V_\Theta(r)})$$

$$\text{filter}_{r,\Theta}(D) = D_{V_\Theta(r)}$$

[37] The authors suggest a two-level hash, keyed on p_r and t_r as the main data structure for GB.

[38] Hopefully it will also be easier to integrate into a declarative DSL, although we will not pursue that path further here.

For each sample s that satisfies $C[s] \neq T[s]$,
generate all rules r that correct the classification of s; increase $pos(r)$.

For each sample s that satisfies $C[s] = T[s]$
generate all predicates p s.t. $p(s) = 1$; for each rule r s.t. $p_r = p$ and $t_r \neq C[s]$ increase $neg(r)$.

1: Find the rule $b = \arg\max_{r \in \mathcal{R}} f(r)$.

If ($f(b) <$ Threshold or corpus learned to completion) **then** quit.

For each predicate p,
let $\mathcal{R}(p)$ be the set of rules whose predicate is p (i.e. $\mathcal{R}(p) = \{r | p_r = r\}$).

For each samples s, s' s.t. $C[s] \neq C[b(s)]$ and $s' \in V(s)$:

- **If** $C[s'] = C[b(s')]$ **then**
  - **For each** predicate p s.t. $p(s') = 1$
    - **If** $C[s'] \neq T[s']$ **then**
      - **If** $p(b(s')) = 0$ **then** { decrease $pos(r)$, where $r = [p, T[s']]$, the rule created with predicate p and target $T[s']$; }
    - **Else**
      - **If** $p(b(s')) = 0$ **then** for all the rules $r \in \mathcal{R}(p)$ s.t. $t_r \neq C[s']$ decrease $neg(r)$;
  - **For each** predicate p s.t. $p(b(s')) = 1$
    - **If** $C[b(s')] \neq T[s']$ **then**
      - **If** $p(s') = 0$ **then** { increase $pos(r)$, where $r = [p, T[s']]$; }
    - **Else**
      - **If** $p(s') = 0$ **then** { **For each** rule $r \in \mathcal{R}(p)$ s.t. $t_r \neq C[b(s')]$ increase $neg(r)$;

  Else
  - **For each** predicate p s.t. $p(s') = 1$
    - **If** $C[s'] \neq T[s']$ **then**
      - **If** $p(b(s')) = 0 \vee C[b(s')] = t_r$ **then** decrease $pos(r)$, where $r = [p, T[s']]$;
    - **Else**
      - **For each** rule $r \in \mathcal{R}(p)$ s.t. $t_r \neq C[s']$ decrease $neg(r)$;
  - **For each** predicate p s.t. $p(b(s')) = 1$
    - **If** $C[b(s')] \neq T[s']$ **then**
      - **If** $p(s') = 0 \vee C[s'] = t_r$ **then** increase $pos(r)$, where $r = [p, T[s']]$;
    - **Else**
      - **For each** rule $r \in \mathcal{R}(p)$ s.t. $t_r \neq C[b(s')]$ increase $neg(r)$;

Repeat from step 1:

Figure 14: The FnTBL algorithm (Ngai and Florian, 2001b).

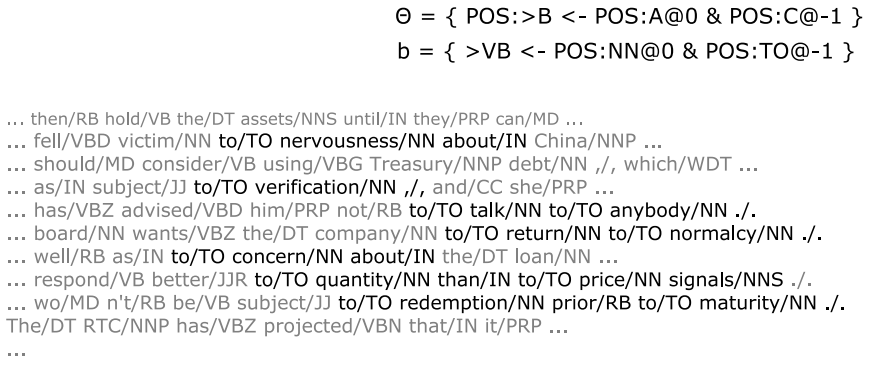


Figure 15: Partitioning a corpus (the Penn treebank) with respect to a rule b and a template set Θ. Black for $D_{V_\Theta(b)}$, grey for $D_{\neg V_\Theta(b)}$ (see text).

Here, we abuse the $V(\cdot)$ notation somewhat by expanding it from samples to rules: $D_{V_\Theta(r)}$ consists of all vicinities of all application sites of r, taken to be elements of a sequence if they are so in D (with any overlapping sequences merged – no samples are duplicated). Analogously, $D_{\neg V_\Theta(r)}$ is everything that did not fit – either subsequences which were left over when vicinities of longer ones were removed, or other unchanged sequences which didn't contain any application sites to boot. Normally, $|D_{V_\Theta(r)}| \ll |D_{\neg V_\Theta(r)}|$. See Figure 15.

We now assume that we have a template set Θ, a data set D, a rule count GB^i_D for the current iteration i, and a best-scoring rule b which partitions D into $D_{V_\Theta(b)}$ and $D_{\neg V_\Theta(b)}$. The goal is to calculate the net effect of applying b, $GB^i_\Delta = GB^{i+1} - GB^i$, in terms of this partition. First, we note that in any iteration i, the good and bad counts contained in GB^i_D can be written (leaving the dependence on Θ implicit, to reduce clutter)

$$GB^i_D = GB^i_{D_{\neg V(r)}} + GB^i_{\Pi(r)} + GB^i_{D_{V(r)}} \tag{7}$$

where $GB^i_{\Pi(r)}$ is a residual term needed to account for sequence boundaries (it depends on templates which applied before the split but not after or vice versa). Thus,

$$GB^{i+1}_D = GB^{i+1}_{D_{\neg V(r)}} + GB^{i+1}_{\Pi(r)} + GB^{i+1}_{D_{V(r)}} \tag{8}$$

The key insight here is that application of b has no influence on either $GB_{\Pi(r)}$ or $GB^i_{D_{\neg V(r)}}$; thus $GB^{i+1}_{\Pi(r)} = GB^i_{\Pi(r)}$ and $GB^{i+1}_{D_{\neg V(r)}} = GB^i_{D_{\neg V(r)}}$. Subtracting (7) from (8) yields

$$GB^i_\Delta = GB^{i+1}_D - GB^i_D = GB^{i+1}_{D_{V(r)}} - GB^i_{D_{V(r)}} \tag{9}$$

In summary, we can calculate the update GB^i_Δ, needed as a consequence of applying the rule b, by:

1. extracting the subcorpus $D_{V(b)}$ of vicinities of all applications sites of b;
2. score all relevant rules for $D_{V(b)}$;
3. apply b to (a copy of) $D_{V(b)}$ and score the result;
4. subtract the result of (2) from the result of (3).

An outline of the algorithm in functional-style pseudocode[39] is given in Figure 16. Several details are omitted; thus, we assume the procedures given in Section 4.2 (with the obvious changes for the functional paradigm to accept necessary parameters and to return values); furthermore, corpus filtering, rule count addition, and rule count subtraction as described above; and finally some sensible definitions of `findBestRule` (cf. page 18), `applyRule` (cf. page 20), and `terminationReached` (cf. page 21).

4.5 Implementation notes

We implemented the algorithm just described in the functional language Haskell (Peyton Jones et al., 2003).[40] Here, we collect a few random observations from the process.

- The algorithm requires efficient access to the best-scoring rule and to the scores of each rule given (p_r, t_r). In a declarative setting, this can be acquired by pairing a functional heap (Okasaki, 1998) and a tree-based map.
- The idea behind the optimization we applied in the initialization step (2b in Section 4.2, `calculate-bad-scores-for-good-rules`) is useful also when scoring subcorpora, but the `-good-rules` part then need to refer to the good rules for the entire corpus. (The pseudocode of Figure 16 ignores this optimization, to reduce clutter).
- Occasionally, application of a rule will create the opportunity for an entirely new rule r', previously unseen. Even if $good(r')$ falls below the scoring threshold s in the current iteration (and r' thus is not likely to be used, as far as we presently can tell), more opportunities may turn up in later iterations.

  However, introducing new rules can be very expensive, as there are no minimal updates to apply: to score any rule created after the initialization stage we will need to search the entire corpus linearly (see next section for some practical measurements). In our experience, disallowing the introduction of new rules does not affect performance noticeably. It would be unwise, however, to make statements of all possible scenarios, so a more flexible solution is to parameterize rule creation by a threshold. New rules not reaching this threshold will be ignored. A reasonable value is s or a bit above it. Ngai and Florian (2001b) are not clear on how they handle this point.

[39] Essentially Haskell with over-explicit function names, java-style signatures in function definitions, and no worries about IO.

[40] Haskell tutorials in suggested reading order are Hudak et al. (2000); Thompson (1999); Hudak (2000); O'Sullivan et al. (2008).

```
scoreInitialCorpus(Corpus cps, Templates tpls) = goodBadScores
  where
    goodScores = calculateGoodScoresForAllRules(cps, tpls)
    goodBadScores = calculateBadScoresForGoodRules(cps, tpls, goodScores)

scoreSubcorpus(Corpus subCps, Templates tpls) = goodBadScores
  where
    goodScores = calculateGoodScoresForAllRules(subCps, tpls)
    goodBadScores = calculateBadScoresForAllRules(subCps, tpls)

nextGoodBadCount(Rule bestRule, RuleCount goodBad, Corpus cps) = goodBad'
  where
    subCps = filter(bestRule, cps)
    subCps' = applyRule(bestRule, subCps)
    deltaScores = scoreSubcorpus(subCps') - scoreSubcorpus(subCps)
    goodBad' = goodBad + deltaScores

tbl(Corpus cps_0, Templates tpls) = until(terminationReached, iter, initialState)
  where
    terminationReached(Corpus cps, RuleCount goodBad, Rules rls) = ...

    iter(Corpus cps, RuleCount goodBad, Rules rls) = (cps', goodBad', rls')
      where
        bestRule = findBestRule(goodBad)
        rls' = append(rls, bestRule)
        cps' = applyRule(bestRule, cps)
        goodBad' = nextGoodBadCount(bestRule, goodBad, cps)

    initialState = (cps_0, goodBad_0, rules_0)
      where
        goodBad_0 = scoreInitialCorpus(cps_0, tpls)
        rules_0 = []


-- until(pred, iter, a)
--   yields the result of repeatedly applying function iter until pred holds,
--   with a as the initial value
```

Figure 16: Fast, declarative TBL in functional-style pseudocode. Names beginning with upper-case letters informally denote types; with lower-case they denote values. `x` and `x'` have the same type, usually understood as `x'` being a modified version of `x`

In addition, any attempt to parallelize code inevitably will bring up design choices. Haskell is a purely functional language with referential transparency, which puts it in a better position than most to explore the increasing number of cores on modern hardware. Nevertheless, compilers for parallel programming is work in progress in any language. If there are any silver bullets, they haven't been found yet, much less fired. The main problem is to find, ahead of time, the right size of the work chunks: if too small, the bookkeeping cost will dwarf any parallelization gain; if too big, some of the cores may idle their time away while their colleagues are sweating over the task.

Haskell offers three main mechanisms for programmer-controlled parallelism. One uses explicit control with transactional memory (similar to other languages). Another, Data Parallel Haskell (Chakravarty et al., 2007), is more original. It offers nested data structures which are specially built to be traversed in parallel, with each core executing exactly the same instruction (a single program counter). The third (Trinder et al., 1998) is much more lightweight; it amounts to speculatively annotating data structures where the compiler should search for parallel opportunities.

With the data description introduced in the previous section, we know that each updating step involves a large number of independent and not entirely trivial tasks. This knowledge is a good base for aspiring to parallelism gains (today, but hopefully even more so with the even smarter compilers of tomorrow). For the purposes here, we will not be very ambitious: we will be satisfied with a demonstration that such gains actually are possible, rather than trying to maximize them. Thus, we restrict ourselves to the simple annotation scheme of Trinder et al. (1998). For a more serious implementation (e.g., aiming at the same performance ballpark as the existing FnTBL implementation) we would have considered Data Parallel Haskell, which can be expected to scale to a much larger number of cores.

4.6 Algorithm time and memory consumption

To measure running times, we used the original setup of Brill (1995a): 26 templates, score threshold 2, accuracy threshold 0.5, the Wall Street Journal corpus. The achieved tagging accuracy was not of interest (and the same in all cases, for a given corpus size). The experiments were run on a dual-core computer.[41] For each corpus size, we recorded the time needed to produce the first rule (Time0, see Figure 17), the time to finish training where new rules were allowed to be created in each iteration (TimeN-create) and where new rules were disallowed (TimeN-nocreate). For comparison, we implemented the basic TBL algorithm (including the common optimization described on p. 40) and ran it on the corpus sizes it could reasonably handle (TimeN-Brill).

As can be seen, for this setup, allowing new rules almost doubled the running time, with no gain. Brill's original algorithm is asymptotically slower, as expected.

Our attempt at parallelization (Figure 17, right) resulted in a speedup on the order of 30%. This is quite a bit from the theoretic maximum, but given the effort (about 20 lines of code), we at least find it interesting.[42]

Figure 18 (left) gives an idea of the state size. The thing to note is that state grows sublinearly in the size of the corpus N. This means that the size of the corpus dominates that of the state and

[41] ghc 6.8 and 6.10, Ubuntu 8.10 on a Lenovo T400, Intel Core Duo, 2.26 GHz, 4GB RAM)

[42] Note that the more recent Haskell compiler actually performed worse. We have not looked further into this matter, but it does strengthen the impression that parallelization is a tricky business.

that total memory usage thus is linear in N (Figure 18, right), which is the best one could hope for.

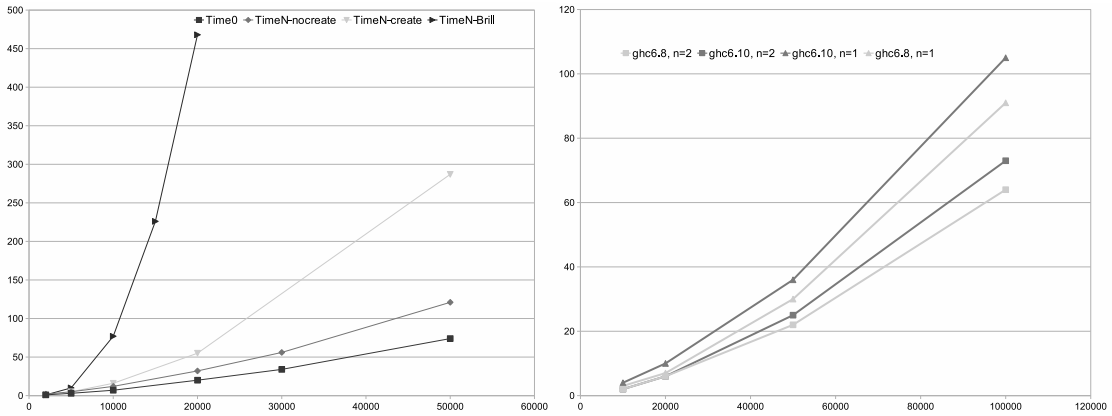


Figure 17: Algorithm performance: Running time (seconds) as a function of corpus size. Left: comparison with Brill's original algorithm. See text for details. Right: parallel performance, one and two cores used on two different Haskell compilers.

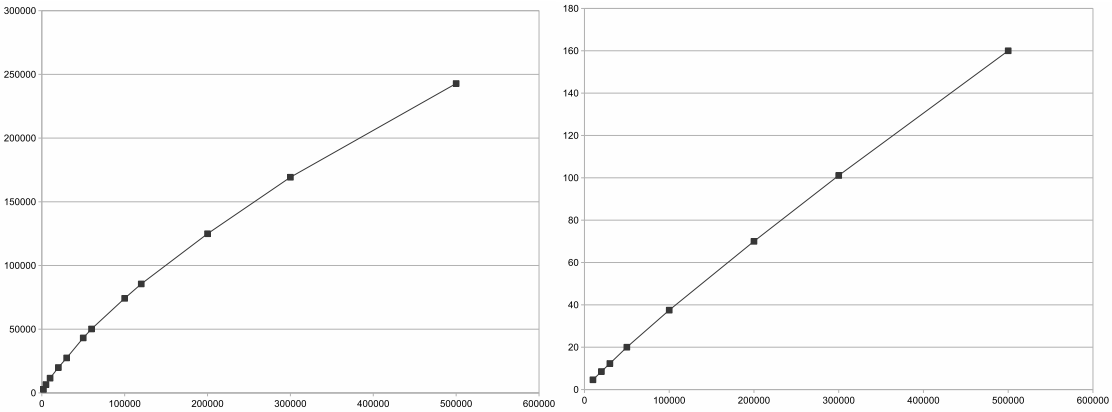


Figure 18: Memory usage as a function of corpus size. Left: state size (nodes in tree). Right: maximum memory residency (MB).

5 Conclusion and future directions

In this thesis, we have provided a survey of Transformation-Based Learning. We have done so without aiming at full formal detail, but hopefully we have been explicit enough to provide a self-contained introduction to the basic method, and to the most important of its later developments. Where the full story is needed, we believe that the pointers provided will be helpful, as well as the comprehensive bibliography.[43] Furthermore, we have tried to fill a small gap by rephrasing a state-of-the-art training algorithm for declarative languages.

In the introduction, in addition to these two concrete aims, we also expressed a rather vague hope of promoting the general interest in TBL. As we have mentioned, TBL is almost unheard of outside the linguistic world. Indeed, the absence of non-linguistic applications in Table 8 is striking, the only exceptions being Bringmann et al. (2002) and Landwehr et al. (2008). We don't see any particular reasons why this should be so. It is true that the kind of local dependencies TBL is especially suited for exploiting are particularly common in language, but they certainly occur elsewhere.

Although rarely sufficient by themselves, good introductions and good toolkits are helpful when propagating ideas. While any of several sources (including, perhaps, this thesis) may serve the first role, the casting of the second is less clear. What qualifications should we look for in the candidate? A prerequisite for making it in the non-linguistic world is clearly that there are no limiting, specifically linguistic assumptions on data or tagsets. Actually, removing such hardcoded assumptions is likely to benefit linguists, too, who are looking for new application domains for a well-tested technique.

For a second requirement, we have made no secret of our view that one of the most flexible ways to interact with a toolkit is via a DSL (Section 3.5), which can hide any amount of imperative machinations behind the appropriate, high-level abstractions of the domain. A well-designed, syntax-light DSL which covers the relevant concepts of the domain should generally be understandable and usable by a domain expert who is not a programmer (especially if it is of the declarative kind). Yet, it exposes all the power of the system (and its underlying base language, when built as an extension of an existing one) to the experienced user.

On the wish list we may also find features such as unicode compliance, open source, prospects of parallelization, performance, and many other things. Different projects will weigh these factors differently and we will not dwell on them here.

With such a list of desiderata, what candidates will we currently find on the world wide web marketplace? As it turns out, "currently" is not very different from ten years ago. To our knowledge, there are currently four TBL implementations publicly available on the web: Brill (C),[44] FnTBL (C++),[45] NLTK (Python),[46] and µ-TBL (Prolog).[47]

Brill was the first implementation, but it is hard-coded for POS tagging and now mostly of

[43] See footnote on p. 10.

[44] The original page has disappeared, but a mirror of the Brill tagger at its latest version is at `http://www.tech.plym.ac.uk/soc/staff/guidbugm/software/RULE_BASED_TAGGER_V.1.14.tar.Z`

[45] `http://www.cs.jhu.edu/~rflorian/fntbl/`, latest update 2001

[46] `http://www.nltk.org/`, Natural Language Toolkit, general tools for language processing; the toolkit is continuously updated, but there has been no non-trivial changes to the TBL module at least since version handling started in 2005

[47] `http://www.ling.gu.se/~lager/mutbl.html`, latest update 2000

historical interest. The others all have different strengths and weaknesses. FnTBL is the fastest and widely used. It implements several of the extensions of Section 3, but it is not so easy to extend or experiment with for non-C++ programmers. The TBL implementation present in NLTK is part of a larger toolkit for natural language processing, actively developed. It thus offers APIs to all kinds of language-processing modules and utilities. However, its TBL part is very basic and clearly geared towards the specific task of part-of-speech tagging. With proper generalization and extension, its impressive infrastructure would make it an interesting tool for TBL applications, but currently it seems less useful than it could be.

In our view, µ-TBL is the most interesting, and we will describe its pros and cons in somewhat more detail. It is well-documented and has been used in several real-world projects. It offers an interactive interface and a scripting language for automation. Code for compiling learned rules into blazingly fast FSTs (Section 3.3.2) can be generated automatically. It contains, to our knowledge, the only attempt to learn Constraint Grammar rules by TBL.

Most importantly, as we have already seen exemplified (Figure 10), its template specifications form a declarative DSL, expressive but concise. It should be reasonably understandable to domain experts without insights into Prolog or even programming. Still, templates may use arbitrary Prolog code for complex tasks.[48]

To be sure, the toolkit has a lot of improvement potential, too. It was developed under Sicstus, a commercial and relatively expensive implementation of Prolog for whose availability no guarantees can be made; ports between Prolog systems are usually possible, but tiresome. Besides Brill's original, it includes the fast randomized training algorithm due to Samuel (1998b) mentioned in Section 3.3.1, but none of the sometimes preferable alternatives (e.g. Ramshaw and Marcus, 1994; Ngai and Florian, 2001b; Hepple, 2000; Santos and Milidiú, 2007). Data is input as Prolog databases, rather than in more widely used formats such as csv or XML. All data partitioning into train and test set must be done by preprocessing. The latest code change is from 2000: thus, the system suffers slightly from general bit rot (no unicode; poor use of later hardware, especially memory). It should be commended for its design around exchangeable training algorithms, but this API still makes unfortunate assumptions on the template format (cf. Table 1 vs. Table 4).

Furthermore, µ-TBL does of course not contain any of the post-2000 developments (and only a few of the earlier ones) described in Section 3. A list of useful extensions includes probabilistic TBL (Section 3.2.1), multidimensional TBL (Section 3.1.1), template-free TBL (Section 3.1.3), TBL with automatically learned templates (Section 3.1.2), TBR (Section 3.2.3), unsupervised TBL (Section 3.4), and means of automatically building ensembles (Section 2.5.3).[49]

Leaving current and planned toolkits aside, we may briefly speculate on future general developments of the TBL method itself. We have mentioned several areas which seem to deserve

[48] As a remark, Prolog may seem like an odd choice of implementation language: generally speaking, Prolog tends to fit very well to some problems and very poorly to others, with the traditional, strongly imperative phrasing of TBL rather belonging to the second group. However, Prolog is an excellent base language for a declarative DSL behind which any amount of imperative detail can be hidden. The details of the hiding is the problem of the implementer, not the user of the DSL.

In addition, it might well be worthwhile to look for other phrasings than the imperative ones. Lager (1999b) presents an implementation which is derived from a logical interpretation of TBL rules; and, by way of a practical argument, Lager (1999a) describes an implementation of a system based on this idea which is an order of magnitude faster than Brill's original, with the Prolog source code fitting on a single page.

[49] We are aware that writing such lists is quicker than implementing them.

wider use (e.g., ETL, Section 3.1.2; TBR, Section 3.2.3; probabilistic TBL, Section 3.2.1; Constraint Grammar-TBL, Section 3.2.2) and others which apparently have been explored little or not at all (weights in multidimensional learning, Section 3.1.1; unsupervised Constrained Grammar-TBL, Section 3.4; alternative scoring schemes, Section 2.2.5; TBL rule purification in ensembles, Section 2.5.3). Time will tell which ones of these are worth their salt.

In a slightly larger perspective, it should be recognized that TBL is (mostly) a supervised machine learning method, and that generally speaking the trend has moved away from most such methods. The current focus clearly lies on unsupervised learning on very large data sets. Nevertheless, not all data sets are gigantic, and we believe there will always be a use for supervised methods in specialized and not-so-specialized domains.

Notes

i The parts-of-speech in this example are taken from the Penn Treebank tagset. So are their sometimes inscrutable abbreviations – VB for verb, NN for noun, IN for preposition, etc. For the purposes of this thesis, it is enough to think of such names as arbitrary, atomic labels; see http://www.cis.upenn.edu/~treebank/ for external definitions.
Admittedly, these labels bear little similarity to what one might have learned about word classes in fifth grade. To remove any possible misunderstandings, parts-of-speech are not given by nature (this becomes very clear when unrelated languages are compared). Instead, for practical purposes the set of allowable class labels, the *tagset*, needs to be specified stipulatively. The Penn Treebank tagset (along with a few others) is commonly used for English, but needs extensive modification to be useful even for closely related languages.

ii This problem formulation, which employs single-letter correspondences, is chosen for illustration rather than efficiency. It works well for many languages, but unfortunately, due to its very irregular orthography, English is not one of them. Some unnaturalness in the mapping is unavoidable; here, we have had to introduce "empty" phonemes (denoted by _). However, the point of this section is to provide a few examples of sequential classification rather than solve problems, so we will gloss over such details here.

iii Most such ambiguities pass unnoticed by humans – we disambiguate on semantic grounds, usually without even noticing that we did. But it is not difficult to come up with examples where also humans will be hesitant. Consider

- I [tripped]$_{\text{V}}$ [the man]$_{\text{NP}}$ [with my umbrella]$_{\text{PP}-\text{V}(?)}$
- I [tripped]$_{\text{V}}$ [the man]$_{\text{NP}}$ [with the black umbrella]$_{\text{PP}-\text{NP}(?)}$
- I [tripped]$_{\text{V}}$ [the man]$_{\text{NP}}$ [with the umbrella]$_{\text{PP}-\text{V}/\text{PP}-\text{NP}??}$

iv We have chosen POS tagging (Example 1) as a recurring example not because we believe it is the only thing TBL is good for, but rather because it is a practical, basic, and well-defined task, often needed as a preprocessing step, and frequently treated in the literature.
We also observe that very often the challenges of any task for a particular language are decided by its typological properties – how many, how regular, how complicated are the inflection patterns, how rigid the word order, how well-defined the word boundaries, etc. POS tagging of Russian is very different from POS tagging of Chinese, and both are different again from POS tagging of English. In this way, the typological variety of the thousands of languages of the world brings to light a wide range of interesting subchallenges. POS tagging, being one of the most basic tasks, has inspired (or at least been used to illustrate) several extensions of popular machine learning algorithms, including TBL.

v For almost all areas of Computational Linguistics, there is an unfortunate but massive preponderance of literature where English is not only the language of communication but also the object of study. As a consequence, linguistic phenomena or subtasks which are judged interesting for English are more researched and (sometimes) better understood than those that are not. We will not dwell on this topic here, but just give a few examples. English exhibits little inflectional morphology (most words have only two or three forms); and word boundary detection can mostly be done trivially, on white space. By contrast, a Turkish verb may have thousands of forms; and word boundary detection is a major challenge in Thai. Conversely, any collection of English text written by non-professionals will contain a very significant amount of spelling errors, which is a challenge for automatic methods. On the other hand, with respect to spelling, a comparable text for Finnish can be expected to be almost perfect.

vi Just to avoid misunderstandings, linguistic classification tasks such as those in Table 8 do not form a closed set. Part-of-speech tagging has been performed by humans for thousands of years, since the early grammarians of Sanskrit and Greek, but word alignment (deciding what words should be paired together in sentences which are translations of each other) or document format processing (e.g., extracting rules for turning poor HTML into well-formed XML) are recent techniques to help satisfy recent needs.

vii *Smoothing* is a generic term for techniques for improving probability estimations of stochastic events drawn from so large event spaces that they may rarely or (more often) never have been seen before. Human language offers many such event spaces. For instance, the sentence "please wait for a while before swallowing the headphone" probably never have occurred before, and in any corpus which does not include this paper it will have zero occurrences. Yet, the sentence clearly has some non-zero probability. Smoothing helps estimating that probability from existing non-zero counts. It is also an entire research field of its own.

References

Abelson, H. and G. J. Sussman (1996). *Structure and Interpretation of Computer Programs*. Cambridge: MIT Press.

Aone, C. and K. Hausman (1996). Unsupervised learning of a rule-based spanish part of speech tagger. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pp. 53–58. Association for Computational Linguistics.

Ayan, N., B. Dorr, and C. Monz (2005). Alignment link projection using transformation-based learning. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pp. 185–192. Association for Computational Linguistics.

Bach, N., N. Ha, and N. Binh (2008). Transformation rule learning without rule templates: A case study in part of speech tagging. In *Advanced Language Processing and Web Information Technology, 2008. ALPIT'08. International Conference on*, pp. 9–14. IEEE.

Becker, M. (1998). Unsupervised part of speech tagging with extended templates. In *Proceedings of ESSLLI 1998, student session*.

Bouma, G. (2000). A finite state and data oriented method for grapheme to phoneme conversion. In *NAACL-2000*, Seattle, WA, pp. 303–310.

Bouma, G. (2003). Finite state methods for hyphenation. *Natural Language Engineering 9*, 5–20.

Bratko, I. (2001). *Prolog Programming for artificial intelligence, 3rd ed* (3 ed.). Pearson/Addison-Wesley.

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

Breiman, L., J. Friedman, R. Olshen, and C. Stone (1984). *Classification and Regression Trees*. Monterrey, CA: Wadsworth and Brooks.

Brill, E. (1993a). Automatic grammar induction and parsing free text: A transformation-based approach. In *Meeting of the Association for Computational Linguistics*, pp. 259–265.

Brill, E. (1993b). Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the workshop on Human Language Technology*, pp. 237–242. Association for Computational Linguistics.

Brill, E. (1993c). *A Corpus-Based Approach to Language Learning*. Ph. D. thesis, University of Pennsylvania, Philadelphia, PA.

Brill, E. (1994). Some advances in transformation-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 722–727.

Brill, E. (1995a). 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. (1995b). Unsupervised learning of disambiguation rules for part of speech tagging. In *Proceedings of the third workshop on very large corpora*, Volume 30, pp. 1–13.

Brill, E. (1996). Learning to parse with transformations. In *Recent Advances in Parsing Technology*. Kluwer.

Brill, E. and P. Resnik (1994). A rule-based approach to prepositional phrase attachment disambiguation. In *Proceedings of COLING'94*, pp. 1198–1204.

Brill, E. and J. Wu (1998). Classifier combination for improved lexical disambiguation. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pp. 191–195. Association for Computational Linguistics.

Bringmann, B., S. Kramer, F. Neubarth, H. Pirker, and G. Widmer (2002). Transformation-based regression. In *Machine learning – International Workshop then Conference*, pp. 59–66. Citeseer.

Carberry, S., K. Vijay-Shanker, A. Wilson, and K. Samuel (2001). Randomized rule selection in transformation-based learning: A comparative study. *Natural Language Engineering 7(2)*, 99–116.

Caruana, R. (1997). Multitask learning. *Machine Learning 28*(1), 41–75.

Chakravarty, M., R. Leshchinskiy, S. Jones, G. Keller, and S. Marlow (2007). Data parallel haskell: a status report. In *Proceedings of the 2007 workshop on Declarative aspects of multicore programming*, pp. 10–18. ACM New York, NY, USA.

Clocksin, W. F. and C. S. Mellish (1994). *Programming in Prolog* (4 ed.). Berlin: Springer-Verlag.

Covington, M. A., D. Nute, and A. Vellino (1997). *Prolog Programming in Depth* (2 ed.). Prentice-Hall.

Curran, J. and R. Wong (1999). Transformation-based learning for automatic translation from HTML to XML. In *Proceedings of the Fourth Australasian Document Computing Symposium (ADCS99)*. Citeseer.

Curran, J. R. and R. K. Wong (2000). Formalization of transformation-based learning. In *ACSC*, pp. 51–57. IEEE Computer Society.

Daelemans, W. (1995). Memory-based lexical acquisition and processing. In P. Steffens (Ed.), *Machine translation and the lexicon*, Lecture notes in Artificial Intelligence, pp. 85–98. Berlin: Springer.

Day, D., J. Aberdeen, L. Hirschman, R. Kozierok, P. Robinson, and M. Vilain (1997). Mixed-initiative development of language processing systems. In *Proceedings of the fifth conference on Applied natural language processing*, pp. 348–355. Association for Computational Linguistics.

Dini, L., V. Di Tomaso, and F. Segond (1998). Error driven word sense disambiguation. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pp. 320–324. Association for Computational Linguistics.

Fernandes, E., C. N. d. Santos, and R. L. Milidiú (2010). A machine learning approach to portuguese clause identification. *Computational Processing of the Portuguese Language*, 55–64.

Florian, R., J. Henderson, and G. Ngai (2000). Coaxing confidences from an old friend: Probabilistic classifications from transformation rule lists. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in NLP and very large corpora*, pp. 26–34. Association for Computational Linguistics. Held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13.

Florian, R., A. Ittycheriah, H. Jing, and T. Zhang (2003). Named entity recognition through classifier combination. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pp. 171. Association for Computational Linguistics.

Florian, R. and G. Ngai (2001). Multidimensional transformation-based learning. In *Proceedings of the Fifth Workshop on Computational Language Learning (CoNLL-2001)*, Volume cs.CL/0107021.

Fordyce, C. (1998). Prosody prediction for speech synthesis using transformational rule-based learning. Master's thesis, Boston University.

Hardt, D. (1998). Improving ellipsis resolution with transformation-based learning. In *AAAI fall symposium*.

Hardt, D. (2001). Transformation-based learning of Danish grammar correction. In *Proceedings of RANLP 2001, Tzigov Chark*. Citeseer.

Hedelin, P., A. Jonsson, and P. Lindblad (1987). Svenskt uttalslexikon. Technical report, Chalmers University of Technology.

Hepple, M. (2000). Independence and commitment: Assumptions for rapid training and execution of rule-based POS taggers. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pp. 278. Association for Computational Linguistics.

Hudak, P. (1996). Building domain-specific embedded languages. *ACM Computing Surveys (CSUR) 28*(4).

Hudak, P. (1998). Modular domain specific languages and tools. In P. Devanbu and J. Poulin (Eds.), *Proceedings: Fifth International Conference on Software Reuse*, pp. 134–142. IEEE Computer Society Press.

Hudak, P. (2000). *The Haskell School of Expression: Learning Functional Programming Through Multimedia*. Cambridge University Press.

Hudak, P., J. Peterson, and J. Fasel (2000). A gentle introduction to haskell. `http://www.haskell.org/tutorial/`.

Jurafsky, D. and J. H. Martin (2008). *An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (2 ed.). Prentice-Hall.

Karlsson, F., A. Voutilainen, J. Heikkilä, and A. Anttila (Eds.) (1995). *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Natural Language Processing, No 4. Mouton de Gruyter.

Kavallieratou, E., E. Stamatatos, N. Fakotakis, and G. Kokkinakis (2000, Sept). Handwritten character segmentation using transformation-based learning. In *Proceedings of the 15th International Conference on Pattern Recognition (ICPR 2000)*, pp. 634–637.

Kim, J., S. E. Schwarm, and M. Ostendorf (2004). Detecting structural metadata with decision trees and transformation-based learning. In *Proceedings of HLT-NAACL'04*, pp. 137–144.

Kuncheva, L. and C. Whitaker (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning 51*(2), 181–207.

Lager, T. (1999a, June 8-12, 1999). μ-TBL lite: A small, extensible transformation-based learner. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL'99)*, Bergen. Poster paper.

Lager, T. (1999b). The μ-TBL system: Logic programming tools for transformation-based learning. In *Proceedings of CoNLL*, Volume 99.

Lager, T. (2001). Transformation-based learning of rules for constraint grammar tagging. In *13th Nordic Conference in Computational Linguistics*, Uppsala, Sweden, pp. 21–22.

Lager, T. and N. Zinovjeva (1999, Oct). Training a dialogue act tagger with the μ-TBL system. In *Third Swedish Symposium on Multimodal Communication*. Linköping University Natural Language Processing Laboratory (NLPLAB).

Landwehr, N., B. Gutmann, I. Thon, L. De Raedt, and M. Philipose (2008). Relational transformation-based tagging for human activity recognition. *Fundamenta Informaticae 89*(1), 111–129.

Mangu, L. and E. Brill (1997). Automatic rule acquisition for spelling correction. In *Machine learning – International Workshop then Conference*, pp. 187–194. Citeseer.

Manning, C. D. and H. Schütze (2001). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: MIT Press.

Mikheev, A. (1997). Automatic rule induction for unknown-word guessing. *Computational Linguistics 23*(3), 405–423.

Milidiú, R. L., C. Crestana, and C. N. d. Santos (2010). A token classification approach to dependency parsing. In *Information and Human Language Technology (STIL), 2009 Seventh Brazilian Symposium in*, pp. 80–88. IEEE.

Milidiú, R. L., J. C. Duarte, and C. N. d. Santos (2007). Evolutionary TBL template generation. *J. Brazilian Computer Society. [online] 13(4)*, 39–50.

Milidiú, R. L., C. N. d. Santos, J. Duarte, and C. do Exército (2008). Phrase chunking using entropy guided transformation learning. In *Proceedings of ACL2008*. Citeseer.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill. otherisbn: 0070428077?

Nahm, U. (2005). Transformation-based information extraction using learned meta-rules. *Computational Linguistics and Intelligent Text Processing*, 535–538.

Ngai, G. and R. Florian (2001a). Transformation-based learning in the fast lane. In *Second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies 2001*, pp. 8. Association for Computational Linguistics.

Ngai, G. and R. Florian (2001b). Transformation based learning in the fast lane: A generative approach. Technical report, Center for Speech and Language Processing, Johns Hopkins University.

Oflazer, K. and G. Tür (1996). Combining hand-crafted rules and unsupervised learning in constraint-based morphological disambiguation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 69–81.

Okasaki, C. (1998). *Purely functional data structures*. Cambridge University Press.

O'Keefe, R. (1990). *The Craft of Prolog*. MIT Press.

O'Sullivan, B., J. Goerzen, and D. Stewart (2008). *Real World Haskell*. Sebastopol: O'Reilly.

Palmer, D. (1997). A trainable rule-based algorithm for word segmentation. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pp. 321–328. Association for Computational Linguistics.

Park, S., J. Chang, and B. Zhang (2004). Korean compound noun decomposition using syllabic information only. *Computational Linguistics and Intelligent Text Processing*, 146–157.

Peyton Jones, S. et al. (2003, Jan). The Haskell 98 language and libraries: The revised report. *Journal of Functional Programming 13*(1), 0–255. http://www.haskell.org/definition/.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

Ramshaw, L. and M. Marcus (1994). Exploring the statistical derivation of transformational rule sequences for part-of-speech tagging. In *Proceedings of the ACL Workshop on Combining Symbolic and Statistical Approaches to Language*, pp. 128–135.

Ramshaw, L. A. and M. P. Marcus (1995, Jun). Text chunking using transformation-based learning. In D. Yarowsky and K. W. Church (Eds.), *Proceedings of the ACL Third Workshop on Very Large Corpora*, Volume cmp-lg/9505040, Somerset, New Jersey, pp. 82–94. Association of Computational Linguistics.

Roche, E. and Y. Schabes (1995). Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics 21*(2), 227–253.

Roth, D. (1998). Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 806–813. JOHN WILEY & SONS LTD.

Ruland, T. (2000). A context-sensitive model for probabilistic LR parsing of spoken language with transformation-based postprocessing. In *Proceedings of the 18th conference on Computational linguistics-Volume 2*, pp. 677–683. Association for Computational Linguistics.

Samuel, K. (1998a). Discourse learning: Dialogue act tagging with transformation-based learning. In *AAAI/IAAI*, pp. 1199.

Samuel, K. (1998b). Lazy transformation-based learning. In *Proceedings of the Eleventh International Florida Artificial Intelligence Research Society Conference*, pp. 235–239. AAAI Press.

Samuel, K., S. Carberry, and K. Vijay-Shanker (1998, Jun). An investigation of transformation-based learning in discourse. In *Machine Learning: Proceedings of the 15th International Conference*.

Samuelsson, C., P. Tapanainen, and A. Voutilainen (1996). Inducing constraint grammars. *Grammatical Interference: Learning Syntax from Sentences*, 146–155.

Santos, C. N. d. (2009). *Entropy Guided Transformation Learning*. Ph. D. thesis, Pontifícia Universidade Católica do Rio de Janeiro.

Santos, C. N. d. and R. Milidiú (2007). Probabilistic classifications with TBL. In A. Gelbukh (Ed.), *Computational Linguistics and Intelligent Text Processing*, Volume 4394 of *Lecture Notes in Computer Science*, pp. 196–207. Springer Berlin / Heidelberg.

Santos, C. N. d., R. Milidiú, C. Crestana, and E. Fernandes (2010). ETL ensembles for chunking, NER and SRL. In A. Gelbukh (Ed.), *Computational Linguistics and Intelligent Text Processing*, Volume 6008 of *Lecture Notes in Computer Science*, pp. 100–112. Springer Berlin / Heidelberg.

Santos, C. N. d., R. Milidiú, and R. Rentería (2008). Portuguese part-of-speech tagging using entropy guided transformation learning. In A. Teixeira, V. de Lima, L. de Oliveira, and P. Quaresma (Eds.), *Computational Processing of the Portuguese Language*, Volume 5190 of *Lecture Notes in Computer Science*, pp. 143–152. Springer Berlin / Heidelberg.

Santos, C. N. d. and R. L. Milidiú (2009). Entropy guided transformation learning. *Foundations of Computational Intelligence, Volume 1*, 159–184.

Santos, C. N. d. and C. Oliveira (2005). Constrained atomic term: Widening the reach of rule templates in transformation based learning. In C. Bento, A. Cardoso, and G. Dias (Eds.), *EPIA*, Volume 3808 of *Lecture Notes in Computer Science*, pp. 622–633. Springer.

Sterling, L. and E. Shapiro (1994). *The Art of Prolog, 2nd ed* (2 ed.). MIT Press.

Thompson, S. (1999). *Haskell — The Craft of Functional Programming, 2nd ed* (2 ed.). Addison-Wesley.

Trinder, P. W., K. Hammond, H.-W. Loidl, and S. Peyton Jones (1998, January). Algorithm + strategy = parallelism. *Journal of Functional Programming 8*(1), 23–60.

Tsuruoka, Y., J. McNaught, and S. Ananiadou (2008). Normalizing biomedical terms by minimizing ambiguity and variability. *BMC bioinformatics 9*(Suppl 3), S2.

van Deursen, A., P. Klint, , and J. Visser (2000). Domain-specific languages: An annotated bibliography. *ACM SIGPLAN Notices 35*(6), 26–36.

Williams, K., C. Dozier, and A. McCulloh (2004). Learning transformation rules for semantic role labeling. In *Proceedings of CoNLL-2004*.

Wilson, G. and M. Heywood (2005). Use of a genetic algorithm in Brill's transformation-based part-of-speech tagger. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, New York, NY, USA, pp. 2067–2073. ACM Press.

Wu, D., G. Ngai, and M. Carpuat (2004, May). Raising the bar: Stacked conservative error correction beyond boosting. In *Fourth International Conference on Language Resources and Evaluation (LREC-2004)*. Lisbon.

Xin, L., H. Xuan-Jing, and W. Li-de (2006). Question classification by ensemble learning. *IJCSNS 6*(3), 147.