Chapter 2
Entropy Guided Transformation Learning

Abstract This chapter details the entropy guided transformation learning algorithm [8, 23]. ETL is an effective way to overcome the transformation based learning bottleneck: the construction of good template sets. In order to better motivate and describe ETL, we first provide an overview of the TBL algorithm in Sect. 2.1. Next, in Sect. 2.2, we explain why the manual construction of template sets is a bottleneck for TBL. Then, in Sect. 2.3, we detail the entropy guided template generation strategy employed by ETL. In Sect. 2.3, we also present strategies to handle high dimensional features and to include the current classification feature in the generated templates. In Sects. 2.4–2.6 we present some variations on the basic ETL strategy. Finally, in Sect. 2.7, we discuss some related works.

Keywords Machine learning · Entropy guided transformation learning · Transformation based learning · Information gain · Decision trees · Transformation rules · Feature selection

2.1 Transformation Based Learning

Transformation based learning is a supervised ML algorithm usually applied to NLP tasks. TBL generates a set of transformation rules that correct classification mistakes of a baseline classifier [3]. The following three rules illustrate the kind of transformation rules learned by TBL.

```
                pos[0]=ART  pos[1]=ART → pos[0]=PREP
pos[0]=ART  pos[1]=V    word[0]=a    → pos[0]=PREP
pos[0]=N    pos[−1]=N     pos[−2]=ART → pos[0]=ADJ
```

These rules were learned for Portuguese POS tagging. They check the following features: `pos[0]`, the POS tag of the current word; `pos[1]`, the POS tag of the next word; `pos[−1]`, the POS tag of the previous word; `pos[−2]`, the POS tag of

C. N. dos Santos and R. L. Milidiú, *Entropy Guided Transformation Learning*: *Algorithms and Applications*, SpringerBriefs in Computer Science,
DOI: 10.1007/978-1-4471-2978-3_2, © The Author(s) 2012

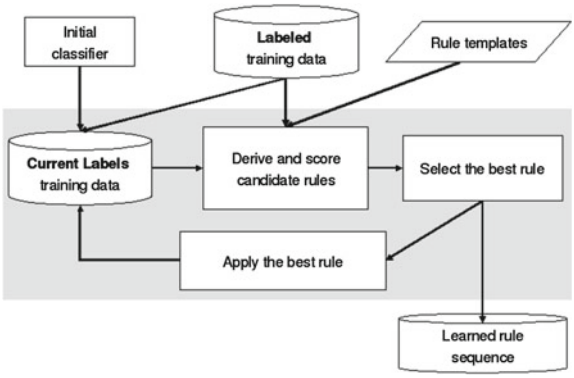

Fig. 2.1 Transformation based learning

the word two before; and `word[0]`, the current lexical item. The first rule should be read as

"**IF** the POS tag of the current word is an *article*
AND the POS tag of the next word is an *article*
THEN change the POS tag of the current word to *preposition*"

TBL rules are composed of two parts: the left hand side and the right hand side. The *left hand side* is a conjunction of feature=value tests, whereas the *right hand side* indicates a value assignment to a target feature. TBL rules must follow patterns, called *rule templates*, that specify which feature combinations should appear in the rule left-hand side. The template set defines the candidate rules space to be searched. Briefly, a template is an uninstantiated rule. The following three templates were used to create the previously shown rules.

```
pos[0] pos[1]
pos[0] pos[1]  word[0]
pos[0] pos[−1]  pos[−2]
```

TBL requires three main inputs:

(i) a correctly labeled training set;
(ii) an initial (baseline) classifier, the *baseline system* (BLS), which provides an initial labeling for the training examples. Usually, the BLS is based on simple statistics of the correctly labeled training set, such as to apply the most frequent class;
(iii) a set of rule templates.

The TBL algorithm is illustrated in Fig. 2.1. The central idea in the TBL learning process is to greedily learn rules that incrementally reduces the number of classification errors produced by the initial classifier. At each iteration, the algorithm learns the rule that has the highest *score*. The score of a rule r is the difference between the number of errors that r repairs and the number of errors that r creates. The learning

process stops when there are no more rules whose score is above a given threshold. The *rule score threshold* is a parameter of TBL.

A pseudo-code of TBL is presented in Algorithm 1. In this pseudo-code, the *apply* function classifies the given training set examples using the given initial classifier or transformation rule. The *isWronglyClassified* function checks whether the example is misclassified or not. This checking is done by comparing the current example class to the correct class. The *instantiateRule* function creates a new rule by instantiating the given template with the given example context values. The *countCorrections* function returns the number of corrections that a given rule would produce in the current training set. Similarly, the *countErrors* function returns the number of misclassifications that a given rule would produce in the current training set. There are also several variants of the TBL algorithm. FastTBL [14] is the most successful, since it achieves a significant speedup in the training time while still achieving the same performance as the standard TBL algorithm. We have also developed a TBL variant that produces probabilistic classifications [9].

When using a TBL rule set to classify new data, we first apply the Initial Classifier to the new data. Then we apply the learned rule sequence. The rules must be applied following the same order they were learned.

2.2 TBL Bottleneck

TBL templates are meant to capture relevant feature combinations. Templates are handcrafted by problem experts. Therefore, TBL templates are task specific and their quality strongly depends on the problem expert skills to build them. For instance, Ramshaw and Marcus [29] propose a set of 100 templates for the phrase chunking task. Florian [12] proposes a set of 133 templates for the named entity recognition task. Higgins [16] handcrafted 130 templates to solve the semantic role labeling task. Elming [11] handcrafted 70 templates when applying TBL for machine translation.

The development of effective template sets is a difficult task. It usually involves a lengthy trial-and-error strategy or the adaptation of an existent, known-to-perform-well-on-a-similar-task template set to the new task [13]. The template developer should indicate the relevant feature combinations, otherwise the TBL algorithm can not learn effective rules. When the number of features to be considered is large, the effort to manually combine them is extremely increased, since there are $2^{|\mathscr{F}|}$ feature combinations, where $\mathscr{F}$ denotes the feature set. On the other hand, the template developer can not generate a very large number of templates, since the training time and memory requirements become intractable. Moreover, Ramshaw and Marcus [28] argue that overfitting is likely when irrelevant templates are included. *Overfitting* is the phenomenon of training too complex a model that do not generalize for new data.

Algorithm 1 Transformation Based Learning Pseudo-Code

```
input LabeledTrainingSet; TemplateSet;
    InitialClassifier; RuleScoreThreshold
 1: LearnedRules ← {}
 2: CurrentTrainingSet ← apply(InitialClassifier, LabeledTrainingSet)
 3: repeat
 4:    CandidateRules ← {}
 5:    for all example ∈ CurrentTrainingSet do
 6:       if isWronglyClassified(example) then
 7:          for all template ∈ TemplateSet do
 8:             rule ← instantiateRule(template, example)
 9:             CandidateRules ← CandidateRules + rule
10:          end for
11:       end if
12:    end for
13:    bestScore ← 0
14:    bestRule ← Null
15:    for all rule ∈ CandidateRules do
16:       good ← countCorrections(rule, CurrentTrainingSet)
17:       bad ← countErrors(rule, CurrentTrainingSet)
18:       score ← good − bad
19:       if score > bestScore then
20:          bestScore ← score
21:          bestRule ← rule
22:       end if
23:    end for
24:    if bestScore > RuleScoreThreshold then
25:       CurrentTrainingSet ← apply(bestRule, CurrentTrainingSet)
26:       LearnedRules ← LearnedRules + bestRule
27:    end if
28: until bestScore > RuleScoreThreshold
output LearnedRules
```

Even when a template set is available for a given task, it may not be effective when we change from a language to another. For instance, dos Santos and Oliveira [10] extend the Ramshaw and Marcus [29] template set, which was handcrafted for English phrase chunking, by adding six templates specifically designed for Portuguese phrase chunking.

Based on the above, it can be concluded that the human driven construction of good template sets is a bottleneck on the effective use of the TBL approach.

2.3 Entropy Guided Template Generation

The main propose of ETL is to overcome the TBL bottleneck. In this section, we explain the strategy for automatic template generation employed by ETL. The template generation process is *entropy guided*. It uses information gain (IG) in order to select the feature combinations that provide good template sets. IG, which is based

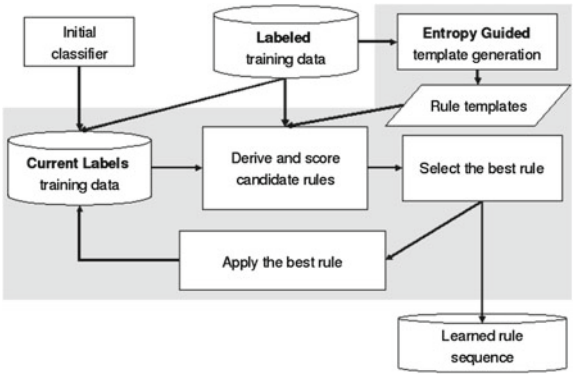

Fig. 2.2 Entropy guided transformation learning

on the data entropy, is a key measure for many feature selection strategies. The ETL algorithm is illustrated in the Fig. 2.2.

The template generation strategy employed by ETL uses decision trees (DT) induction to obtain entropy guided feature selection. The most popular DT learning algorithms [27, 30] use the IG measure in their feature selection step. Moreover, most DT induction algorithms are efficient in terms of CPU and memory usage [30]. Hence, they provide a quick way to obtain entropy guided feature selection. However, DT algorithms that use IG are usually ineffective to deal with high dimensional features. Therefore, in ETL, we developed an effective way to handle high dimensional features. ETL also enables the inclusion of the *current classification* feature in the generated templates. This kind of feature, which changes during the learning process, is not used in ML algorithms like DTs.

The remainder of this section is organized as follows. First, we review the IG measure and the DT learning method. Next, we show how to automatically generate templates from decision trees. Next, we present the ETL true class trick, which enables the generation of templates that use the *current classification* feature. Finally, we detail how ETL handles high dimensional features.

2.3.1 Information Gain

Information gain is a statistical measure commonly used to assess feature relevance [7, 15, 26]. IG is based on the Entropy concept, which characterizes the impurity of an arbitrary collection of examples. Given a training set T whose examples assume classes from the set C. The entropy of T relative to this classification is defined as

$$H(T) = -\sum_{i=1}^{|C|} P_T(c_i) \log_2 P_T(c_i) \tag{2.1}$$

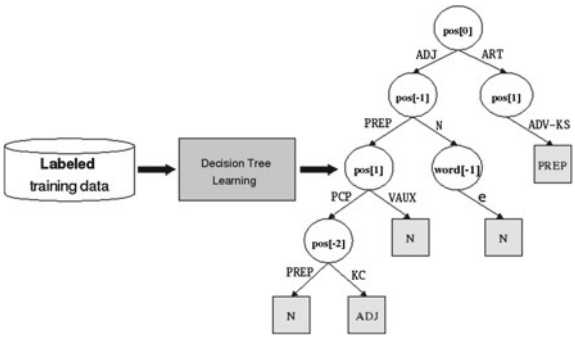

Fig. 2.3 Decision tree learning

where c_i is a class label from C, $|C|$ is the number of classes and $P_T(c_i)$ is estimated by the percentage of examples belonging to c_i in T.

In feature selection, information gain can be thought as the expected reduction in entropy $H(T)$ caused by using a given feature A to partition the training examples in T. The information gain $IG(T, A)$ of a feature A, relative to an example set T is defined as

$$IG(T, A) = H(T) - \sum_{v \in Values(A)} \frac{|T_v|}{|T|} H(T_v) \tag{2.2}$$

where $Values(A)$ is the set of all possible values for feature A, and T_v is the subset of T for which feature A has value v [24]. When using information gain for feature selection, a feature A is preferred to feature B if the information gain from A is greater than that from B.

2.3.2 Decision Trees

Decision tree induction is a widely used machine learning algorithm [26]. Quinlan's C4.5 [27] system is the most popular DT induction implementation. It recursively partitions the training set using the feature providing the largest information gain. This results into a tree structure, where the nodes correspond to the selected features and the arc labels to the selected feature values. After the tree is grown, a pruning step is carried out in order to avoid overfitting.

In Fig. 2.3, we illustrate the DT induction process for Portuguese POS tagging. Here, the five selected features are: `pos[0]`, the POS tag of the current word; `pos[−1]`, the POS tag of the previous word; `pos[1]`, the POS tag of the next word; `pos[−2]`, the POS tag of the word two before; and `word[−1]`, the previous lexical item. The feature values are shown in the figure as arc labels.

We use the C4.5 system to obtain the required entropy guided selected features. We use pruned trees in all experiments shown here.

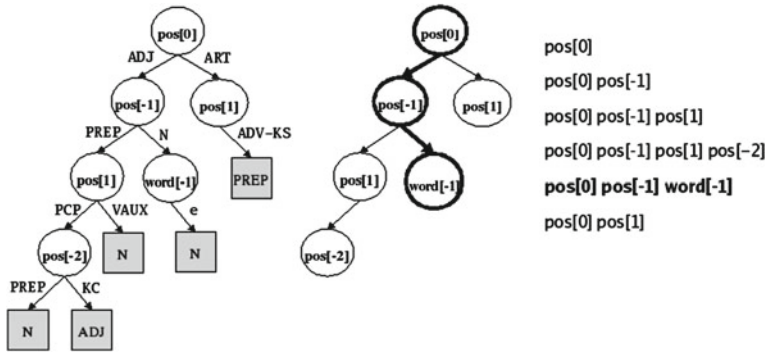

Fig. 2.4 Decision tree template extraction

2.3.3 Template Extraction

In a DT, the more informative features appear closer to the root. Since we just want to generate the most promising templates, we combine first the more informative features. Hence, as we traverse the DT from the root to a leaf, we collect the features in this path. This feature combination provides an information gain driven template. Additionally, paths from the root to internal nodes also provide good templates.

It is very simple to obtain these templates from C4.5's output. From the given DT, we eliminate the leaves and the arc labels. We keep only the tree structure and the node labels. Next, we execute a depth-first traversal of the DT. For each visited tree node, we create a template that combines the features in the path from the root to this node. Figure 2.4 illustrates the template extraction process. In this figure, the template in bold is extracted from the tree path in bold.

2.3.4 True Class Trick

TBL learning can access intermediate results of the classification process as a feature. Moreover, due to the iterative nature of the error correction approach adopted by TBL, the information in the *current classification feature* becomes more precise throughout the rule application process. For instance, when applying TBL to NLP tasks, one usual feature is the current classification of the words within a context window. This behavior is very desirable for NLP tasks, where local classification dependencies play an important role. In order to explore this TBL property in the ETL approach, the current classification feature must be available for selection in the template generation process. We include this feature by providing the DT learner with the initial and true classification of the words. We call this strategy as the *true class trick*.

When creating DT examples, we use the following information as the value of the current classification features: (1) the initial class label for the target word; and (2) the true class label for the neighbor words. Using the true class labels for the neighbor words produce better results, since they contain a precise information. This reflects what the TBL algorithm finds in the context window after some iterations. At this point, the total number of remaining errors is small, dispersed and spread throughout the data. Hence, around an incorrectly classified word is very likely that all the words are correctly classified. Using the true class labels for the target word is not allowed, since it would imply the use of the task solution as an input feature.

The use of the true class labels at training time is a general modeling strategy. It is usual for algorithms that do not have access to intermediate classification results, such as DT and support vector machines (SVM). For instance, when training an AdaBoost system for SRL, Surdeanu et al. [31] use the true class labels of the left side words. At test time, they use the class labels predicted by the classifier, since the classification is done in a left to right fashion. Kudo and Matsumoto [19] use a similar strategy when applying SVM for Phrase Chunking. The advantage of TBL over these algorithms is that, by default, it has access to the current classification of the words in both left and right sides, both at training and at test time. Furthermore, the ETL *true class trick* allows the effective use of this TBL property.

2.3.5 High Dimensional Features

High dimensional features are characterized by having a large number of possible values. For instance, the feature *words* in a text is high dimensional, since it can assume thousands of different values. This kind of feature is ineffective when information gain is used as the feature informativeness measure. IG has a bias that favors high dimensional features over those with a few values [24]. Since we use information gain in the ETL method, through DT learning, we must overcome this problem.

Although the C4.5 system uses the IG ratio measure to avoid the IG bias, the use of high dimensional features is still ineffective. When these features are present in the training set, usually the C4.5 system does not make use of them. Therefore, we include a preprocessing step. This step consists in pruning the high dimensional features in order to retain only their most informative values.

Let T be a training set that contains the high dimensional feature A. For each value v that A assumes in T, we compute its individual information gain using the following equation.

$$IG(T, A, v) = H(T) - \frac{|T_v|}{|T|} H(T_v) \tag{2.3}$$

where $H(T)$ is the entropy of the training set T, T_v is the subset of T for which feature A has value v, and $H(T_v)$ is the entropy of the subset T_v. After the individual IGs are computed, we sort the feature values in decreasing order of IG. Let SV be

Table 2.1 ETL Template evolution

Phase	Template set
1	pos [0]
2	pos [0] pos [−1]
	pos [0] pos [1]
3	pos [0] pos [−1] pos [1]
	pos [0] pos [−1] word [−1]
4	pos [0] pos [−1] pos [1] pos [−2]

the set containing the top z IG values. Then, for each example in T whose value for A is not in SV, we replace that value by a common dummy value. Observe that z is a parameter of the ETL algorithm.

Note that this preprocessing is used only at the DT learning stage. All feature values are maintained at the transformation rule learning stage.

2.4 Template Evolution

TBL training time is highly sensitive to the number and complexity of the applied templates. Curran and Wong [6] argued that we can better tune the *training time* versus *templates complexity* trade-off by using an evolutionary template approach. The main idea is to apply only a small number of templates that evolve throughout the training. When training starts, templates are short, consisting of few feature combinations. As training proceeds, templates evolve to more complex ones that contain more feature combinations. In this way, only a few templates are considered at any point in time. Nevertheless, the descriptive power is not significantly reduced.

ETL provides an easy scheme to implement the template evolution strategy. First, we partition the learned template set by template size. Let T_k be the template set containing all templates of size k, where $k = 1, ..., K$ and K equals to the largest template size. Next, we split the TBL step into K consecutive phases. In phase k, TBL learns rules using only templates from T_k. For instance, using the tree shown in Fig. 2.4, we have four TBL training phases. In Table 2.1, we show the template sets used in the four TBL phases when the tree shown in Fig. 2.4 is used.

Using the template evolution strategy, the training time is decreased by a factor of five for the English phrase chunking task. This is a remarkable reduction, since we use an implementation of the *fastTBL* algorithm [25] that is already a very fast TBL version. Training time is a very important issue when modeling a system with a corpus-based approach. A fast ML strategy enables the testing of different modeling options, such as different feature sets. The efficacy of the rules generated by ETL template evolution is quite similar to the one obtained by training with all the templates at the sametime.

2.5 Template Sampling

Although the template evolution strategy produces a significant speedup in the ETL training time, it has a poor performance when it is necessary to learn all the possible rules at each consecutive TBL phase. If an excessive number of rules are learned in the earlier phases, the posterior phases will have just a few errors to correct. However, the main problem is that the templates of the earlier phases are very simple and may generate poor rules if all the rules with positive score are learned. Therefore, in cases where it is necessary to learn the largest rule set possible, template evolution is not suitable. On the other hand, in many cases this is not a problem. Usually, in order to avoid overfitting, we only learn rules whose score is at least two.

Nevertheless, there are cases where the learning of the largest rule set is necessary. For instance, when training an ensemble of classifiers using different training data sets, overfitting can be beneficial. This is because, in this specific case, overfitting can introduce diversity among the ensemble members. As an example, some DT ensemble learning methods do not use pruning [1, 2, 17].

In our ETL implementation, we also include the *template sampling* functionality, which consists in training the ETL model using only a randomly chosen fraction of the generated templates. Besides being simple, this strategy provides a speed up control that is very useful when multiple ETL models are to be learned.

2.6 Redundant Transformation Rules

As previously noticed by Florian [13], the TBL learning strategy shows a total lack of redundancy in modeling the training data. Only the rule that corrects the largest number of errors is selected at each learning iteration. All alternative rules that may correct the same errors, or a subset of the errors, are ignored. This greedy behavior is not a problem when the feature values tested in the alternative rules and the ones tested in the selected rule always co-occur. Unfortunately, this is not always the case when dealing with sparse data.

Florian includes redundancy in his TBL implementation by adding to the list of rules, after the training phase has completed, all the rules that do not introduce error. Florian shows that these additional rules improve the TBL performance for tasks were a word classification is independent of the surrounding word classifications.

In our ETL implementation, we also include redundancy in the TBL step, but in a different way. At each iteration, when the best rule b is learned, the algorithm also learns all the rules that do not include errors and correct exactly the same examples corrected by b. These redundant rules do not alter the error-driven learning strategy, since they do not provide any change in the training data. Their inclusion is compatible with the standard TBL framework, in the sense that applying the resulting rule set for the training data results in the same number of errors, with or without redundant rules. This kind of redundancy is more effective for low scored rules, since they are

more likely to use sparse feature values and their selection is supported by just a few examples.

For the four tasks presented in this work, the inclusion of redundant rules does not improve the performance of single ETL classifiers. Actually, in some cases there is a decrease in the classification performance. We believe this performance degradation is due to a greater overfitting. Redundant rules increase the overfitting because more information from the training set is included in the learned model. However, the inclusion of redundancy improves the classification quality when several classifiers are combined. Since overfitting can be beneficial when multiple classifiers are used.

2.7 Related Work

Liu et al. [20] present a method for automatic template generation that uses DT. In their method, the tree-guided transformation-based learning (TTBL), two DT's are generated: one that uses all the examples and another that uses only the examples wrongly classified by the initial classifier. They produce the template set by extracting templates from the two trees. They use the second tree with the aim of producing templates that focus on the errors in the initial classification. Liu et al. apply the TTBL strategy to eliminate homograph ambiguity in a Mandarin text-to-speech system. For this task, TTBL obtains results comparable to the ones of handcrafted templates. TTBL is similar to ETL in the sense that they extract templates from DTs. However, the ETL template generation process is more versatile, since it uses the initial classification as an input feature. The ETL *true class trick* enables the use of the current classification feature in an effective way. Moreover, ETL [8] has been published earlier than TTBL [20].

An evolutionary scheme based on genetic algorithms (GA) to automatically generate TBL templates is presented by Milidiú et al. [21, 22]. Using a simple genetic coding, the generated template sets show an efficacy close to the one of handcrafted templates. The main drawback of this strategy is that the GA step is computationally expensive, since it is necessary to run the TBL algorithm in order to compute the fitness for each individual of the population. If we need to consider a large context window or a large number of features, it becomes infeasible.

Corston-Oliver and Gamon [5] present a combination of DTs and TBL. They derive candidate rules from the DT, and use TBL to select and apply them. Their work is restricted to binary features only. ETL strategy extracts more general knowledge from the DT, since it builds rule templates. Furthermore, ETL is applied to any kind of discrete features.

Carberry et al. [4] introduce a randomized version of the TBL framework. For each error, they try just a few randomly chosen templates from the given template set. This strategy speeds up the TBL training process, enabling the use of large template sets. However, they use handcrafted templates and variations of them, which implies that a template designer is still necessary.

Hwang et al [18] use DT decomposition to extract complex feature combinations for the weighted probabilistic sum model (WPSM). They also extract feature combinations that use only some nodes in a tree path. This is required to improve the effectiveness of the WPSM learning. Their work is similar to ours, since they use DT's for feature extraction. Nevertheless, the ETL learned template set is simpler than theirs, and is enough for effective TBL learning.

References

1. Banfield, R.E., Hall, L.O., Bowyer, K.W., Kegelmeyer, W.P.: Ensemble diversity measures and their application to thinning. Inf. Fusion **6**(1), 49–62 (2005)
2. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001). doi:10.1023/A:1010933404324
3. Brill, E.: Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. Comput. Linguist. **21**(4), 543–565 (1995)
4. Carberry, S., Vijay-Shanker, K., Wilson, A., Samuel, K.: Randomized rule selection in transformation-based learning: a comparative study. Nat. Lang. Eng. **7**(2), 99–116 (2001). doi:10.1017/S1351324901002662
5. Corston-Oliver, S., Gamon, M.: Combining decision trees and transformation-based learning to correct transferred linguistic representations. In: Proceedings of the Ninth Machine Tranlsation Summit, pp. 55–62. Association for Machine Translation in the Americas, New Orleans (2003)
6. Curran, J.R., Wong, R.K.: Formalisation of transformation-based learning. In: Proceedings of the ACSC, pp. 51–57, Canberra (2000)
7. Dash, M., Liu, H.: Feature selection for classification. Intell. Data Anal. **1**, 131–156 (1997)
8. dos Santos, C.N., Milidiú, R.L.: Entropy guided transformation learning. Technical Report 29/07, Departamento de Informática, PUC-Rio (2007). http://bib-di.inf.puc-rio.br/techreports/2007.htm
9. dos Santos, C.N., Milidiú, R.L.: Probabilistic classifications with TBL. In: Proceedings of Eighth International Conference on Intelligent Text Processing and Computational Linguistics—CICLing, pp. 196–207, Mexico (2007)
10. dos Santos, C.N., Oliveira, C.: Constrained atomic term: widening the reach of rule templates in transformation based learning. In: Portuguese Conference on Artificial Intelligence—EPIA, pp. 622–633 (2005)
11. Elming, J.: Transformation-based corrections of rule-based MT. In: Proceedings of the EAMT 11th Annual Conference, Oslo (2006)
12. Florian, R.: Named entity recognition as a house of cards: classifier stacking. In: Proceedings of CoNLL-2002, pp. 175–178, Taipei (2002)
13. Florian, R.: Transformation based learning and data-driven lexical disambiguation: syntactic and semantic ambiguity resolution. Ph.D. Thesis, The Johns Hopkins University (2002)
14. Florian, R., Henderson, J.C., Ngai, G.: Coaxing confidences from an old friend: probabilistic classifications from transformation rule lists. In: Proceedings of Joint Sigdat Conference on Empirical Methods in NLP and Very Large Corpora. Hong Kong University of Science and Technology, Kowloon (2000)
15. Forman, G., Guyon, I., Elisseeff, A.: An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. **3**, 1289–1305 (2003)
16. Higgins, D.: A transformation-based approach to argument labeling. In: Ng, H.T., Riloff, E. (eds.) HLT-NAACL 2004 Workshop: Eighth Conference on Computational Natural Language Learning (CoNLL-2004), pp. 114–117. Association for Computational Linguistics, Boston (2004)

17. Ho, T.K.: The random subspace method for constructing decision forests. IEEE Trans. Pattern Anal. Mach. Intell. **20**(8), 832–844 (1998). doi:10.1109/34.709601
18. Hwang, Y.S., Chung, H.J., Rim, H.C.: Weighted probabilistic sum model based on decision tree decomposition for text chunking. Int. J. Comput. Process. Orient. Lang. **16**(1), 1–20 (2003)
19. Kudo, T., Matsumoto, Y.: Chunking with support vector machines. In: Proceedings of the NAACL-2001 (2001)
20. Liu, F., Shi, Q., Tao, J.: Tree-guided transformation-based homograph disambiguation in mandarin TTS system. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4657–4660, Cambridge (2008)
21. Milidiú, R.L., Duarte, J.C., dos Santos, C.N.: Evolutionary TBL template generation. J. Braz. Comput. Soc. **13**(4), 39–50 (2007)
22. Milidiú, R.L., Duarte, J.C., dos Santos, C.N.: TBL template selection: an evolutionary approach. In: Proceedings of Conference of the Spanish Association for Artificial Intelligence—CAEPIA, Salamanca (2007)
23. Milidiú, R.L., dos Santos, C.N., Duarte, J.C.: Phrase chunking using entropy guided transformation learning. In: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies—ACL-08: HLT, Columbus (2008)
24. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
25. Ngai, G., Florian, R.: Transformation-based learning in the fast lane. In: Proceedings of North Americal ACL, pp. 40–47 (2001)
26. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986). doi:10.1023/A:1022643204877
27. Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann, San Francisco (1993)
28. Ramshaw, L., Marcus, M.: Exploring the statistical derivation of transformational rule sequences for part-of-speech tagging. In: Proceedings of the Balancing Act-Workshop on Combining Symbolic and Statistical Approaches to Language, pp. 86–95. Association for Computational Linguistics, Toulouse (1994). http://www.citeseer.ist.psu.edu/article/ramshaw94exploring.html
29. Ramshaw, L., Marcus, M.: Text chunking using transformation-based learning. In: Armstrong, S., Church, K., Isabelle, P., Manzi, S., Tzoukermann, E., Yarowsky, D. (eds.) Natural Language Processing Using Very Large Corpora. Kluwer, Dordrecht (1999)
30. Su, J., Zhang, H.: A fast decision tree learning algorithm. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence—AAAI (2006)
31. Surdeanu, M., Johansson, R., Meyers, A., Màrquez, L., Nivre, J.: The CoNLL 2008 shared task on joint parsing of syntactic and semantic dependencies. In: CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning, pp. 159–177. Coling 2008 Organizing Committee, Manchester (2008). http://www.aclweb.org/anthology/W08-2121

http://www.springer.com/978-1-4471-2977-6

Entropy Guided Transformation Learning: Algorithms and Applications

dos Santos, C.N.; Milidiú, R.L.

2012, XIII, 78 p. 10 illus., Softcover
ISBN: 978-1-4471-2977-6