4
Machine Learning Methods and Classification Models

The presented chunk derivation heuristic is well suited for any corpus that already contains phrase structure annotations, requiring only adaptations regarding the corpus format and the available phrase kinds. However, we also want to be able to determine our proposed chunk structure for corpora with no previous syntactic annotation, i.e., unstructured text data.

Machine Learning methods come into play in this situation. Given that we already have a corpus with "golden" chunk annotation, that is, chunk tags that are correct according to our predefined rules, we are able to use this corpus as training data for a supervised learning algorithm. Such a supervised learning method discovers patterns in data labeled with golden tags and other features, and then can reapply the learned patterns to unseen data in order to annotate it automatically.

We can cite two particularly interesting characteristics of this approach. First, the use of such an automatic extraction method minimizes the need for domain experts when solving a particular task, provided that a considerable amount of previously annotated data is already available. Second, it is possible to apply a myriad of supervised learning techniques to the same task and using the same training data, and evaluate which one performs best. The many approaches tried for text chunking at CoNLL-2000 attest this fact.

We choose to apply the Entropy Guided Transformation Learning technique, an evolution of Transformation-Based Learning, in this work. We describe both of them in the next sections. Afterwards, we give design details about the models we propose for text chunking.

4.1
Machine Learning algorithms

TBL (9) is a rule-based error-driven learning algorithm. As will be clarified later in this section, it depends on the previous definition of *rule templates* that serve as the basis for its learned rules.

ETL (21) is an evolution of TBL which eliminates the requirement of setting those rule templates, determining them automatically through an

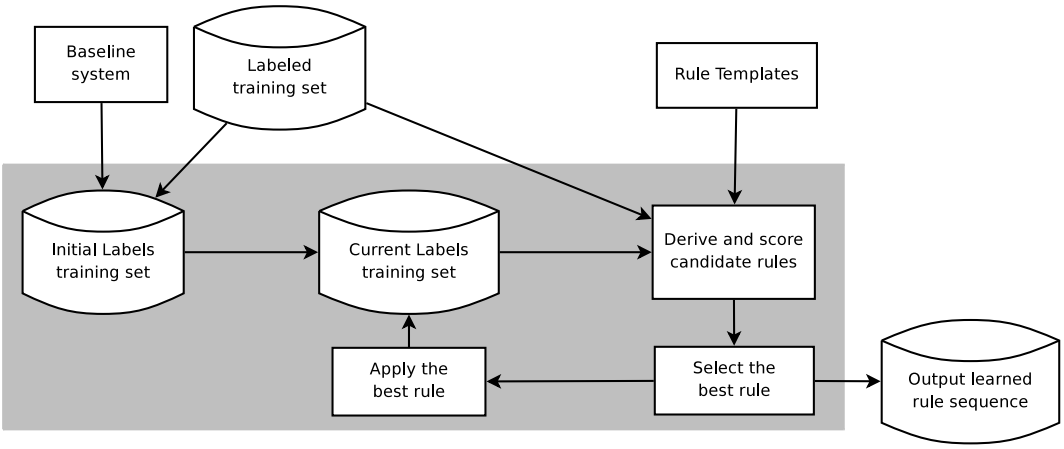

Figure 4.1: A schematic of the Transformation-Based Learning algorithm

pos[-1] word[0] ck[0]

Figure 4.2: A Transformation-Based Learning rule template

entropy-oriented method.

4.1.1 Transformation-Based Learning

As reported by dos Santos (21), TBL enjoyed success when applied to a number of NLP tasks, like part-of-speech tagging (9, 25), noun phrase extraction (51, 43), spelling correction (39), appositive extraction (26), named entity recognition (45) and semantic role labeling (31). Since it is a supervised learning algorithm, it naturally requires a labeled corpus as input. The input is also composed of the aforementioned rule templates and a baseline classifier. This classifier determines the initial classification based on which the algorithm generates the resulting transformation rules. Figure 4.1 shows a schematic summarizing the TBL learning phase.

A TBL rule template is simply a set of attributes from a token and the tokens in its vicinity. One example of template is shown in figure 4.2. This template takes into account the actual word and chunk tag of the current token and the POS tag of the token immediately before it.

When particular values for those template attributes are set and a transformation is specified, the result is one of TBL's rules. Figure 4.3 demonstrates two possible rules for the given template.

The role of rule templates is to limit the space of considered rules. Therefore, it is important to choose templates that are relevant to the task at hand. This typically depends on having specific domain knowledge, and limits TBL's versatility somewhat.

pos[-1]=**preposition** word[0]=**os** ck[0]=**I-NP** → ck[0]=**B-NP**
pos[-1]=**article** word[0]=**homem** ck[0]=**B-NP** → ck[0]=**I-NP**

Figure 4.3: Transformation-Based Learning rules

TBL is an iterative method, and greedily selects a new rule to be applied after each step. Its first action is to apply the baseline classifier to the provided training corpus. This baseline system, being a naive classifier, generates an output which expectedly has many tagging errors to be corrected. Thus, in every step, based on the rule templates given as input, TBL derives all possible correction rules and uses a score function to evaluate the effectiveness of each rule. The best scoring rule is then selected, applied to the current state of the training corpus and saved. This rule selection process is repeated until all possible next rules yield a score below a specified threshold, meaning that no remaining rule is good enough to be selected.

The score function for rules can be as simple as the difference between the number of corrections and the number of errors a given rule produces when applied to the corpus. Indeed, this is the function used by our ETL models.

4.1.2 Entropy Guided Transformation Learning

ETL has been thrivingly used for many of the NLP tasks to which TBL has also been applied, like POS tagging (22), noun phrase extraction (42), named entity recognition (44, 19) and semantic role labeling (19), improving TBL's results consistently. It has also been utilized to tackle text chunking for other languages (42). Many ETL-based systems can be freely used through the F-EXT-WS service[1]. (24).

ETL solves what is referred to as the TBL bottleneck: the construction of relevant rule template sets (21). That enables the effective application of ETL without specific domain knowledge for the task considered. It does that by applying the Information Gain concept through means of a Decision Tree. Information Gain is based on data entropy, and several feature selection strategies rely on it to restrict the feature set and make learning algorithms more efficient. Our implementation of ETL employs the C4.5 Decision Tree algorithm (50) for template selection.

ETL's learning phase is very similar to TBL's, except for the added initial step for template generation. Following the application of the chosen baseline classifier to the training corpus, the Decision Tree algorithm is invoked,

[1] `http://www.learn.inf.puc-rio.br`

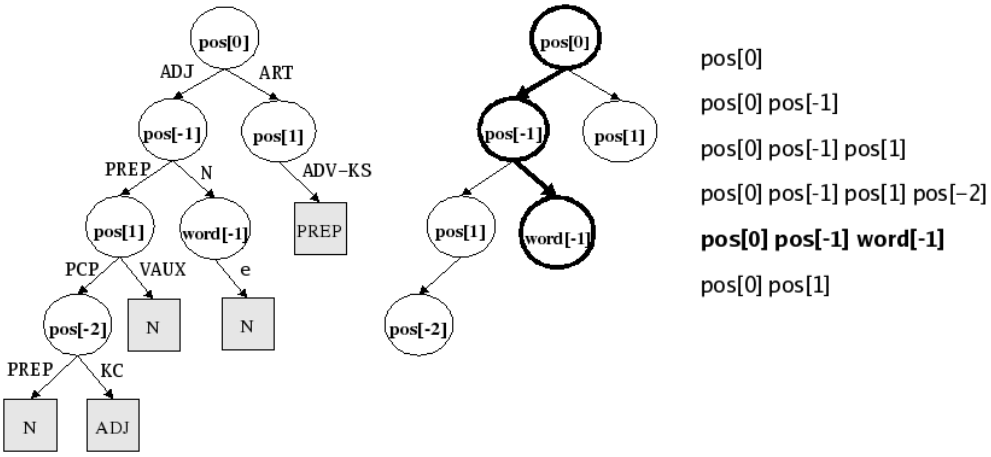

Figure 4.4: Template generation in Entropy Guided Transformation Learning

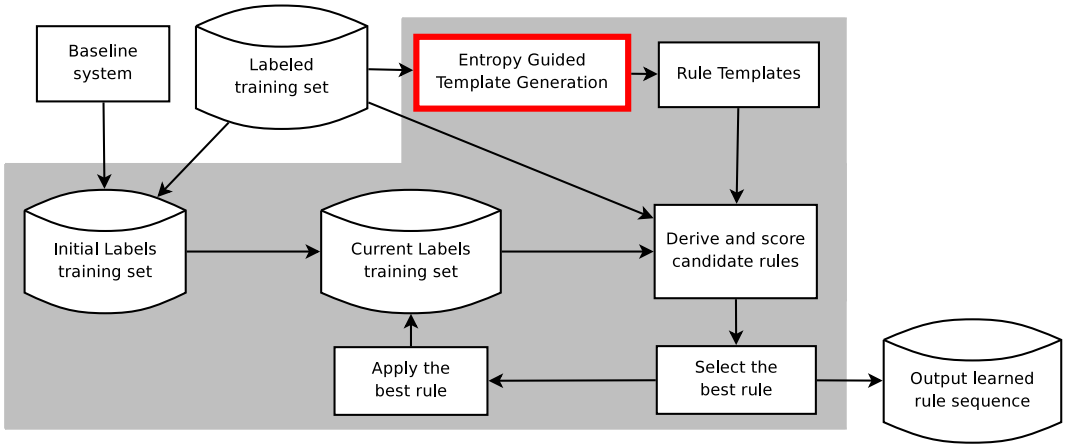

Figure 4.5: A schematic of Entropy Guided Transformation Learning

generating a tree targeted at the classification of the attribute wanted. The features taken into account when instantiating the tree are the ones of the token being classified and those of the surrounding tokens. The number of surrounding tokens watched is set through the ETL "window" parameter.

After the creation of the tree, the paths from the root of the tree to every leaf are set as the model's rule templates. In other words, given one of the paths of the tree, the nodes in this path become the attributes in a derived template. Figure 4.4 elucidates this process, displaying a Decision Tree and its resulting templates.

ETL then uses the generated templates in the same way TBL does, by greedily selecting the best scoring rule at each step until no rule score reaches a given threshold. A complete schematic of ETL is shown in figure 4.5.

This work also makes use of a technique intended to lower the training time and memory consumption of an ETL model for some of the tackled tasks. This technique is called *template evolution*, and is described by dos Santos in

his work (21). It consists in processing groups of templates divided by their size in a progressive manner. This means that, at the beginning of the rule learning phase, only rules from templates with one attribute are generated; when no more one-attribute rules can be created, then only two-attribute templates are taken into account, and so on.

At the end, the application of rules during extraction follows the order in which the rules were created, exactly like the regular TBL and ETL extraction process. The reduced training time comes at the cost of very little performance loss.

4.2 Chunk extraction models

We propose two models for the extraction of chunks in Portuguese corpora. One model is a direct classifier, extracting chunk tags in the IOB2 format. The second model decomposes the chunking task into three smaller subtasks, and thus we refer to it as the "subtasks classifier".

In our experiments, we apply the ETL technique as the classification algorithm for these approaches, but any of them can be replicated using another supervised learning algorithm.

More details about these approaches are given in the next sections.

4.2.1 Direct classifier

Our direct classifier applies the ETL algorithm to determine IOB2 tags for tokens in a corpus in a single step. The only features this corpus needs to have beforehand are the words and POS classes for each token, which are the fundamental attributes used by text chunking learning models in general. We also create *derived features* based on these two provided features in order to make the learning model consider certain patterns more easily. These features are described in a following section.

The baseline system employed for this model is the same one described in the CoNLL-2000 section in chapter 2. For each POS tag, we determine the chunk tag most frequently associated with it among the training corpus examples. This BLS classifies a given token precisely with the chunk tag corresponding to its POS tag according to the above procedure.

4.2.2 Subtasks classifier

The subtasks classifier relies on two minor classification tasks and a subsequent matching task. The classification subtasks are responsible for the detection of chunk boundaries. The first subtask finds tokens that start a chunk, tagging these tokens according to the type of chunks they start. The second subtask is similar to the first, doing the same procedure for tokens that end chunks. Finally, the last subtask takes the results of the previous ones and matches start tokens and end tokens appropriately. Table 4.1 shows the tokens of a sentence, their tags corresponding to the mentioned classification subtasks, and the expected final result after the matching task in IOB2 format.

		Chunks		
Word	**POS**	**Start**	**End**	**IOB2**
A	art	NP	X	B-NP
Juve	prop	X	NP	I-NP
vive	v-fin	VP	VP	B-VP
a	art	NP	X	B-NP
transição	n	X	NP	I-NP
para	prp	PP	PP	B-PP
uma	art	NP	X	B-NP
nova	adj	X	X	I-NP
temporada	n	X	NP	I-NP
.	.	X	X	O

Table 4.1: Example of output for the intermediate steps of the subtask classifier

Once more, our approach to the classification subtasks is based on the ETL technique. As denoted by table 4.1, the tag set for these tasks consists simply in the group of chunk types considered, and an additional `X` tag for tokens that do not begin or end a chunk. A pertinent detail is that the chunk end classifier uses the results from the chunk start extractor as a feature, after it is trained and reapplied on the training corpus. This provides the former with extra information while also taking the patterns learned by the latter into consideration.

The matching task is not solved through a tagging approach. Rather, a straightforward heuristic is applied. It iterates through every pair (T_S, T_E) of tokens, where T_S is a start token and T_E is an end token of the same type and appearing later in the sentence. Such a pair is considered valid if there is no other start or end token between T_S and T_E. The heuristic then generates the chunks corresponding to every valid pair found. This procedure guarantees that the resulting chunks are non-recursive and non-overlapping.

Since the chunk end subtask depends on the chunk start one and the matching task depends on the previous two, they are executed in sequence. We also use the derived features mentioned in the direct classifier section for the two classification subtasks to improve their results. Additionally, the baseline systems used for the two first subtasks follow the same principle as the one employed by the direct classifier: they detect the class most frequently associated with a POS tag and attribute that class to all tokens that have the given POS tag.

4.2.3 Derived features

When designing a Machine Learning system, one of the most important actions is to provide the model with a set of features that exposes useful patterns in the dataset. Although text chunking systems are typically given only two fundamental features, the word value of each token and their POS tags, there are ways to manipulate this basic data and create more features that directly improve the effectiveness of the learning algorithm. Examples of feature manipulation have been shown in chapter 2, when we describe state-of-the-art approaches for English text chunking.

Along the development of this work, a number of those derived features have been created and applied to the described models. Two of the tested features resulted in substantial performance gains and were kept in our final settings.

One of the improving attributes is called *predecessor verb*. For every token, it holds the word value of the closest verb to the left.

The other derived feature is *capitalization*. It is generated from the word feature and may assume one of the following classes:

1. first letter is uppercase;
2. all letters are uppercase;
3. all letters are lowercase;
4. plain number;
5. number with embedded "-" or "/" characters;
6. punctuation;
7. other, i.e., none of the above.

It is possible to verify that, because ETL rules depend on the matching of discrete class values and only consider a fixed number of surrounding tokens, some of the details captured by these derived features would be ignored by the model if they were not explicitly extracted. For example, the capitalization feature allows the model to create rules that involve all words that have a starting capital letter. This kind of generalization would not be possible if only the word attribute were available.