4
Algoritmos de Aprendizado

Este capítulo apresenta os algoritmos utilizados ao longo da dissertação e alguns utilizados como base por eles. Os algoritmos adotados são todos de aprendizado supervisionado. Isso significa que eles aprendem a partir de exemplos anotados. Começamos apresentando o classificador bayesiano ingênuo; em seguida são apresentados o classificador unigrama, as árvores de decisão, o TBL e o ETL.

4.1
Classificador Bayesiano Ingênuo

O classificador bayesiano ingênuo, mais conhecido como classificador *Naïve Bayes*, é uma versão simplificada do classificador bayesiano. Como todo classificador, sua função é definir a qual classe um dado item pertence. Neste caso, isso é feito a partir da probabilidade do item pertencer a cada uma das classes. Assim, temos:

$$classe = \arg\max_{classe_i} Pr[classe_i|f_1, ..., f_n]$$

onde $f_1, ..., f_n$ são as características que compõe o item que se quer classificar, $classe_i$ é uma das possíveis i classes e $classe$ é a classificação.

Utilizando o Teorema de Bayes, que serve para inverter probabilidades condicionais, temos:

$$classe = \arg\max_{classe_i} \frac{Pr[classe_i] * Pr[f_1, ..., f_n|classe_i]}{P[f_1, ..., f_n]} .$$

Como a probabilidade do item é igual para todas as classes, o denominador da equação pode ser ignorado, e assim ficamos com:

$$classe = \arg\max_{classe_i} Pr[classe_i] * Pr[f_1, ..., f_n|classe_i] .$$

Neste ponto, o classificador ingênuo assume que a probabilidade das características do item são independentes. Isso nos permite simplificar a

equação anterior, obtendo

$$classe = \arg\max_{classe_i} Pr[classe_i] * \prod_{j=1}^{n} Pr[f_j|classe_i] \ .$$

Em muitos dos casos essa afirmação de independência não está de acordo com a realidade, por isso ele é considerado ingênuo. Nesses casos, não podemos assumir que as probabilidades calculadas estejam corretas. Mesmo assim, este método simples gera usualmente um classificador de boa qualidade.

4.2 Classificador Unigrama

Um classificador unigrama atribui a cada *token* sua etiqueta mais frequente no corpus de treino. Por exemplo, a palavra *dever* aparece 18 vezes como substantivo e 5 vezes como verbo no corpus Mac-Morpho [9]. Sendo assim, se treinarmos um classificador unigrama com esse corpus, ele sempre irá classificar a palavra *dever* como substantivo. Apesar de simples, ele apresenta 78,26% de acurácia para a palavra *dever* no Mac-Morpho.

Utilizamos esse classificador para auxiliar na tradução do corpus na Seção 5.2.1 e para classificar palavras conhecidas apresentado na Seção 6.1.

4.3 Árvores de Decisão

Árvores de decisão são funções de classificação representadas por árvores. Elas são bastante utilizadas devido a sua simplicidade e transparência no processo de classificação. Cada nó da árvore representa um teste que deve ser feito em um determinado atributo do objeto a ser classificado. Os galhos que saem deste nó são as possíveis respostas ao teste e as folhas são as classes as quais o objeto pode pertencer.

A Figura 5.3 mostra uma árvore de decisão criada para classificar meios de transporte.

Um dos algoritmos mais utilizados para a construção de árvores de decisão é o C4.5 [10]. O algoritmo de geração da árvore de decisão utiliza uma técnica gulosa, visando construir a menor árvore que particione o conjunto de treino em classes. A cada etapa, é escolhido o atributo que particiona o conjunto de treino, gerando o maior ganho de informação naquele ponto. O teste deste atributo é então adicionado ao modelo como um novo nó. Esse novo nó irá particionar o conjunto de treino naquele ponto. Se não há mais atributos a serem escolhidos, ou se não há mais ganho de informação, é criada uma folha com a classe predominante do que sobrou do conjunto de treino naquele ponto.

Após criada, a árvore é podada para evitar que seu modelo seja especializado no conjunto de treino.

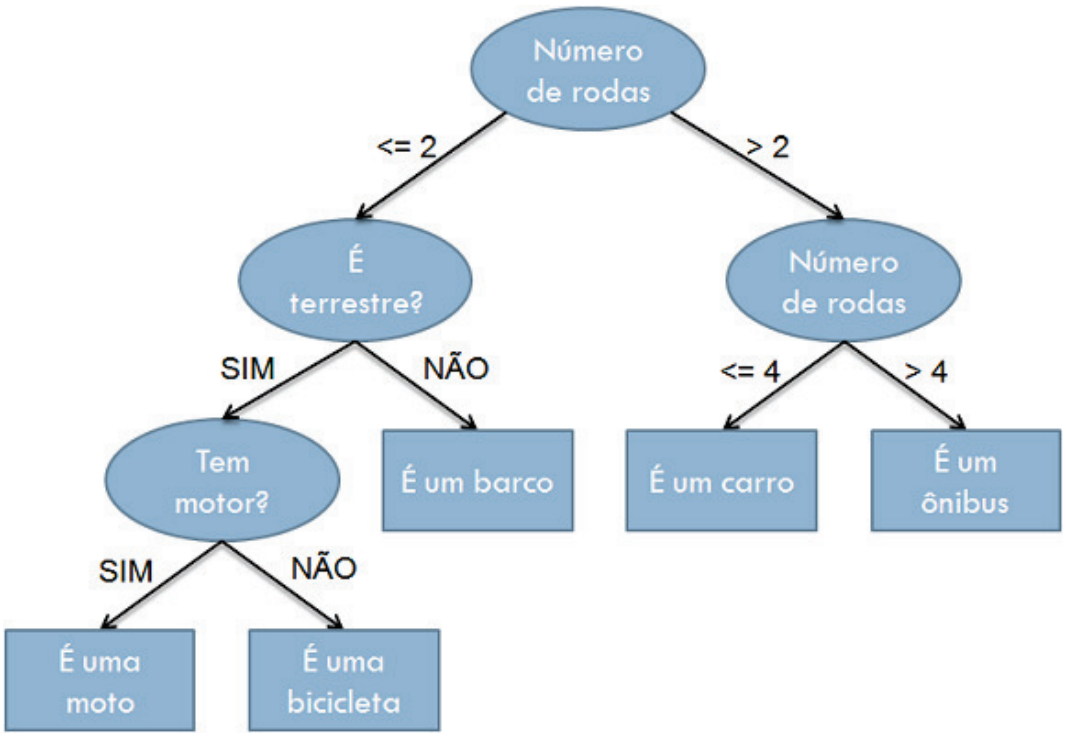

Figura 4.1: Exemplo de árvore de decisão que classifica meios de transporte.

4.4 Transformation-Based Learning

Em 1995, Eric Brill propôs um novo algoritmo para o processamento de linguagem natural, com uma abordagem baseada em correção de erros, o *Transformation-Based Learning* (TBL) [11]. Desde então, ele foi usado com sucesso, alcançando o estado da arte em várias tarefas, como por exemplo, *Named Entity Recognition* e *Noun Phrase Chunking*.

Para treinar o TBL é necessário ter os seguintes itens:

- um corpus anotado;
- um classificador simples;
- gabaritos de geração de regras de correção de erros.

Os gabaritos indicam quais informações devem ser consideradas para gerar regras de correção. A geração desses gabaritos é o gargalo do TBL. É necessário um grande conhecimento do domínio em que se está trabalhando,

para saber que informações são interessantes para se gerar regras. Em nosso caso, o domínio é a anotação morfossintática do português-twitter.

A Figura 4.2, por exemplo, apresenta um gabarito utilizado para gerar regras de correção considerando o POS da palavra sendo corrigida (**pos[0]**), o POS da palavra anterior a ela (**pos[-1]**) e a palavra seguinte (**word[1]**).

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

Figura 4.2: Exemplo de um gabarito de geração de regras de correção.

O algoritmo TBL tem como objetivo minimizar as diferenças entre a classificação predita e a classificação anotada no corpus de treino, gerando regras que corrijam a classificação inicial dada ao corpus de treino. A Figura 4.3 ilustra a relação entre os componentes que compõe o treinamento do TBL.

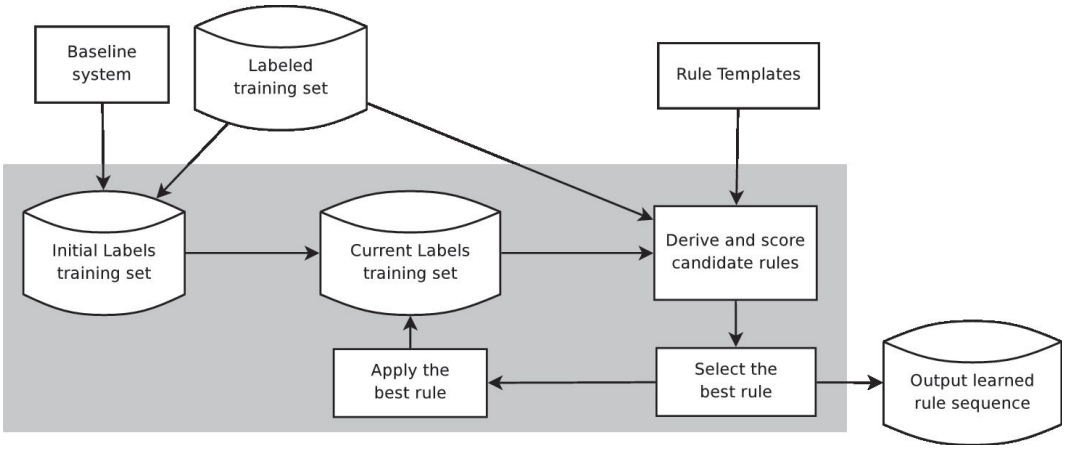

Figura 4.3: Treinamento do TBL.

De forma mais detalhada, o treinamento do TBL funciona da seguinte maneira:

1. O classificador inicial - *baseline system* (BLS) - tenta predizer a classificação de cada *token* de uma versão não anotada do corpus de treino.

2. As classificações preditas e as anotadas do corpus de treino são comparadas e para cada erro de classificação, são criadas regras a partir dos gabaritos.

   A Figura 4.4 mostra o exemplo de uma regra de correção gerada a partir do modelo de regra apresentado na Figura 4.2. Essa regra indica que se o POS da palavra corrente for um adjetivo, o POS da palavra anterior

for um verbo e a palavra seguinte for "para", então o POS da palavra corrente deverá ser alterado para advérbio.

$$\textbf{pos[0]}=ADJ \quad \textbf{pos[-1]}=V \quad \textbf{word[1]}=\textit{'para'} \implies \textbf{ADV}$$

Figura 4.4: Uma regra gerada a partir do gabarito da Figura 4.2.

3. Essas regras, se aplicadas a todo corpus, podem corrigir outras entradas, ou gerar novos erros. Para cada regra gerada, é atribuída uma pontuação calculada, subtraindo o total de entradas que aquela regra corrige, pelo erros que ela gera.

4. A regra de maior pontuação é adicionada ao modelo e é atribuída a classificação predita corrigindo-a.

5. O processo se repete a partir do passo 2, até que seja atingido o critério de parada. O critério de parada é a pontuação mínima que uma regra deve ter para fazer parte do modelo. O valor padrão atribuído a esse limite é 1, o que quer dizer que enquanto for possível gerar regra que corrija mais do que estrague a classificação do corpus, o processo irá continuar.

O modelo gerado pelo treinamento é composto do classificador inicial e das regras de correção.

4.5 Entropy Guided Transformation Learning

O algoritmo *Entropy Guided Transformation Learning* (ETL) foi desenvolvido em 2007 [12]. Ele utiliza o algoritmo de árvore de decisão para resolver o gargalo do TBL, automatizando a geração de gabaritos de regras de correção. A Figura 4.5 ilustra a relação entre os componentes que compõe o treinamento do ETL e destaca o elemento que o difere do treinamento do TBL. Ele foi aplicado com sucesso para tarefas de anotação morfossintática [13, 14], *phrase chunking* [13, 7, 8], *named entity recognition* [13, 8], *clause identification* [15], e *semantic role labeling* [8]; gerando resultados pelo menos tão bons quanto os obtidos pelo TBL com modelos de regras escritos a mão.

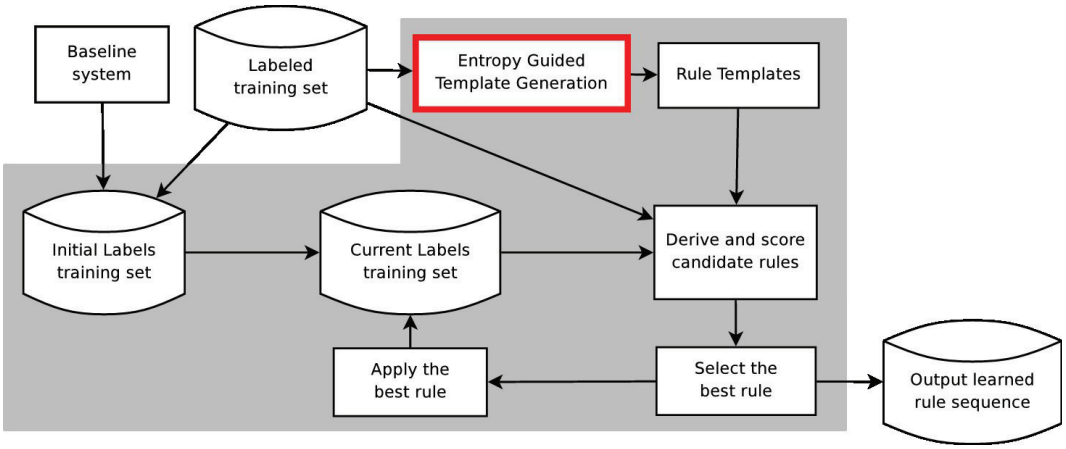

Figura 4.5: Treinamento do ETL.

Para criar os gabaritos de regras, o ETL gera uma árvore de decisão a partir do corpus anotado. Nessa árvore, é feita uma busca em profundidade, gerando as correspondentes regras de decisão. De cada uma destas regras, extraímos apenas a lista de atributos verificados pela mesma. Esta lista é então um gabarito de regra. Assim, o conjunto de gabaritos é gerado automaticamente a partir do conjunto de regras de decisão. Finalmente, o ETL aplica o TBL, utilizando o conjunto de gabaritos construído na etapa anterior. O processo de geração de *templates* é ilustrado na Figura 4.6.

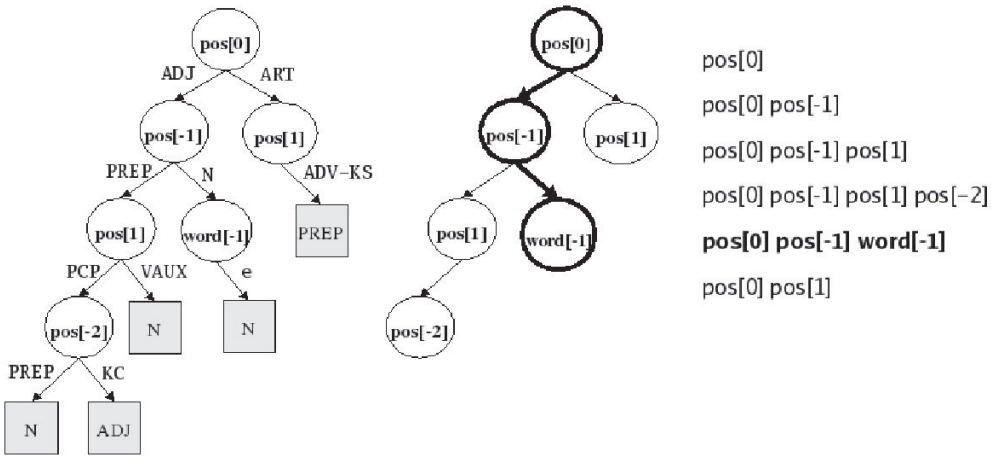

Figura 4.6: Geração dos templates.

Para treinar o ETL é necessário ter os seguintes items:

– um corpus anotado;

– um classificador simples;

– um tamanho de janela.

A janela atribui contexto aos *tokens* durante a etapa de geração dos gabaritos pela árvore de decisão. Por exemplo, na frase da Figura 4.7, ao analisar a palavra 'Twitter' com uma janela de tamanho 3, a árvore de decisão deverá considerar os atributos das palavras 'O' e 'é', como atributos da palavra 'Twitter'.

[**O/ART Twitter/NPROP é/V**] muito/ADV legal/ADJ ./.

Figura 4.7: Exemplo de janela no ETL.

Consequentemente, os atributos do *token*, representado pela palavra 'Twitter', seriam: **word[0]**=Twitter **pos[0]**=NPROP **word[-1]**=O **pos[-1]**=ART **word[1]**=é **pos[1]**=V

Assim como o TBL, o modelo gerado pelo treinamento é composto do classificador inicial e das regras de correção.