UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA

Disciplina Processamento de Linguagem Natural
Docente Carlos Augusto Prolo
Discente Felipe Cortez de Sá

Parts-of-speech tagger

1 Introdução

Este relatório descreve a implementação de dois algoritmos para realizar *parts-of-speech tagging* e os resultados obtidos utilizando o EVALB para avaliação.

2 Programa

O programa foi desenvolvido utilizando a linguagem de programação Python 3. Antes de realizar o *tagging*, é preciso treinar o modelo para cada estratégia. O programa criará um diretório `models` onde os modelos são salvos em arquivos `.pkl`, formato *pickle* que serializa estruturas de dados e podem ser posteriormente recuperados em outras execuções. Para rodar o programa, digite na linha de comandos

```
$ python3 [-a naive | viterbi] [-t "treebank" | -s "frase" | -i "arquivo com frases"]
```

A flag `-a` é utilizada para selecionar o algoritmo desejado e as opções são *naive* e *viterbi*. Caso a flag seja omitida, o algoritmo padrão é o Viterbi.

A flag `-t` treina o algoritmo selecionado (ingênuo ou Viterbi) com o *treebank* do corpus passado como argumento e salva o modelo em um arquivo `.pkl`.

A flag `-s` executa o tagger para uma frase passada como argumento.

A flag `-i` executa o tagger para o arquivo de entrada com uma frase por linha. O programa escreverá os resultados do *tagging* no arquivo de mesmo nome da entrada com o sufixo `.tst`.

Os algoritmos implementados são derivados da classe `Tagger` definida em `common.py` e devem implementar os métodos abstratos `train`, `load_model` e `tag_sentence`. A função `tag_file` é implementada pela classe base e chama `tag_sentence` em todas as frases de um arquivo de entrada.

3 Estratégias

3.1 Algoritmo ingênuo

3.1.1 Treinamento

Para armazenar as partes do discurso mais comuns para cada palavra, um *hash map* é indexado por palavras e partes do discurso, e o valor é a quantidade de ocorrências de cada parte para cada palavra. `words_pos["word"]["NN"]`, por exemplo, retorna a quantidade de terminais na árvore de treino com formato `(NN word)`.

A função `tag_sentence` escolhe para cada palavra a *tag* com maior frequência, ou seja, $\arg\max_{tag}$ words_pos[word][tag] . Caso a palavra não seja encontrada, é escolhida a *tag* mais comum "NN".

3.2 Viterbi

3.2.1 Treinamento

São calculados os bigramas, adicionados ao *hash map* `bigrams`, e as frequências de palavras para cada parte do discurso. Em vez de `words_pos[word][tag]`, a ordem é invertida e utiliza-se `pos_words[tag][word]`. Ao terminar a contagem de frequências, os dois *hash maps* são normalizados, transformando as contagens em probabilidades.

Para calcular os bigramas para começo e fim de frase, foram criadas as partes do discurso `"START"` e `"END"`.

3.2.2 Tagging

O *hash map* `viterbi` armazena os valores do algoritmo de Viterbi e o *hash map* `backpointer` armazena os ponteiros para determinar a sequência mais provável no final do algoritmo.

Uma dificuldade encontrada foi a baixa ocorrência de algumas palavras no corpus. Uma palavra rara resultava em baixas probabilidades, podendo zerar um passo da matriz. Como os passos seguintes são calculados com base nos anteriores, o erro era propagado, sendo impossível determinar *tags* para as palavras a partir daquele ponto.

A medida inicial tomada para tratar o problema foi repetir as probabilidades do passo anterior caso fosse detectado que o passo não tinha probabilidades maiores que zero.

Uma segunda estratégia foi identificar, no treinamento, quais palavras ocorriam menos de x vezes no corpus, sendo x determinado pela constante `UNKNOWN_TRESHOLD`. As palavras raras encontradas são contabilizadas como uma única palavra com *string* dada pela constante `UNKNOWN_STR`. No processo de *tagging*, caso uma palavra não seja encontrada no corpus, ela é tratada como `UNKNOWN_STR`, recebendo a probabilidade de ter certa *tag* dado que é uma palavra rara.

4 Resultados

Utilizou-se o corpus do Penn Treebank para treinamento, desenvolvimento e validação. Os algoritmos foram treinados utilizando o arquivo `traindata`, ajustes no algoritmo foram feitos com o arquivo `bank0` e as árvores em `section23` foram convertidas em frases com o programa `fromTreeToSentence`. Utilizou-se então o EVALB para comparar o resultado do *tagger* utilizando as frases da `section23` para comparar com o *gold standard* `section23`.

4.1 Ingênuo

90% com EVALB.

4.2 Viterbi

93.64% sem heurística, 94.33% com heurística de palavras desconhecidas usando o Penn Treebank.