Linguística Computacional - MAC5725
Instituto de Matemática e Estatística
Universidade de São Paulo

Treinamento de Etiquetadores Morfossintáticos por HMMs

Fernando Leandro dos Santos

30 de Setembro de 2018

Nota: A-/B+- Boa apresentação- Ausência (imperdoável) de r

1 Introdução

Definição do problema

Este artigo se propõe à apresentar o desenvolvimento de um etiquetador morfossintático através de um modelo oculto de Markov, de agora em diante, abreviado por HMM. Em outras palavras, o programa aqui apresentado é capaz de classificar palavras do português em determinadas classes morfossintáticas. Nessa implementação são realizadas algumas simplificações, utilizamos uma quantidade fixa de etiquetas possíveis, uma quantidade fixa de palavras conhecidas e utilizamos um *corpus* já etiquetado para treinamento do modelo desenvolvido.

Para implementação do HMM foi necessária a utilização de um *corpus* de sentenças etiquetadas, isto é, uma grande variedade de sentenças do português com as respectivas classificações morfossintáticas para cada palavra. O *CETENFolha-1.0_jan2014.cg.gz*, utilizado neste trabalho, está disponível em `https://linguateca.pt/`.

Solução proposta

Neste trabalho foram desenvolvidas duas implementações de etiquetadores. A primeira delas é um modelo basal simples que faz a contagem de palavras e etiquetas para prever a etiqueta mais provável para cada palavra. O segundo etiquetador é uma implementação do algoritmo de Viterbi e será apresentado em mais detalhes nos próximos capítulos.

Para comparação e validação desses modelos a métrica utilizada foi a acurácia, ou seja, a quantidade de palavras em que o modelo faz a classificação correta da etiqueta sobre a quantidade total de palavras analisadas.

2 Modelos de Markov

Foi você que inventou as Cadeias de Markov?Se não foi, é OBRIGATÓRIO re

Um modelo de Markov é um modelo estocástico utilizado para modelar mudanças em um sistema qualquer. Como característica básica, esse modelo assume a hipótese de Markov, isto é, ele assume que a mudança de estado para estados futuros depende exclusivamente do estado atual e não dos estados anteriores. Na maioria dos casos onde utilizamos uma rede de markov, essa hipótese não é verdadeira, mas torna possível a computação das probabilidades desse modelo. No caso deste trabalho, sabemos que em uma sentença, a etiqueta de uma palavra depende bastante das etiquetas das palavras anteriores, mas assumimos que ela depende exclusivamente da etiqueta anterior e não das etiquetas antecedentes. Mesmo com essa forte hipótese, veremos que conseguimos resultados interessantes.

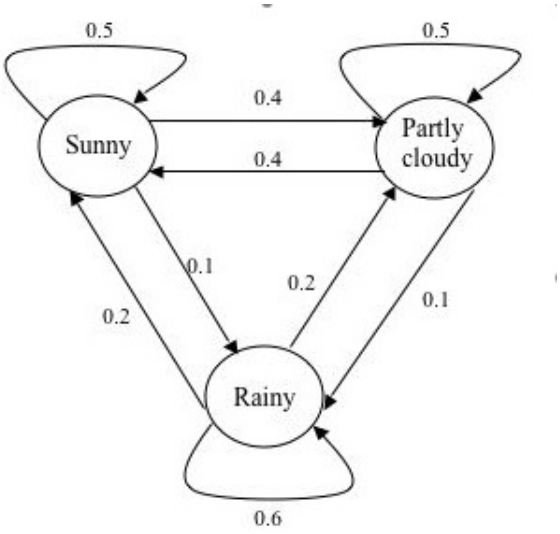

Figura 1: Exemplo de uma cadeia de markov para os estados "Cloudy", "Rainy"e "Partly cloudy"

A versão mais simples de um modelo de Markov é uma cadeia de Markov, isto é, um conjunto de estados onde as probabilidades de mudança para o próximo estado estão definidas exclusivamente pelo estado atual, como mostrado na figura 1. Já um modelo oculto de Markov é uma cadeia de Markov onde os estados não são necessariamente observáveis, e por isso, são observadas características ou atributos relacionados ao estado. Por exemplo, se pensarmos no modelo da figura 1 como uma rede oculta, não conseguiríamos observar diretamente se estamos no estado "Sunny"ou "Rainy", mas poderíamos, por exemplo, observar o evento de "Ver pessoas com guarda-chuvas"ou "Ver pessoas com roupas de praia". Dessa forma, temos um modelo que consegue prever e estimar probabilidades para um estado utilizando outras observações referentes àquele estado. No caso deste trabalho, estaremos estimando a probabilidade de uma etiqueta morfossintática aparecer em uma determinada posição da sentença (estado) dada a sequência de palavras (observação).

Modelo Oculto de Markov de ordem 2

Agora que já temos uma noção do que são modelos de Markov, segue uma definição mais formal do modelo desenvolvido nesse trabalho. Nessa implementação, consideramos um modelo de Markov de ordem 2, isto é, estamos assumindo que o estado atual depende exclusivamente

dos 2 últimos estados da cadeia. Em outras palavras, a probabilidade da etiqueta na posição i depende somente das etiquetas nas posições $i-1$ e $i-2$.

Imagine uma sentença $x_i...x_n$ onde $x_i \in V$. V é o conjunto de palavras do vocabulário e n é o número de palavras da sentença. Para essa sentença, temos uma sequência de etiquetas $y_i...y_{n+1}$ onde $y_i \in S$ e $y_{n+1} = STOP$. S é o conjunto de etiquetas possíveis e $STOP$ é a etiqueta que define o final da sentença. A partir dessas definições temos que a probabilidade conjunta da sentença e do seu conjunto de etiquetas é:

$$p(x_i...x_n, y_i...y_{n+1}) = \prod_{i=1}^{n+1} q(y_i|y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i|y_i)$$

onde $q(y_i|y_{i-2}, y_{i-1})$ é a probabilidade de vermos a etiqueta y_i logo após a sequência y_{i-2} y_{i-1} e $e(x_i|y_i)$ é a probabilidade de vermos a palavra x_i dado que a etiqueta dessa posição i é y_i. Para isso, assumimos também que $x_0 = x_{-1} = *$ (símbolo que identifica o início da sentença).

Essa definição da probabilidade da sentença é muito útil, uma vez que o valor máximo dessa probabilidade nos dá a sequência de etiquetas mais provável para uma determinada sentença $x_i...x_n$. E essa sequência é justamente o que queremos neste trabalho. Assim, o que queremos encontrar na verdade, é a sequência de etiquetas $y_i...y_n$ que maximiza a probabilidade p, isto é:

$$\arg\max_{y_i...y_n} p(x_i...x_n, y_i...y_{n+1})$$

Dada essa definição, é necessário estimar os parâmetros da probabilidade de emissão e e da probabilidade de transição q para um modelo oculto de Markov de ordem 2:

- $e(x|s)$ para qualquer $s \in S, x \in V$
- $q(s|u, v)$ para qualquer $s \in S \cup \{STOP\}, u, v \in S \cup \{*\}$

No capítulo de implementação, será dada uma descrição mais detalhada de como esses parâmetros foram estimados neste trabalho.

Algoritmo de Viterbi

Como apresentado na seção anterior, para classificar uma sentença com a sua sequência de etiquetas mais provável, é necessário identificar a sequência de etiquetas que maximiza a probabilidade p. Entretanto, uma busca por força bruta em todas as sequências possíveis é inviável, dado que teríamos uma complexidade de tempo exponencial com relação ao tamanho da sequência n. Mais especificamente, teríamos $|S|^n$ sequências possíveis, onde $|S|$ é o número de etiquetas possíveis. Esse problema é tratado com o algoritmo de Viterbi, apresentado a seguir.

- Temos como entrada do algoritmo a sentença $x_i...x_n$ e os parâmetros $q(s|u, v)$ e $e(x|s)$
- Definimos o conjunto possível de etiquetas S_k de acordo com a posição k na sentença da seguinte forma:

$$S_{-1} = S_0 = \{*\}$$
$$S_k = S, k \in \{1...n\}$$

- Definimos um estado $\pi(k, u, v)$ como a probabilidade máxima de uma sequência de etiquetas qualquer acabar com as etiquetas u e v, respectivamente, na posição k, ou seja, $u \in S_{k-1}$ e $v \in S_k$.
- Inicializamos π com $\pi(0, *, *) = 1$
- Segue o algoritmo que nos retorna a probabilidade máxima para a sequência de entrada:

Faltou legenda para referência no texto.

---

for $k = 1...n$:
    for $u \in S_{k-1}$:
        for $v \in S_k$:
            $\pi(k, u, v) = \max_{w \in S_{k-2}}(\pi(k-1, w, u) * q(v|w, u) * e(x_k|v))$
retorne $\max_{u \in S_{n-1}, v \in S_n}(\pi(n, u, v) * q(STOP, u, v))$

- Entretanto, precisamos dos argumentos que geram essa probabilidade e não da probabilidade em si. Para isso fazemos o uso de uma outra tabela que guardará as etiquetas de cada probabilidade máxima, resultando no seguinte algoritmo:

---

for $k = 1...n$:
    for $u \in S_{k-1}$: iDEM
        for $v \in S_k$:
            $\pi(k, u, v) = \max_{w \in S_{k-2}}(\pi(k-1, w, u) * q(v|w, u) * e(x_k|v))$
            $bp(k, u, v) = \arg\max_{w \in S_{k-2}}(\pi(k-1, w, u) * q(v|w, u) * e(x_k|v))$
$(y_{n-1}, y_n) = \arg\max_{u \in S_{n-1}, v \in S_n}(\pi(n, u, v) * q(STOP, u, v))$
for $k = (n-2)...1$:
    $y_k = bp(k+2, y_{k+1}, y_{k+2}$:
retorne a sequência $y_1...y_n$

Analisando esse algoritmo, vemos uma complexidade de tempo bem menor do que por força bruta. Temos uma complexidade da ordem de $O(n|S|^3)$ ao invés de $O(|S|^n)$, dado que iteramos 3 vezes sobre S para para cada palavra da sentença de tamanho n. No próximo capítulo, será dada uma descrição mais detalhada de como o algoritmo foi implementado e que estruturas foram utilizadas.

3 Implementação

O desenvolvimento deste trabalho foi realizado com a linguagem Python (v3.6) e alguns pacotes auxiliares como *sklearn*, *nltk*, e *numpy*. O sistema operacional utilizado para desenvolvimento e teste foi o Windows 10, com 16GB de memória RAM e um processador Intel Core i7 2.20GHz.

Pré-processamento das sentenças

A primeira parte da implementação consiste em ler o *corpus CETENFolha* e separá-lo em sentenças, além de identificar corretamente a etiqueta morfossintática de cada palavra das sentenças. Para essa tarefa foram utilizadas simplesmente algumas expressões regulares para remover informações não utilizadas e a quebra de sentenças através da sequência dupla de nova linha ("\n\n"). Em mais detalhes, foi removido dos dados qualquer texto entre tags <> e colchetes *[ ]*, conforme define a função *clean_and_split*.

Seleção de etiquetas

Após a limpeza e separação das sentenças, o próximo passo foi a seleção de etiquetas. O corpus não é perfeitamente etiquetado e possui alguns erros ou anotações fora do padrão esperado. Dessa forma, foi necessário filtrar palavras identificadas como etiquetas que não eram as etiquetas esperadas. Para isso, foi feita uma análise e definida a seguinte lista de etiquetas permitidas.

- Etiquetas: 'N', 'DET' ,'PRP', 'V', 'PROP', 'ADJ' ,'ADV', 'NUM', 'KC','SPEC','PERS', 'KS', 'IN', 'EC', 'PRON', '__PUNCT__'

Qualquer palavra detectada como uma etiqueta não pertencente à essa lista foi removida da base, e consequentemente, a sentença dessa palavra foi também completamente removida. Foram removidas 1.469 sentenças com essa filtragem, totalizando 1.693.101 de sentenças após o pré-processamento.

Outra estratégia de implementação que vale ressaltar aqui é a pontuação. Inicialmente, a implementação foi feita tratando cada pontuação possível como uma etiqueta única (isto é, cada pontuação existente era uma etiqueta separada: vírgula, ponto final, dois pontos, abre parenteses, fecha parenteses, etc..), porém essa estratégia nos levou à um total de 42 etiquetas, o que tornou a implementação do algoritmo de Viterbi muito lenta (devido à complexidade cúbica do algoritmo com relação ao número de etiquetas). Por esse motivo, foi feita uma simplificação onde qualquer pontuação existente (detectada pelo caractere inicial $) foi marcada com a etiqueta __PUNCT__. Após essa definição, ficamos com 16 possíveis etiquetas. O gráfico mostra a distribuição dessas etiquetas no *corpus* pré-processado.

Palavras raras

Para fazer uma validação mais realista do sucesso da implementação, é necessário separar uma parte dos dados para treinar o modelo e uma parte dos dados não vista pelo treinamento. Essa é uma prática muito comum quando se trabalha com aprendizado de máquina. Assim, pode acontecer de aparecer palavras na base de teste que nunca foram vistas na base de treino. Para resolver esse problema, a seguinte estratégia foi desenvolvida: durante a fase de treino (contagem das palavras), qualquer palavra que tenha frequência total na base de treino menor ou igual a 5, foi convertida para a palavra __RARE__ e de forma análoga, antes da avaliação dos modelos na fase de teste, palavras da base de teste que não foram vistas no treino também foram convertidas para __RARE__. Dessa maneira, é possível simular a aparição de palavras pouco comuns na base de treino para que seja feita uma previsão para palavras não vistas na base de teste.

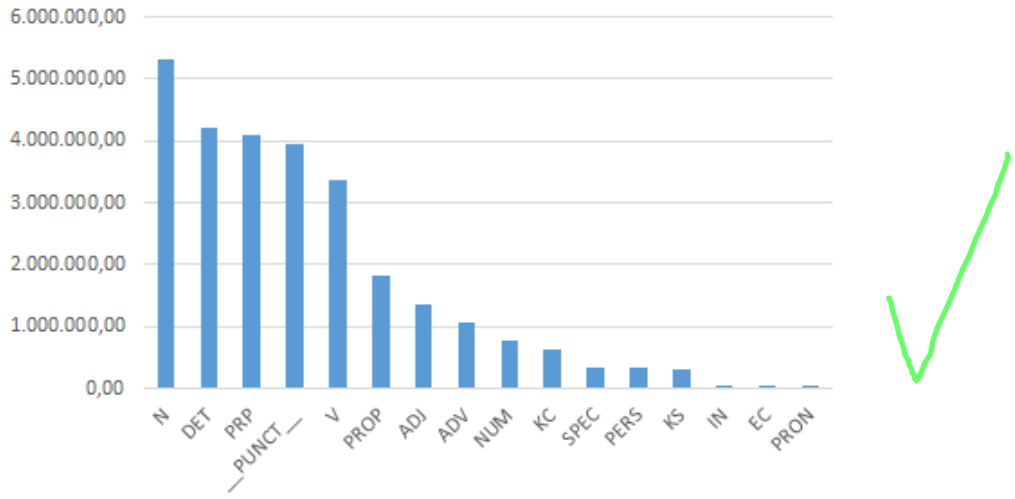

Figura 2: Frequência de etiquetas no corpus pré-processado

Modelo basal: Contagem de frequências

Essa implementação é simplesmente uma contagem de palavras. Através de uma iteração completa por todas as sentenças e palavras da base de treino, é criado um dicionário python onde as chaves são palavras e o valor é outro dicionário com as etiquetas possíveis para àquela palavra e a frequência de cada etiqueta. Com essa simples estrutura, é possível então, iterar sobre todas as sentenças da base de teste e classificar cada palavra com a sua etiqueta mais provável (etiqueta de maior frequência dada a palavra). Isso é realizado com uma busca única no dicionário para cada palavra da base. Esse procedimento é realizado na função *get_train_counts*.

Vale ressaltar que a complexidade de tempo para uma busca em um dicionário de n chaves no python é, amortizadamente, O(1) (constante) e não linear, visto que a implementação de dicionários no python é feita com HashMaps.

Parâmetros do modelo oculto de Markov

Os parâmetros $e(x|s)$ e $q(s|u,v)$ são calculados, respectivamente, pelas funções *get_emission_probs* e *get_transition_probs*. A probabilidade de emissão é simplesmente uma contagem da freqûencia de uma palavra x com etiqueta s sobre o total de ocorrências da etiqueta s. Vale ressaltar que essa contagem foi suavizada pela suavização de La Place, ou seja, somando 1 à todas as frequências. Dessa forma, casos onde a palavra nunca foi emitida através daquela etiqueta terão a frequência mínima 1.

A probabilidade de transição de etiquetas $q(s|u,v)$ é um pouco mais complexa. Através da utilização de uma função do pacote *nltk* para cálculo de bi-gramas e tri-gramas, foram calculados a contagem de cada possível bi-grama e tri-grama dos dados de treino. A partir disso, foi calculada a frequência de todos os possíveis tri-gramas sobre a frequência do respectivo bi-grama (formado pelos dois primeiros termos do tri-grama).

Os dois parâmetros foram armazenados como dicionários Python onde as chaves são tuplas (s,v) e triplas (u,v,s), para, e e q, respectivamente. Essa estratégia foi tomada no intuito de minimizar os tempos de buscas para essas probabilidades, dado que possuem complexidade

média de tempo constante.

Implementação do algoritmo de Viterbi

O algoritmo de Viterbi foi implementado de forma direta na função *viterbi*. Recebe como parâmetros as sentenças de entrada, a lista de etiquetas possíveis e os parâmetros e e q. As tabelas de programação dinâmica π e bp foram implementadas também como dicionários python com suas chaves sendo a tripla (k, u, v).

Avaliação

Os testes foram realizados com o método de validação cruzada com 5 pastas. O *corpus* pré-processado foi dividido aleatoriamente em 5 partes e os testes foram feitos em 5 rodadas diferentes. Cada rodada usou 1 parte diferente da base para teste e as 4 partes restantes para treino dos parâmetros, garantindo assim, que o modelo seja capaz de generalizar as suas previsões para diferentes versões de dados nunca antes observadas durante o treinamento. Foram utilizadas 5 ao invés de 10 pastas (o que é mais comum na literatura), devido ao longo tempo de processamento para cada pasta (aproximadamente 5 horas). Essa divisão da base de dados foi feita com o auxílio do pacote *sklearn*, que possui funções específicas para esse propósito.

Conforme apresentado na introdução, a métrica utilizada para avaliação do modelo foi a acurácia. Para cada modelo, o basal e o de Viterbi, foi calculado o número de palavras classificadas corretamente sobre o total de palavras. Mais detalhes dessa avaliação estão na função *evaluate_viterbi*.

4 Resultados

Apresentamos na tabela abaixo o resultado final comparativo dos dois modelos implementados. É possível perceber valores bem similares entre as diferentes rodadas de execução, o que indica boa consistência dos dados e dos modelos de forma geral, indicando que estão robustos à variações de palavras em diferentes partes do *corpus*.

Percebe-se também que o modelo basal, apesar de simples e de rápida execução, já possui uma performance considerável no contexto de etiquetamento morfossintático, com acurácia de 94.58%. Dado que essa acurácia já é relativamente alta, é necessário bastante esforço e processamento para melhorá-la em alguns pontos percentuais. É o que nos indica as métricas da coluna do algoritmo de Viterbi. Foram necessárias 25 horas de processamento para avaliar o modelo de Viterbi nas 5 pastas e obter uma acurácia média de 96.27% enquanto que o processamento para o modelo basal é quase que instantâneo.

Rodada	*Acertos Viterbi*	*Acurácia Viterbi*	*Tempo de processamento (horas)*	*Acertos Basal*	*Acurácia Basal*	*Total de palavras*
$k = 1$	5.297.854	96.27%	5.19	5.204.619	94.58%	5.502.802
$k = 2$	5.294.017	96.27%	5.05	5.201.878	94.59%	5.499.002
$k = 3$	5.317.937	96.28%	5.00	5.224.428	94.59%	5.523.073
$k = 4$	5.313.382	96.29%	4.96	5.219.220	94.58%	5.517.860
$k = 5$	5.292.300	96.26%	5.00	5.199.382	94.57%	5.497.557
Média		**96.275%**	**25.20**		**94.587%**	**27.540.294**

5 Conclusão

Foi apresentado nesse artigo o desenvolvimento de uma solução para o problema de etiquetamento morfossintático para sentenças do português. Dois modelos foram desenvolvidos, um modelo basal que faz a contagem de palavras e etiquetas e classifica palavras pela etiqueta mais provável e também o modelo seguindo a implementação do algoritmo de Viterbi que utiliza um modelo oculto de Markov de ordem 2 para encontrar a sequência de etiquetas mais provável para uma determinada sentença.

O código foi desenvolvido em Python e está bem organizado. Entretanto, seria possível implementar esse mesmo modelo em linguagens de maior performance. Devido à limitação do tempo de processamento, uma grande simplificação foi feita, onde todas as pontuações foram consideradas de forma igual, como o termo __PUNCT__. Muito provavelmente essa simplificação prejudica o modelo, uma vez que um ponto final, uma vírgula ou o ponto-e-vírgula são tratados como o mesmo caractere. Essa seria uma possível melhoria para a implementação aqui desenvolvida. Outro ponto de melhoria seria no tratamento de palavras raras. Neste trabalho somente foi feita uma tratativa para converter palavras raras para o termo __RARE__. Porém, esse tratamento poderia ser melhorado convertendo esses termos raros e não vistos para categorias mais detalhadas, como, por exemplo, números, datas, siglas e outras categorias através de um pré-processamento desses termos raros. Dessa forma teríamos um modelo mais assertivo em sentenças que possuem palavras raras. Outros métodos poderiam ser desenvolvidos uma vez que temos as probabilidades iniciais do modelo de Markov, como por exemplo o algoritmo Forward-Backward.

Por fim, o desenvolvimento deste trabalho trouxe bastante aprendizado para o autor. Considero que as principais dificuldades dessa implementação estiveram relacionadas com o cálculo das probabilidades de transição $q(s|u, v)$ e também com o entendimento de como tratar os dados entre treino e teste para lhe dar com emissões e transições não vistas na base de treino.

REFERÊNCIAS???