Instituto de Matemática e Estatística

Linguística Computacional

MAT5725

# Treinamento de Etiquetadores Morfossintáticos por HMMs

*Autor:*
Diego Andrés Méndez

8 de outubro de 2018

# Sumário

## 1. Introdução

O objetivo deste artigo é descrever a implementação de um etiquetador morfossintático utilizando o algoritmo de Viterbi, por meio de modelos de Markov ocultos (HMMs) através da linguagem Python. A implementação encontra-se no programa "Etiquetador_Morfossintático_DAM.ipynb", disponibilizado juntamente com este artigo.

O programa utiliza o córpus CETENFolha-1.0_jan2014.cg.gz, disponível gratuitamente no site linguateca.pt. Esse córpus possui, aproximadamente, 24 milhões de palavras em português brasileiro, extraídas de edições do jornal Folha de São Paulo durante o ano de 1994. Foram usadas somente as classificações morfossintáticas das palavras, equanto que as anotações sintáticas de Gramáticas de Restrições, também presentes do cóprpus, foram descartadas.

Para o entendimento adequado deste artigo, espera-se que o leitor possua conhecimentos básicos de programação em Python e de lógica matemática. Durante a leitura, serão apresentados conceitos de preprocessamento de texto, de HMMs e do algoritmo de Viterbi, além de detalhes específicos da implementação do etiquetador morfossintático.

## 2. Modelos de Markov ocultos (HMMs)

Uma cadeia de Markov M de ordem k é uma sequência de variáveis aleatórias $X_{-k}, X_{-k+1}, \ldots X_{-1}, X_0, X_1, \ldots X_n, X_{n+1}$ que assumem valores em um conjunto E, tal que a distribuição de $X_i$ depende somente de $X_{i-k}, ..., X_{i-1}$, para todo $i(0 \leq i \leq n+1)$. $X_{-k}, \ldots, X_{-1}$ são ditos os estados iniciais, cujas distribuições são dadas, e $X_{n+1}$ é dito o estado final de M.

Um modelo de Markov oculto H é uma cadeia de Markov, tal que, para cada elemento $X_i(0 \leq i \leq n)$, existe uma variável aleatória $Y_i$ que assume valores em um conjunto F e a distribuição de $Y_i$ depende somente de $X_i$. F é dito o conjunto dos estados observáveis, enquanto E é dito o conjunto dos estados ocultos.

Exemplo de modelo de Markov oculto de ordem 1: desde que acorda, a cada hora, um bebê pode chorar ou sorrir, dependendo se está com fome ou se está satisfeito. Sabe-se que, quando acorda, o bebê está sempre com fome. A qualquer hora o bebê pode dormir e a observação se encerra. O modelo de Markov oculto é totalmente determinado pelos seguintes dados:
$E = \{satisfeito, fome, dormir\}$, $F = \{chorar, sorrir\}$,

$X_{-1} = fome$
Para todo $i(0 \leq i \leq n+1)$:
$P(X_i = satisfeito | X_{i-1} = satisfeito) = 0,4$,
$P(X_i = fome | X_{i-1} = satisfeito) = 0,3$,
$P(X_i = dormindo | X_{i-1} = satisfeito) = 0,3$,
$P(X_i = satisfeito | X_{i-1} = fome) = 0,7$,
$P(X_i = fome | X_{i-1} = fome) = 0,2$,
$P(X_i = dormndo | X_{i-1} = fome) = 0,1$,

$P(Y_i = chorar | X_i = satisfeito) = 0,3$,
$P(Y_i = sorrir | X_i = satisfeito) = 0,7$,
$P(Y_i = dormir | X_i = satisfeito) = 0$,
$P(Y_i = chorar | X_i = fome) = 0,9$,
$P(Y_i = sorrir | X_i = fome) = 0,1$,
$P(Y_i = dormir | X_i = fome) = 0$,
$P(Y_i = chorar | X_i = dormindo) = 0$,
$P(Y_i = sorrir | X_i = dormindo) = 0$,
$P(Y_i = dormir | X_i = dormindo) = 1$,

Resumindo as informações, temos uma matriz $A = \begin{bmatrix} 0,4 & 0,4 & 0,3 \\ 0,7 & 0,2 & 0,1 \end{bmatrix}$, onde os elementos $a_{rs}$ representam as probabilidades de transição dos estado $X_{i-1}$ para o estado $X_i$ e temos uma matrix $B = \begin{bmatrix} 0,3 & 0,7 & 0 \\ 0,9 & 0,1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, onde os elementos $b_{rs}$ reprensentam as probabilidades de $Y_i$, dado $X_i$. Note que a soma dos elementos de uma mesma linha é sempre igual a 1, tanto na matriz A como na matriz B. A é dita a matriz de transições de estados ocultos, e B é dita a matriz de emissões de estados visíveis. Como A e B caracterizam H, frequentemente deixamos essa dependência explícita denotando H(A,B) em vez de H.

## 3. O algoritmo de Viterbi

Seja H(A,B) um modelo de Markov oculto de ordem k, onde A é a matriz de probablilidades de transições de estados ocultos e B é a matriz de probabilidade de emissões de estados visíveis. Denote que os valores ocultos de H(A,B) assumam valores em E e os valores visíveis de H(A.B) assumam valores em F. Dadas as distribuições para os estados iniciais $(X_{-k}, \ldots, X_{-1})$ e a sequência de observações $(o_0, o_1, \ldots o_n)$ geradas por H(A,B), o objetivo do algoritmo de Viterbi é determinar a sequência de estados ocultos $\{q_0, q_1, \ldots, q_n\}$ mais provável para o cenário. O algoritmo pode ser calculado recursivamente da seguinte forma. Para cada $o_i (0 \leq i \leq n+1)$, calcule

$$p^i_{s_{i-k},\ldots,s_i} =$$
$$P(X_i = s_i | (X_{i-k}, \ldots, X_{i-1}) = (s_{i-k}, \ldots, s_{i-1})) \cdot$$
$$P(Y_i = o_i | X_i = s_i) \cdot$$
$$P((X_{i-k}, \ldots, X_{i-1}) = (s_{i-k}, \ldots, s_{i-1}))$$

, para todo $(s_{i-k}, \ldots, s_i) \in E^{k+1}$.
A expressão acima é equivalente, para todo $i (1 \leq i \leq n+1)$, a:

$$p^i_{s_{i-k},\ldots,s_{i-1}} = a_{rs} \cdot b_{tu} \cdot \max_{s_{i-k-1} \in E} p^{i-1}_{s_{i-k-1},\ldots,s_{i-1}}$$

, onde $a_{rs}$ é o elemento de A que representa a probabilidade de transição dos estados $s_{i-k}, \ldots, s_{i-1}$ para o estado $s_i$ e $b_{tu}$ é o elemento de B que representa a probabilidade de emissão do estado observável $o_i$ dado o estado oculto $s_i$.
Note que $p^0_{s_{-k},\ldots,s_0}$ é conhecido para todo $(s_{-k}, \ldots, s_0) \in E^{k+1}$, pois as distribuições de $X_{-k}, \ldots, X_{-1}$ são dadas, o que permite calcular $P((X_{i-k}, \ldots, X_{i-1}) = (s_{i-k}, \ldots, s_{i-1}))$ explicitamente, sendo esta a condição inicial da recursão.
Para finalmente encontrar a sequência de estados ocultos $(q_0, \ldots, q_n)$ mais provável, temos que

$$(q_{n+1-k}, \ldots, q_{n+1}) = \underset{(s_{n+1-k},\ldots,s_{n+1}) \in E^{k+1}}{argmax} p^{n+1}_{s_{n+1-k},\ldots,s_{n+1}}.$$

Em seguida, definimos $q_{i-k} = \underset{s_{i-k} \in E}{argmax}\, p^i_{s_{i-k},q_{i-k+1},\ldots,q_i}$, para todo $i (k \leq i \leq n)$ e portanto encontramos a sequência completa $(q_0, \ldots q_n)$ dos estados ocultos mais prováveis para as observações $o_0, \ldots o_n$.

## 4. Finalidade do programa

O objetivo principal do programa é implementar o algoritmo de Viterbi para etiquetagem morfossintática utilizando cadeias de Markov ocultas (HMMs). Isso foi feito através de aprendizado supervisionado, onde 80% do córpus CETENFolha-1.0_jan2014.cg.gz foram a base para treino e os outros 20% foram a base para teste. No caso do CETENFolha, as classificações morfossintáticas utilizadas foram:
. → Pontuações que, em geral, determinam o fim das sentenças. Ex: . ! ?
, → Pontuações que, em geral, não determinam o fim das sentenças. Ex: , ) [ ”
ADJ → Adjetivos. Ex: chato, japonês, dolorosas
ADV → Advérbios. Ex: onde, não, mais, muito
DET → Determinantes. Ex: os, uma, este
EC → Elementos compostos. Ex: inter, aero, pan
FIM → Fim das sentenças. Artificialmente representado pela palavra %%%%%
IN → Interjeições Ex: rarará, ai, ops
INICIO → Inícios de sentença
KC → Conjunções coordenativa. Ex: e, nem, ou
KS → Cconjunções subordinativa. Ex: se, que, pois
N → Nomes. Ex: ligação, pessoas, formação
NUM → Numerais. Ex: 3, 6, 2
PERS → Pronomes pessoais se, lo, ele
PROP → Nomes próprios. Ex: Ibirapuera, Clinton, China
PRP → Preposições. Ex: em, de, por
SPEC → Pronomes não pessoais. Ex: que, quem, nada, tudo
V → Verbos. Ex: fazer, deverão, existe

O algoritmo pode ser replicado utilizando outro(s) córpus para treinamento e/ou teste, porém é recomendado sempre que o contexto utilizado em todos eles seja o mesmo, para que não haja grandes variações na linguagem utilizada. O requisito para esses córpus é que haja separadores entre as sentenças.

Ao utilizar o programa, as próprias classificações morfossintáticas podem ser o objetivo final ou elas podem ser um resultado intermediário para outras análises. Por exemplo, é possível utilizar tais classificações para ajudar na detecção de entidades mencionadas ou relações de causa e efeito dentro dos textos.

## 5. Preprocessamento

O córpus CETENFolha-1.0_jan2014 possui 1.215.767.302 caracteres, distribuídos em um total de 35.236.586 linhas. Primeiramente, foi necessário fazer um pré-processamento do córpus, pois ele estava em um formato parecido a XML, porém a sintaxe não estava correta. Por exemplo, ao iniciar uma sentença o correto seria abrir uma tag <s> e fechá-la com </s> quando a sentença acabasse, porém isso nem sempre ocorria. Também foi necessário, para cada palavra, filtrar as etiquetas morfossintáticas base e eliminar as anotações sintáticas da Gramática de Restrições, bem como informações sobre gênero, número e conjugação das palavras. Para isso, foram utilizadas expressões regulares (regex) através da biblioteca re.

As etiquetas morfossintáticas que apareceram muito raramente no corpus foram eliminadas, juntamente com as sentenças que as acompanhavam. Por exemplo, a etiqueta PRP_DET aparecia somente uma vez no corpus inteiro, no caso para a palavra "pra"(para+a). Isso prejudica o algoritmo de Viterbi, pois a probabilidade de uma palavra nunca vista ser classificada como PRP_DET aumenta muito, vistoar que existe somente uma outra palavra emitida por essa etiqueta e é realizada uma suavização que atribui certa probabilidade a todas as palavras não vistas.

A pontuação não estava classificada no corpus, por isso foi utilizado o seguinte critério: ! ? . : foram classificados como "."e os outros símbolos de pontuação foram classificados como ",". O intuito foi separar os sinais de pontuação que normalmente terminam sentenças dos que normalmente aparecem no meio de sentenças. Todas as letras foram convertidas para minúscula, pois, em geral, a etiquetagem morfossintática não deve ser afetada pelo uso de maiúsculas e minúsculas.

Ao terminar o preprocessamento, foi gerada uma lista onde cada elemento é uma lista de duplas. Cada dupla contém uma palavra do córpus e a sua etiqueta morfossintática. As listas de duplas são nada mais do que as sentenças e a lista de listas de tuplas são todas as senteças.

Para obter um resultado preliminar para a atiquetagem morfossintática, foi encontrada a etiqueta mais comum do corpus inteiro para cada palavra. Foi então analisada a porcentagem de vezes no corpus inteiro em que essa etiqueta mais comum de cada palavra era a etiqueta correta e o resultado obtido foi de aproximadamente 95,54%. Vale lembrar que o corpus não foi dividido em partes de treino e teste para este experimento, portanto a porcentagem obtida será mais baixa caso a mesma etiquetagem seja aplicadas a um trecho novo de texto.

## 6. Implementação do algoritmo de Viterbi

Para a o treinamento da HMM e teste do algoritmo de Viterbi, 80% das sentenças do córpus foram utilizadas para treino e os outros 20% foram reservados para teste, de forma aleatória. A HMM treinada é de órdem 2. Foi utilizada a biblioteca numpy para manipular matrizes de números. No pré-processamento, foram identificados 18 tipos de etiquetas possíveis e por isso foi criada uma matriz A, correspondente ao parametro A da HMM, com $18^2 = 324$ linhas e 18 colunas. Cada linha corresponde a uma dupla de etiquetas e cada coluna corresponde a uma etiqueta. As entradas da matriz A representam o número de vezes que duas etiquetas das linhas passaram a uma etiqueta das colunas na palavra seguinte. Em seguida, foi adicionado 0,1 a todas as entradas da matriz e as entradas foram divididas pela soma dos valores da linha correspondente. Dessa forma, os números passaram a representar probabilidades e foi aplicada a suavização de Laplace para que todas as transições sejam possíveis.

Também foi criado um dicionário B, correspondente ao parâmetro B da HMM. O dicionário B possui 18 chaves, uma para cada etiqueta possível. O valor de cada chave é outro dicionário, onde cada chave deste são as palavras que assumiram aquela etiqueta alguma vez no córpus e os valores destas são o número de vezes que a palavra assumiu aquela etiqueta no córpus. Para cada um destes últimos dicionários, foi adicionada uma palavra "#####"com 0,1 ocorrências e as ocorrências de todas as outras palavras foi aumentada em 0,1. Em seguida, esses valores foram divididos pelo número total de ocorrências da etiqueta, para que os números representem probabilidades. Todas as palavras que nunca apareceram com determinada etiqueta são mapeadas para a ocorrência de "#####". O artifício da palavra "#####"foi utilizado como uma suavização similar à de Laplace.

Posteriormente, foi aplicado o logaritmo natural a todos os valores de A e B. Isso foi feito para evitar *underflow* ao múltiplicar números muito pequenos ao calcular probabilidades de vários eventos simultâneos. Vale mencionar que é possível fazer essa modificação, pois a função log é estritamente crescente, mantendo a relação de ordem total entre os números, e porque foi aplicada a suavização, então o log não está sendo aplicado a nenhuma entrada com valor 0. Esse ajuste implica que os valores devem ser somados, em vez de multiplicados, ao aplicar o algoritmo de Viterbi, porque o produto dos números originais corresponde à soma dos logs.

Tendo os parâmetros A e B da HMM, para cada sentença, foram criados mais dois dicionários: viterbi e viterbi_old. As chaves do dicionário viterbi são sempre duplas de etiquetas e o valor retornado é o caminho mais provável que termina com aquelas duas etiquetas, naquele momento. Cada caminho é uma lista de duplas, onde cada dupla corresponde a uma palavra da sentença, o primeiro elemento da dupla é a etiqueta da palavra e o segundo é a probabilidade do caminho até aquela etiqueta. Dessa forma, o segundo elemento da última dupla representa a probabilidade total do caminho. viterbi_old assume os valores de viterbi e viterbi é resetado a cada palavra que passa da sentença. Dessa forma, para cada palavra da sentença, para cada combinação possível de duplas de etiquetas e para cada possível etiqueta nova, são múltiplicados os valores correspondentes de A, de B e de viterbi_old e armazenados em viterbi. Ao final de todas as palavras da sentenca, basta escolher o caminho de viterbi que tem a maior probabilidade na última dupla.

## 7. Resultados e tempo de processamento

O notebook onde foi feitas a implementação e todos os testes possui processador Intel(R) Core (TM) i5-5300U CPU @ 2.30 GHz 2.29GHz, memória RAM de 8,00 Gb e sistema operativo Windows (64-bit) e o código foi desenvolvido na linguagem Python.

O tempo de preprocessamento foi de aproximadamente 11 minutos. Em certo momento do pré-processamento, a utilização de memória RAM chegou a 5 Gb.

A aplicação do algoritmo de Viterbi demorou, em média 2h para 10.000 sentenças, o que equivale a aproximadamente 0,72 segundos por sentença, quando o notebook estava dedicado à tarefa. Quando o notebook estava sendo utilizado para outros fins (como navegar na internet), o tempo aumentou para 0,81 segundos por sentença

Foram utilizadas 1.276.098 sentenças para o treinamento da HMM. Os testes foram feitos em 6 conjuntos isolados de 10.000 sentenças. Os resultados do algoritmo de Viterbi foram comparados com a porcentagem obtida ao simplesmente aplicar nas amostras de teste a etiqueta mais observada para cada palavra na amostra de treinamento. Seguem os resultados obtidos:

| **Acurácia (%)** | |
|---|---|
| **Etiquetagem mais observada** | **Algoritmo de Viterbi** |
| 94,51888574944963 | 96,93991344392634 |
| 94,47643691770972 | 96,90869919309656 |
| 94,47269613713545 | 96,83368913110135 |
| 94,57838261029751 | 96,97833195173621 |
| 94,5728491086711 | 96,9702967991337 |
| 94,57353635097653 | 96,88855896603713 |
| **Total** | **Total** |
| 94,53214614202956 | 96,91988384915801 |

Observamos que a simples etiquetagem com as etiquetas mais comuns do treinamento já traz um bom resultado, em torno de 94,5% de acurácia. No entanto, aplicando o algoritmo de Viterbi, a acurária sobe para aproximadamente 96,9%, tendo uma melhoria de 2,4% sobre o outro método mais simples.

Portanto, é possível concluir que o algoritmo de Viterbi é bastante eficaz, pois aumentar a acurácia em qualquer porcentagem quando ela já está mais alta do que 90% realmente representa muito. Invertendo um pouco a perspectiva, podemos dizer que houve uma redução de aproximadamente 44% na quantidade de erros.

## 8. Considerações finais

Ao criar este programa, houveram várias dificuldades que não são necessariamente aparentes ao iniciar um projeto relacionado ao processamento de texto.

O pré-processamento do texto é essencial para garantir a qualidade dos dados e suportar o viés que seja desejado dar ao algoritmo, seja ao eliminar sentenças problematicas ou ao transformar todas as letras em minúsculas. Posteriormente, existe a dificuldade da implementação do algoritmo de forma eficiente. Isso é especialmente importante quando se tem quantidades muito grandes de dados como era o caso. Rodar os testes também exige atenção de acompanhar o programa periodicamente e comparar os resultados obtidos.

Neste projeto, acredito que o melhor ponto a ser melhorado seria implementar o algoritmo de Viterbi de forma mais eficiente, bem como fazer um k-fold validation para diminuir a possibilidade de algum viés nos dados de treinamento e de teste.

## 9. Bibliografia

[1] Marcelo Finger adapted from Raymond J. Moooney University of Texas at Austin - Natural Language Processing: Part-Of-Speech Tagging, Sequence Labeling, and Hidden Markov Models (HMMs)

Instituto de Matemática e Estatística - Universidade de São Paulo, São Paulo - SP, Brasil
*Email address*: diegoam00@gmail.com