Modelos Ocultos de Markov

Felipe Alves Ferreira Ligabo Lucas Ferreira de Almeida
Sabrina Eloise Nawcki

18/11/2019

Conteúdo

1 Introdução

Toda massa de dados pode ser transformada em estatística através de fórmulas matemáticas e agrupamento de informações. Em inteligência artifical a extração de dados de dados é de extrema importância como Thomas H. Davenport, especialista em analytics diz no excerto do The Wall Street Journal: "Seres humanos podem, normalmente, criar um ou dois modelos bons por semana; machine learning pode criar milhares de modelos por semana.".

De forma geral, o modelo de Markov modela os dados e é muito conhecido no uso de aplicações de aprendizagem por padrões e no reconhecimento de padrões temporais, como fala, escrita à mão, reconhecimento de gestos, acompanhamento musical, descargas parciais e bioinformática.

O objetivo do presente trabalho é desenvolver um trabalho acadêmico de pesquisa com foco em modelos ocultos de Markov, tendo as informações sobre o

que é o modelo, quais bibliotecas ajudam no desenvolvimento do modelo na linguagem computacional Python, como essas bibliotecas funcionam, a implementação de uso da biblioteca com um exemplo de problema e quais os resultados da implementação.

O trabalho é dirigido às pessoas que se interessam em se aprofundar em modelos probabilísticos, especificamente no modelo oculto de Markov.

Para o desenvolvimento do presente trabalho foram utilizadas pesquisas em publicações científicas e em publicações voltadas à programação em Python utilizando o modelo oculto de Makrov. A estrutura do trabalho é dividida em três partes, sendo a primeira parte uma explicação do que é o modelo oculto de Makrov, a segunda parte sendo a implementação em Python e a terceira parte sobre os resultados obtidos através do código implementado.

2 Fundamentação Teórica

Um modelo de Markov é um modelo estocástico que modela dados temporais ou sequenciais, ou seja, dados que são ordenados. Fornecendo uma maneira de modelar as dependências da informação atual com informações anteriores, composto por estados, esquema de transição entre estados,e emissão de saídas (discretas ou contínuas). Com esta implementação podem ser alcançados objetivos como aprender estatísticas de dados sequenciais, fazer previsões ou estimativas e reconhecer padrões. Dele surge o Modelo Oculto de Markov (HMMs), onde os estados do modelo estão ocultos e cada um deles pode emitir uma saída que é observada.

Segundo Jacob Schreiber o Modelo Oculto de Markov é um modelo probabilístico estruturado que forma uma distribuição de probabilidade de sequências, em oposição a símbolos individuais. Possuindo uma estrutura gráfica direcionada em que os nós representam distribuições de probabilidade , enquanto suas arestas representam transições e codificam probabilidades de transição.

Um Modelo Oculto de Markov pode ser pensado como um modelo geral de mistura com mais uma matriz de transição, em que cada componente no modelo geral de Mistura corresponde a um nó e a matriz de transição informa a probabilidade de que símbolos adjacentes na transição de sequência sejam gerado de um componente para outro. Um ponto forte é que eles podem modelar sequências de tamamho de variável, enquanto outros modelos geralmente exigem um conjunto de recursos fixos. Sendo amplamente utilizados nas áreas de processamento de linguagem natural para modelar fala, bioinformática para modelar biosequências e robótica para modelar movimento.

3 Implementação

Para exemplificação do modelo oculto de Markov foi implementado um programa em *Python* utilizando duas bibliotecas distintas, a *pomegranate* e a *hmmlearn*, para o mesmo problema.

3.1 Problema

O problema abordado nessa implementação tem o objetivo de prever qual será o clima de acordo com a ação que uma pessoa faz, usando como massa dados as previsões dos dias anteriores e as respectivas ações feitas nos dias.

O modelo possui três estados: Sol, Chuva e Nuvens. Para o problema é necessário saber qual a probabilidade do dia iniciar com cada clima citado e o restante do modelo de transição e a distribuição discreta para as ações. Esses dados estão respectivamente nas tabelas 1, 2 e 3.

Chuva	Sol	Nuvens
0.5	0.2	0.3

Tabela 1: Tabela de inicialização.

Matriz	Chuva	Sol	Nuvens
Chuva	0.5	0.3	0.2
Sol	0.35	0.45	0.2
Nuvens	0.2	0.3	0.5

Tabela 2: Tabela de transições.

Matriz	Caminhar	Comprar	Limpar
Chuva	0.1	0.4	0.5
Sol	0.6	0.3	0.1
Nuvens	0.2	0.1	0.7

Tabela 3: Tabela de distribuição.

3.2 Código

3.2.1 Biblioteca *Pomegranate*

O uso da biblioteca *Pomegranate* é intuitiva e de simples discernimento, pois suas funções são bem nomeadas e a biblioteca recalcula automaticamente possíveis erros do modelo de transição. Abaixo há um tutorial passo à passo de codificação utilizando a biblioteca.

1. Importar a biblioteca

   ```
   from pomegranate import *
   ```

2. Criar o modelo oculto de Markov

   ```
   modelo2 = HiddenMarkovModel( name="previsao-tempo" )
   ```

3. Criar os estados com os valores da tabela de distribuição 3

```
chuva = State(
    DiscreteDistribution(
    { 'caminhar': 0.1, 'comprar': 0.4, 'limpar': 0.5 }
    ),
    name='Chuva'
)
sol = State(
    DiscreteDistribution(
        { 'caminhar': 0.6, 'comprar': 0.3, 'limpar': 0.1 }
    ),
    name='Sol'
)
nuvens = State(
    DiscreteDistribution(
        { 'caminhar': 0.2, 'comprar': 0.1, 'limpar': 0.7 }
    ),
    name='Nuvens'
)
```

4. Criar o modelo de transição inicial da tabela de inicialização 1

```
modelo2.add_transition(
    modelo2.start, chuva, 0.5
)
modelo2.add_transition(
    modelo2.start, sol, 0.2
)
modelo2.add_transition(
    modelo2.start, nuvens, 0.3
)
```

5. Criar o restante do modelo de transição da tabela de transições 2. Note que a biblioteca necessita que seja subtraído 0.5 dos valores da tabela e que se adione o modelo de transição final

```
modelo2.add_transition(
    chuva, chuva, 0.45
)
modelo2.add_transition(
    chuva, sol, 0.25
)
modelo2.add_transition(
    chuva, nuvens, 0.15
)
```

```
modelo2.add_transition(
    sol, chuva, 0.3
)
modelo2.add_transition(
    sol, sol, 0.4
)
modelo2.add_transition(
    sol, nuvens, 0.15
)
modelo2.add_transition(
    nuvens, chuva, 0.15
)
modelo2.add_transition(
    nuvens, sol, 0.25
)
modelo2.add_transition(
    nuvens, nuvens, 0.45
)

modelo2.add_transition(
    chuva, modelo2.end, 0.15
)
modelo2.add_transition(
    sol, modelo2.end, 0.15
)
modelo2.add_transition(
    nuvens, modelo2.end, 0.15
)
```

6. Chamar a função bake

```
modelo2.bake( verbose=True )
```

7. Criar a sequência a ser observada

```
sequencia = [
    'comprar', 'limpar', 'caminhar',
    'limpar', 'limpar', 'caminhar', 'limpar'
]
```

3.2.2 Resultados

Depois de criar o modelo, a biblioteca possui funções para calcular as probabilidades resultantes dos dados obtidos. Abaixo podemos visualizar como utilizar essas funções e quais os resultados obtidos a partir do uso de cada uma.

1. Criar funções resultantes

```
print (
    math.e**modelo2.forward(
        sequencia
    )[
        len(sequencia), modelo2.end_index
    ])
#A probabilidade de observar a sequencia eh de
#3.71232063676406e-05

print (
    math.e**modelo2.forward_backward(
    sequencia
    )[1]
    [ 2, modelo2.states.index( chuva ) ])
#A probabilidade de limpar como terceira acao da sequencia eh de
#0.1418166223832125

print (
    math.e**modelo2.backward(
        sequencia
        )[ 3, modelo2.states.index( sol ) ])
#A probabilidade da sequencia ocorrer sendo sol o quarto estado eh de
#0.0013228796250000004

print (
    " ".join(
        state.name for i, state in modelo2.maximum_a_posteriori(
        sequencia
        )[1] ))
#Por ultimo o resultado a probabilidade sera
#"Chuva Chuva Sol Nuvens Nuvens Sol Nuvens"
```

3.2.3 Biblioteca *Hmmlearn*

Assim como a biblioteca *Pomegranate* o uso da biblioteca *Hmmlearn* é intuitiva, pois suas funções são bem nomeadas, porém, a biblioteca não consegue recalcular os possíveis erros do modelo de transição e os estados são um vetor, sendo mais difícil a visualização dos resultados. Abaixo há um tutorial passo à passo de codificação utilizando a biblioteca.

1. Importar a biblioteca

   ```
   from hmmlearn import hmm
   ```

2. Criar o modelo oculto de Markov

   ```
   modelo1 = hmm.MultinomialHMM(
   ```

```
    n_components = 3
) #estados -> chuva, sol, nuvens
```

3. Criar o modelo de transição inicial da tabela de inicialização 1

```
modelo1.startprob_ = numpy.array(
    [0.5, 0.2, 0.3]
) #probabilidades de chuva, sol, nuvens
```

4. Criar o restante do modelo de transição da tabela de transições 2

```
modelo1.transmat_ = numpy.array([
    [0.5, 0.3, 0.2], #[chuva -> chuva, sol, nuvens]
    [0.35, 0.45, 0.2], #[sol -> chuva, sol, nuvens]
    [0.2, 0.3, 0.5] #[nuvens -> chuva, sol, nuvens]
])
```

5. Criar os estados com os valores da tabela de distribuição 3

```
modelo1.emissionprob_ = numpy.array([
    [0.1, 0.4, 0.5], #probabilidade de chuva em caminhar, comprar e limpar
    [0.6, 0.3, 0.1], #probabilidade de sol em caminhar, comprar e limpar
    [0.2, 0.1, 0.7] #probabilidade de nuvens em caminhar, comprar e limpar
])
```

6. Criar a sequência a ser observada

```
logprob, seq = modelo1.decode(
    numpy.array([[1,2,0]]
    ).transpose()) #{‘‘comprar’’, ‘‘limpar’’, ‘‘caminhar’’}
```

3.2.4 Resultados

Novamente, assim como a biblioteca analisada anteriormente, depois de criar o modelo, a biblioteca *Hmmlearn* possui funções para calcular as probabilidades resultantes dos dados obtidos. Abaixo podemos visualizar como utilizar essas funções e quais os resultados obtidos a partir do uso de cada uma.

1. Criar as funções resultado

```
print(
    math.exp(
        modelo1.score(
            numpy.array([[0]]))
        )
```

```
    )
#Probabilidade da primeira observacao sendo ‘‘caminhar’’
#0.2 x 0.1 + 0.5 x 0.6  + 0.3 x 0.5 = 0.23 (23%)

print(
    math.exp(
        modelo1.score(
            numpy.array([[1]]))
        )
    )
# 0.29000000000000004

print(
    print(
        math.exp(
             modelo1.score(
                numpy.array([[2]]))
            )
        )
# 0.48
print(
    math.exp(
        modelo1.score(
            numpy.array([[2,2,2]]))
        )
    )
# 0.095256

print(math.exp(logprob))
# dada a sequencia
#{‘‘limpar’’, ‘‘limpar’’, ‘‘limpar’’}
# 0.025724999999999994

print(seq)
#O resultado sera
#{‘‘nuvens’’, ‘‘nuvens’’, ‘‘nuvens’’} [2 2 2]
```

4 Conclusão

Tendo em vista o exposto, podemos afirmar que o modelo oculto de Markov é de excelente uso para o estudo de probabilidades.

Além disso, não se pode ignorar o fato que as bibliotecas criadas para seu uso facilitam na utilização do modelo na programação, tendo os cálculos de fácil implementação.

Pode-se dizer que estas informações podem beneficiar qualquer um, que tenha o interesse de usar o modelo oculto de Markov em *Python*.

A maior dificuldade apresentada na apresentação destes fatos foi encontrar as bibliotecas e progrmá-las com o mesmo problema, sendo que elas possuem diferentes implementações.

5 Referências

pomegranate.readthedocs.io/en/latest/HiddenMarkovModel.html

cecas.clemson.edu/~ahoover/ece854/refs/Ramos-Intro-HMM.pdf

medium.com/@kangeugine/hidden-markov-model-7681c22f5b9

github.com/jmschrei/pomegranate/blob/master/examples/hmm_rainy_sunny.ipynb

en.wikipedia.org/wiki/Hidden_Markov_model#A_concrete_example%3Earticle

hmmlearn.readthedocs.io/en/latest/api.html#hmmlearn-hmm