simulacao_cadeias_com_memoria_alcance_variavel

November 6, 2017

Aluno: Cloves Adriano Paiva Sousa
nº USP: 9292218
Prof: Antônio Galves
Curso: Aprendizagem Estatística

1 Cadeias estocásticas com memória de alcance variável

Constituem uma família de cadeias estocásticas de ordem finita em um alfabeto finito. A ideia é que para cada passado, apenas um sufixo finito, chamado contexto, é suficiente para predizer o próximo símbulo.

1) Formule uma questão que lhe pareca interessante, pertinente e factível em aproximadamente 10 minutos utilizando noções e resultados apresentados no curso. As melhores questoes cairam na terceira prova do curso.

Resposta:
Dada a amostra (0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0), diga qual é a árvore de contextos t de altura menor ou igual a 3, obtida aplicando o Algoritmo Contexto, utilizando δ = 0.05 no critério de poda.

2) Usando símbolos do alfabeto A = {0, 1}, escolha duas árvores de contexto de altura menor ou igual a 3 e atribua probabilidades de transição aos contextos dessas duas árvores.

Resposta:
t1 = {0, 01, 11}, com probabilidades de transição p(0|0)=0.2, p(0|10)=0.4 e p(0|11)=0.6;
t2 = {00, 10, 1}, com probabilidades de transição p(0|00)= 1/3, p(0|01)=1/3 e p(0|1)=3/8.

a) Descreva um algoritmo de simulação de cadeias com memoria de alcance variável dada uma árvores e uma familia de probabilidades de transição associada.

```
In [1]: '''
        1º Defina uma árvore de contextos no alfabeto A={0, 1};
        2º Defina as probabilidade de transição dessa árvore;
        3º obtenha uma amostra [x1, x2, ..., xn] de uma distribuiçao uniforme
           tomando valores no intervalo [0, 1];
        4º inicialise a cadeia com algum(ns) valor(es) do alfabeto A;
        5º para cada xi da amostra, i=0,...,n:
                yi = 1 se xi > p(0|contexto(cadeia))
```

```
            yi = 0 caso contrário
            yi é adiconada à cadeia.
    ''';
```

b) Escreva o código na sua linguagem de programação preferida implementando esse algoritmo.

```
In [2]: import scipy.stats as stats

In [3]: def getContext(lista, arvore):
            '''
            recebe uma lista de caracteres do alfabeto A={0, 1}
            e uma lista de contextos 'arvore' e retorna o contexto
            necessário para prever próximo caracter
            '''
            cadeia = ''.join(lista)
            context = ''
            n = len(cadeia)
            for i in range(n):
                context = cadeia[n-1] + context
                if context in arvore:
                    return context
                n -= 1

            return None

        class Cadeia():
            '''
            Esta classe representa uma cadeia com memoria alcance variavel
            que recebe os seguintes atributos:
                t = uma lista dos contextos da árvore
                prob = um dicionário com todos os contextos como chave
                       e as suas respectivas probabilidades de transição
                       para o caracter zero.
            '''

            def __init__(self, t, prob):
                self.arvore = t
                self.prob_trans = prob
                self.cadeia = 'ainda nao foi gerada'

            def gerarCadeia(self, inicio_cadeia, tamanho_cadeia):
                '''
                Este método recebe:
                  inicio_cadeia = uma string com o início da cadeia
                  tamanho_cadeia = um inteiro informando o tamanho desejado
                                   da cadeia
                Este método retorna:
```

```
            cadeia com memoria alcance variavel com contextos
            self.arvore e probabilidades de transição self.prob_trans
        '''
        amostra_uniforme = stats.uniform.rvs(
            size=tamanho_cadeia-len(inicio_cadeia))
        cadeia = []
        cadeia.append(inicio_cadeia)

        for item in amostra_uniforme:
            context = getContext(cadeia, self.arvore)
            if context == None:
                pass
            else:
                if item > self.prob_trans[context]:
                    y = '1'
                else:
                    y = '0'
                cadeia.append(y)

        self.cadeia = ''.join(cadeia)
        return self.cadeia
```

```
In [4]: arvore = ['0', '01', '11']

In [5]: prob = {'0': 0.2, '01': 0.4, '11': 0.6}

In [6]: cadeia = Cadeia(arvore, prob)

In [7]: cadeia.cadeia

Out[7]: 'ainda nao foi gerada'

In [8]: cadeia.gerarCadeia(inicio_cadeia = '01', tamanho_cadeia = 1000)

Out[8]: '0110110110011101011011110001111111011011001101011011011011001101101110110
```

c) Gere 5 amostras de comprimento 1000 para cada uma dessas cadeias usando o algoritmo apresentado no item b).

```
In [9]: cadeia1 = Cadeia(['0', '01', '11'], {'0': 0.2, '01': 0.4, '11': 0.6})
        cadeia2 = Cadeia(['00', '10', '1'], {'00': 0.33, '10': 0.33, '1': 0.375})

In [10]: amostra_cadeia1 = []
         amostra_cadeia2 = []

In [11]: for i in range(5):
             amostra_cadeia1.append(cadeia1.gerarCadeia('01', 1000))
             amostra_cadeia2.append(cadeia2.gerarCadeia('01', 1000))
```

d) Diga que estatísticas podem ser usada para verificar a correção do algoritmo de simulação e de seus códigos.

Resposta:

Uma estatística seria obter a frequência relativa amostral de cada contexto para o caracter '0' e obter a média dessas frequências relativas de cada contexto e comparar com suas respectivas probabilidades de transições.

Outra estatística possível seria o 'Erro quadrático Médio' das probabilidades(usando-se a maximoverossimilhança) de transição baseados nas amostras.

e) Aplique essas estadísticas nas amostras geradas e comente os resultados.

```
In [12]: def freqRelativa(cadeia, contexto):
             n = float(cadeia.count(contexto))
             evento = cadeia.count(contexto + '0')
             return evento/n

In [13]: def media_freq_relativa(cadeias, prob):
             contextos = prob.keys()
             dic = {}
             for contexto in contextos:
                 for cadeia in amostra_cadeia1:
                     l = [freqRelativa(cadeia, contexto)]
                 dic[contexto] = sum(l)/float(len(l))
             return dic

In [14]: print cadeia1.prob_trans
         media_freq_relativa(amostra_cadeia1, cadeia1.prob_trans)

{'11': 0.6, '0': 0.2, '01': 0.4}


Out[14]: {'0': 0.16279069767441862, '01': 0.3081967213114754, '11': 0.8333333333333

In [15]: print cadeia2.prob_trans
         media_freq_relativa(amostra_cadeia2, cadeia2.prob_trans)

{'1': 0.375, '10': 0.33, '00': 0.33}


Out[15]: {'00': 0.25396825396825395,
          '1': 0.49755301794453505,
          '10': 0.19672131147540983}
```