Aula 22: 06/NOV/2018

Aula passada

Programação dinâmica: *longest commom subsequence* (LCS)

Modelos computacionais e simulação em pesquisa científica: paradoxo do aniversário

Hoje

Mais modelos computacionais e simulação em pesquisa científica.

Possíveis assuntos

- colecionador de figurinhas
- máquina de votação

No diretório `py` os subdiretórios têm um arquivo `main.py` que o cliente da classe principal:

- `Colecionador` (`colecionador.py`) para o colecionador de figurinhas
- `Votacao` (`votacao.py`) para a máquina de votação

Todos os diretórios tem um arquivo `config.py` com constantes para os programas. As constantes poderiam ser parâmetros.

Método científico:

Está presente em tudo que faremos.

- **observar** algum aspecto da natureza
- **hipotetizar** um modelo consistente com as observações
- **prever** eventos usando as hipóteses
- **verificar** as predições fazendo mais observações
- **validar** repetindo até que hipoteses e obeservações estajam de acordo

Colecionador de figurinhas

Coupon collector's problem

Suponha que temos álbum de `n` figurinhas que são numeradas de 0 a `n-1`.

Quantas figurinhas temos que comprar para completar o álbum?

Modelagem

Suporemos que cada figurinha é igualmente provável de ser comprada.

Código

```
import random

class Colecionador:
    #----------------------------------------
    def __init__(self, n, t):
        self.n = n
        self.t = t
        no_figurinhas = 0
        for i in range(t):
            no_figurinhas += self.experimento()
        self.n_medio = no_figurinhas/t

    #----------------------------------------
    def mean(self):
        return self.n_medio

    #----------------------------------------
    def experimento(self):
        n = self.n
        album = set()
        no_compradas = 0
        while len(album) != n:
            fig = random.randrange(n)
            no_compradas += 1
            album |= {fig}
        return no_compradas
```

Cliente

```
from colecionador import Colecionador

#-----------------------------------------------------
def main(argv=None):
    # número de figurinhas
    n = int(argv[1])
```

```
# número de experimentos
t = int(argv[2])

# realize os no_exp experimentos
exp = Colecionador(n, t)

# média aritmética
print("número de figurinhas compradas =", exp.mean(), "(%g)"%(n*H(n)))
```

Cálculo do valor esperado

Seja T o número de figurinhas a serem compradas para completarmos o álbum. Seja t_i o número de figurinhas que compramos entre obtermos a $(i-1)$-ésima e a i-ésimo figurinha distintas. T e t_i são variáveis aleatórias e $T = t_1 + t_2 + \cdots + t_n$. A probabilidade p_i de conseguirmos uma figurinha distinta, dado que já conseguimos $i-1$ figurinhas distintas é $(n-(i-1))/n$. Seja $q_i = 1 - p_i = 1/(i-1)$. Assim, o valor esperado de t_i é dado por

$$\begin{aligned} E(t_i) &= 1p_i + 2p_iq_i + 3p_iq_i^2 + \cdots \\ &= p_i(1 + 2q_i + 3q_i^2 + 4q_3^3 + \cdots \\ &= p_i 1/(1-q_i)^2 = 1/p_i. \end{aligned}$$

Desta forma, temos que

$$\begin{aligned} E(T) &= E(t_1) + E(t_2) + \cdots + E(t_n) \\ &= n/n + n/(n-1) + n/(n-2) + \cdots n/1 = n \times H(n) < n\ln(n+1). \end{aligned}$$

Portanto, temos que comprar aproximadamente $n \ln n$ figurinhas para completarmos um álbum de n figurinhas (supondo que a probabilidade ...).

Máquinas de votação

Máquina de votação. Suponha que em uma população de 100 milhões de eleitores, 51% votam para o candidato A e 49% votam para o candidato B. Entretanto, a máquina de votação é propensa a cometer erros e 5% das vezes ela contabiliza um voto de maneira errada.

Supondo que os erros ocorrem uniformemente ao acaso, 5% é uma taxa de erro suficiente para invalidar a eleição? Qual a taxa de erro que pode ser tolerada?

Modelagem

Suponha que `no_votos` é o número total de votos, `por_A` é a porcentagem dos eleitores que votaram no candidadto `A` (valor entre 0 e 100). Portanto a porcentagem do eleitores de `B` é `por_B = 100 - por_A`. Suponha ainda que `por_erro` é a porcentagem de votos errados.

No arquivo `config.py` temos as definicões

```
NO_VOTOS  = 10**8 # 100 milhões
POR_A     = 51    # porcentagem de eleitores de A
POR_B     = 49    # porcentagem de eleitores de B
POR_ERRO  = 5     # porcentagem de erro
T         = 100   # número de testes
```

Podemos calcular o número esperado `n_erros` de votos errados e sortear esses `n_erros` votos. Se o voto pertence ao candidato `A` o voto vai para o candidado `B` e se o voto pertence ao candidato `B` ele vai para o candidado `A`.

Para selecionar o voto a ser alterado basta selecionarmos um inteiro `voto` aleatoriamente em 'range(no_votos). `Após selecionarmos devemos fazer`no_votos -= 1‘.

Se `votos_A` é o número de votos no candito `A`, então consideramo que o `voto` sorteado é do candidato `A` se `voto in range(votos_A)` e do candidato `B` em caso contrário.

Código

```
class Votacao:
    def __init__(self, no_votos, por_A, por_erro, t):
        '''(Votacao, int, float, float, int) ->

        Recebe

             - no_votos: número de votos
             - por_A: porcentagem de votos para o candidato A
             - por_erro: porcentagem de votos errados
             - t: número de experimentos
        determina a probabilidade do resultado da eleição ser alterado
        '''
        self.no_votos = no_votos
        self.por_A = por_A
        self.por_erro = por_erro
```

```
    # realize os no_t experimentos
    alterou = 0
    for i in range(t):
       alterou += experimento(p_erro)
    self.p = t/alterou

#-------------------------------------------------------------
def mean(self):
    return self.p

#-------------------------------------------------------------
def experimento(self):
    '''(int) -> bool

    Recebe a porcentagem de votos que são contados de forma errada
    e retorna True se o resultado da eleição foi alterado e False
    em caso contrário.

    Supõe que a porcentagem PORCEN_A de eleitores de A é
    maior que porcentagem PORCEN_B de eleitores de B.
    '''
    no_votos   = self.no_votos

    # calcule número de votos errados
    n_erros = (no_votos * self.por_erro) // 100

    # calcule número de votos em A e em B
    votos_A  = (no_votos * self.por_A) // 100

    # print(votos, n_erros, limiar)
    # número líquido de votos de B para A
    muda = 0

    # supor que os votos de A estão em range(votos_A)
    # supor que os votos de B estão em range(voto_A, no_votos)
    for i in range(n_erros):
        # escolhe um voto para alterarmos o resultado
        voto = random.randrange(no_votos)
        if voto < votos_A:
            # transfere um voto de A para B
            muda -= 1
            # diminui os votos de A que podem ser transferidos
            votos_A -= 1
        else:
            # transfere um voto de B para A
            muda += 1
        no_votos -= 1

    votos_A = (self.no_votos * por_A) // 100
```

```
        votos_B =  votos - votos_A

        return  votos_A + muda <= votos_B - muda
```

Cliente

```
def main():
    no_votos  = int(...)
    por_A     = float(...)
    por_erro = float(...)
    votacao = Votacao(no_votos, por_A, por_erro, t)
    print("prob mudar resultado = %g"votacao.mean()
```

Plot e plot adaptativo

A seguir há duas versões de "plotagem". Uma em que os pontos são plotados em um passo de `GAP` e outro em que a plotagem é adaptativa.

Plot de passo fixo

```
# parâmetros: GAP, ERR, X0, Y0, X1, Y1, T
from config import *

import matplotlib.pyplot as plt

#------------------------------------------------------------
def grafico():
    '''
    Referência:
    https://panda.ime.usp.br/algoritmos/static/algoritmos/10-matplotlib.html
    '''
    plt.title("MAC0122 Paradoxo do Aniversário")
    plt.xlabel("número de pessoas")
    plt.ylabel("probabilidade")
    x = []
    y = []
    x0, y0 = X0, Y0
    for n in range(X0, X1, GAP):
        # realize os no_exp experimentos
        ani = Aniversario(n, T)
        x1 = n
        y1 = ani.mean()
        x.append(x1)
        y.append(y1)
        plt.plot([x0,x1],[y0,y1], 'b-') #mlines.Line2D([x0,y0], [x1,y1])
        x0, y0 = x1, y1
    plt.plot(x, y, 'go') # green bolinha
    plt.show()
```

Plot adaptativo

Decisões de projeto:

- para quanto valores realizarmos os experimentos
- para quais valores realizarmos os experimentos
- quantos experimentos para cada valor

Plotagem adaptatica de uma função $\texttt{f}(\texttt{x})$ em um intervalo $[\texttt{x}_0, \texttt{x}_1]$:

- pare se o intervalo é suficiente pequeno (menor que `GAP`)
- divida o intervalo pela metade e compute $\texttt{f}(\texttt{x}_\texttt{m})$
- pare se $\texttt{f}(\texttt{x}_\texttt{m})$ é próximo de $(\texttt{f}(\texttt{x}_0) + \texttt{f}(\texttt{x}_1))/2$
- recursivamente plot o gráfico no intervalo $[\texttt{x}_0, \texttt{x}_\texttt{m}]$

- plot o ponto $(x_m, f(x_m))$
- recursivamente plot o gráfico no intervalo $[x_m, x_1]$

```
from estimativa import Estimativa

import matplotlib.pyplot as plt

# parâmetros
from config import *

#-------------------------------------------------------------
def adaptative_plot(n):
    '''
    Referência:
    https://panda.ime.usp.br/algoritmos/static/algoritmos/10-matplotlib.html
    '''
    # import matplotlib.lines as mlines
    plt.title("MAC0122 Percolação")
    plt.xlabel("probabilidade de abrir um sítio")
    plt.ylabel("probabilidade de percolar")
    # desenhe o ponto inicial e final
    plt.plot([X0, X1], [Y0, Y1], 'go') # green bolinha
    # desenhe a curva entre eles
    curva(n, X0, Y0, X1, Y1)
    # mostre o gráfico
    plt.show()

#-------------------------------------------------------------
def curva(n, x0, y0, x1, y1):
    xm = (x0 + x1) / 2;
    ym = (y0 + y1) / 2; # float

    # realiza o experimento
    perc = Estimativa(n, xm, T)
    fxm = perc.mean()
    perc = None # evita 'loitering': essencial quando o espaço é importante

    # base da recursão
    if x1 - x0 < GAP or abs(ym - fxm) < ERR:
        # desenha a linha entre [x0,y0] e [x1, y1]
        plt.plot([x0,x1],[y0,y1], 'b-')
        return None

    # desenhe recursivamente a curva entre [x0,y0] e [xm,fm]
    curva(n, x0, y0, xm, fxm);

    # desenhe o ponto [xm,fm]
    plt.plot(xm, fxm, 'go') # green bolinha

    # desenhe recursivamente a curva entre [xm,fxm] e [x1,y1]
```

```
curva(n, xm, fxm, x1, y1);
```

Apêndice

Lei dos grandes números

A **lei dos grandes números** diz que a média aritmética dos resultados da realização de um mesmo experimento repetidas vezes tende a se aproximar do valor esperado à medida que mais tentativas se sucederem. Em outras palavras, quanto mais tentativas são realizadas, mais a probabilidade da média aritmética dos resultados observados irá se aproximar da probabilidade real.

Método de Monte Carlo

O método de Monte Carlo é uma classe de algoritmos computacionais que se baseiam em repetições amostragens aleatórias para se obter um resultado numérido. A ideia central é usar aleatoriedade para resolver problemas que podem ser determinísticos em princípio. O método é frequentemente utilizado em física e matemática quando se é difícil ou impossível de se utilizar outros métodos. O método de Monte Carlo é usado principalmente em três classes de problemas de otimização, integração numérica e geração de sorteios como uma certa distribuição de probabilidade;