Atividade prática 4
Álgebra Linear Numérica

Pedro Henrique Coterli

21 de maio de 2024

Exercício 1

Item a)

Primeiramente, precisamos passar essa equação para uma forma polinomial, de modo que consigamos trabalhar adequadamente com mínimos quadrados. Para isso, utilizemos do logaritmo natural:

$$\begin{aligned} P &= bL^{\alpha}K^{1-\alpha} \\ ln(P) &= ln(bL^{\alpha}K^{1-\alpha}) \\ ln(P) &= ln(b) + ln(L^{\alpha}) + ln(K^{1-\alpha}) \\ ln(P) &= ln(b) + \alpha ln(L) + (1-\alpha)ln(K) \\ ln(P) &= ln(b) + \alpha ln(L) + ln(K) - \alpha ln(K) \\ ln(P) &= ln(K) + ln(b) + \alpha(ln(L) - ln(K)) \\ ln(P) - ln(K) &= ln(b) + \alpha(ln(L/K)) \\ ln(P/K) &= ln(b) + \alpha(ln(L/K)) \end{aligned} \tag{1}$$

Com isso, obtemos uma equação da forma $y = ax + b$, que podemos facilmente resolver por mínimos quadrados.

Agora, vamos criar vetores com os nossos dados.

```
P = [100; 101; 112; 122; 124; 122; 143; 152; 151; 126; 155; 159; 153; 177; 184;␣
 ↪169; 189; 225; 227; 223; 218; 231; 179; 240];
L = [100; 105; 110; 117; 122; 121; 125; 134; 140; 123; 143; 147; 148; 155; 156;␣
 ↪152; 156; 183; 198; 201; 196; 194; 146; 161];
K = [100; 107; 114; 122; 131; 138; 149; 163; 176; 185; 198; 208; 216; 226; 236;␣
 ↪244; 266; 298; 335; 366; 387; 407; 417; 431];
```

Em seguida, montemos a matriz A e o vetor b do nosso método de mínimos quadrados com base na equação encontrada anteriormente. A matriz A será formada por uma coluna de 1s, que será multiplicada pelo $ln(b)$, e pelo vetor $ln(L/K)$, que será multiplicado pelo α, e o vetor b será o vetor $ln(P/K)$.

```
A = [ones(P) log(L ./ K)];
disp("A", A(1:5, :))

b = log(P ./ K);
disp("b", b(1:5))
```

```
 "A"

  1.   0.
  1.  -0.0188685
  1.  -0.0357181
  1.  -0.0418471
  1.  -0.0711763

 "b"

  0.
 -0.0577083
 -0.0176996
  0.
 -0.0549158
```

Por fim, precisamos resolver o sistema $A^T A\overline{x} = A^T b$. Para isso, utilizaremos nossa clássica `Gaussian_Elimination_4`:

```
function [x, C, P]=Gaussian_Elimination_4(A, b)
    C=[A,b];
    [n]=size(C,1);
    // Inicializando a matriz P
    P = eye(n,n)
    for j=1:(n-1)
        // Se o pivô não for o maior valor de sua coluna, troca as linhas
        if max(abs(C(j:n,j))) ~= abs(C(j,j)) then
            // Encontrando a linha com o maior pivô em módulo
            moduled_vector = abs(C(j:n,j))
            max_pivot_index = find(moduled_vector == max(moduled_vector))
            // Trocando essas linhas na C e na P
            C([j j+max_pivot_index-1],:) = C([j+max_pivot_index-1 j],:)
            P([j j+max_pivot_index-1],:) = P([j+max_pivot_index-1 j],:)
        end
        //O pivô está na posição (j,j)
        for i=(j+1):n
            //O elemento C(i,j) é o elemento na posição (i,j) of L na␣
↪decomposição LU de A
            C(i,j)=C(i,j)/C(j,j);
            //Linha i  Linha i - C(i,j)*Linha j
            //Somente os elementos da diagonal ou acima dela são computados
            //(aqueles que compõem a matrix U)
```

```
            C(i,j+1:n+1)=C(i,j+1:n+1)-C(i,j)*C(j,j+1:n+1);
        end
    end
    x=zeros(n,1);
    // Calcula x, sendo Ux=C(1:n,n+1)
    x(n)=C(n,n+1)/C(n,n);
    for i=n-1:-1:1
        x(i)=(C(i,n+1)-C(i,i+1:n)*x(i+1:n))/C(i,i);
    end
    C=C(1:n,1:n);
endfunction
```

Dessa forma, resolvendo o sistema, obtemos o seguinte vetor $\overline{x}$:

```
x_barra = Gaussian_Elimination_4(A'*A, A'*b)
```

```
 x_barra  =
   0.0070440
   0.7446062
```

No entanto, apesar do segundo elemento ser justamente o α que estávamos procurando, o primeiro ainda é o logaritmo natural de b. Então, calculemos o verdadeiro b:

```
alpha = x_barra(2)

e = %e;
b = e^(x_barra(1))
```

```
 alpha  =
   0.7446062
 b  =
   1.0070689
```

Portanto, obtemos assim a nossa estimativa da equação de Cobb-Douglas:

$$P = 1.0070689 \cdot L^{0.7446062} \cdot K^{0.2553938}$$

Item b)

Para calcular nossas previsões para 1910 e 1920, vamos definir uma função para a equação de Cobb-Douglas:

```
// Função de Cobb-Douglas
function [P] = Cobb_Douglas(L, K)
    // Calculando P com os parâmetros encontrados
    P = 1.0070689 * L^(0.7446062) * K^(1 - 0.7446062)
endfunction
```

Desse modo, façamos nossas estimativas utilizando os dados fornecidos na tabela:

```
P_1910 = Cobb_Douglas(147, 208)
P_1920 = Cobb_Douglas(194, 407)
```

```
 P_1910  =
   161.76185
 P_1920  =
   236.07216
```

Para 1910, obtivemos uma previsão de aproximadamente 162, enquanto o dado real é de 159. E, para 1920, obtivemos 236 quando o verdadeiro valor foi de 231. Portanto, pode-se dizer que foram boas estimativas.

Exercício 2

Inicialmente, vamos carregar e organizar nossos dados, separando-os em features e targets de treino e de teste:

```
data_train = csvRead("cancer_train_2024.csv", ";");
data_test = csvRead("cancer_test_2024.csv", ";");

X_train = data_train(:, 1:10);
X_test = data_test(:, 1:10);
y_train = data_train(:, 11);
y_test = data_test(:, 11);
```

Antes de calcularmos nosso modelo, primeiro precisamos adicionar uma coluna de 1s tanto no conjunto de treino quanto no de teste, referente ao bias dos dados.

```
X_train_bias = [ones(y_train) X_train];
disp("X_train_bias", X_train_bias(1:5, :))

X_test_bias = [ones(y_test) X_test];
disp("X_test_bias", X_test_bias(1:5, :))
```

```
  "X_train_bias"

   1.    0.64     0.2643    0.6515    0.4006    0.8182    0.8037    0.7031    0.7311
0.7957    0.8078
   1.    0.7318   0.4524    0.705     0.5306    0.5856    0.2277    0.2036    0.3488
0.5961    0.5816
   1.    0.7005   0.541     0.6897    0.4814    0.7574    0.4629    0.4625    0.6357
0.6806    0.6157
   1.    0.4063   0.5188    0.4116    0.1545    0.9848    0.8219    0.5656    0.5229
0.8543    1.
   1.    0.4429   0.3997    0.438     0.1909    0.8832    0.4922    0.3697    0.402
0.6865    0.7813

  "X_test_bias"
```

```
   1.    0.7123    0.6152    0.6929    0.4866    0.7038    0.6374    0.6044    0.5551
0.6975    0.6843
   1.    0.4544    0.6475    0.4303    0.1884    0.5172    0.3936    0.1879    0.162
0.6933    0.6526
   1.    0.7327    0.7767    0.7207    0.4986    0.661     0.7135    0.6281    0.6691
0.8754    0.7801
   1.    0.3826    0.6058    0.3577    0.1337    0.6536    0.2592    0.0632    0.093
0.5769    0.6893
   1.    0.4179    0.5911    0.3937    0.1612    0.5418    0.2987    0.092     0.0785
0.5492    0.652
```

Agora, podemos utilizar o método dos mínimos quadrados com o conjunto de treino para aprender nosso modelo. Para isso, utilizaremos nossa função `Gaussian_Elimination_4` para resolver o sistema $A^T A\overline{x} = A^T b$, onde A é o nosso `X_train_bias` e b é o nosso `y_train`.

```
x_barra = Gaussian_Elimination_4(X_train_bias'*X_train_bias,␣
 ↪X_train_bias'*y_train)
```

```
 x_barra  =
  -6.2101493
   15.902409
   1.5568757
  -5.0718598
  -7.1846562
   1.2702227
  -0.9298812
   0.5285964
   1.9535131
  -0.0470564
   0.7701829
```

Perfeito, temos nossos parâmetros, ou seja, os coeficientes do nosso hiperplano! Com ele, podemos agora ver como ele se ajustou aos dados de treinamento. Para isso, calculemos primeiro as previsões para esse conjunto:

```
y_train_pred = X_train_bias * x_barra;
y_train_pred(1:5)
```

```
 ans  =
   0.8729158
   0.4847284
   1.2751264
   0.3981943
   0.0766325
```

Agora, confiramos nossas previsões para o conjunto de treinamento. Se multiplicarmos elemento a elemento nossa previsão com os rótulos verdadeiros, valores positivos indicarão previsões corretas (previsão e rótulo positivos ou previsão e rótulo negativos).

```
y_train_pred_conf = y_train_pred .* y_train;
train_accuracy = sum(y_train_pred_conf > 0)/size(y_train)(1)
```

```
 train_accuracy  =
   0.9214286
```

Portanto, nosso modelo obteve uma acurácia de 92% no conjunto de treinamento. Agora, vamos fazer o mesmo processo para o conjunto de teste:

```
y_test_pred = X_test_bias * x_barra;
y_test_pred(1:5)
```

```
 ans  =
   1.2639506
  -0.3352808
   1.7880925
  -0.6495086
  -0.7104726
```

```
y_test_pred_conf = y_test_pred .* y_test;
test_accuracy = sum(y_test_pred_conf > 0)/size(y_test)(1)
```

```
 test_accuracy  =
   0.8892857
```

Tivemos uma acurácia de aproximadamente 89% no conjunto de teste, o que é bem razoável para um modelo tão simples como esse.

Por fim, vamos calcular a matriz de confusão e suas medidas decorrentes para o conjunto de teste. Para isso, primeiro precisamos calcular o número de previsões de cada categoria: verdadeiro positivo (TP), falso positivo (FP), verdadeiro negativo (TN) e falso negativo (FN).

```
TP = sum((y_test_pred >= 0) .* (y_test >= 0));
FP = sum((y_test_pred >= 0) .* (y_test < 0));
TN = sum((y_test_pred < 0) .* (y_test < 0));
FN = sum((y_test_pred < 0) .* (y_test >= 0));

confusion_matrix = [TP FP; FN TN]
```

```
 confusion_matrix  =
   80.   25.
   6.    169.
```

Agora, vamos calcular algumas medidas interessantes resultantes dessa matriz.

```
accuracy = (TP + TN)/(TP + FP + FN + TN)
precision = TP/(TP + FP)
recall = TP/(TP + FN)
fpr = FP/(FP + TN)
fnr = FN/(FN + TP)
```

```
accuracy  =
  0.8892857
precision  =
  0.7619048
recall  =
  0.9302326
fpr  =
  0.1288660
fnr  =
  0.0697674
```

Analisemos essas medidas:

- A **acurácia** é a proporção de registros classificados corretamente, sejam eles positivos ou negativos. Assim, pela nossa medida, nosso modelo acerta a previsão em 89% das vezes.
- A **precisão** é a proporção de registros classificados como positivos que são realmente positivos. Dessa forma, se nosso modelo diz que uma pessoa tem câncer de mama, a chance de ela realmente ter é de 76%.
- A **revocação** ou **recall** é a proporção de registros positivos que são classificados como positivos. Desse modo, a probabilidade de uma pessoa com câncer de mama ser detectada pelo nosso modelo é de 93%.
- A **probabilidade de falso alarme** ou **false positive rate (FPR)** é a chance de um registro negativo ser classificado como positivo. Assim, uma pessoa que não tem câncer de mama tem 13% de chance de receber diagnóstico positivo por esse modelo.
- E a **probabilidade de falsa omissão de alarme** ou **false negative rate (FNR)** é a probabilidade de um registro positivo ser classificado como negativo. Dessa forma, uma pessoa com câncer de mama tem 7% de chance de ser diagnosticada como não tendo pelo nosso modelo.

Uma análise interessante é que há um trade-off entre a precisão e a revocação (recall), ou seja, quanto maior um, menor o outro. No nosso caso, nossa revocação é alta enquanto nossa precisão é baixa, e isso é bom! Nesse contexto, é muito menos prejudicial classificar incorretamente um negativo como positivo do que um positivo como negativo. Uma pessoa que não tem câncer mas que foi classificada como tendo câncer vai apenas fazer outros exames para conferência e eventualmente descobrirá que na verdade não tinha. Por outro lado, uma pessoa que tem câncer e que foi classificada como não tendo câncer possivelmente não fará exames de garantia e terá o estado de sua doença piorado, podendo levar até a uma situação irreversível. Portanto, nosso modelo acabou escolhendo certo qual das duas métricas manter superior.

Exercício 3

Podemos tirar algumas conclusões com base no vetor de parâmetros `x_barra`:

```
x_barra
```

```
x_barra  =
 -6.2101493
  15.902409
  1.5568757
```

```
  -5.0718598
  -7.1846562
   1.2702227
  -0.9298812
   0.5285964
   1.9535131
  -0.0470564
   0.7701829
```

Cada um desses valores refere-se a uma das características do nosso dado (exceto o primeiro, que refere-se ao bias). Assim, podemos observar que algumas variáveis, como a primeira (referente ao 2° elemento do vetor) e a quarta (referente ao 5° elemento do vetor) possuem um peso maior na previsão dos registros. Em contrapartida, variáveis como a sétima e a nona têm pouca influência no resultado. Dessa forma, vamos tentar eliminar essas variáveis uma de cada vez para ver como o desempenho do modelo se comporta em vista de sua simplificação.

Primeiramente, eliminemos a penúltima variável, que possui o menor parâmetro relacionado.

```
X_test_one_less = [X_test_bias(:, 1:9) X_test_bias(:, 11)];
x_barra_one_less = x_barra([1:9, 11])
```

```
 x_barra_one_less  =
  -6.2101493
   15.902409
   1.5568757
  -5.0718598
  -7.1846562
   1.2702227
  -0.9298812
   0.5285964
   1.9535131
   0.7701829
```

Agora, vamos fazer a nova previsão e calcular sua acurácia.

```
y_test_pred_one_less = X_test_one_less * x_barra_one_less;
y_test_pred_one_less_conf = y_test_pred_one_less .* y_test;
test_one_less_accuracy = sum(y_test_pred_one_less_conf > 0)/size(y_test)(1)
```

```
 test_one_less_accuracy  =
   0.8678571
```

Tivemos uma perda de aproximadamente 2% de acurácia, o que é bem razoável! Eliminar aquela variável não surtiu tanto efeito na previsão.

Para finalizar, vamos fazer isso em loop até não sobrar nenhuma variável no modelo (até termos apenas o bias). Abaixo está o código do loop e as acurácias ao longo da remoção das variáveis.

```
// Vetor para armazenar a acurácia de cada teste
accuracies = [test_accuracy; zeros((10, 1))];
```

```
// Inicializando os dados e o vetor de pesos reduzidos
reduced_X_test = X_test_bias;
reduced_x_barra = x_barra;

// Em cada iteração...
for i = 1:10
    // Encontra o índice do menor peso em módulo do vetor x_barra
    weaker_variable = find(abs(reduced_x_barra) == min(abs(reduced_x_barra(2:(12 - i)))));
    // Remove do dado a coluna correspondente a esse peso
    reduced_X_test = reduced_X_test(:, [1:(weaker_variable - 1) (weaker_variable + 1):(12 - i)]);
    // Remove o próprio peso do vetor
    reduced_x_barra = reduced_x_barra([1:(weaker_variable - 1) (weaker_variable + 1):(12 - i)]);

    // Faz uma nova predição do conjunto de teste
    reduced_y_test_pred = reduced_X_test * reduced_x_barra;
    // Calcula a acurácia
    reduced_y_test_pred_conf = reduced_y_test_pred .* y_test;
    reduced_accuracy = sum(reduced_y_test_pred_conf > 0)/size(y_test)(1);
    // Salva-a no vetor
    accuracies(i + 1) = reduced_accuracy;
end

accuracies
```

```
 accuracies  =
   0.8892857
   0.8678571
   0.9
   0.8857143
   0.9071429
   0.8142857
   0.6964286
   0.6928571
   0.7964286
   0.425
   0.6928571
```

É possível observar que, removendo algumas das variáveis menos relevantes (até 4 delas), a acurácia permanece alta, até mesmo aumentando em alguns casos. Isso ocorre provavelmente pela redução de ruído ocasionada pela remoção dessas características. No entanto, quando temos poucas variáveis restantes, o desempenho começa a diminuir, dado que há poucas informações à disposição para realizar a análise e a predição corretas.

Uma informação curiosa é o fato de que o último ponto, referente ao caso em que todas as variáveis foram removidas, restando apenas o bias, possui quase 70% de acurácia, e isso ocorre porque, como o

valor do bias é negativo, então todos os registros são classificados como negativos. Assim, dado que quase 70% dos registros são realmente negativos, esse acaba sendo exatamente o valor da acurácia.