Gabriel de Sousa Araujo (9299341)
Lucas Hattori Costa (10335847)

Métodos Numéricos e Aplicações

Primeiro Exercício Programa: Fatoração de Matrizes para Sistemas de Classificação de Machine Learning

Brasil

2019

Gabriel de Sousa Araujo (9299341)
Lucas Hattori Costa (10335847)

Métodos Numéricos e Aplicações

Primeiro Exercício Programa: Fatoração de Matrizes para Sistemas de Classificação de Machine Learning

Exercício Computacional: Fatoração de Matrizes para Sistemas de Classificação de Machine Learning

Universidade de São Paulo - USP

Escola Politécnica

Turma 11

MAP 3121

Orientador: Prof. Pedro Peixoto

Brasil

2019

Lista de ilustrações

Sumário

1 Introdução

A proposta do primeiro Exercício-Programa consiste em empregar conceitos de métodos númericos e de *Machine Learning* para encontrar padrões em grandes quantidades de dados, tornando possível sua classificação e resumo, de forma que possam ser processadas por um humano.

Para a abordagem em Python, os dados foram consolidados na forma matricial e computados por meios de processos, como Rotações de Givens, escalonamento e fatoração, para viabilizar a resolução de sistemas sobredeterminados e simultâneos.

Dessa forma, tornou-se possivel treinar classificadores e capacitar a máquina para o reconhecimento de dígitos manuscritos. A alteração das condições iniciais de cada um desses sistemas, como a dimensão das matrizes e o número de imagens para treino, implica diretamente na acurácia das respostas obtidas.

Nosso algoritmo foi capaz de obter os resultados em um tempo computacional aceitável, com mais de 85% de acerto na classificação.

2 Primeira Tarefa

Como descrito no enunciado do Exercício-Programa, foi desenvolvido um algoritmo para efetuar sucessivas rotações de Givens em uma matriz W de dimensão nXm a fim de se obter uma matriz R de mesma dimensão, porém triangular superior, bem como um vetor $\tilde{b}$ também rotacionado. Com essas duas novas matrizes, resolvemos um sistema sobredeterminado, obtendo uma matriz H cujas colunas são os vetores solução de cada sistema.

Dividimos esse problema em duas partes conceituais:

1. Fatoração QR de matrizes
2. Resolver sistemas simultâneos

2.1 Fatoração QR de matrizes

A fatoração de matrizes foi estruturada de acordo com o algoritmo descrito no enunciado, ou seja, se utilizando de sucessivas rotações de Givens representadas por matrizes Q_i que, multiplicadas, resultam em Q. A partir de Q podemos encontrar uma matriz R que equivale à matriz original escalonada. O trecho de código abaixo representa esse algoritmo:

2.1.1 Item A

Para essa matriz A descrita no enunciado, obtemos a matriz A_resposta tal qual demonstrada.
Matriz A: n = m = 64, Wi,i = 2, i = 1, n, Wi,j = 1, se |i j| = 1 e Wi,j = 0, se |i j| > 1. Use b(i) = 1, i = 1, n. B = [0.49, 0.01, 0.47, 0.03, 0.46, 0.04, 0.44, 0.06, 0.43, 0.07, 0.41, 0.09, 0.40, 0.10, 0.38, 0.12, 0.36, 0.13, 0.35, 0.15, 0.33, 0.16, 0.32, 0.18, 0.30, 0.20, 0.29, 0.21, 0.27, 0.23, 0.26, 0.24, 0.24, 0.26, 0.23, 0.27, 0.21, 0.29, 0.20, 0.30, 0.18, 0.32, 0.16, 0.33, 0.15, 0.35, 0.13, 0.36, 0.12, 0.38, 0.10, 0.40, 0.09, 0.41, 0.07, 0.43, 0.06, 0.44, 0.04, 0.46, 0.03, 0.47, 0.01, 0.49]

2.1.2 Item B

n = 20, m = 17, Wi,j = 1/(i + j 1), se |i j| 4 e Wi,j = 0, se |i j| > 4. Use b(i) = i, i = 1, n.

Resposta: [52.52, -43.48, -42.48, -50.51, -21.18, 81.86, 44.15, 55.12, 10.04, 101.07, -69.3, -53.66, -56.78, -9.39, 86.36, 205.51, 283.06]

2.2 Vários sistemas simultâneos

2.2.1 Item C

Resolvendo sistemas simultâneos para a amtriz dada no enunciado, obtemos a matriz demonstrada abaixo:

Input-> Matriz A: n = p = 64, Wi,i = 2, i = 1, n, Wi,j = 1, se |ij| = 1 e Wi,j = 0, se |ij| > 1. Defina m = 3, resolvendo 3 sistemas simultâneos, com A(i, 1) = 1, A(i, 2) = i, A(i, 3) = 2i 1, i = 1, n.

Out-> [[0.49, -0.5, -1.48], [0.01, 0.98, 1.95], [0.47, -0.48, -1.44], [0.03, 1.96, 3.9], [0.46, -0.47, -1.39], [0.04, 2.95, 5.86], [0.44, -0.45, -1.34], [0.06, 3.93, 7.81], [0.43, -0.44, -1.3], [0.07, 4.92, 9.76], [0.41, -0.42, -1.25], [0.09, 5.9, 11.72], [0.4, -0.41, -1.2], [0.1, 6.89, 13.67], [0.38, -0.39, -1.16], [0.12, 7.87, 15.63], [0.36, -0.37, -1.11], [0.13, 8.86, 17.58], [0.35, -0.36, -1.07], [0.15, 9.84, 19.53], [0.33, -0.34, -1.02], [0.16, 10.83, 21.49], [0.32, -0.33, -0.97], [0.18, 11.81, 23.44], [0.3, -0.31, -0.93], [0.2, 12.8, 25.39], [0.29, -0.3, -0.88], [0.21, 13.78, 27.35], [0.27, -0.28, -0.84], [0.23, 14.76, 29.3], [0.26, -0.27, -0.79], [0.24, 15.75, 31.26], [0.24, -0.25, -0.74], [0.26, 16.73, 33.21], [0.23, -0.24, -0.7], [0.27, 17.72, 35.16], [0.21, -0.22, -0.65], [0.29, 18.7, 37.12], [0.2, -0.21, -0.6], [0.3, 19.69, 39.07], [0.18, -0.19, -0.56], [0.32, 20.67, 41.03], [0.16, -0.17, -0.51], [0.33, 21.66, 42.98], [0.15, -0.16, -0.47], [0.35, 22.64, 44.93], [0.13, -0.14, -0.42], [0.36, 23.63, 46.89], [0.12, -0.13, -0.37], [0.38, 24.61, 48.84], [0.1, -0.11, -0.33], [0.4, 25.6, 50.79], [0.09, -0.1, -0.28], [0.41, 26.58, 52.75], [0.07, -0.08, -0.24], [0.43, 27.56, 54.7], [0.06, -0.07, -0.19], [0.44, 28.55, 56.66], [0.04, -0.05, -0.14], [0.46, 29.53, 58.61], [0.03, -0.04, -0.1], [0.47, 30.52, 60.56], [0.01, -0.02, -0.05], [0.49, 31.5, 62.52]]

2.2.2 Item D

n = 20, p = 17, Wi,j = 1/(i + j 1), se |i j| 4 e Wi,j = 0, se |i j| > 4. Defina m = 3, resolvendo 3 sistemas simultâneos, com A(i, 1) = 1, A(i, 2) = i, A(i, 3) = 2i 1, i = 1, n.

[[2.83, 52.52, 102.21], [-1.81, -43.48, -85.15], [-1.55, -42.48, -83.41], [-1.70, -50.51, -99.31], [-0.03, -21.18, -42.33], [5.75, 81.86, 157.98], [3.36, 44.15, 84.93], [3.63, 55.12, 106.61], [1.19, 10.04, 18.88], [6.02, 101.07, 196.12], [-2.43, -69.30, -136.17], [-1.52, -53.66, -105.79], [-1.54, -56.78, -112.02], [0.91, -9.39, -19.7], [6.21, 86.36, 166.51], [11.42, 205.51, 399.60], [14.52, 283.06, 551.60]]

3 Segunda Tarefa

Para a segunda etapa do exercício, implementamos o método de *alternating least squares* para efetuar uma *non-negative matrix factorization* (nmf), ou seja, estamos fatorando uma matriz A de dimensão nXm em duas outras matrizes, W e H, de dimensões nXp e pXm com elementos positivos, respectivamente. Para isso, fizemos um algoritmo iterativo, onde, a partir de uma W aleatória, computamos uma matriz H através de uma resolução de sistemas simultâneos. A partir dessa H, computamos uma nova W e calculamos a diferença entre a A' aproximada e A original, de acordo com a equação (3.1).

$$\begin{aligned} E &= ||A - WH||^2 \quad onde\ A' = WH \\ &= \sum_{i=1}^{n}\sum_{j=1}^{m}(A_{i,j} - (WH)_{i,j})^2 \end{aligned}$$

```
def nmf (A,p):
    start_time = time.time()
    W = np.random.rand(A.shape[0],p)
    #gera uma matriz com elementos aleatorios
    error = 1
    i = 0
    while abs(error) > 1e-5 and i<100:
        W = normalizar(W)
        A_aux = A.copy()
        H = simult(W,A_aux).clip(0)
        A_aux = A.copy()
        W = simult(H.T,A_aux.T).T.clip(0)
        #Diferen a entre a A aproximada e A original
        matrix_error = (A - np.einsum('ij,jk->ik',W,H))
        #Calculo do erro quadratico da matriz
        error = np.sqrt(np.einsum('ij,ij->', matrix_error, matrix_error))
        i+=1
    end_time = time.time()
    print(i,' itera es ',end_time-start_time,' dura o ',W)
    return W
```

Para testar o algoritmo, temos a seguinte decomposição exata fornecida pelo enunciado:

$$A = WH \tag{3.1}$$

$$\begin{pmatrix} \frac{3}{10} & \frac{3}{5} & 0 \\ \frac{1}{2} & 0 & 1 \\ \frac{4}{10} & \frac{4}{5} & 0 \end{pmatrix} = \begin{pmatrix} \frac{3}{5} & 0 \\ 0 & 1 \\ \frac{4}{5} & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 0 & 1 \end{pmatrix} \tag{3.2}$$

Utilizando o algoritmo supracitado obtemos, com $p = 2$:

$$\begin{aligned} [Out] : & 7 iterações \\ & 0.015604496002197266 duração(s) \\ & array([[1.38286502e-06, 6.00001383e-01], \\ & [1.00000024e+00, 2.44078353e-07], \\ & [1.84382002e-06, 8.00001844e-01]]) \end{aligned} \tag{3.3}$$

A função retorna um vetor com termos da ordem de grandeza de e^{-6} e e^{-7}, os quais podem ser aproximados para 0. Dessa forma, os resultados podem ser escritos como:

$$\mathrm{W} = \begin{pmatrix} 0.6 & 0.0 \\ 0.0 & 1.0 \\ 0.8 & 0.0 \end{pmatrix}$$

Que era o valor esperado.

Buscando mensurar as implicações causadas devido à função "random"como chute inicial, analisamos 10 situações. Os resultados apresentados na Tabela 3 mostram que a moda do número de iterações é 7 («<100), e o sistema é solucionado em um tempo médio de 10,3 ms. Por ser um sistema com soluções exatas, o resultado está de acordo com o esperado.

Número de iterações	Tempo transcorrido (ms)
7	15,6
8	9,8
2	10,5
7	10,0
8	10,0
7	15,0
7	8,4
5	10,1
7	3,9
7	9,6

Tabela 1 – Resultados da Segunda Tarefa para solução exata

Vamos fazer um teste agora com a fatoração de outra matriz, dessa vez não-exata. Criamos uma matriz A', com valores aproximadamente próximos:

$$\text{A'} = \begin{pmatrix} 0.31 & 0.6 & 0 \\ 0.5 & 0 & 0.98 \\ 0.4 & 0.81 & 0 \end{pmatrix}$$

Como A'≈A, podemos esperar que W'*approx*W. Utilizando o código supra mencionado 10 vezes seguidas, obtemos em todas o seguinte resultado:

$$\text{W'} = \begin{pmatrix} 0.59833061 & 0.00912622 \\ 0. & 0.99999352 \\ 0.80164114 & 0. \end{pmatrix}$$

É evidente que W' aproxima W com elevado grau de similaridade, com diferenças na ordem de 10^{-3}. O resultado da convergência, apesar de haver diferença entre a matriz W e W' pode ser interpretado como o que ocorrerá entre as imagens de treinamento, e as que são usadas no treinamento. De forma aplicada, se imagens similares (ou seja, que formem matrizes similares tais quais A e A') gerarão matrizes classificadoras similares (tais quais W e W').

4 Tarefa Principal

Na primeira seção, de treinamento, fazemos uma fatoração não negativa para cada matriz formada pelas imagens de treino, ou seja, encontramos uma matriz W classificadora para cada dígito em cada instância de treinamento. No código abaixo, pode-se ver que optamos por construir um tensor de dimensão 10 X 784 X p, composta por 10 matrizes W de dimensão 784 X p cada qual referente ao um dígito diferente.

```
def treinamento():
    ndig_treino = int(input('N mero de imagens usadas para treino[int]:'))
    p = int(input('Dimens o de W[int]:'))
    start_time = time.time()
    W_dig = []
    for i in range(10):#Vamos treinar uma W para cada d gito
        print('Treinando para o algarismo',i)
        dig_time = time.time()
        file_name = ('train_dig'+str(i)+".txt")
        #seleciona as primeiras ndig_treino colunas para treino
        cols = np.arange(0,ndig_treino)
        A = np.loadtxt(file_name, usecols = cols)
        W_i = nmf(A, p)
        W_dig.append(W_i)
        print(time.time() - dig_time,' tempo', i)
    W_dig = np.array(W_dig)
    file_name = 'W_dig para '+str(ndig_treino)+' imagens de treino e dim '+
    str(p)
    np.save(file_name, W_dig)
    end_time = time.time()
    print(end_time - start_time,' para treinamento de ',ndig_treino,'x',p)
```

No treinamento serão executados 9 testes, com o intuito de treinar os classificadores e avaliar o impacto da dimensão das matrizes e número de imagens de treino na eficiência da identificação dos dígitos.

Test number	*ndig_treino*	*p*	*n_test*
i	100	5	10.000
ii	100	10	10.000
iii	100	15	10.000
iv	1.000	5	10.000
v	1.000	10	10.000
vi	1.000	15	10.000
vii	4.000	5	10.000
viii	4.000	10	10.000
ix	4.000	15	10.000

Tabela 2 – Condições iniciais

Começamos treinando os classificadores por meio do cálculo das matrizes Wd para cada dígito, para cada uma das situações propostas na Tabela 2. O tempo de treino para cada algarismo, assim como o tempo total para o treinamento, pode ser encontrado na Tabela 3. Como eesperado, o tempo de treino para cada algarismo tende a uma média comum, mas conforme há o aumento no volume de dados trabalhados (representados por p e *ndig_treino*), há o aumento no tempo para tal atividade.

	Algarismos										
Test	0	1	2	3	4	5	6	7	8	9	Total
i	4.76	4.61	4.63	4.67	4.70	4.81	4.62	4.97	4.91	4.92	47.66
ii	8.74	8.53	8.59	9.33	9.28	8.73	8.40	8.80	8.58	8.70	87.79
iii	12.43	12.45	12.62	13.40	13.20	12.70	12.47	14.46	14.72	13.02	131.55
iv	15.60	15.73	15.62	15.64	15.65	16.01	15.68	15.79	15.67	15.73	157.14
v	27.62	27.34	27.73	27.65	27.78	27.66	27.84	27.64	27.58	30.97	279.86
vi	45.34	40.30	39.77	40.23	39.68	39.27	39.09	39.82	39.78	39.91	403.27
vii	50.77	48.73	48.83	48.62	50.21	48.96	49.18	48.64	48.92	48.36	491.26
viii	86.81	91.32	91.97	101.14	96.26	86.67	87.90	88.05	87.45	87.26	904.87
ix	124.81	125.57	124.69	125.17	125.13	125.35	124.28	125.76	124.95	124.85	1250.62

Tabela 3 – Tempo de treino para cada algarismo (s)

Evidentemente, o tempo de execução do algoritmo cresce para maiores p e mais imagens de treinamento. Podemos ver também que o aumento no tempo tem característica aproximadamente linear, ou seja, o tempo total das condições i ii e iii estão em proporção aproximada de 1:2:3, assim como o número de colunas da matriz classificadora (p). Isso é esperado porque o número de iterações na rotina de treinamento é diretamente proporcional ao número de colunas de W. Entretanto, vemos que a alteração no número de imagens teste não causa um aumento linear da mesma forma já que esse aumento não causa um aumento nas iterações de treinamento mas sim na rotina de resolver os sistemas simultâneos de forma não linear.

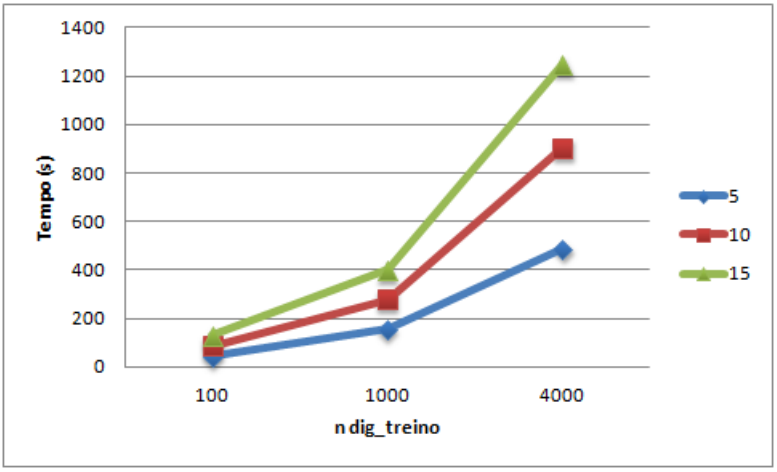

Figura 1 –

Usamos, então, o conjunto de testes para encontrar o percentual de acerto de cada um dos classificadores.

Há o cálculo da norma euclidiana.

$$||cj|| = \sqrt{\sum_{i=1}^{784} c_{i,j}^2} \tag{4.1}$$

Assumimos que a imagem é um dígito d se o valor de erro correspondente $e_j = ||c_j||$ for menor que o obtido para os outros dígitos.

O banco de dados fornecia a seguinte quantidade de imagens para cada dígito:

Dígito	Quantidade
0	980
1	1135
2	1032
3	1010
4	982
5	892
6	958
7	1028
8	974
9	1009
Total	10000

Tabela 4 – Número de imagens de teste para cada algarismo.

	Algarismos										
Test	0	1	2	3	4	5	6	7	8	9	Total
i	955	1128	877	868	827	737	884	902	784	876	
%	97.4	99.4	84.9	85.9	84.2	82.6	92.2	87.7	80.4	86.8	88.4
ii	953	1128	936	867	851	757	922	941	764	881	
%	97.2	99.3	90.7	85.8	86.6	84.8	96.2	91.5	78.4	87.3	90.0
iii	958	1129	934	882	867	756	910	966	819	891	
%	97.7	99.4	90.5	87.3	88.2	84.7	94.9	93.9	84.0	88.3	91.1
iv	958	1124	899	926	822	780	918	913	834	879	
%	97.7	99.0	87.1	91.6	83.7	87.4	95.8	88.8	85.6	87.1	90.5
v	965	1129	912	927	884	794	923	945	867	912	
%	98.4	99.4	88.3	91.7	90.0	89.0	96.3	91.9	89.0	90.3	92.6
vi	965	1128	944	927	911	794	928	940	859	930	
%	98.4	99.4	91.4	91.7	92.7	89.0	96.8	91.4	88.1	92.1	93.2
vii	956	1127	926	925	854	786	921	920	843	886	
%	97.5	99.2	89.7	91.5	86.9	88.1	96.1	89.4	86.5	87.8	91.4
viii	963	1128	943	942	914	804	923	943	856	916	
%	98.2	99.3	91.3	93.2	93.1	90.1	96.3	91.7	87.8	90.7	93.3
ix	965	1124	935	931	927	820	930	937	872	931	
%	98.4	99.0	90.6	92.1	94.3	91.9	97.0	91.1	89.5	92.2	93.7

Tabela 5 – Acertos

	p = 5	p = 10	p = 15
ndig = 100	88.4	90.0	91.1
ndig = 1000	90.5	92.6	93.2
ndig = 4000	91.4	93.3	93.7

Tabela 6 – Implicação da variação de p e Ndig treino na porcentagem de acertos

Avaliando os resultados presentes nas Tabelas 5 e 6 é possível concluir que o aumento do p promove o aumento no índice de acertos (analisando cada linha), assim como o número de imagens do dataset também melhora a acurácia dos resultados (analisando coluna a coluna).

Conclui-se então que o aumento no número de sistemas simultâneos, assim como a quantidade de imagens usadas para o treinamento, como o esperado, promovem a melhora na classificação dos dígitos.