Gustavo Lopes Oliveira - 10335490
Lucas Hideki Takeuchi Okamura - 9274315

Relatório EP1
Machine Learning

São Paulo-SP, Brasil

21 de maio de 2019

Gustavo Lopes Oliveira - 10335490
Lucas Hideki Takeuchi Okamura - 9274315

Relatório EP1
Machine Learning

Universidade de São Paulo – USP

Escola Politécnica

MAP3121

Orientador:
Profº Dr. Pedro da Silva Peixoto

São Paulo-SP, Brasil

21 de maio de 2019

Resumo

O presente trabalho compõe o desenvolvimento de fatoração de matrizes para sistemas de classificação de *Machine Learning*. O processo de classificação de *Machine Learning* inicia-se desde a fatoração de matrizes de formas diversas até a implementação desses métodos para a usos mais complexos, como o exemplo a ser mostrado ao longo deste trabalho. Com a programação em linguagem *Pyhton*, foram elaborados códigos para fatorações do tipo QR e WH, que resolvem sistemas simultâneos. Assim, foi realizada uma aplicação realística que utilizava as fatorações citadas para classificação de dígitos manuscritos, que verifica qual dígito aparece em uma determinada imagem, a partir da avaliação do erro quadrático da decomposição WH.

Palavras-chaves: *Machine Learning*, *Pyhton*, fatoração

Lista de ilustrações

Figura 1 – Determinação de cosθ e senθ para a rotação de Givens 7
Figura 2 – Rotina em *Python* para calcular cosθ e senθ. 7
Figura 3 – Pseudo-código para aplicação da rotação de Givens 8
Figura 4 – Rotina em *Python* para aplicação da rotação de Givens 8
Figura 5 – Pseudo-código para resolução de sistemas lineares 8
Figura 6 – Rotina em *Python* para resolução de sistemas lineares 9
Figura 7 – Rotina em *Python* para criação das matrizes W e b do item a) 10
Figura 8 – Tabela indicativa dos resultados do sistema linear da tarefa a 11
Figura 9 – Rotina em *Python* para criação das matrizes W e b do item b) 12
Figura 10 – Tabela indicativa dos resultados do sistema linear da tarefa b 13
Figura 11 – Pseudo-código para resolução de sistemas simultâneos 14
Figura 12 – Rotina em *Python* para resolução de matrizes 15
Figura 13 – Rotina em *Python* para criação da matriz b 16
Figura 14 – Tabela indicativa dos resultados do sistema linear da tarefa c 18
Figura 15 – Tabela indicativa dos resultados do sistema linear da tarefa d 20
Figura 16 – Pseudo-código para o Método dos mínimos quadrados alternados . . . 21
Figura 17 – Rotina em *Python* para o Método dos mínimos quadrados alternados . 22
Figura 18 – Rotina em *Python* para o cálculo do erro entre A e WH 22
Figura 19 – Matrizes A, W e H para a segunda tarefa 23
Figura 20 – Rotina em *Python* para o treinamento de dígitos 25
Figura 21 – Rotina em *Python* para a classificação de dígitos 26
Figura 22 – Índice total de acertos para os 9 conjuntos de classificadores 27
Figura 23 – Gráfico dos erros para treino do dígito 0 28
Figura 24 – Exemplares do dígito 7 28
Figura 25 – Índices de acerto dos classificadores para cada dígito 29
Figura 26 – Exemplares para os dígitos 6, 0 e 1 29
Figura 27 – Digítos com escrita fora do padrão 30
Figura 28 – ndig$_t reino = 100, ndig_t est = 10000, p = 5$ 34
Figura 29 – ndig$_t reino = 100, ndig_t est = 10000, p = 10$ 34
Figura 30 – ndig$_t reino = 100, ndig_t est = 10000, p = 15$ 34
Figura 31 – ndig$_t reino = 1000, ndig_t est = 10000, p = 5$ 35
Figura 32 – ndig$_t reino = 1000, ndig_t est = 10000, p = 10$ 35
Figura 33 – ndig$_t reino = 1000, ndig_t est = 10000, p = 15$ 35
Figura 34 – ndig$_t reino = 4000, ndig_t est = 10000, p = 5$ 36
Figura 35 – ndig$_t reino = 4000, ndig_t est = 10000, p = 10$ 36
Figura 36 – ndig$_t reino = 4000, ndig_t est = 10000, p = 15$ 36

Sumário

Introdução 5

I FATORAÇÃO DE MATRIZES PARA SISTEMAS DE CLASSIFICAÇÃO DE *MACHINE LEARNING* 6

1 FATORAÇÃO QR DE MATRIZES E SOLUÇÃO DE SISTEMAS LINEARES 7
1.1 Rotações de Givens 7
1.2 Fatoração QR 7
1.3 Resolvendo sistemas lineares através da Fatoração QR 8
1.4 Primeira Tarefa 9
1.4.1 Sistemas lineares simples 9
1.4.2 Vários sistemas lineares simultâneos 13

2 FATORAÇÃO POR MATRIZES NÃO NEGATIVAS 21
2.1 Método dos mínimos quadrados alternados 21
2.2 Segunda tarefa 22

3 CLASSIFICAÇÃO DE DÍGITOS MANUSCRITOS 25
3.1 Treinamento de um dígito d 25
3.2 Classificação de um dígito d 26
3.3 Tarefa principal 27

4 CONCLUSÃO 31

REFERÊNCIAS 32

II APÊNDICES 33

5 GRÁFICOS 34
5.1 Gráficos de erro para treino de um dígito 34

Introdução

Desde o surgimento do computador e da internet, a tecnologia vem avançando e se sofisticando cada vez mais. Sistemas digitais e máquinas cada vez mais tornam-se parte do nosso cotidiano e realizam tarefas complexas com uma rapidez surpreendente. Assim, com o grande número de dados circulando em todo o mundo, há uma crescente necessidade de se analisar dados automaticamente, sem a intervenção humana.

Nesse sentido, tem sido frequente o uso do termo *Machine Learning*,que é fundamentado na inteligência artificial e o que as máquinas podem realizar com o recebimento de dados. É importante no *Machine Learning* o seu aspecto iterativo, visto que os modelos expostos a novos dados são capazes de se adaptarem independentemente. Eles aprendem com dados anteriores para aprimorar e produzir resultados confiáveis e cada vez melhores.

Neste contexto, muitas ferramentas de aprendizado de máquina fazem uso de fatorações de matrizes de dados. Diversos métodos de fatoração agilizam o trabalho e diminuem o tempo de execução para determinadas tarefas, sendo muito útil, pois, a implementação desses métodos em certos códigos.

Este relatório apresenta o uso de fatorações em código de linguagem *Pyhton* para aplicações realísticas. Serão abordados a fatoração QR, a partir das Rotações de Givens; a solução de sistemas simultâneos a partir da fatoração QR; fatoração por matrizes não negativas ("non-negative matrix faxtorization - NMF), feita a partir da avaliação do erro quadrático entre a matriz original e a fatoração. Tais tópicos foram utilizados para a classificação de digitos manuscritos, ou seja, por meio de código, a máquina deve descobrir qual número está representado em determinada imagem.

Primeiramente, um dígito d é treinado para reconhecer determinado algarismo e após este treinamento, pode-se usá-lo para avaliar se um outro dígito é de fato d ou não.

Através do código elaborado, foram avaliados dados extraídos da base de dados MNIST[(1)], avaliando dígitos escritos a mão e retornando qual número estava representado na imagem. O resultado do código apresentado neste relatório apresentou boa acurácia na avaliação dos dígitos, em um tempo de execução relativamente curto.

Parte I

Fatoração de Matrizes para Sistemas de Classificação de *Machine Learning*

1 Fatoração QR de matrizes e solução de sistemas lineares

Será desenvolvido o método de fatoração QR de matrizes $n \times m$,com $n \geq m$, a partir de rotações de Givens, descritas a seguir:

1.1 Rotações de Givens

Rotações de Givens são transformações lineares ortogonais de R^n em R^n. Uma transformação de R^2 em R^2 é dada por:

$$\mathbf{Q} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Ao ser aplicada uma rotação de Givens $Q(i, j, \theta)$ a uma matriz $W_{n,m}$, apenas as linhas i e j de W são modificadas.

Para realizar as rotações de Givens, foram definidas as expressões de $c = cos\theta$ e de $s = sen\theta$, de acordo com as seguintes expressões:

$$c = \frac{w_{i,k}}{\sqrt{w_{i,k}^2 + w_{j,k}^2}} \quad \text{e} \quad s = -\frac{w_{j,k}}{\sqrt{w_{i,k}^2 + w_{j,k}^2}}$$

Figura 1 – Determinação de cosθ e senθ para a rotação de Givens

Transformando as expressões em uma função em linguagem *Pyhton*, obtemos:

```
def sencos(W,i,j,k): #Funcao que calcula sen e cos para rotacao de givens
    c = W[i,k]/math.sqrt(W[i,k]*W[i,k]+W[j,k]*W[j,k])
    s = -W[j,k]/math.sqrt(W[i,k]*W[i,k]+W[j,k]*W[j,k])
    return c, s
```

Figura 2 – Rotina em *Python* para calcular cosθ e senθ.

1.2 Fatoração QR

Para a implementação da fatoração QR, foi seguido o pseudo-código a seguir, apresentado no enunciado do EP:

Rot-givens(W,n,m,i,j,c,s)

inteiros m,n,i,j,r

reais W(n,m), c, s, aux

para r=1,m

aux = c * w(i,r) - s * w(j,r)

w(j,r) = s * w(i,r) + c * w(j,r)

w(i,r) = aux

fim do para

Figura 3 – Pseudo-código para aplicação da rotação de Givens

E passando para o código em linguagem *Pyhton*, temos:

```
def givensrot(W,i,j,m,c,s): #Funcao que aplica rotacao de givens para W e b
    Wi = W[i,:]
    Wj = W[j,:]
    aux = Wi*c-s*Wj
    W[j,:] = Wi*s+c*Wj
    W[i,:] = aux
    bi = b[i]
    bj = b[j]
    aux = bi*c-s*bj
    b[j] = bi*s+bj*c
    b[i] = aux
    return W, b
```

Figura 4 – Rotina em *Python* para aplicação da rotação de Givens

1.3 Resolvendo sistemas lineares através da Fatoração QR

Para solução do sistema $Wx = b$ utilizando a fatoração QR, deve-se resolver $Rx = \tilde{b}$, onde R é a matriz triangular superior oriunda de W após aplicadas as rotações de Givens e $\tilde{b}$ é a matriz b modificada após as rotações de Givens.

Novamente, foi seguido o pseudo-código apresentado no enunciado do EP:

Para k=1 a m faça

Para j=n a k+1 com passo -1 faça

i=j-1

Se $w_{j,k} \neq 0$ aplique $Q(i, j, \theta)$ à matriz W (com c e s definidos por $w_{i,k}$ e $w_{j,k}$) e ao vetor b.

Fim do para

Fim do para

Para k=m a 1 com passo -1

$x_k = (b_k - \sum_{j=k+1}^{m} w_{k,j} x_j)/w_{k,k}$

Fim do para

Figura 5 – Pseudo-código para resolução de sistemas lineares

O primeiro código para resolução de sistemas lineares simples foi elaborado com duas funções, além daquelas já citadas (*sencos*, *givensrot*). A primeira transforma a matriz W em uma matriz triangular e aplica a rotação de Givens em b e a segunda, que de fato resolve o sistema linear escalonado, retorna o valor da matriz x, ou seja, os valores das raízes da matriz W inicial para um A pré definido. São elas, *solver* e *solve*, respectivamente:

```
def solver(K,A,n,m): #Função que resolve varios sistemas simultâneos
    b = np.transpose(A) #Usa-se a transposta de A para ficar mais fácil escolher uma coluna separadamente
    for k in range(0,m): #Essa seção se baseia na aplicação do algoritmo apresentado no enunciado
        for j in range(n-1,k,-1):
            i=j-1
            if K[j,k] != 0:
                c,s = sencos(K,i,j,k)
                K, b = givensrot(K,i,j,m,c,s)
            A = np.transpose(b) #transpoe-se novamente b
    x = solve(K,b,n,m) #chama a funcao que resolve o sistema
    return x


def solve(W,A,n,m): #funcao que resolve o sistema
    x = np.zeros(m) #cria matriz para acondicionar os resultados
    for k in range(m-1,-1,-1): #Aplica o algoritmo apresentado no enunciado
        aux = 0
        for j in range(k,m):
            aux += W[k,j]*x[j] #Armazena a soma
        x[k] = (A[k] - aux)/W[k,k]
    return x
```

Figura 6 – Rotina em *Python* para resolução de sistemas lineares

1.4 Primeira Tarefa

A primeira tarefa consiste em receber uma matriz $Wn \times m$ e um vetor $b \in R^n$ e através de sucessivas rotações de Givens, transformar W na matriz triangular superior R e o vetor b no vetor modificado $\tilde{b}$, resolvendo em seguida o sistema $Rx = \tilde{b}$.

1.4.1 Sistemas lineares simples

Foram fornecidos dados de W e de b para serem aplicados os métodos e a serem resolvidos os sistemas. A seguir mostra-se os resultados obtidos e o processo de resolução de acordo com o código elaborado:

a) $n = m = 64, W_{i,i} = 2, i = 1, n, W_{i,j} = 1$, se $|i - j| = 1$ e $W_{i,j} = 0$, se $|i - j| > 1$. Use $b(i) = 1, i = 1, n$

Para a resolução do item a), foram criadas funções para designar os valores corretos para as matrizes W e b, mostradas na figura a seguir:

```python
def matrixbuild(n,m): #matriz utilizada para criar a matriz W, dadas suas dimensões e lei de formacao
    K = np.zeros((n,m))
    for i in range(0,n):
        for j in range(0,m):
            if i == j:
                K[i,j] = 2
            elif abs(i-j) == 1:
                K[i,j] = 1
            else:
                K[i,j] = 0
    return K

def bbuild(n): #matriz utilizada para criar a matriz b, dadas suas dimensoes e lei de formacao
    b = np.zeros(n)
    for i in range(0,n):
        b[i] = 1
    return b

W = matrixbuild(64,64)
b = bbuild(64)
```

Figura 7 – Rotina em *Python* para criação das matrizes W e b do item a)

Obtendo as seguintes matrizes:

$$\mathbf{W_{64,64}} = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 2 \end{pmatrix}$$

$$\mathbf{b_{64,1}} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ \dots \\ \dots \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Definidas as matrizes W e b, foram aplicadas as rotações de Givens para ambas, a partir das funções *sencos* e *givensrot*, descritas anteriormente, obtendo os seguintes resultados (apenas parte dos resultados foram colocados neste relatório, devido ao tamanho da matriz) :

$$\mathbf{R_{64,64}} = \begin{pmatrix} 2.23607 & 1.78885 & 0.447214 & 0 & 0 & ... & 0 & 0 & 0 \\ 0 & 1.67332 & 1.91237 & 0.597614 & 0 & ... & 0 & 0 & 0 \\ 0 & 0 & 1.46385 & 1.9518 & 0.68313 & ... & 0 & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & 0 & ... & 1.0241 & 1.99981 & 0.97647 \\ 0 & 0 & 0 & 0 & 0 & ... & 0 & 1.02372 & 1.99982 \\ 0 & 0 & 0 & 0 & 0 & ... & 0 & 0 & 0.217344 \end{pmatrix}$$

$$\tilde{\mathbf{b}}_{\mathbf{64,1}} = \begin{pmatrix} 1.34164 \\ 0.956183 \\ 1.07349 \\ ... \\ ... \\ 0.999906 \\ 1.00027 \\ 0.107 \end{pmatrix}$$

Assim, para tais matrizes aplicam-se as funções resolvedoras de sistemas lineares (Figura 6), obtendo-se o seguinte resultado:

X					
1	0.492308	**23**	0.323077	**45**	0.153846
2	0.0153846	**24**	0.184615	**46**	0.353846
3	0.476923	**25**	0.307692	**47**	0.138462
4	0.0307692	**26**	0.2	**48**	0.369231
5	0.461538	**27**	0.292308	**49**	0.123077
6	0.0461538	**28**	0.215385	**50**	0.384615
7	0.446154	**29**	0.276923	**51**	0.107692
8	0.0615385	**30**	0.230769	**52**	0.4
9	0.430769	**31**	0.261538	**53**	0.0923077
10	0.0769231	**32**	0.246154	**54**	0.415385
11	0.415385	**33**	0.246154	**55**	0.0769231
12	0.0923077	**34**	0.261538	**56**	0.430769
13	0.4	**35**	0.230769	**57**	0.0615385
14	0.107692	**36**	0.276923	**58**	0.446154
15	0.384615	**37**	0.215385	**59**	0.0461538
16	0.123077	**38**	0.292308	**60**	0.461538
17	0.369231	**39**	0.2	**61**	0.0307692
18	0.138462	**40**	0.307692	**62**	0.476923
19	0.353846	**41**	0.184615	**63**	0.0153846
20	0.153846	**42**	0.323077	**64**	0.492308
21	0.338462	**43**	0.169231		
22	0.169231	**44**	0.338462		

Figura 8 – Tabela indicativa dos resultados do sistema linear da tarefa a

b) $n = 20, m = 17, W_{i,j} = 1/(i + j - 1)$, se $|i - j| \leq 4$ e $W_{i,j} = 0$, se $|i - j| > 4$. Use $b(i) = i, i = 1, n$

Assim como no item a), no item b) foram designadas funções para criar as matrizes W e b, vistas na figura a seguir:

```
def matrixbuild(n,m): #matriz utilizada para criar a matriz W, dadas suas dimensões e lei de formacao
    K = np.zeros((n,m))
    for i in range(0,n):
        for j in range(0,m):
            if abs(i-j) <= 4:
                K[i,j] = 1/(i+1+j+1-1)
            else:
                K[i,j] = 0
    return K

def bbuild(n): #matriz utilizada para criar a matriz b, dadas suas dimensoes e lei de formacao
    b = np.zeros(n)
    for i in range(0,n):
        b[i] = i + 1
    return b

W = matrixbuild(20,17)
b = bbuild(20)
```

Figura 9 – Rotina em *Python* para criação das matrizes W e b do item b)

Obtendo as seguintes matrizes:

$$\mathbf{W_{20,17}} = \begin{pmatrix} 1 & 0.5 & 0.333333 & \dots & \dots & 0 & 0 & 0 \\ 0.5 & 0.333333 & 0.25 & \dots & \dots & 0 & 0 & 0 \\ 0.333333 & 0.25 & 0.2 & \dots & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \dots & 0.03125 & 0.030303 & 0.0294118 \\ 0 & 0 & 0 & \dots & \dots & 0.030303 & 0.0294118 & 0.0285714 \\ 0 & 0 & 0 & \dots & \dots & 0 & 0.0285714 & 0.0277778 \end{pmatrix}$$

$$\mathbf{A_{20,1}} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ \dots \\ \dots \\ 18 \\ 19 \\ 20 \end{pmatrix}$$

E, novamente, aplicam-se as rotações de Givens, obtendo:

$$\mathbf{R_{20,17}} = \begin{pmatrix} 1.2098 & 0.68882 & 0.492014 & 0.385411 & ... & ... & 0 & 0 \\ 0 & 0.193193 & 0.18681 & 0.171496 & ... & ... & 0 & 0 \\ 0 & 0 & 0.1131 & 0.103167 & ... & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & ... & 0.0361266 & 0.0246044 & 0.0188544 \\ 0 & 0 & 0 & 0 & ... & 0 & 0.0410288 & 0.033789 \\ 0 & 0 & 0 & 0 & ... & 0 & 0 & 0.0281374 \\ 0 & 0 & 0 & 0 & ... & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ... & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ... & 0 & 0 & 0 \end{pmatrix}$$

$$\tilde{\mathbf{b}}_{\mathbf{20,1}} = \begin{pmatrix} 4.13292 \\ 8.07637 \\ 7.07216 \\ ... \\ ... \\ -1.87506 \\ -10.9695 \\ 0.749369 \end{pmatrix}$$

E, então, aplicadas as funções resolvedoras de sistema linear:

	X		
1	56.3578	**10**	109.163
2	-45.8748	**11**	-72.8728
3	-43.489	**12**	-54.3629
4	-48.5765	**13**	-51.4444
5	-30.1402	**14**	-25.0148
6	89.8121	**15**	98.5719
7	48.7136	**16**	218.659
8	59.2392	**17**	298.4
9	11.4464		

Figura 10 – Tabela indicativa dos resultados do sistema linear da tarefa b

1.4.2 Vários sistemas lineares simultâneos

Nesta tarefa, o objetivo é minimizar $||A - WH||$, onde A é uma matriz nxm, W é nxp e H é pxm. Para isso, tem-se as matrizes W e A dadas, sendo necessário determinar a matriz H, ou seja, encontrar as raízes dos múltiplos sistemas.

O produto de W por uma coluna h_j de H deve aproximar a coluna a_j de A. Em outras palavras, a intenção é resolver m sistemas sobredeterminados, com a mesma

matriz W simultaneamente, um para cada coluna da matriz A (sendo a solução a coluna correspondente da matriz H).

Para isso nos utilizamos do pseudo-código a seguir:

Para k=1 a p faça

    Para j=n a k+1 com passo -1 faça

        i=j-1

        Se $w_{j,k} \neq 0$ aplique $Q(i, j, \theta)$ à matriz W (com c e s definidos por $w_{i,k}$ e $w_{j,k}$) e à matriz A.

    Fim do para

Fim do para

Para k=p a 1 com passo -1

    Para j=1 a m faça

        $h_{k,j} = (a_{k,j} - \sum_{i=k+1}^{p} w_{k,i} h_{i,j})/w_{k,k}$

    Fim do para

Fim do para

Figura 11 – Pseudo-código para resolução de sistemas simultâneos

É necessário modificar as funções *givensrot*, *solve* e *solver* para que recebam matrizes H com mais colunas, ou seja, mais sistemas. Vale notar que a variável d no código é equivalente à variável p. O código em *Python* e as matrizes estão demonstrados na figura a seguir:

```python
def givensrot(W,b,i,j,g,m,c,s): #Funcao que aplica rotacao de givens para W e b
    Wi = W[i,:]
    Wj = W[j,:]
    aux = Wi*c-s*Wj
    W[j,:] = Wi*s+c*Wj
    W[i,:] = aux
    bi = b[:,i]
    bj = b[:,j]
    aux = bi*c-s*bj
    b[:,j] = bi*s+bj*c
    b[:,i] = aux
    return W, b


def solver(K,A,n,m,g): #Função que resolve varios sistemas simultâneos
    b = np.transpose(A) #Usa-se a transposta de A para ficar mais fácil escolher uma coluna separadamente
    for k in range(0,g): #Essa seção se baseia na aplicação do algoritmo apresentado no enunciado
        for j in range(n-1,k,-1):
            i=j-1
            if K[j,k] != 0:
                c,s = sencos(K,i,j,k) #
                K, b = givensrot(K,b,i,j,g,m,c,s)
    A = np.transpose(b) #transpoe-se novamente b
    x = solve(K,A,n,m,g) #chama a funcao que resolve o sistema
    return x



def solve(W,A,n,m,g): #funcao que resolve o sistema
    x = np.zeros((g,m)) #cria matriz para acondicionar os resultados
    for k in range(g-1,-1,-1): #Aplica o algoritmo apresentado no enunciado
        for j in range(0,m):
            aux = 0
            for i in range(k,g):
                aux += W[k,i]*x[i,j] #Armazena a soma
            x[k,j] = (A[k,j] - aux)/W[k,k]
    return x
```

Figura 12 – Rotina em *Python* para resolução de matrizes

É importante notar que o código em *Python* para realizar a rotação de Givens é um pouco diferente do algoritmo apresentado no enunciado, ao dispensar o uso do *loop for* para percorrer as matrizes, multiplicando diretamente as linhas e colunas de W e b (transposta de A), respectivamente. Optou-se por essa solução ao verificar a necessidade de reduzir o tempo necessário para a execução da função em fases posteriores do programa, onde se trabalhou com matrizes de dimensões elevadas. Optou-se, também, por usar a transposta de A, devido às situações onde se era necessário separar as colunas de A para resolver os sistemas, evitando a necessidade de transpor a transformação que seria aplicada durante a rotação de Givens. Porém, em essência, estão sendo realizadas as mesmas multiplicações do que o proposto no enunciado.

Para o teste inicial dos algoritmos, foram designados 2 casos a serem considerados. Eles e seus respectivos resultados estão apresentados a seguir:

c) $n = p = 64, W_{i,i} = 2, i = 1, n, W_{i,j} = 1$, se $|i - j| = 1$ e $W_{i,j} = 0$, se $|i - j| > 1$. Defina $m = 3$, resolvendo 3 sistemas simultâneos, com $A(i, 1) = 1, A(i, 2) = i, A(i, 3) = 2i - 1, i = 1, n$.

A matriz W é igual ao do item a), assim, a matriz foi construída a partir do código *matrixbuild* da figura 7. Já a matriz A foi elaborada a partir de outra função *bbuild*, vista a seguir:

```
def bbuild(n,m): #matriz utilizada para criar a matriz b, dadas suas dimensoes e lei de formacao
    b = np.zeros((n,m))
    for i in range(0,n):
        b[i,0]=1
        b[i,1]=i + 1
        b[i,2]=2*(i+1)-1
    return b
```

Figura 13 – Rotina em *Python* para criação da matriz b

Obtendo as seguintes matrizes:

$$\mathbf{W_{64,64}} = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & 2 \end{pmatrix}$$

$$\mathbf{A_{64,3}} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 5 \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ 1 & 62 & 123 \\ 1 & 63 & 125 \\ 1 & 64 & 127 \end{pmatrix}$$

Definidas as matrizes W e b, aplicamos para ambas as rotações de Givens, obtendo:

$$\mathbf{R_{64,64}} = \begin{pmatrix} 2.23607 & 1.78885 & 0.447214 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1.67332 & 1.91237 & 0.597614 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1.46385 & 1.9518 & 0.68313 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1.0241 & 1.99981 & 0.97647 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1.02372 & 1.99982 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0.217344 \end{pmatrix}$$

$$\tilde{\mathbf{b}}_{\mathbf{64,3}} = \begin{pmatrix} 1.34164 & 1.78885 & 2.23607 \\ 0.956183 & 2.86855 & 4.78091 \\ 1.07349 & 3.9036 & 6.73371 \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ 0.999906 & 62.994 & 124.988 \\ 1.00027 & 63.9941 & 126.988 \\ 0.107 & 6.955 & 13.803 \end{pmatrix}$$

Observa-se que a matriz R é igual a matriz R do item a), visto que a matriz W é identica e os fatores a serem multiplicados pelo método das rotações de Givens são os mesmo também. A primeira coluna de $\tilde{b}$ é idêntica ao item a), visto que o sistema é o mesmo. A segunda e a terceira coluna são diferentes devido ao fato de se referirem a outros sistemas lineares.

Resolvendo esse sistema simultâneo, obtemos:

E, então, aplicadas as funções resolvedoras de sistema linear temos a matriz seguinte, onde cada coluna é o resultado do respectivo sistema:

	X		
1	0.492308	-4.568E-15	-0.492308
2	0.0153846	1	1.98462
3	0.476923	-1.37E-14	-0.476923
4	0.0307692	2	3.96923
5	0.461538	-2.28E-14	-0.461538
6	0.0461538	3	5.95385
7	0.446154	-3.09E-14	-0.446154
8	0.0615385	4	7.93846
9	0.430769	-4.13E-14	-0.430769
10	0.0769231	5	9.92308
11	0.415385	-5.33E-14	-0.415385
12	0.0923077	6	11.9077
13	0.4	-6.54E-14	-0.4
14	0.107692	7	13.8923
15	0.384615	-8.09E-14	-0.384615
16	0.123077	8	15.8769
17	0.369231	-1.01E-13	-0.369231
18	0.138462	9	17.8615
19	0.353846	-1.15E-13	-0.353846
20	0.153846	10	19.8462
21	0.338462	-1.23E-13	-0.338462
22	0.169231	11	21.8308
23	0.323077	-1.20E-13	-0.323077
24	0.184615	12	23.8154
25	0.307692	-1.17E-13	-0.307692
26	0.2	13	25.8
27	0.292308	-1.14E-13	-0.292308
28	0.215385	14	27.7846
29	0.276923	-1.25E-13	-0.276923
30	0.230769	15	29.7692
31	0.261538	-1.42E-13	-0.261538
32	0.246154	16	31.7538

33	0.246154	-1.63E-13	-0.246154
34	0.261538	17	33.7385
35	0.230769	-1.84E-13	-0.230769
36	0.276923	18	35.7231
37	0.215385	-2.05E-13	-0.215385
38	0.292308	19	37.7077
39	0.2	-2.12E-13	-0.2
40	0.307692	20	39.6923
41	0.184615	-2.06E-13	-0.184615
42	0.323077	21	41.6769
43	0.169231	-2.13E-13	-0.169231
44	0.338462	22	43.6615
45	0.153846	-2.20E-13	-0.153846
46	0.353846	23	45.6462
47	0.138462	-2.13E-13	-0.138462
48	0.369231	24	47.6308
49	0.123077	-2.00E-13	-0.123077
50	0.384615	25	49.6154
51	0.107692	-1.79E-13	-0.107692
52	0.4	26	51.6
53	0.0923077	-1.73E-13	-0.0923077
54	0.415385	27	53.5846
55	0.0769231	-1.66E-13	-0.0769231
56	0.430769	28	55.5692
57	0.0615385	-1.45E-13	-0.0615385
58	0.446154	29	57.5538
59	0.0461538	-1.11E-13	-0.0461538
60	0.461538	30	59.5385
61	0.0307692	-7.63E-14	-0.0307692
62	0.476923	31	61.5231
63	0.0153846	-4.16E-14	-0.0153846
64	0.492308	32	63.5077

Figura 14 – Tabela indicativa dos resultados do sistema linear da tarefa c

Observa-se que na segunda coluna há números elevados a grandes potências negativas, o que permite concluir que o real resultados desses números tende a zero.

d) $n = 20, m = 17, W_{i,j} = 1/(i + j - 1)$, se $|i - j| \leq 4$ e $W_{i,j} = 0$, se $|i - j| > 4$. Defina $m = 3$, resolvendo 3 sistemas simultâneos, com $A(i, 1) = 1, A(i, 2) = i, A(i, 3) = 2i - 1, i = 1, n$.

Aqui a matriz W é igual a matriz W do item b) e a matriz A igual a do item c) até a linha 20:

$$\mathbf{W_{20,17}} = \begin{pmatrix} 1 & 0.5 & 0.333333 & \dots & \dots & 0 & 0 & 0 \\ 0.5 & 0.333333 & 0.25 & \dots & \dots & 0 & 0 & 0 \\ 0.333333 & 0.25 & 0.2 & \dots & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \dots & 0.03125 & 0.030303 & 0.0294118 \\ 0 & 0 & 0 & \dots & \dots & 0.030303 & 0.0294118 & 0.0285714 \\ 0 & 0 & 0 & \dots & \dots & 0 & 0.0285714 & 0.0277778 \end{pmatrix}$$

$$\mathbf{A_{20,3}} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 5 \\ ... & ... & ... \\ ... & ... & ... \\ 1 & 18 & 35 \\ 1 & 19 & 37 \\ 1 & 20 & 39 \end{pmatrix}$$

Aplicando as rotações de Givens para as matrizes acima, temos:

$$\mathbf{R_{20,17}} = \begin{pmatrix} 1.2098 & 0.68882 & 0.492014 & 0.385411 & ... & ... & 0 & 0 \\ 0 & 0.193193 & 0.18681 & 0.171496 & ... & ... & 0 & 0 \\ 0 & 0 & 0.1131 & 0.103167 & ... & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & ... & 0.0361266 & 0.0246044 & 0.0188544 \\ 0 & 0 & 0 & 0 & ... & 0 & 0.0410288 & 0.033789 \\ 0 & 0 & 0 & 0 & ... & 0 & 0 & 0.0281374 \\ 0 & 0 & 0 & 0 & ... & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ... & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ... & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{\tilde{b}_{20,3}} = \begin{pmatrix} 1.88737 & 4.13292 & 6.37848 \\ 1.51558 & 8.07637 & 14.6372 \\ 1.03653 & 7.07216 & 13.1078 \\ ... & ... & ... \\ ... & ... & ... \\ -0.0848568 & -1.87506 & -3.66526 \\ -0.530541 & -10.9695 & -21.4084 \\ 0.0329633 & -0.0848568 & 1.46577 \end{pmatrix}$$

E por fim, resolvendo os sistemas, temos a matriz seguinte, onde cada coluna é o resultado do respectivo sistema:

	X		
1	2.88155	56.36	109.834
2	-1.83376	-45.87	-89.9159
3	-1.51399	-43.49	-85.4641
4	-1.52191	-48.58	-95.6312
5	-0.453795	-30.14	-59.8266
6	5.8567	89.81	173.768
7	3.42193	48.71	94.0054
8	3.65656	59.24	114.822
9	1.20368	11.45	21.689
10	6.12534	109.16	212.2
11	-2.47971	-72.87	-143.266
12	-1.47793	-54.36	-107.248
13	-1.2039	-51.44	-101.685
14	0.128415	-25.01	-50.158
15	6.50159	98.57	190.642
16	11.4911	218.66	425.827
17	14.5805	298.40	582.22

Figura 15 – Tabela indicativa dos resultados do sistema linear da tarefa d

Deve ressaltar que, por se tratar de um sistema sobre-determinado, houve aproximações por parte do programa, o que faz com que alguns valores não fiquem tão próximos do resultado real do sistema.

2 Fatoração por matrizes não negativas

Sendo A uma matriz $n \times m$, gostaríamos de decompô-la em um produto entre duas matrizes, W, $n \times p$, e H, $p \times m$. Porém, na maioria dos casos essa decomposição exata não existe, então buscamos minimizar o erro E = || A - WH $||^2$. Ainda assim, existe o problema de possivelmente existirem diversos pontos de mínimo local para E, e existem diversos métodos para tentar contornar esse problema. O método utilizado nesse exercício programa é o método dos mínimos quadrados alternados.

2.1 Método dos mínimos quadrados alternados

O método dos mínimos quadrados alternados vem sendo amplamente utilizado em aplicações de *Machine Learning*. Seu pseudo-código fornecido no enunciado do exercício programa está apresentado na figura .

Inicialize randomicamente a matriz W com valores positivos

Armazene uma cópia da matriz A

Repita os seguintes passos até que a norma do erro se estabilize (diferença entre as normas do erro em dois passos consecutivos $< \epsilon$ (use $\epsilon = 10^{-5}$ no seu programa) ou que um número máximo de iterações ($itmax$, escolha $itmax = 100$) seja atingido

- Normalize W tal que a norma de cada uma de suas colunas seja 1 ($w_{i,j} = w_{i,j}/s_j$, com $s_j = \sqrt{\sum_{i=1}^{n} w_{i,j}^2}$)
- Resolva o problema de mínimos quadrados $WH = A$, determinando a matriz H (são m sistemas simultâneos! Cuidado, pois A é modificada no processo de solução. Na iteração seguinte deve-se usar a matriz A original novamente. Por isso armazena-se uma cópia de A!)
- Redefina H, com $h_{i,j} = \max\{0, h_{i,j}\}$
- Compute a matriz A^t (transposta da matriz A original)
- Resolva o problema de mínimos quadrados $H^t W^t = A^t$, determinando a nova matriz W^t. (são n sistemas simultâneos!)
- Compute a matriz W (transposta de W^t)
- Redefina W, com $w_{i,j} = \max\{0, w_{i,j}\}$

Fim do Repita

Figura 16 – Pseudo-código para o Método dos mínimos quadrados alternados

Para implementá-lo por meio de uma rotina em *Python* criou-se uma função que seguia o pseudo-código, utilizando a função *Solver* para múltiplos sistemas lineares quando fosse necessário resolver o problema de mínimos quadrados, e também uma função *Error*, para o cálculo do erro de cada aproximação. O código está apresentado na figura .

```python
def MMQA(A,n,m,g):
    ead = 0.00001 #diferenca maxima ente os erros de duas iterações do MMQA
    itmax =100 # número maximo de iterações admissível
    W = setting(n,g) #chama a funcao que cria a matriz randomica
    t=1 #variavel que armazena o numero da iteracao
    W = normalize(W,n,g) #normaliza a matriz aleatoria gerada
    An = np.array(A) # Cria copia de A
    At = np.transpose(An) #Armazena a transposta de A
    while t <= itmax: # Enquanto a quantidade de iteracoes for menos que o maximo estipulado
        print(t)
        h = solver(np.array(W),np.array(A),n,m,g) # Encontra h para W e A (resolve multiplos sistemas)
        # É utilizado deepcopy nessas funcoes pelo fato de alterarem as matrizes originais durante sua execucao
        for i in range(0,g): # Essa secao igual os elementos de h menores que 0 a 0
            for j in range(0,m):
                if h[i,j] < 0:
                    h[i,j] = 0
        ht = np.transpose(h) # transpoe h
        Wt = solver(np.array(ht),np.array(At),m,n,g) # Encontra Wt para ht e At
        W = np.transpose(Wt) # Encontra W a partir da transposta de Wt
        for i in range(0,n): # Zera os valores negativos de W
            for j in range(0,g):
                if W[i,j] < 0:
                    W[i,j] = 0
        W = normalize(W,n,g) # Normaliza W
        en = error(np.array(W),np.array(A),np.array(h),n,m,g) # Calcula o erro
        if t == 1: # Na primeira execucao, igual o erro antigo ao erro novo
            eant = en
            t+=1 # armazena a informacao de que se passou uma iteracao
        elif t > 1: # nas outras iteracoes, verifica se a diferenca enter os erros esta na precisao esperada
            if abs((en-eant)/en) < ead: # se estiver, termina a iteracao
                t = itmax+1
            else: # se nao estiver, igual o erro antigo ao erro novo e itera novamente
                eant = en
                t+=1
    return W
```

Figura 17 – Rotina em *Python* para o Método dos mínimos quadrados alternados

Já na figura a seguir, está apresentado o código para a função *Error*, que consiste na soma quadrada das diferenças entre os elementos da matriz A e da matriz produto entre W e H.

```python
def error(K,A,x,n,m,g): #funcao que calcula o erro quadrado do MMQA
    soma = 0
    r = np.matmul(K,x) #multiplicam-se as matrizes W e x
    e = np.zeros((n,g)) #cria-se uma matriz para armazenar o erro
    for i in range(n):
        for j in range(g):
            soma += (A[i,j] - r[i,j])*(A[i,j] - r[i,j]) #calcula o erro quadrado
    return soma
```

Figura 18 – Rotina em *Python* para o cálculo do erro entre A e WH

Como não foi explícito no enunciado o momento em que deveria ser calculado o erro, optou-se por calculá-lo após a transposição da matriz W^t. Devido ao fato de que os erros tendem a crescer conforme a matriz A cresce, optou-se, também, por utilizar a diferença relativa entre duas iterações, uma vez que ao medir a diferença absoluta, seria muito difícil obter uma resolução tão pequena quanto 10^{-5}.

2.2 Segunda tarefa

Na segunda tarefa proposta pelo enunciado do exercício programa, solicitou-se verificar que a decomposição de uma determinada matriz A era exata, comparando os

resultados com o produto WH encontrado. Na figura 15 estão apresentadas as matrizez A, W e H.

$$A = \begin{pmatrix} 3/10 & 3/5 & 0 \\ 1/2 & 0 & 1 \\ 4/10 & 4/5 & 0 \end{pmatrix}$$

$$W = \begin{pmatrix} 3/5 & 0 \\ 0 & 1 \\ 4/5 & 0 \end{pmatrix}$$

$$H = \begin{pmatrix} 1/2 & 1 & 0 \\ 1/2 & 0 & 1 \end{pmatrix}$$

Figura 19 – Matrizes A, W e H para a segunda tarefa

Quando estabelecido precisão do erro absoluta de 10^{-5}, os resultados obtidos foram (onde C equivale a matriz produto de W e H):

$$\mathbf{W_{3,2}} = \begin{pmatrix} 0.6 & 4 - 3108e - 05 \\ 5.09283e - 06 & 1 \\ 0.8 & 5.74773e - 05 \end{pmatrix}$$

$$\mathbf{H_{2,3}} = \begin{pmatrix} 0.499784 & 1 & 0 \\ 0.499985 & 0 & 1 \end{pmatrix}$$

$$\mathbf{C_{3,3}} = \begin{pmatrix} 0.299796 & 0.6 & 8.15023e - 05 \\ 0.5 & 0 & 1 \\ 0.399728 & 0.8 & 0.00010867 \end{pmatrix}$$

Percebe-se que as matrizes estão muito próximas da solução exata, porém a precisão utilizada nos cálculos as torna satisfatórias. Quando se força o método a realizar o número de iterações máximas, verifica-se que se atinge o resultado desejado.

$$\mathbf{W_{3,2}} = \begin{pmatrix} 0.6 & 0 \\ 0 & 1 \\ 0.8 & 0 \end{pmatrix}$$

$$\mathbf{H_{2,3}} = \begin{pmatrix} 0.5 & 1 & 0 \\ 0.5 & 0 & 1 \end{pmatrix}$$

Também vale notar que, devido à origem aleatória de W, algumas vezes a solução convergia para as matrizes com as linhas trocadas, no caso de W, e colunas trocadas, no caso de H, que, ainda assim, quando multiplicadas eram equivalentes a A.

$$\mathbf{W_{3,2}} = \begin{pmatrix} 0 & 0.6 \\ 1 & 0 \\ 0 & 0.8 \end{pmatrix}$$

$$\mathbf{H_{2,3}} = \begin{pmatrix} 0.5 & 0 & 1 \\ 0.5 & 1 & 0 \end{pmatrix}$$

3 Classificação de dígitos manuscritos

Uma das maiores aplicações do *Machine Learning* atualmente está relacionada com a visão computacional. Trazer às máquinas a capacidade de absorver informações por meio da visualização do mundo pode trazer grandes benefícios, possibilitando desde a leitura de dígitos manuscritos, processo que será apresentado nesse exercício programa, até o reconhecimento facial e de objetos, processos muito mais complexos que se baseiam, no fundo, na mesma ideia básica: ensinar o computador.

3.1 Treinamento de um dígito d

Para que se possa ensinar o computador, é necessário fornecer a ele uma base de dados. No caso desse exercício de simulação, foi utilizada uma base de dados extraída do banco de dados MNIST, mundialmente utilizada para esse tipo de procedimento, pré-processada pelos responsáveis pela disciplina.

Enquanto o banco de dados do MNIST consistem em milhares de imagens exemplos para cada dígito, com dimensões de 28x28 pixels. O pré-processamente realizados transformou as imagens em arquivos .txt que consistem em matrizes de dimensão $784 \times n$, onde n é a quantidade máxima de dígitos que podem ser utilizados para treinamento do computador e cada elemento da matriz representa a tonalidade da cor, onde 0 equivale ao branco e 255 ao preto.

O processo de treinamento de um dígito d se baseia na leitura do arquivo de treinamento do dígito, seleção de uma quantidade $ndig_treino$ de exemplos para a formação de uma matriz A, sobre a qual será aplicado o método dos mínimos quadrados alternados com um valor de p determinados. O processo retornará uma matriz W_d de dimensões $n \times p$ que armazenará, de certa forma, características do dígito em seus p parâmetros.

A função de treinamento utilizado no código está apresentada na figura a seguir.

```
def treino(ndig_treino,p): #Funcao que treina os Wd
    W=[] # Cria a matriz que irá armazenar os W para cada dígito
    n=784 # Define a dimensao n da matriz, sempre e784 por conta do tamanho das imagens utilizadas
    for i in range(0,10): # Para os 10 digitos
        print("Treinando digito: "+str(i))
        file = "train_dig"+str(i)+".txt" # Cria a string do nome do arquivo
        aux = np.loadtxt(file) # lê o arquivo
        aux1 = aux[:,:ndig_treino]/255 # Seleciona a quantidade de numeros que se deseja utilizar para treinar e ja divide a matriz por 255
        W.append(MMQA(aux1,n,ndig_treino,p)) # Armazena o Wd na matriz W
    return W
```

Figura 20 – Rotina em *Python* para o treinamento de dígitos

3.2 Classificação de um dígito d

Após o treinamento, é necessário verificar se a máquina é capaz de reconhecer os dígitos. Para isso, além dos arquivos para treinamento de cada um dos dígitos, é fornecido um arquivo para realização de testes dos classificadores. Esse arquivo possui 10000 exemplos de dígitos dispostos sem uma aparente ordenação. Além disso, há outro arquivo em que existe a indicação sobre qual número é cada um desses.

O processo consiste em resolver, para todos os classificadores, o sistema W_dH = A. Após isso mede-se o erro entre A e W_dH. O classificador d que minimiza esse erro é o que melhor classifica o exemplo. Dessa forma, é realizada a solução do sistema para todos W_d, a verificação dos erros e a classificação, por meio do armazenamento do dígito do classificador que o minimizou o erro em uma matriz.

A figura a seguir apresenta o código em *Python* para a realização dos testes.

```
def test(ndig_test,W,p): # Funcao que testa os classificadores criados
    n = 784
    file = "test_images.txt"
    test = np.loadtxt(file) #Carrega a matriz com os digitos a serem testados
    test = test[:,:ndig_test]/255 # Seleciona a quantidade de digitos a serem testados e ja divide a matriz por 255
    file = "test_index.txt"
    index = np.loadtxt(file) #Carrega a matriz com os digitos verdadeiros de cada numero testado
    Ct = [] # matrizes que armazenarao os erros
    for k in range(0,10): # para cada digito
        H = solver(cp.deepcopy(W[k]),cp.deepcopy(test),n,ndig_test,p) # resolve os multiplos sistemas
        R = test - np.matmul(W[k],H) # verifica o erro entre a matriz A e o produto WH
        c=[]
        for j in range(0,ndig_test):
            cj = np.sqrt(np.sum(R[:,j]**2))
            c.append(cj)
        Ct.append(c) # Armazena o erro de cada classificação
    result = []
    for j in range(0,ndig_test): # Essa secao procura a classificacao com menor erro, que teoricamente classifica corretamente o numero
        for i in range(0,10): # primeiro escolhe um dos numeros testados e depois ve qual classificador de digito o classificou com menor erro
            if i == 0:
                k = 0
                cmin = Ct[i][j]
            if Ct[i][j] < cmin:
                k = i
                cmin = Ct[i][j]
        result.append(k) # Matriz de resultados armazena o digito que obteve menor erro
    acertos = 0 # Essa secao consiste em verificar a acuracia dos classificadores
    digito = np.zeros(10) # Armazena quantas vezes o digito aparece
    digitocerto = np.zeros(10) # Armazena quantas vezes cada digito foi classificado corretamente
    for i in range(0,ndig_test): # Para cada um numero testado
        digito[int(index[i])] += 1 # Adiciona um a contagem de quantas vezes o digito apareceu (utiliza o indice de digitos conhecidos)
        if result[i] == index[i]: # Se classificou-se corretamente
            acertos +=1 # Conta-se mais um na quantidade de acertos geral
            digitocerto[int(index[i])] += 1 # Conta-se mais um na quantidade de acertos para aquele digito
    percentualtotal = acertos/ndig_test # Calcula a porcentagem total de acertos
    percentual = []
    for i in range(0,10): #Calcula a porcentagem de acertos para cada digito
        percentual.append(digitocerto[i]/digito[i])
    return result, percentualtotal, percentual, digito, digitocerto
```

Figura 21 – Rotina em *Python* para a classificação de dígitos

Vale notar, que a parte final do código faz uma avaliação sobre a acurácia dos classificadores. A variável *percentualtotal* armazena a acurácia total, ou seja, mede a razão entre a quantidade de acertos total e quantidade de dígitos testada. Já a variável *percentual* consiste em uma matriz que armazena a acurácia para cada dígito, calculada pela divisão de quantas vezes se acertou a classificação de um determinado dígito e quantas vezes ele apareceu no conjunto de teste.

3.3 Tarefa principal

A tarefa principal desse exercício programa consiste no treinamento de classificadores e a verificação de seu desempenho a partir da base de dados MNIST pré-processada pelos responsáveis pela disciplina. O objetivo da tarefa é verificar como o índice de acertos é influenciado pela quantidade de exemplos utilizado no treinamento e a quantidade p de parâmetros utilizada.

Para tanto, variaram-se os parâmetros *ndig-treino* entre 100, 1000 e 4000 e p entre 5, 10 e 15, e os classificadores gerados serviram para classificar 10000 exemplares de um arquivo de teste. Os índices de acerto obtidos estão expressos na figura a seguir.

p \ ndig_treino	100	1000	4000
5	88,63%	90,92%	91,60%
10	89,76%	92,88%	93,25%
15	90,76%	93,52%	93,82%

Figura 22 – Índice total de acertos para os 9 conjuntos de classificadores

De modo geral, é visível que o erro na classificação tende a diminuir conforme se usa uma maior quantidade de exemplos para o treinamento e uma maior quantidade de parâmetros, o que é bastante esperado. Quando se acresce a quantidade de exemplares para o treinamento, o classificador é criado baseado em uma maior quantidade de exemplos, que podem tornar o classificador cada vez mais abrangente. Já o aumento na quantidade de parâmetros reflete, de certa forma, no aumento da quantidade de características que definem os dígitos que o classificador é capaz de reconhecer.

É notável a influência da quantidade de exemplares utilizados no treinamento, chegando a diferenças de até 3,5% para p = 10, quando se compara a acurácia dos estimadores treinados com 100 e com 4000 exemplares. Ao ser apresentado à uma maior quantidade de exemplos de um dígito, mais características que definem aquele dígito são armazenadas no classificador, o que pode ajudar na sua classificação. No geral, os ganhos entre 100 e 1000 são muito maiores do que entre 1000 e 4000 devido a esse fator, ou seja, um conjunto de 1000 traz muito mais características sobre o dígito do que um conjunto de 100, porém um conjunto de 4000 não traz tantas informações novas assim em relação ao conjunto de 1000.

Vale ressaltar que nos treinos de dígito para grandes valores de p (10, 15) os parâmetros tendem mais a não atingir a convergência de 10^{-5} da diferença das normas dos erros totais requisitada na tarefa, ocorrendo para cada dígito 100 iterações cada. Na figura a seguir está representada a variação entre as normas dos erros para $ndigtreino = 100$,

$ntest = 10000$ e $p = 15$, sendo que foi omitido parte das iterações para ficar evidenciado a razoável diferença entre os erros:

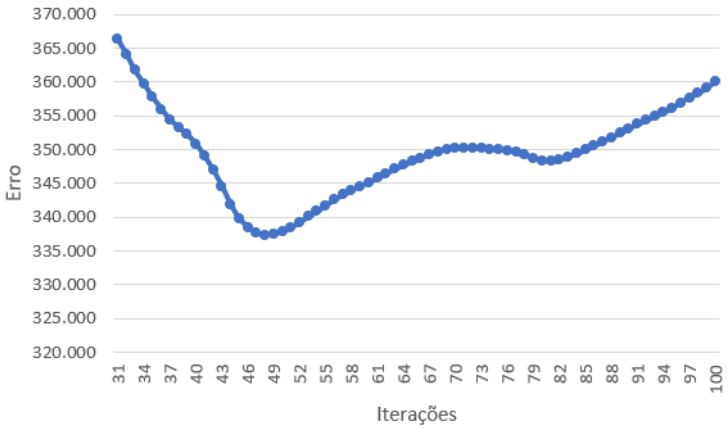

Figura 23 – Gráfico dos erros para treino do dígito 0

A figura a seguir mostra 5 exemplos do dígito 7 que possuem uma grande diferenciação. Aumentar a quantidade de dígitos para o treino faz com que o classificador tenha contato com uma maior gama de exemplos, com maior diversidade de características, e aumentar sua quantidade de parâmetros é uma forma de aumentar sua capacidade de armazenar essas características sobre os exemplares.

Figura 24 – Exemplares do dígito 7

Dessa forma, pode-se esperar que ao aumentar o número de exemplares treinados, seja requisitada uma maior quantidade de parâmetros para que o aprendizado seja eficaz. Isso se nota quando se mede a diferença do efeito do aumento dos dígitos treinados de acordo com o valor de p. Para p = 5, o aumento de 100 para 1000 da quantidade de exemplos utilizada no treino resultou num crescimento relativo de 2,5% no índice. Já com p = 10, o índice de acertos cresceu aproximadamente 3,4%. A influência da quantidade de parâmetros também é visível quando se comparam os resultados para ndigtreino = 1000, p=10 e ndigtreino = 4000, p=5. Por mais que o processo de treinamento dos classificadores no segundo caso custe mais em processamento, a acurácia dos classificadores do primeiro caso é maior. Isso pode indicar certa saturação do classificador, pois, ainda que seja apresentado a uma quantidade maior de números, ele não tem a mesma capacidade de aprender suas características.

Também é interessante avaliar o índice de acertos para cada dígito. A figura a seguir apresenta, para cada conjunto de classificador criado, o índice de acertos de cada um dos dígitos testado.

ndig_treir	100	100	100	1000	1000	1000	4000	4000	4000
p	5	10	15	5	10	15	5	10	15
dígito									
0	97,35%	97,76%	98,06%	97,65%	98,27%	98,98%	97,55%	98,67%	98,78%
1	99,38%	99,65%	99,38%	99,56%	99,47%	99,47%	99,38%	99,56%	99,12%
2	84,59%	89,73%	89,24%	88,86%	82,07%	91,96%	89,73%	91,67%	91,38%
3	85,84%	85,05%	85,35%	90,20%	91,58%	93,27%	92,08%	92,18%	92,97%
4	84,93%	84,73%	89,21%	86,25%	92,87%	92,97%	87,47%	93,48%	93,58%
5	83,86%	85,54%	84,98%	88,23%	88,34%	91,26%	87,67%	89,46%	91,48%
6	93,11%	95,51%	95,41%	94,89%	96,03%	96,56%	96,76%	97,08%	97,29%
7	87,94%	91,05%	95,14%	89,20%	90,86%	92,32%	88,91%	91,25%	92,61%
8	80,29%	78,75%	83,16%	86,55%	89,01%	89,22%	89,43%	87,06%	88,19%
9	87,31%	88,11%	87,61%	86,62%	89,49%	88,40%	85,93%	91,08%	92,07%

Figura 25 – Índices de acerto dos classificadores para cada dígito

É notável a diferença dos índices de acerto para cada um dos dígitos avaliado. Embora alguns dígitos obtenham grande índice de acertos, como é o caso do 0, 1 e o 6, também existem dígitos cuja acurácia dos classificadores é muito menor, como o 3, 5 e 8. É possível fazer algumas inferências sobre a fonte dessa diferença na acurácia da classificação, que poderia servir para a otimização do treinamento dos classificadores em um momento posterior.

Os dígitos cujo índice de acertos é maior, normalmente são bastante distintos dos outros, como é o caso do 0 e do 1, e também são escritos de poucas formas diferentes, como o 6. Ao acessar o conjunto de dados isso se torna visível. Por mais que existam diferenças na forma de escrita desses dígitos, elas são muito pequenas, o que torna sua classificação mais fácil. Essa uniformidade é evidenciada na figura a seguir.

Figura 26 – Exemplares para os dígitos 6, 0 e 1

Já nos dígitos com menor índice de acertos, normalmente sua escrita é realizada de formas muito distintas, como é o caso do dígito 4 e do dígito 2. Porém, no caso do dígito 8, pode se verificar que devido ao seu formato, existem partes que podem ser confundidas com os outros dígitos, o que também pode acarretar numa maior quantidade de erros durante a classificação.

Além disso, ao acessar à base de dados é possível encontrar certos *outliers*, ou seja, exemplos em pouca quantidade cuja escrita é realizada de forma muito distinta dos outros exemplos. É difícil para o classificador encontrar uma boa classificação para esses

casos, pois sua grande diferenciação e pouca ocorrência dificultam seu aprendizado pelos classificadores. A figura a seguir traz alguns exemplos desses exemplares.

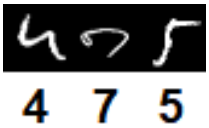

Figura 27 – Digítos com escrita fora do padrão

Os sites fornecidos no enunciado levam à índices de acertos de diversos classificadores, desde aqueles que usam árvores de decisão para a classificação, máquinas de vetores suporte e até redes neurais de alta complexidade. Um fator que muitos daqueles que possuem grande acurácia têm em comum é o fato de realizarem um pré-processamento mais avançado na base de dados que será utilizada no treinamento. Redimensionar, rotacionar e realizar outros procedimentos sobre os exemplares pode auxiliar os classificadores na missão de reconhecer padrões em cada dígito, garantindo maior acurácia na classificação.

4 Conclusão

O exercício programa além de trazer aplicações do Método Numérico para solução de sistemas (Fatoração por meio da rotação de Givens), solução de métodos dos mínimos quadrados alternados e uma boa noção sobre o trabalho com matrizes de elevada dimensão em soluções computacionais, trouxe à tona conceitos de *Machine Learning*, uma das grandes tendências da análise de dados atualmente.

Outro ponto bastante explorado foi a questão da eficiência da linguagem de programação utilizada na realização do treinamento e classificação dos dígitos. Quando se utilizaram diretamente as funções criadas em *Python*, verificou-se que o programa era executado muito lentamente, fator que foi contornado ao utilizar um módulo chamado *Numpy*, baseado na linguagem C, que facilitou a execução de processos envolvendo operações com as matrizes de elevadas dimensões utilizadas no programa.

Os resultados, no geral, foram satisfatórios, comprovando o funcionamento e apresentando as complicações dos métodos numéricos propostos para a resolução dos problemas. Na aplicação do *Machine Learning*, além de requisitar a aplicação dos métodos e a otimização do programa, uma análise do banco de dados MNIST possibilitou um bom entendimento sobre as possíveis fontes de erro.

Referências

1 MNIST. *Base de dados de dígitos escritos a mão* Disponível em:<http://yann.lecun.com/exdb/mnist/> Citado na página 5.

Parte II

Apêndices

5 Gráficos

5.1 Gráficos de erro para treino de um dígito

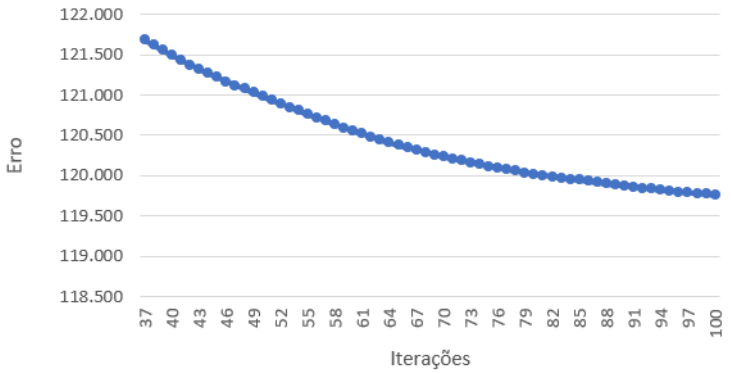

Figura 28 – ndig$_t reino = 100, ndig_t est = 10000, p = 5$

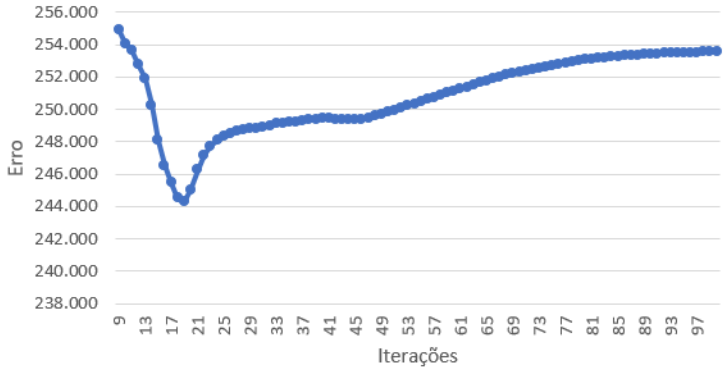

Figura 29 – ndig$_t reino = 100, ndig_t est = 10000, p = 10$

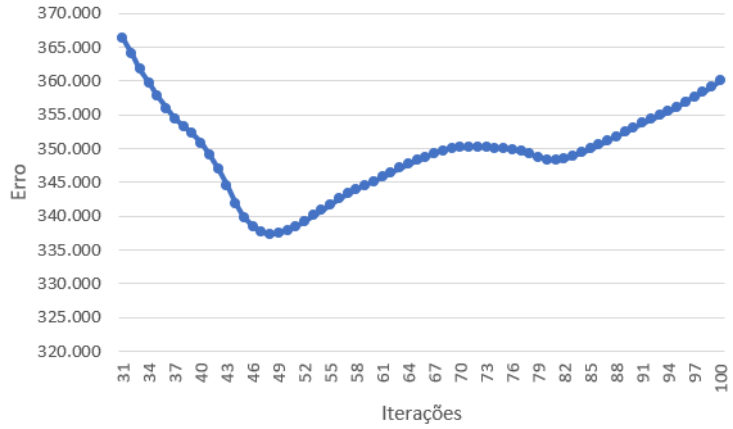

Figura 30 – ndig$_t reino = 100, ndig_t est = 10000, p = 15$

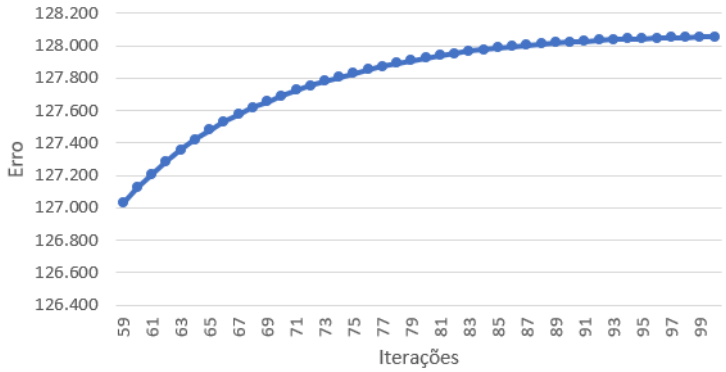

Figura 31 – $\text{ndig}_t reino = 1000, ndig_t est = 10000, p = 5$

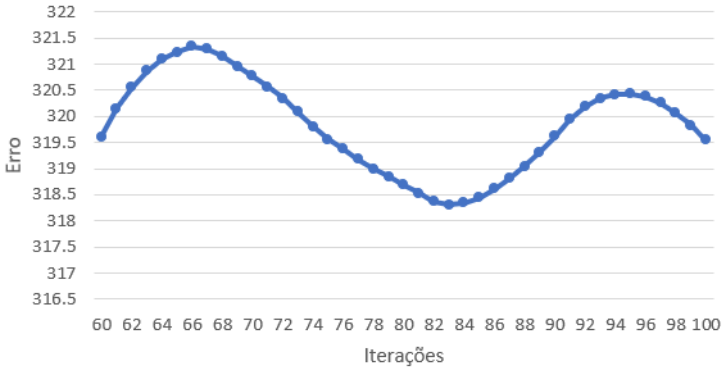

Figura 32 – $\text{ndig}_t reino = 1000, ndig_t est = 10000, p = 10$

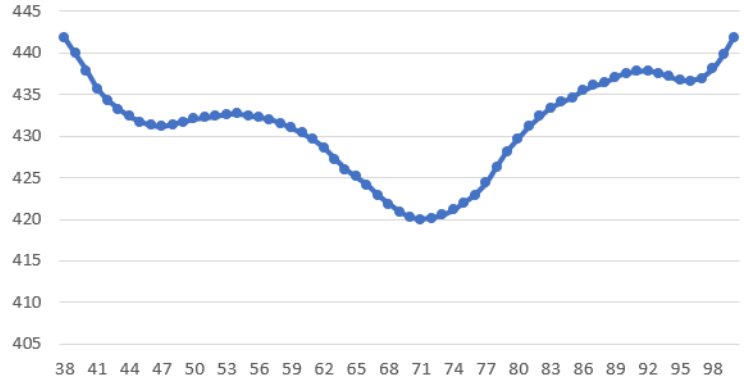

Figura 33 – $\text{ndig}_t reino = 1000, ndig_t est = 10000, p = 15$

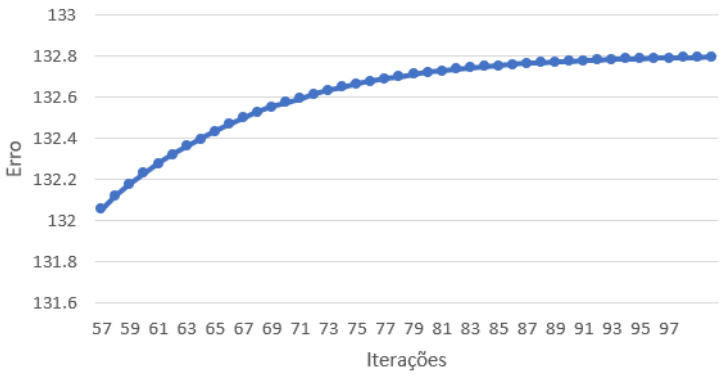

Figura 34 – ndig$_t reino = 4000, ndig_t est = 10000, p = 5$

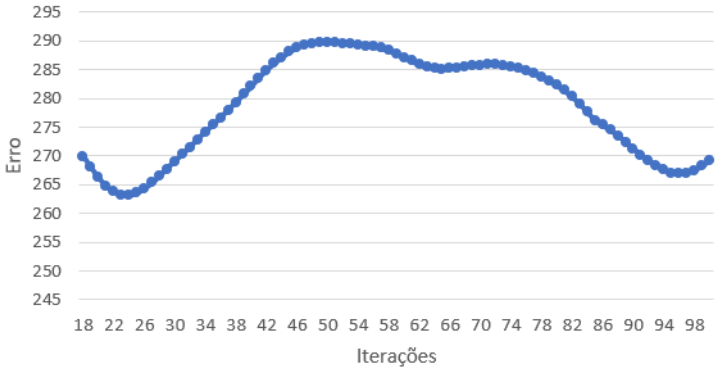

Figura 35 – ndig$_t reino = 4000, ndig_t est = 10000, p = 10$

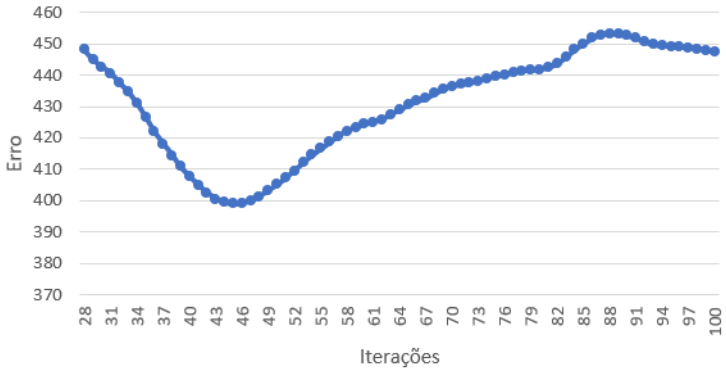

Figura 36 – ndig$_t reino = 4000, ndig_t est = 10000, p = 15$