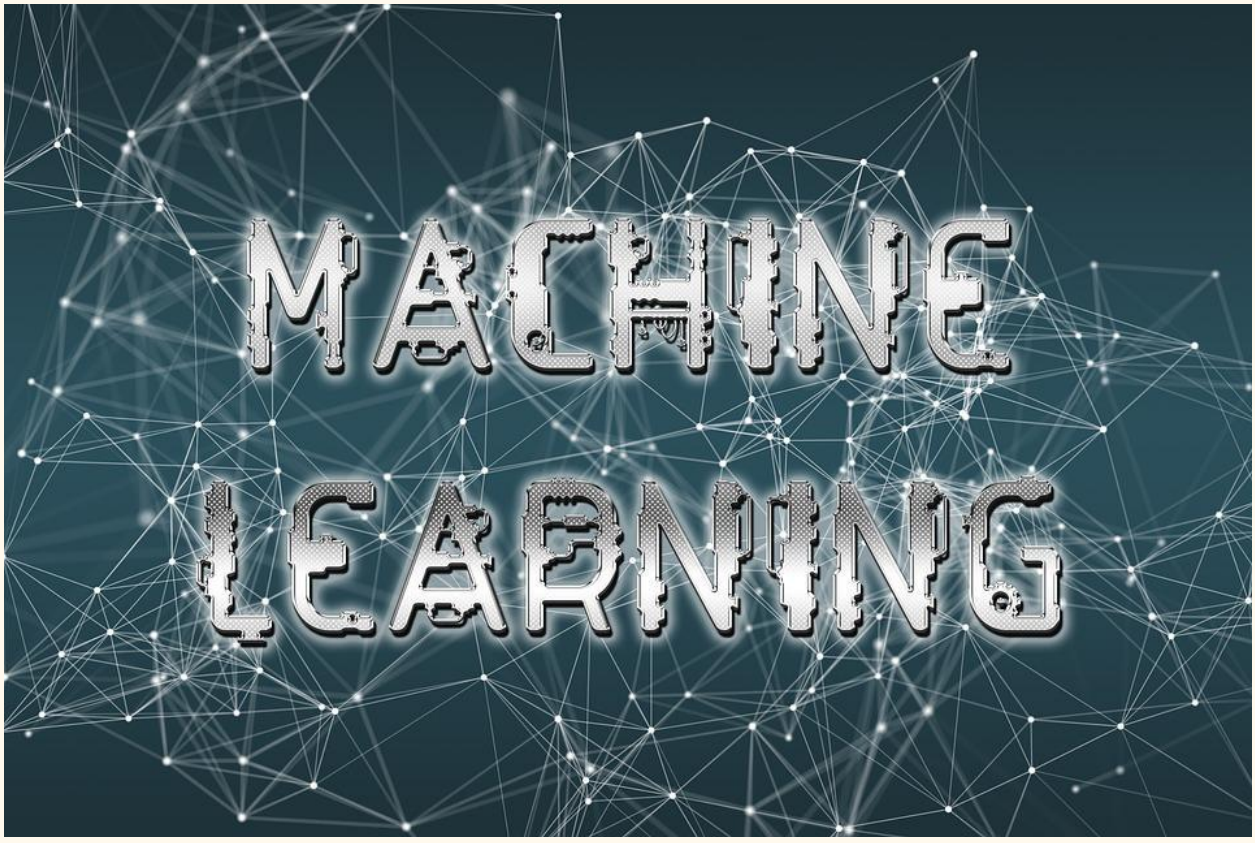
ESCOLA POLITÉCNICA DA USP

CÁLCULO NUMÉRICO 1º EXERCÍCIO PROGRAMA
NOME DO PROFESSOR: Clodoaldo Grotta Ragazzo

—

Marcos Paulo Holanda Nº USP: 9835449

Hericlis Lima dos Santos Nº USP: 9900528



São Paulo -19 de maio

1. INTRODUÇÃO

Trabalho com o intuito de aprendizagem de máquina para reconhecimento de dígitos manuscritos.

Foi utilizado a linguagem *python* na sua versão mais atualizada, 3.7.3 com a importação das bibliotecas *math, random, time e numpy*. Com a biblioteca *numpy* utilizamos apenas as funções de multiplicação e subtração de matrizes.

2. FATORAÇÃO QR E USO NA SOLUÇÃO DE SISTEMAS LINEARES

a) Rotações de givens

Aplicou-se as rotações partindo da coluna k até m-ésima coluna da matrizes A, isto é, ao mesmo tempo que aplica-se a rotação em W, aplica-se a mesma em todas as colunas de A para as linhas i e j.

Esse processo foi feito a partir da concatenação das matrizes W e A na função *QR_factorization* para minimizar o número de cálculos e poupar memória da máquina.

b) Fatoração QR

Para deixar o programa numericamente mais estável foi adotado o parâmetro τ para definição do seno e cosseno. Com isso obtivemos a triangularização de W e transformação do sistema Wx=b em Rx=Qt*b.

Ex de triangularização:

Partindo da matriz W_{10x3}:

```
[[ 1  4  1]
 [ 5  2  5]
 [ 2  3  5]
 [ 7  2  1]
 [ 7  9 81]
 [ 2  4 89]
 [21  1  4]
 [ 3  5  6]
 [ 7  3  2]
 [ 5  6  4]]
```

Figura 2.1

E A_{10x4}:

```
[[ 2  6  7  3]
 [ 2  8  4  7]
 [ 4  6  8  4]
 [ 3  2  6  8]
 [ 4  7  1  9]
 [ 6  4  7  4]
 [ 2  8  8  4]
 [ 1  9  5  7]
 [ 7  5 33  5]
 [ 3  6  7  4]]
```

Figura 2.2

Obtemos W após as rotações de Givens sendo:

```
[[ 2.56124969e+01  7.49634057e+00  3.60761390e+01]
 [-1.11022302e-16 -1.20334899e+01 -7.55858015e+01]
 [ 0.00000000e+00  0.00000000e+00 -8.71280598e+01]
 [-2.22044605e-16  0.00000000e+00  0.00000000e+00]
 [-8.88178420e-16 -4.44089210e-16 -8.88178420e-16]
 [-1.77635684e-15  0.00000000e+00 -1.77635684e-15]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]
```

Figura 2.3

E a matriz resultante H_{3x4}(no programa sai a sua transposta):

```
[[0.13623222 0.31371555 0.56803782 0.23834463]
 [0.42242263 1.11310408 1.33401257 0.95823372]
 [0.0211326  0.         0.         0.        ]]
```

Figura 2.4

Tal que $A_{10x4} = W_{10x3} x H_{3x4}$

c) Primeira tarefa

Na primeira tarefa temos a solução do sistemas Wx=b, onde A seria apenas o vetor b, e vale notar que não foi adotado o critério h(i,j)=max{0,h(i,j)}. O resultado obtido segue-se conforme figura 2.5 abaixo.

```
Olá. Seja bem-vindo ao Machine learning
Digite a tarefa(tarefa 1, tarefa 2, ML): tarefa 1
#####Teste; Rot-givens, itens a e b#####

Vetor resultante, ítem a
[[0.492 0.015 0.477 0.031 0.462 0.046 0.446 0.062 0.431 0.077 0.415 0.092
  0.4   0.108 0.385 0.123 0.369 0.138 0.354 0.154 0.338 0.169 0.323 0.185
  0.308 0.2   0.292 0.215 0.277 0.231 0.262 0.246 0.246 0.262 0.231 0.277
  0.215 0.292 0.2   0.308 0.185 0.323 0.169 0.338 0.154 0.354 0.138 0.369
  0.123 0.385 0.108 0.4   0.092 0.415 0.077 0.431 0.062 0.446 0.046 0.462
  0.031 0.477 0.015 0.492]]

Vetor resultante, ítem b
[[ 56.358 -45.875 -43.489 -48.577 -30.14   89.812  48.714  59.239  11.446
  109.163 -72.873 -54.363 -51.444 -25.015  98.572 218.659 298.4  ]]
```

Figura 2.5

Já para a solução de vários sistemas simultâneos a função *QR_factorization* utiliza o parâmetro m, isto é, resolve-se m-sistemas simultaneamente e já aplica o critério h(i,j)=max{0,h(i,j)}. O resultado segue abaixo na figura 2.6.

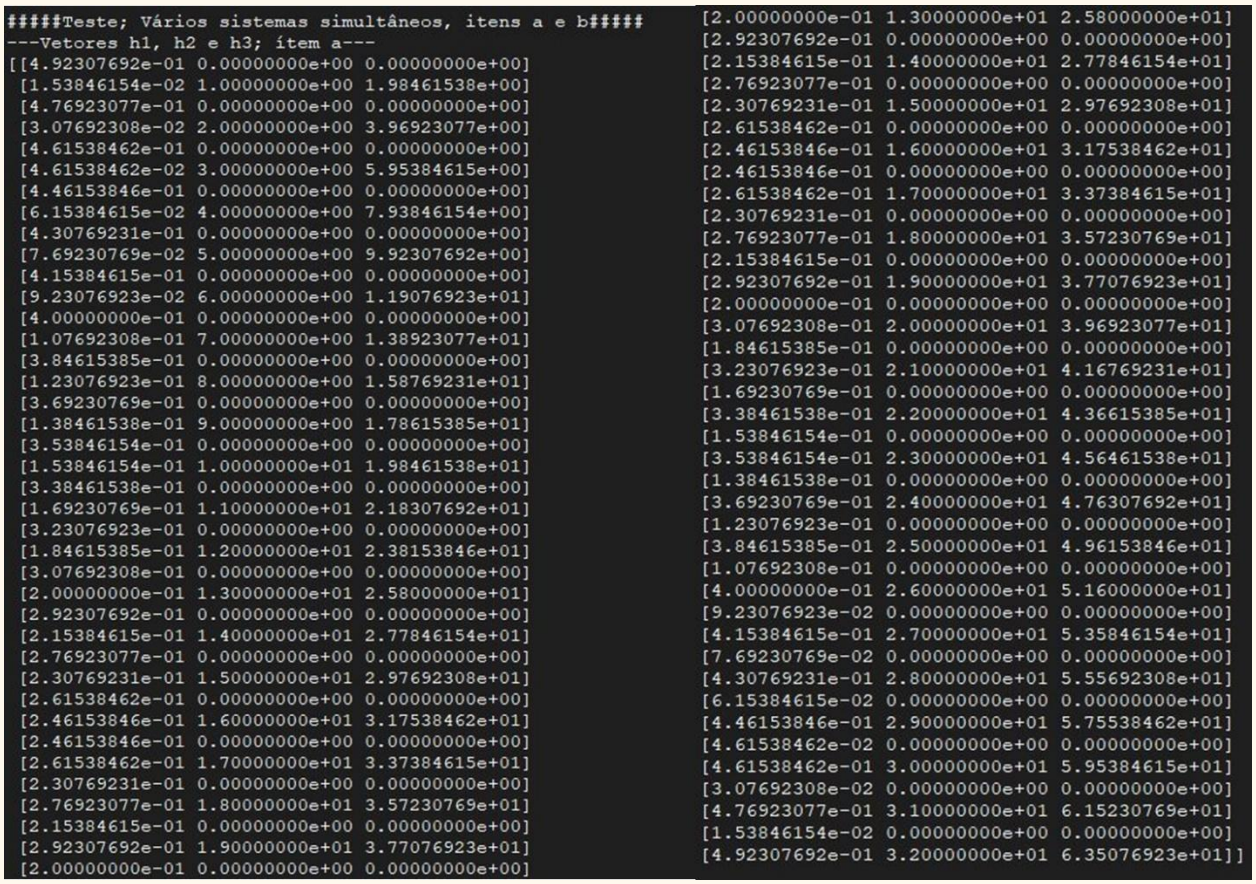


Figura 2.6

3. FATORAÇÃO POR MATRIZES NÃO NEGATIVAS

Função definida pela abreviação NMF de "*non-negative matrix factorization*"; utiliza como sub-funções primeiramente a normalização de W e posteriormente a *QR_factorization*. E, finalmente, para otimizar ainda mais a compilação do programa utiliza ao final de cada loop a função *Frobenius*.

Observe que o loop da função já começa com H transporta proveniente do retorna da *QR_factorization*, pois na mesma os acréscimos de hi's foram feito com o método '*append',* então logo em seguida define-se H aplicando a transposta da transposta.

Segunda tarefa

Obtivemos a aproximação exata da matriz. Com o retorno de W e H proveniente da função *NMF* utilizamos a multiplicação de matrizes a partir da biblioteca numpy; os elementos da matriz resultante podem aparecer em forma de exponenciais. O resultado segue abaixo na figura 3.1:

Figura 3.1

```
Aproximação exata
[[2.99999265e-01 6.00000367e-01 3.67491249e-07]
 [5.00000000e-01 0.00000000e+00 1.00000000e+00]
 [3.99999020e-01 8.00000490e-01 4.89988332e-07]]
```

4. CLASSIFICAÇÃO DE DÍGITOS MANUSCRITOS

Fase de treinamento

Para cada dígito definimos a função '*classif_Wd'* que recebe os parâmetros ndig_treino e p para treinar a máquina em relação a um dígito com a quantidade de imagens definidas por ndig_treino.

Para maior comodidade da avaliação, foi executado o programa com a função *GuardaWd* em que salvamos as matrizes Wd da fase de treinamento em arquivos txt. O tempo de execução para a fase de treinamento foi em torno de 16 horas. E obtivemos as Wd's com as dimensões para p=5, p=10 e p=15. Foram usadas 100,1000 e 4000 imagens para cada p respectivamente e esses arquivos são nomeados como no esquema: Wd_p5, Wd_p10 ou Wd_p15

Caso queira importar essas Wd salvas, apenas anula a fase de treinamento e retire as aspas que anulam a importação das mesmas.

Vale notar também que os arquivos txt train_digd exportados estavam com uma linha em branco no final do bloco de nota. Caso dê algum problema de '*index out of range*', por favor, verificar a quantidade de linhas nos blocos de nota.

Aproximações A=WH

Feito a fase de treinamento, determinamos as aproximações de A pelas Wd definidas como: WH0, WH1,...,WH9 para p igual a 5,10 e 15. Salvamos também essas aproximações em arquivos txt para conferência. Esses arquivos estão nomeados no esquema WHd_p5, WHd_p10 e WHd_p15, tal que $0 \leq d \leq 9$. O tempo de execução foi em torno de 3 dias em que usamos duas máquinas simultaneamente para definir essas aproximações.

Classificação

Para cada aproximação da matriz A de teste obtivemos a diferença ||A-WdH|| e aplicamos a função norma euclidiana para definir o vetor erro para cada dígito. Feito todos os vetores erros chama-se a função 'classificadora' que monta uma matriz(C) com esses vetores e define sua transposta(Ct), pois aí em cada linha da Ct(de comprimento 10) ela procura pelo menor valor(erro) e aplica seu método *index* e salva na variável 'ind'. Com a variável ind define-se o número da j-ésima imagem de A a partir do vetor referência 'num'.

Comparação

Feito a classificação partimos para conferência em que se usa a função 'compara'. A função tem como parâmetro um vetor proveniente da classificação que nomeamos como 'ML'. Logo em seguida a função abre o arquivo de resposta das verdadeiras classificações das j-ésimas imagens de A e simplesmente analisa se cada elemento de índice v do ML é igual a cada elemento de índice v do vetor V que possui as respostas das classificações.

Para p=5 obtivemos 13,42 % de acerto e o tempo de execução da classificação foi de 23 mins e 33 s.

```
Olá. Seja bem-vindo ao Machine learning
Digite a tarefa(tarefa 1, tarefa 2, ML): ML
Insira o número de dígitos(100,1000,4000) p/ os classificadores do ML: 100
Insira o valor de p(5,10,15):5
Insira o números de imagens a serem classificadas(10000): 10000
Fim do treinamento: 0.0
lenCt 10000 lenCt[0] 10
len-classification 10000
13.42 % de acerto
Tempo de execução 1413.0 s
>>>
```

Figura 4.1

Para p=10 e números de dígitos=1000, obtemos um 23,66% de acerto, e o um tempo de execução da classificação foi de 35 min e 43s.

Para p=15 e número de dígitos=4000, obtemos 69,90% de acerto, e foi de 50 min e 20s.

Para um p pequeno, como igual a 5, a fatoração de A não é tão representativa, e isso pode ser um dos fatores que explicam a baixa porcentagem de acerto, a medida que se aumenta o número de colunas em W e o números de dígitos, a porcentagem de acerto, consequentemente, aumenta.

5. CONCLUSÃO

O presente trabalho tornou-se interessante e ‘divertido’ pelo fato da aplicação matemática, teoria, cálculos e métodos numéricos em problema realístico e de atual pesquisa na área de ciência da computação. Com isso, houve muitos cálculos e utilização da memória da máquina e fez com que utilizássemos não apenas nossos conhecimentos em programação como também tivemos que aprimorá-los para construir um programa eficiente que utilizasse a menor quantidade de cálculos possíveis.

Um exemplo foi na implementação das rotações de givens; primeiramente tentamos construir matrizes com os coeficientes de givens e multiplicamos para cada rotação a matriz W e o vetor b, após percebemos que o programa não saia nem da 1º iteração decidimos mudar o algoritmo conforme o edital.