Escola Politécnica - USP

MAP3121 - Método Numéricos e Aplicações - 2019

Diego Jun Sato Kurashima - 10274231
Felipe Gomes de Melo D'Elia - 10340624

Exercício Programa 1

Fatoração de Matrizes para Sistemas de Classificação de Machine Learning

11 de Maio de 2019

1. Introdução

O EP1 da disciplina foi escrito em linguagem Python 3. As tarefas foram divididas em três, Primeira Tarefa, Segunda Tarefa e Tarefa Principal. Nesse relatório, as resoluções para cada tarefa serão explicadas separadamente, bem como as discussões em relação aos resultados de cada testes

2. Primeira Tarefa

A Primeira Tarefa consiste na aplicação do método de fatoração QR de matrizes para resolução de sistemas. Foram introduzidas conceitos como rotação de givens, fatoração QR e suas aplicações para resolução de sistemas lineares.

Primeiramente foram definidos funções “def calc_c (a,b)” e “def calc_s (a,b) ”, que dados dois números a e b, definem os parâmetros de cosseno e seno usados na operação de Rotação de Givens de uma matriz conforme a página 3 do enunciado e as equações (3) e (4). Posteriormente foi definida a própria Rotação de Givens como “ def rot_givens(W,n,m,i,j,c,s) “ como explícito pelo pseudo-código da página 4.

Conforme a página 3, foi indicado um exemplo de como anular um elemento de dada matriz W (5x5) a partir da Rotação de Givens. Foi criada a função “def zera_elemento (W,Wc,i,j,k)“, que zera o elemento W[j][k] a partir de seno e cosseno definido conforme uma matriz Wc. Foi testado para o exemplo e o resultado foi idêntico ao esperado conforme o mostrado no enunciado. É necessário notar que a matriz W pode ser diferente de Wc, conforme será mostrado posteriormente.

O objetivo da primeira tarefa é a solução de sistemas lineares a partir da fatoração QR. Foi inserido uma função “def fatorar_qr (W) “ que aplica sucessivas Rotações de Givens para a matriz W, tornando-a uma matriz triangular superior, conforme o pseudo-código fornecido na página 3. A função foi testada para a matriz de exemplo da mesma página e a transformação ocorreu com sucesso.

Finalmente, a função “ def resolver_sist (W, A) “ para duas entradas W e A, resolve os sistema W*x = A, a partir de fatoração QR e rotações de Givens conforme o pseudo-código da página 5. A função funciona tanto para um array A de apenas uma coluna 1 como especificado na página 4, como para sistemas simultâneos.

Para o item a) de teste a matriz W após submetida à fatoração QR apresentou valores superiores à zero para elementos abaixo da diagonal principal, apesar de ser uma matriz triangular superior. Entretanto a ordem de grandeza desses valores era desprezível (ordem de e-19) logo pode-se considerar que a fatoração ocorreu corretamente e o erro. O resultado gerado foi :

```
H =
[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]]

Para o item b) novamente a fatoração QR gerou uma matriz R com valores abaixo da matriz diagonal não nulas mas desprezíveis (ordem de e-19). A resolução da matriz é :

H = [[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]]

No item c), o teste era para resolução de sistemas simultâneos com a mesma matriz W do item a). A resposta é:

H (c) =
[[0.492 0. -0.492]
[0.015 1. 1.985]
[0.477 0. -0.477]
[0.031 2. 3.969]
[0.462 0. -0.462]

[0.046 3.   5.954]
[0.446 0.   -0.446]
[0.062 4.   7.938]
[0.431 0.   -0.431]
[0.077 5.   9.923]
[0.415 0.   -0.415]
[0.092 6.   11.908]
[0.4   0.   -0.4]
[0.108 7.   13.892]
[0.385 0.   -0.385]
[0.123 8.   15.877]
[0.369 0.   -0.369]
[0.138 9.   17.862]
[0.354 0.   -0.354]
[0.154 10.   19.846]
[0.338 0.   -0.338]
[0.169 11.   21.831]
[0.323 0.   -0.323]
[0.185 12.   23.815]
[0.308 0.   -0.308]
[0.2   13.   25.8]
[0.292 0.   -0.292]
[0.215 14.   27.785]
[0.277 0.   -0.277]
[0.231 15.   29.769]
[0.262 0.   -0.262]
[0.246 16.   31.754]
[0.246 0.   -0.246]
[0.262 17.   33.738]
[0.231 0.   -0.231]
[0.277 18.   35.723]
[0.215 0.   -0.215]
[0.292 19.   37.708]
[0.2   0.   -0.2]
[0.308 20.   39.692]
[0.185 0.   -0.185]
[0.323 21.   41.677]
[0.169 0.   -0.169]
[0.338 22.   43.662]
[0.154 0.   -0.154]
[0.354 23.   45.646]
[0.138 0.   -0.138]
[0.369 24.   47.631]
[0.123 0.   -0.123]
[0.385 25.   49.615]
[0.108 0.   -0.108]

```
 [ 0.4   26.    51.6  ]
 [ 0.092  0.    -0.092]
 [ 0.415 27.    53.585]
 [ 0.077  0.    -0.077]
 [ 0.431 28.    55.569]
 [ 0.062  0.    -0.062]
 [ 0.446 29.    57.554]
 [ 0.046  0.    -0.046]
 [ 0.462 30.    59.538]
 [ 0.031  0.    -0.031]
 [ 0.477 31.    61.523]
 [ 0.015  0.    -0.015]
 [ 0.492 32.    63.508]]
```

No item d), repetindo a execução do algoritmo obtivemos:

```
H (d)  =
[[   2.882   56.358  109.834]
 [  -1.834  -45.875  -89.916]
 [  -1.514  -43.489  -85.464]
 [  -1.522  -48.577  -95.631]
 [  -0.454  -30.14   -59.827]
 [   5.857   89.812  173.768]
 [   3.422   48.714   94.005]
 [   3.657   59.239  114.822]
 [   1.204   11.446   21.689]
 [   6.125  109.163  212.2  ]
 [  -2.48   -72.873 -143.266]
 [  -1.478  -54.363 -107.248]
 [  -1.204  -51.444 -101.685]
 [   0.128  -25.015  -50.158]
 [   6.502   98.572  190.642]
 [  11.491  218.659  425.827]
 [  14.581  298.4    582.22 ]]
```

3. Segunda Tarefa

Para a Segunda Tarefa, o objetivo era realizar a fatoração de uma matriz A em W e H não negativas. Nem sempre a fatoração obtida é exata, ou seja, há um erro e = A -WH, no cálculo. Para minimizar esse erro, é utilizado o método dos mínimos quadrados alternados conforme especifica o enunciado.

A rotina “def resolver_mmq(A, W0)” implementa o pseudo-código para fatoração por matrizes não negativas conforme a página 6. Como é necessário normalizar as colunas das

matrizes durante o algoritmo foi criado as instâncias “ def resíduo (A,W, H)” que calcula o erro quadrático das matrizes ‘A’ e ‘W’*’H’ calculados a cada iteração da fatoração, e a instância “ def normaliza (M)”, que realiza a normalização de cada coluna de ‘M’ para 1. A entrada ‘W0’ deve ser uma matriz aleatória usada como ponto de partida nas iterações da fatoração não negativa. O critério de parada das iterações ou quando a diferença do erro quadrático entre cada iteração, cujo é calculado pela função“def resíduo (A,W,H)”, ser menor a 1e-5 ou um número máximo de iterações definido em 100.

Como rotina de teste foi usado o exemplo com matrizes ‘A’, ‘W’ e ‘H’ definidos na página 6 do enunciado. ‘A’ matriz ‘W’ é a matriz inicial ‘W0’ usada para gerar a fatoração não negativa de ‘A’. Como a fatoração de ‘A’ em ‘W‘ e ‘H’ é exata não há erro quadrático associado à fatoração, além disso W já é normalizada nas colunas, logo a função executa apenas uma iteração do laço.

Se aplicarmos uma nova matriz ‘W0’ = [[12/13, 0], [5/13, 1], [0, 0]] como entrada, as saídas geradas são as mesmas matrizes ‘W’ e ‘H’ do exemplo de teste do enunciado, entretanto foram necessários 2 iterações do laço para se minimizar o erro quadrático.

Interessante notar que em alguma situações, as matrizes ‘W’’ e ‘H’’ geradas têm respectivamente suas colunas e linhas trocadas em relação ao exemplo do enunciado, o que não compromete pois a multiplicação entre essas matrizes continua com a mesma igualdade ’A’.

Para os seguintes exemplos com diferentes matrizes de entrada W0, temos os seguintes número de interações para se chegar na resposta W e H fatorados para o mesmo A:

1. W0 = [[1.0 , 0.0], [0.0. 1.0], [0.0, 0.0]] ; 1 iteração
2. W0 = [[123.0, 1.0], [456.0, 0.0], [789.0, 0.0]] ; 2 iterações (note que W0 não está normalizado na primeira coluna)
3. W0 = [[12.0, 3.0]. [5.0, 0.0], [0.0, 4.0]] ; 3 iterações (note que W0 não está normalizado para as duas colunas)
4. W0 = [[96759699.3224, -2424353.0], [2245.78, 2535.0], [0.34434, 34343.434]]; 3 iterações (note que W0 possui elemento negativo)

No geral, uma das conclusões é que a complexidade do cálculo aumenta enquanto as exigências da fatoração não negativa não são respeitadas (matrizes não negativas e normalizadas), o que fica evidente nos testes acima com o aumento de iterações para se chegar na solução da fatoração de ‘A’, que no exemplo é exata.

4. Tarefa Principal

A Tarefa Principal consiste na aplicação das funções escritas nas tarefas anteriores em um problema clássico no campo de reconhecimento de imagem, a classificação de dígitos manuscritos da base de dados MNIST. Para tal, foi feito um pré-processamento dos dados, no qual as imagens de 28x28 pixels foram convertidas em colunas de 784 elementos e concatenadas, de modo a construir uma matriz de dimensões 784xN, onde N é a quantidade de imagens utilizadas. Por fim, dividiu-se a matriz por 255, de forma que todos os valores estejam contidos no intervalo de 0 a 1.

A tarefa de reconhecimento de imagem se divide em duas etapas, uma de treinamento e outra de teste. No treinamento, constrói-se uma matriz A pela concatenação de *n_train* imagens de um mesmo dígito *d*, de forma a obter uma matriz de dimensões (784 x *n_train*). Em seguida, calcula-se iterativamente uma fatoração Wd*H = A para cada conjunto de dígitos, onde Wd e H são matrizes não-negativas. Define-se a matriz Wd com dimensões 784 x p, onde **p** é um meta parâmetro do modelo (*n_train* = 100, 1000 ou 4000; p = 5, 10 ou 15).

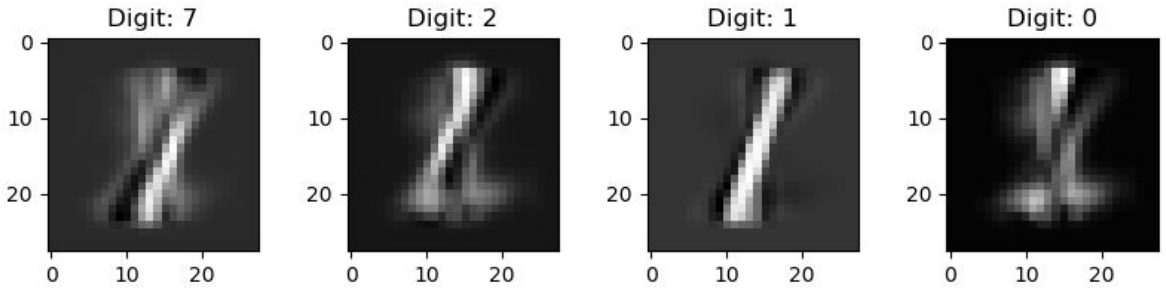
Após a obtenção de uma matriz Wd para cada um dos 10 dígitos diferentes, passamos para a etapa de teste. De maneira análoga ao treino, construiu-se uma nova matriz A, de dimensões 784 x *n_test* (*n_test* = 1000), pela concatenação de diversas imagens a serem classificadas.

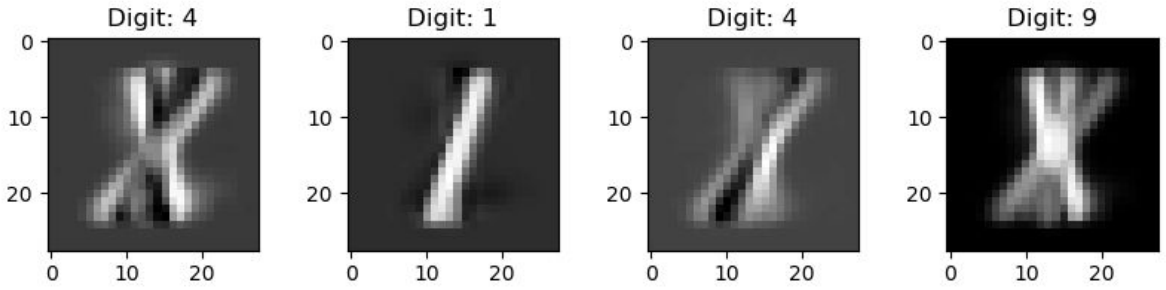
5. Resultados

Modelo treinado	**Fração de acertos para cada dígito**										
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
100-5	96.5%	100.0%	83.6%	74.8%	75.5%	75.9%	92.0%	88.9%	74.2%	89.4%	85.2%
100-10	98.8%	100.0%	89.7%	80.4%	78.2%	78.2%	93.1%	89.9%	69.7%	87.2%	86.8%
100-15	92.9%	100.0%	84.5%	79.4%	80.0%	73.6%	94.3%	90.9%	82.0%	86.2%	86.6%
1000-5	95.3%	99.2%	81.9%	90.7%	83.6%	81.6%	92.0%	90.9%	83.1%	84.0%	88.4%
1000-10	98.8%	100.0%	83.6%	87.9%	89.1%	83.9%	93.1%	90.9%	82.0%	85.1%	89.6%
1000-15	98.8%	100.0%	86.2%	89.7%	90.9%	87.4%	94.3%	96.0%	85.4%	87.2%	91.7%
4000-5	95.3%	100.0%	88.8%	92.5%	81.8%	88.5%	95.4%	91.9%	83.1%	85.1%	90.4%
4000-10	98.8%	100.0%	85.3%	91.6%	89.1%	88.5%	97.7%	91.9%	85.4%	85.1%	91.4%

Podemos perceber que tanto um aumento no número de casos de treino quanto na quantidade de linhas de W levam a um aumento sensível na taxa de acertos na classificação de novos algarismos. Porém, deve-se esperar que essa relação não se estende indefinidamente, uma vez que pode haver *overfitting* do modelo. Vale notar que a taxa de acerto para o algarismo 1 foi o mais elevado, o que pode ser indicado pela simplicidade do traço do algarismo.

Alguns exemplos de aproximação para diferentes algarismos do dataset de treino:





Como exemplificado no enunciado, quando o dígito testado corresponde ao esperado, a aproximação obtida consegue representar de maneira fiel a imagem em análise.

6. Discussão

O problema de reconhecimento de imagens possui inúmeros desafios e aplicações em diversos ramos da engenharia. Nos foi proposto um método de classificação composto por um algoritmo simples e extremamente poderoso, capaz de produzir excelentes resultados. Pode-se perceber que o trabalho extenso com matrizes de grandes dimensões pode levar a programas que requerem um maior poder de processamento e tempo de execução. Dessa forma, se mostra latente a necessidade de implementações eficientes dos algoritmos de manipulação de matrizes. A linguagem Python caracteriza uma ferramenta muito útil na implementação do algoritmo proposto, no entanto apresenta um maior tempo de execução devido ao seu caráter interpretado, em comparação com linguagens compiladas como C, C++ ou Julia. O uso da biblioteca Numpy representa um salto em desempenho do algoritmo, uma vez que recorre à chamada de funções compiladas de C.