MAP3121 – MÉTODOS NUMÉRICOS E APLICAÇÕES

EXERCÍCIO PROGRAMA 1

“FATORAÇÃO DE MATRIZES DE SISTEMAS DE CLASSIFICAÇÃO DE MACHINE LEARNING”

Turma 10

Bruno Alicino Anutto 10335170

Guilherme Donatoni Urbano 9835985

22/05/2019

Sumário

1. Introdução

Nos últimos anos, a análise de uma grande quantidade de dados vem se tornando cada vez mais importante para as pessoas como uma forma de entender melhor o mundo ao nosso redor. Foi desse interessante pensamento que surgiu a ideia de “machine learning”, que nada mais é que o uso de algoritmos para treinar uma máquina a realizar uma tarefa a partir de muitos dados e então fazer uma predição sobre como executar tal operação.

Nesse caso, foi “ensinado” à máquina a descobrir que número aparece em uma imagem a partir de métodos numéricos. Para isso, a máquina primeiramente foi treinada a partir de uma base de dígitos manuscritos MNIST, tornando-a hábil a tentar classificar se a imagem é ou não um dígito e qual seria esse. Foram simulados vários testes com diversos cenários para que seja discutido a importância de cada variável no modelo, assim como a taxa de erro associada.

O problema descrito será resolvido com Python 3.7 (com NumPy e Matplotlib) a partir da fatoração não negativa de matrizes, cujo algoritmo foi criado e testado para casos particulares, antes de ser aplicado a base de dados de dígitos. Os resultados dessas tarefas são descritos a seguir.

2. Fatoração QR de matrizes e seu uso em sistemas lineares

Para a máquina “aprender” como classificar um dígito a partir de uma imagem, é necessário fatorar a matriz A (que contém as informações do dígito) pelo produto de matrizes W e H, a partir de um método que nos permite fazer uma análise mais rápida dos dados. Para isso, foi elaborado a metodologia descrita a seguir baseada no método de fatoração QR de matrizes $n\,x\,m$, com $n \geq m$, por meio de rotações de Givens.

Primeiramente, foi desenvolvida uma função (*rotgivens)* que implementa a rotação de Givens em uma matriz, alterando os dados das linhas i e j somente. Para que seja facilitado a solução do sistema linear, é interessante que a matriz W seja triangular superior.

Para isso, foi criado uma função (*inner_fatoracao)* que realiza sucessivas vezes essa transformação para as matrizes W e b, obtendo as matrizes triangular superior (R) e $\tilde{b}$, respectivamente. É necessário aplicar essa função também para b, porque é possível trabalhar com sistemas múltiplos, onde o número de colunas de b dirá com quantos sistemas estamos lidando simultaneamente. Assim, o laço principal da fatoração foi resumida em outra função (*fatoração*).

Para resolver propriamente o sistema linear $Rx = \tilde{b}$, foi criado a função (*solve*), sendo que x é uma matriz cuja coluna representa uma solução para um dos p sistemas tratados (quando $p = 1$ resolve-se um sistema apenas). Note que essa função pode resolver até mesmo sistemas sobre determinados, isto é, quando $n > m$.

2.1Teste inicial

Para testar o algoritmo usado, foi usado esse sistema linear:

$$\begin{pmatrix} -1 & 2 & 1 \\ 2 & -4 & 1 \\ 1 & -1 & 2 \end{pmatrix} \times x = \begin{pmatrix} 1 \\ 1 \\ 2.5 \end{pmatrix}$$

Que tem como solução exata $x = \begin{pmatrix} 1 \\ 0.5 \\ 1 \end{pmatrix}$

A fatoração por meio da função *fatoracao* nos dá para o lado direito:

$$\begin{pmatrix} -2.4494897427831783 & 4.4907311951024935 & -1.224744871391589 \\ 0.0 & -0.9128709291752769 & -1.643167672515499 \\ 0.0 & 0.0 & 1.3416407864998736 \end{pmatrix}$$

E para o lado esquerdo:

$$\begin{pmatrix} -1.4288690166235205 \\ -2.0996031371031374 \\ 1.3416407864998734 \end{pmatrix}$$

Chegamos pela função *solve* a $x = \begin{pmatrix} 1.0000000000000004 \\ 0.5000000000000003 \\ 0.9999999999999999 \end{pmatrix}$ por meio do algoritmo, que é muito próxima da solução exata.

2.2 Resultados dos itens a) e b) da Primeira Tarefa

a) Nesse caso $n = m = 64$, o que significa uma matriz com um número de elementos elevado. Porém, a matriz é tri diagonal, ou seja, há vários termos nulos nos lados das três diagonais, o que diminui o número de operações a serem feitas. Chegamos a essa solução:

[0.4923076923076926,
0.015384615384614749,
0.4769230769230779,
0.030769230769229487,
0.4615384615384632,
0.04615384615384422,
0.4461538461538484,
0.061538461538459,
0.43076923076923357,
0.07692307692307393,
0.41538461538461857,
0.09230769230768886,
0.40000000000000374,
0.10769230769230376,
0.3846153846153887,
0.12307692307691893,
0.3692307692307735,
0.138461538461534,
0.3538461538461586,
0.15384615384614891,
0.3384615384615436,
0.16923076923076388,
0.32307692307692865,
0.1846153846153788,
0.3076923076923137,
0.19999999999999382,
0.2923076923076985,
0.2153846153846093,
0.27692307692308293,

0.23076923076922476,
0.2615384615384676,
0.24615384615384006,
0.24615384615385227,
0.26153846153845545,
0.23076923076923678,
0.27692307692307094,
0.21538461538462148,
0.2923076923076861,
0.2000000000000063,
0.3076923076923013,
0.18461538461539106,
0.3230769230769165,
0.16923076923077585,
0.33846153846153193,
0.15384615384616018,
0.3538461538461477,
0.1384615384615444,
0.3692307692307634,
0.12307692307692861,
0.38461538461537936,
0.10769230769231264,
0.3999999999999954,
0.09230769230769653,
0.41538461538461163,
0.0769230769230801,
0.43076923076922813,
0.06153846153846365,
0.44615384615384457,
0.046153846153847364,
0.4615384615384607,
0.03076923076923135,
0.47692307692307656,
0.015384615384615581,
0.4923076923076923]

b) Vemos que $n \geq m$, o que corresponde a um sistema sobre determinado.

[[56.35780801605245],

[-45.87484885218083],

[-43.489023741406044],

[-48.57654910203988],

[-30.140200009515077],

[89.81211894051893],

[48.71364799972669],

[59.23924714116414],

[11.446356967034486],

[109.16273558902543],

[-72.87275483205295],

[-54.3628926884665],

[-51.4444171766165],

[-25.01481463399865],

[98.57193881689835],

[218.65885869212497],

[298.4002275093944]]

2.3 Resultados dos itens c) e d) da Primeira Tarefa

c) Nesse caso, tem-se a mesma matriz W do item a), porém resolvemos 3 sistemas simultaneamente. Nota-se que alguns termos da solução têm ordem de grandeza muito pequena, devido ao fato que fazemos operações com pontos do tipo ponto flutuante (*float*).

Visto que o lado direito (b) do sistema linear possui 3 colunas, cada coluna a seguir corresponde a uma solução de um dos sistemas:

[[0.4923076923076926, -2.1945601914705892e-14, -0.49230769230768257],
[0.015384615384614749, 1.000000000000044, 1.9846153846153654],
[0.4769230769230779, -6.583143855859772e-14, -0.47692307692304786],
[0.030769230769229487, 2.0000000000000875, 3.969230769230731],
[0.4615384615384632, -1.0978878773239164e-13, -0.4615384615384157],
[0.04615384615384422, 3.000000000000132, 5.953846153846099],
[0.4461538461538484, -1.5377843508989605e-13, -0.44615384615378245],
[0.061538461538459, 4.000000000000175, 7.938461538461465],
[0.43076923076923357, -1.9715690322989252e-13, -0.4307692307691461],
[0.07692307692307393, 5.00000000000022, 9.92307692307683],
[0.41538461538461857, -2.4292963113379167e-13, -0.4153846153845133],
[0.09230769230768886, 6.000000000000266, 11.907692307692196],
[0.40000000000000374, -2.8721799576276733e-13, -0.3999999999998793],
[0.10769230769230376, 7.000000000000308, 13.892307692307563],
[0.3846153846153887, -3.283003125548596e-13, -0.38461538461524675],
[0.12307692307691893, 8.000000000000348, 15.876923076922932],
[0.3692307692307735, -3.6606571124095106e-13, -0.36923076923061326],
[0.138461538461534, 9.000000000000382, 17.861538461538295],
[0.3538461538461586, -3.9220641951027456e-13, -0.35384615384596824],
[0.15384615384614891, 10.000000000000401, 19.846153846153648],
[0.3384615384615436, -4.0484962287964823e-13, -0.338461538461325],
[0.16923076923076388, 11.000000000000405, 21.830769230769],
[0.32307692307692865, -4.038223772299239e-13, -0.3230769230766844],
[0.1846153846153788, 12.000000000000401, 23.815384615384374],
[0.3076923076923137, -3.957104241121148e-13, -0.3076923076920578],
[0.19999999999999382, 13.000000000000387, 25.79999999999975],
[0.2923076923076985, -3.771421137766305e-13, -0.2923076923074429],

[0.2153846153846093, 14.000000000000368, 27.784615384615137],

[0.27692307692308293, -3.615956740582898e-13, -0.27692307692282886],

[0.23076923076922476, 15.000000000000355, 29.76923076923052],

[0.2615384615384676, -3.491683785973353e-13, -0.26153846153820753],

[0.24615384615384006, 16.000000000000345, 31.753846153845895],

[0.24615384615385227, -3.3993510148547473e-13, -0.24615384615359084],

[0.26153846153845545, 17.000000000000334, 33.73846153846129],

[0.23076923076923678, -3.271390179402221e-13, -0.23076923076897746],

[0.27692307692307094, 18.000000000000323, 35.72307692307666],

[0.21538461538462148, -3.1419687635206067e-13, -0.21538461538434686],

[0.2923076923076861, 19.000000000000302, 37.70769230769204],

[0.2000000000000063, -2.8744244976995835e-13, -0.19999999999974613],

[0.3076923076923013, 20.000000000000266, 39.69230769230746],

[0.18461538461539106, -2.468197956782536e-13, -0.18461538461517157],

[0.3230769230769165, 21.00000000000023, 41.67692307692288],

[0.16923076923077585, -2.1288472041765076e-13, -0.16923076923058936],

[0.33846153846153193, 22.0000000000002, 43.661538461538306],

[0.15384615384616018, -1.8569085579091954e-13, -0.1538461538460248],

[0.3538461538461477, 23.000000000000174, 45.646153846153744],

[0.1384615384615444, -1.5839623859753568e-13, -0.13846153846145026],

[0.3692307692307634, 24.000000000000142, 47.63076923076915],

[0.12307692307692861, -1.2411737194185595e-13, -0.12307692307683962],

[0.38461538461537936, 25.000000000000114, 49.615384615384535],

[0.10769230769231264, -1.0355059028597546e-13, -0.10769230769221642],

[0.3999999999999954, 26.000000000000096, 51.599999999999916],

[0.09230769230769653, -8.983980198394714e-14, -0.09230769230762634],

[0.41538461538461163, 27.00000000000008, 53.584615384615326],

[0.0769230769230801, -7.609376511782226e-14, -0.07692307692303167],

[0.43076923076922813, 28.00000000000007, 55.56923076923074],

[0.06153846153846365, -6.924008862908849e-14, -0.06153846153844396],

[0.44615384615384457, 29.000000000000064, 57.553846153846145],

[0.046153846153847364, -4.850987188598887e-14, -0.04615384615383719],

[0.4615384615384607, 30.000000000000036, 59.538461538461526],

[0.03076923076923135, -2.774229510149053e-14, -0.030769230769225175],

[0.47692307692307656, 31.00000000000002, 61.52307692307692],

[0.015384615384615581, -1.3881640759758542e-14, -0.015384615384614497],

[0.4923076923076923, 32.00000000000001, 63.50769230769231]]

d) Temos a mesma matriz do item b), mas agora resolveu-se para sistemas simultâneos. Visto que o lado direito (b) do sistema linear possui 3 colunas, cada coluna a seguir corresponde a uma solução de um dos sistemas:

[[2.881550631037151, 56.35780801605245, 109.83406540106775],

[-1.8337625317224808, -45.87484885218083, -89.91593517263911],

[-1.5139890394982671, -43.489023741406044, -85.46405844331387],

[-1.5219076200074202, -48.57654910203988, -95.63119058407267],

[-0.4537946136967329, -30.140200009515077, -59.82660540533304],

[5.856698891194266, 89.81211894051893, 173.7675389898436],

[3.421928906529927, 48.71364799972669, 94.00536709292338],

[3.6565621946904665, 59.23924714116414, 114.82193208763789],

[1.2036845409770842, 11.446356967034486, 21.689029393091932],

[6.12534248834075, 109.16273558902543, 212.20012868971003],

[-2.479713963737203, -72.87275483205295, -143.26579570036867],

[-1.477931471299946, -54.3628926884665, -107.24785390563316],

[-1.2039025835760688, -51.4444171766165, -101.68493176965728],

[0.12841468982764045, -25.01481463399865, -50.158043957824255],

[6.501588998362404, 98.57193881689835, 190.64228863543414],

[11.49105599189249, 218.65885869212497, 425.82666139235727],

14.580525821156321, 298.4002275093944, 582.2199291976327]]

3. Fatoração por matrizes não negativas

Foi implementado o método dos mínimos quadrados alternados para que seja obtida a fatoração negativa: $A_{n\times m} = W_{\ n\times p} \times H_{p\times m}$, sendo p um número menor que n e m.

Criaram-se funções auxiliares as quais foram usadas para implementar a função de fatoração não negativa (*fnn*), seguindo os passos do algoritmo dado no enunciado:

- *Normaliza* – normaliza uma matriz;
- *Transposta* – retorna a transposta de uma matriz;
- *Prodmatriz* – retorna o produto entre duas matrizes;
- *Geramatriz* – retorna uma matriz com elementos gerados randomicamente;
- *Removenegativos* – troca os elementos negativos de uma matriz por zero;
- *Copia* – armazena uma cópia de uma matriz;
- *Fnn* – faz a fatoração não negativa de uma matriz.

Vale lembrar que a fatoração ocorre repetidamente até que uma das condições sejam satisfeitas: a diferença entre as normas de erros de iterações consecutivas seja menor que 10^{-5}; ou sejam realizadas 100 iterações.

3.1 Resultados da Segunda Tarefa

Na fatoração não negativa da matriz $A = \begin{pmatrix} 3/10 & 3/5 & 0 \\ 1/2 & 0 & 1 \\ 4/10 & 4/5 & 0 \end{pmatrix}$, com p = 2,

temos as matrizes W e H a seguir:

W: [[0.6000836582118317, 8.37282672238825e-05],

[0, 0.9999996496742953],

[0.8001115442824424, 0.00011163768963184335]],

□: [[0.4995814752762067, 1.0, 0],

[0.5000001751629137, 0, 1.0000003503258275]]

O teste acima chega em resultados muitos próximos as matrizes W e H do enunciado.

Repetindo o teste:

W: [[4.263527374644112e-05, 0.6000394073672827],

[1.0004868096684092, 0.0004893796044890553],

[5.684703166189665e-05, 0.800052543156377]],

H: [[0.4985335395183681, 0, 1.0000013445312612],

[0.4997892131734745, 1.0000055582528817, 0]]

Comparando os testes acima as matrizes W e H aparecem com colunas trocadas, porém sempre o produto $W \times H$ corresponde à matriz fatorada A.

4. Classificação de dígitos manuscritos

Agora, será ensinado à máquina como identificar imagens de dígitos manuscritos diferentes (de 0 a 9), a partir tanto da resolução de sistemas lineares com fatoração QR quanto fatoração não negativa de matrizes. Basicamente, há 3 etapas principais do processo: treinamento do classificador, testes com uma base de dados e avaliação dos classificadores.

Na fase de treinamento, serão desenvolvidas matrizes W_d com o que foi aprendido para cada dígito *d*, que serão usadas para classificar os dígitos, na fase seguinte. Isso foi implementado por meio da função *treino*, que calcula W_d de tamanho $n \times p$ a partir da decomposição da matriz $A_{n \times ndig\_treino}$ (do arquivo "*train_digX.txt").* Vale lembrar que *p* é escolhido e simboliza quantos "padrões comuns" de um dígito são encontrados quando se faz o treinamento de W_d. Além disso, pode-se especificar *ndig_treino*, que é a quantidade de imagens da matriz A que serão lidas.

Após o treinamento, chega-se a fase de testes, representadas pela função *teste*. Ela calcula uma matriz com os erros correspondentes a cada uma das *n_test* imagens (do arquivo *"teste_imagens.txt*") para cada dígito d, considerando as matrizes W_d já calculadas. Primeiramente, através da decomposição QR, resolve-se o sistema linear $W_d H = A$, obtendo a matriz $H_{p \times n\_test}$. Após isso é possível obter o erro c_j para cada uma das imagens com a diferença entre as matrizes: $C = A - W_d H$. Logo, para cada coluna de C, temos um erro relacionado na forma de:

$$\|c_j\| = \sqrt{\sum_{i=1}^{784} {c_{ij}}^2}$$

Ao calcular os erros por meio da função *teste,* caso o novo erro para uma determinada imagem for menor que o erro anterior, o dígito com menor erro é armazenado como o mais provável, juntamente com o esse erro. Essa é o papel da função *provavel,* que retorna os dígitos mais prováveis das imagens dados os erros da função *teste.* Inicialmente, foi escolhido que o dígito zero é o mais provável para todas e o erro seria em relação a esse mesmo dígito.

Finalmente, é possível realizar de fato os testes e avaliar o desempenho dos classificadores, o que foi feito na função *main* do algoritmo. Para avaliar a qualidade do classificador, comparou-se a lista com os dígitos mais prováveis com a o verdadeiro dígito da imagem (do arquivo *"test_index.txt"*), chegando-se a:

- Porcentagem total de acertos total
- Acertos de cada dígito (de 0 a 9)
- Porcentagem de acertos por dígito (de 0 a 9)
- Tempo de execução

4.1 Resultados da Tarefa Principal

Para cada um dos testes a seguir, foram recebidas essas saídas do programa (as listas para dígitos estão nessa ordem: [*0,1,2,3,4,5,6,7,8,9]*)

a) *n_test = 10000, ndig_treino = 100, p = 5*

Porcentagem de acertos = 87.8%
Acertos para cada dígito = [943, 1128, 878, 860, 792, 741, 901, 898, 771, 868]
Ocorrências para cada dígito = [1028, 1245, 934, 917, 926, 882, 1007, 1011, 916, 1134]
Porcentagem de acertos para cada dígito = [91.73151750972762, 90.60240963855422, 94.00428265524626, 93.78407851690294, 85.52915766738661, 84.01360544217687, 89.47368421052632, 88.82294757665677, 84.17030567685589, 76.54320987654322]
Tempo de execução (min): 12.656015368302663

b) *n_test = 10000, ndig_treino = 100, p = 10*

Porcentagem de acertos = 90.01%
Acertos para cada dígito = [956, 1129, 927, 874, 853, 770, 912, 932, 771, 877]
Ocorrências para cada dígito = [1028, 1226, 974, 938, 956, 925, 978, 1063, 857, 1055]
Porcentagem de acertos para cada dígito = [92.99610894941634, 92.08809135399673, 95.17453798767967, 93.17697228144989, 89.22594142259415, 83.24324324324324, 93.25153374233129, 87.6763875823142, 89.96499416569428, 83.12796208530806]
Tempo de execução (min): 20.03337154388428

c) *n_test = 10000, ndig_treino = 100, p = 15*

Porcentagem de acertos = 90.82%
Acertos para cada dígito = [962, 1127, 937, 858, 857, 757, 912, 965, 808, 899]
Ocorrências para cada dígito = [1031, 1213, 995, 908, 914, 885, 965, 1072, 934, 1083]
Porcentagem de acertos para cada dígito = [93.3074684772066, 92.91014014839241, 94.17085427135679, 94.49339207048457, 93.76367614879649, 85.53672316384181, 94.50777202072538, 90.01865671641791, 86.50963597430408, 83.0101569713758]
Tempo de execução (min): 30.103134353955586

d) *n_test = 10000, ndig_treino = 1000, p = 5*

Porcentagem de acertos = 90.77%
Acertos para cada dígito = [954, 1126, 920, 923, 829, 778, 920, 914, 830, 883]
Ocorrências para cada dígito = [1049, 1204, 969, 997, 918, 872, 989, 1017, 939, 1046]
Porcentagem de acertos para cada dígito = [90.94375595805529, 93.52159468438538, 94.94324045407637, 92.5777331995988, 90.30501089324619, 89.22018348623853, 93.02325581395348, 89.87217305801377, 88.39190628328008, 84.4168260038241]
Tempo de execucção (min): 83.11506400108337

e) *n_test = 10000, ndig_treino = 1000, p = 10*

Porcentagem de acertos = 92.84%
Acertos para cada dígito = [962, 1131, 924, 921, 905, 793, 926, 939, 866, 917]
Ocorrências para cada dígito = [1033, 1224, 972, 994, 954, 848, 972, 1003, 971, 1029]
Porcentagem de acertos para cada dígito = [93.12681510164569, 92.40196078431373, 95.06172839506173, 92.65593561368209, 94.86373165618448, 93.51415094339623, 95.26748971193416, 93.61914257228315, 89.18640576725026, 89.1156462585034]
Tempo de execução (min): 134.4604331533114

f) *n_test = 10000, ndig_treino = 1000, p = 15*

Porcentagem de acertos = 93.71%

Acertos para cada dígito = [965, 1128, 947, 930, 920, 803, 925, 954, 873, 926]
Ocorrências para cada dígito = [1023, 1185, 1001, 994, 966, 842, 963, 1027, 969, 1030]
Porcentagem de acertos para cada dígito = [94.54631510164569, 95.12336078431373, 94.91472839506173, 93.67967561368209, 95.71128165618448, 95.51415684339623, 96.26744578193416, 92.619147391228315, 90.1864046525026, 89.9158586585034]
Tempo de execução (min): 180.4604331533114

g) *n_test = 10000, ndig_treino = 4000, p = 5*

Porcentagem de acertos = 91.68%
Acertos para cada dígito = [957, 1127, 929, 940, 851, 787, 925, 921, 854, 877]
Ocorrências para cada dígito = [1033, 1192, 974, 1021, 961, 859, 986, 993, 941, 1040]
Porcentagem de acertos para cada dígito = [92.64278799612778, 94.54697986577182, 95.37987679671458, 92.0666013712047, 88.55359001040583, 91.61816065192083, 93.8133874239351, 92.74924471299094, 90.75451647183847, 84.32692307692308]
Tempo de execução (min): 274.62271860440575

h) *n_test = 10000, ndig_treino = 4000, p = 10*

Porcentagem de acertos = 93.69%
Acertos para cada dígito = [964, 1131, 945, 937, 914, 810, 929, 957, 855, 927]
Ocorrências para cada dígito = [1032, 1200, 982, 1014, 964, 862, 970, 1017, 937, 1022]
Porcentagem de acertos para cada dígito = [93.41567127659575, 94.252, 96.23991967871486, 92.40518371400198, 94.25231899265477, 93.32523364485982, 95.3474661105318, 94.54340594059406, 91.89756359875905, 90.25782524271844]
Tempo de execucção (min): 547.5796846023788

i) *n_test = 10000, ndig_treino = 4000, p = 15*

Porcentagem de acertos = 93.75%

Acertos para cada dígito = [968, 1124, 946, 942, 919, 804, 928, 950, 869, 925]
Ocorrências para cada dígito = [1034, 1188, 996, 1007, 953, 856, 959, 1010, 967, 1030]
Porcentagem de acertos para cada dígito = [93.61702127659575, 94.61279461279462, 94.97991967871486, 93.54518371400198, 96.43231899265477, 93.92523364485982, 96.7674661105318, 94.05940594059406, 89.86556359875905, 89.80582524271844]
Tempo de execucção (min): 747.5969856023788

5. Discussão dos resultados

Após essas simulações, é possível entender melhor como a máquina "aprende" a classificar os dígitos em função das variáveis que podemos escolher. Nota-se que, pelo tempo de execução dos testes, o número de imagens analisadas apresenta uma quantidade significativa de dados para uma máquina computacional. Para os testes finais, o tempo de o programa fornecer as saídas aumentou demasiadamente.

Verifica-se que quanto maior o valor de *p* (testes *a,b* e *c*), maior a porcentagem de acertos dos classificadores. Isso se deve ao fato de que estamos trabalhando com cada vez mais "padrões" para um dígito, que são armazenados em cada coluna das matrizes do tipo W_d. Logo, é de se esperar que o erro em relação ao dígito verdadeiro diminua, sendo mais fácil o acerto do classificador.

Além disso, observou-se que aumentando o tamanho do conjunto de treinamento (testes *a* e *d*), isto é, para números *ndig_treino* maiores, a porcentagem de acertos também aumentou. Isso porque os "padrões" dos classificadores são mais robustos, vide o fato de os classificadores serem treinados com mais variações de um determinado dígito e, logo, os erros devem diminuir.

Analisando conjuntamente os testes, observa-se que o dígito 1 foi entre todos o que gerou um maior acerto da máquina, vide o fato de que sua imagem é bem simples (quase um traço reto) e fácil de distinguir dos outros dígitos. Já o dígito 5 foi o que obteve menos acertos, já que a

imagem desse dígito tem uma fisionomia semelhante a outros dígitos, como o dígito 6, gerando mais erros.

De modo geral, o exercício proporcionou grande aprendizado para o grupo e revelou uma aplicação importante dos assuntos vistos na aula. A ideia de “ensinar” máquinas a partir de treinamentos com algoritmos é extremamente presente na atualidade e relacionada aos conceitos de cálculo numérico.