Escola de Ciências da Universidade do Minho
Departamento de Informática
Mestrado em Matemática e Computação
Mestrado Integrado em Engenharia Informática

Multi-logistic Classifiers versus Softmax

Cecília Coelho, PG37016
Fernando Arraz, PG37014
João Alves, PG22591
Joel Morais, A70841
Tiago Fraga, A74092

2 de junho de 2019

Conteúdo

1 Resumo

A função *softmax* é usada em problemas de classificação multinomial. O objetivo deste trabalho foi o estudo de outras estratégias de classificação com o intuito de usar outras técnicas de classificação além do *softmax*. Para isto, foram desenvolvidos códigos em *Python* das abordagens *One vs All*, *One vs One* e Dicotómica.Todas as estratégias foram avaliadas utilizando o *dataset* de números manuscritos (digits) com 1797 imagens com dimensões 8x8 e um vetor com 64 *features*, que caracteriza cada imagem, da biblioteca *scikit-learn*.

Este relatório está dividido em 3 partes: primeiramente é feita uma introdução teórica às abordagens em estudo; de seguida são apresentados os resultados obtidos em termos de precisão e velocidade de processamento; por último são apresentadas as conclusões finais.

2 Introdução

A classificação multiclasse refere-se a problemas em que existem mais de dois valores possíveis como label da classe. Para a resolução destes problemas foram estudadas 4 abordagens diferentes:

One vs All

A estratégia *One vs All* (OVA) permite transformar problemas de classificação multiclasse em binários. Dado um problema de classificação com N possíveis soluções, a OVA utiliza um classificador binário por cada *label*, ou seja, L classificadores. Cada classificador prevê se um *label* pertence a um dado valor da classe [1]. O algoritmo desta abordagem é dado por:

- For t=(1:L)
- Amostras positivas se $xi : t \in yi$
- Amostras negativas se $xi : t \notin yi$
- $f_t(x)$: decisão do classificador t (um valor de f_t grande indica uma elevada probabilidade de x pertencer à classe t)
- Previsão: $f(x) = arg\max_t f_t(x)$

One vs One

A estratégia *One vs One* (OVO) utiliza $L(L-1)/2$ classificadores binários para um problema com L classes. Cada um recebe um par de classes que corresponde a todas as combinações do número de classes duas a duas. Se a *label* estiver presente num par, este é identificado com um 1. A previsão é feita através da contagem da incidência dos elementos em cada um dos classificadores [1].

- For t=(1:L(L-1))
- Amostras positivas se $xi : s \in yi$
- Amostras positivas se $xi : t \in yi$
- $f_(t, s)(x)$: se $f_(t, s)(x)$ for elevado então a classe s tem maior probabilidade do que a classe t
- $f_(t, s)(x) = -f_(s, t)(x)$
- Previsão é dada por: $f(x) arg\max_s(\sum_t f_(s, t)(x))$

Dicotómica

A estratégia dicotómica utiliza, no mínimo, $\log_2 L$ classificadores binários para um problema com L classes. Por exemplo, para classificar 8 dígitos são necessários 3 classificadores, sendo que os dígitos são agrupados em 2 subconjuntos cada, a união destes subconjuntos é necessariamente o conjunto original. Generalizando, dado um conjunto D, é preciso utilizar pelo menos $\log_2 L$ classificadores, podendo conter diferentes partições e combinações de valores da classe, e ao ser classificado, o valor da classe vencedora será o que obtiver a maior incidência de um determinado *label* da classe. Desta forma, pode-se testar diferentes partições e combinações a fim de hierarquizar a aplicação de técnicas para classificação, a depender da qualidade de previsão dos classificadores e da dificuldade em classificar certos elementos do *dataset*.

A vantagem desta técnica é que há divisões/agrupamentos de classes de forma a aumentar a chance de um *label* pertencer a uma certa partição, além de reduzir a quantidade de testes, comparando com outras técnicas de classificação. Outra vantagem desta técnica é ter a certeza de que um certo elemento estará dentro de uma partição com mais de um elemento, que combinando ao contexto pode ser suficiente para uma correta classificação. Por exemplo, se numa frase/palavra uma certa letra não é bem classificada individualmente, mas bem classificada numa certa divisão, a depender da palavra/frase, a letra que fizer mais sentido entre as demais será a letra escolhida a completar a palavra/frase.

Softmax

A função *Softmax* designa probabilidades a cada classe de um problema multinomial, a soma destas probabilidades é igual a 1. A equação é representada por:

$$\sigma s_j = \frac{e^{s_j}}{\sum_{j=1}^{J} e^{s_j}}$$

A vantagem de usar a *Softmax* é que, uma rede que utilize esta função, quanto maior for o valor de uma dada classe, menores serão os valores de todas as outras. Esta estratégia encoraja a rede a prevêr um output com uma probabilidade elevada [2].

3 Resultados

Nesta primeira abordagem foram usados apenas 4 números para uma primeira colecção de resultados. Os códigos desenvolvidos permitem a escolha da quantidade de números a utilizar e também a escolha dos próprios números. Todas as funções foram inicialmente testadas com 4 números: pouco semelhantes; muito semelhantes; sequenciais e em seguida com os 10 números presentes no *dataset*. Os testes para o *dataset* com todos os dígitos encontram-se nesta secção. Os outros 3 testes e os seus respetivos resultados encontram-se em anexo.

3.1 *One vs All*

- **10 números:** Foram utilizadas 1348 entradas para treino e 449 para teste. Esta função demorou 8.13 segundos e obteve uma precisão de aproximadamente 0.96. Na figura 1 está representada a matriz de confusão.

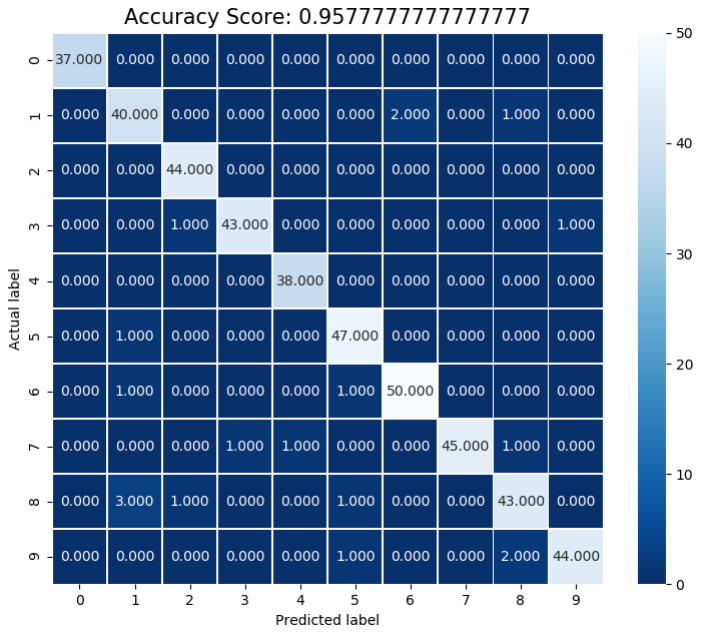

Figura 1: Matriz de confusão *One vs All* para todos os dígitos do *dataset*

Através da tabela de confusão podemos verificar que, para 10 números, o *One vs All* apresenta uma precisão aceitável. O número em que houve mais classificações erradas foi o 8.

3.2 *One vs One*

- **10 números:** Foram utilizadas 1348 entradas para treino e 449 para teste. Esta função demorou 0.13 segundos e obteve uma precisão de aproximadamente 0.95. Na figura 4 está representada a matriz de confusão.

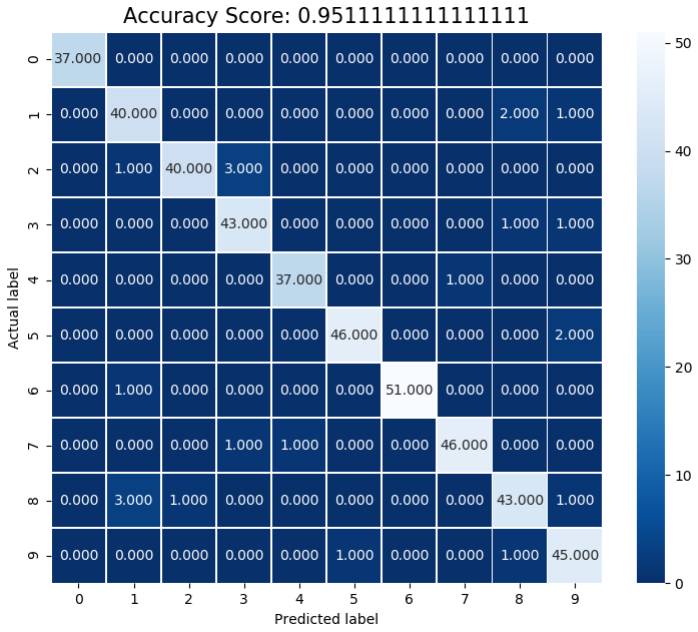

Figura 2: Matriz de confusão *Softmax* para todos os dígitos do *dataset*

Através da tabela de confusão podemos verificar que, para 10 números, o *Softmax* apresenta uma precisão aceitável. O número em que houve mais classificações erradas foi o 8.

3.3 *Dicotómica*

- **10 números:** Foram utilizadas 1348 entradas para treino e 449 para teste. Esta função demorou 4.96 segundos e obteve uma precisão de aproximadamente 0.93. Na figura 3 está representada a matriz de confusão.

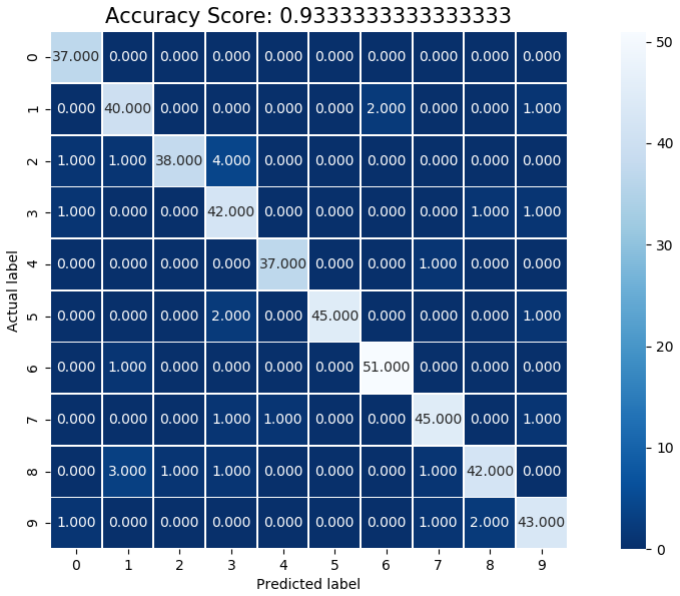

Figura 3: Matriz de confusão *Dicotómica* para todos os dígitos do *dataset*

Através da tabela de confusão podemos verificar que, para 10 números, o *Dicotómica* apresenta uma precisão aceitável. O número em que houve mais classificações erradas foi o 8.

3.4 *Softmax*

- **10 números:** Foram utilizadas 1348 entradas para treino e 449 para teste. Esta função demorou 0.18 segundos e obteve uma precisão de 0.95. Na figura 4 está representada a matriz de confusão.

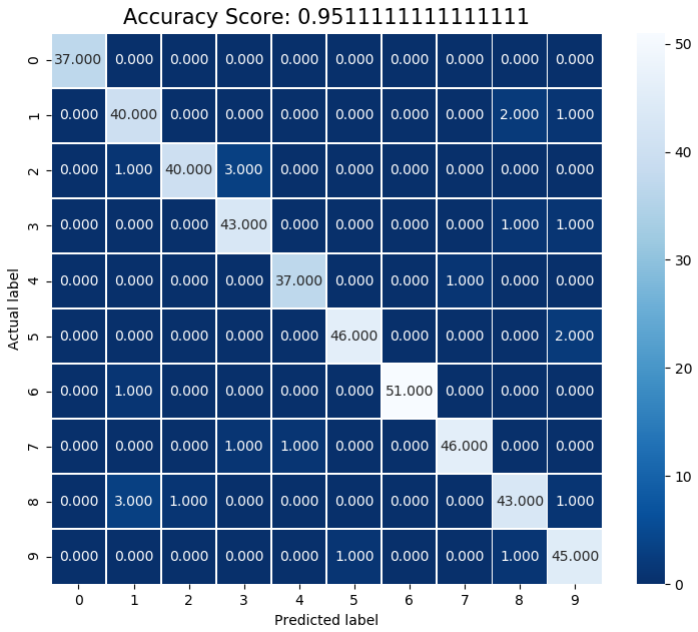

Figura 4: Matriz de confusão *Softmax* para todos os dígitos do *dataset*

Através da tabela de confusão podemos verificar que, para 10 números, o *Softmax* apresenta uma precisão aceitável. O número em que houve mais classificações erradas foi o 8.

4 Conclusão

Neste trabalho foram estudadas 4 técnicas diferentes: *One vs One*, *One vs All*, *Dicotómica* e *Softmax*. O objetivo foi comparar a precisão e tempo de cada uma de forma a concluir qual será a melhor escolha na presença de um *dataset* multinomial. O *dataset* escolhido foi o *digits* da biblioteca sklearn. Nos próximos parágrafos são discutidos os resultados obtidos em todos os testes (incluindo os que estão em anexo), assim como as conclusões retiradas a partir destes. Todos os resultados estão resumidos nas tabelas 1 e 2.

Para o teste de 4 números sequenciais, as técnicas obteram precisões equivalentes, sendo que a *One vs One* e a *One vs All* correm no mesmo tempo. Apesar disto, o OVA tem dificuldade na classificação do número 3 enquanto que o OVO erra também no número 2. O *Softmax* obteve uma precisão equivalente aos últimos 2 mas a computação demorou apenas 0.0625 segundos. No caso da aproximação dicotómica, foi possível ter uma precisão aproximadamente igual à das outras técnicas mas em apenas 0.04 segundos. Posto isto, podemos concluir que, para 4 números sequenciais, a melhor técnica é a dicotómica.

Para o teste de 4 números semelhantes, a OVA foi a que obteve a pior precisão(96.20%). O OVO, no mesmo tempo que a OVA (1.82 segundos), conseguiu obter uma precisão de 99.46%. O *One vs One* apenas classificou incorretamente um 6, enquanto que a *One vs All* teve dificuldades nos números 8 e 9. O *Softmax* conseguiu uma precisão igual á do OVO mas em apenas 0.0625 segundos, sendo que classificou de forma incorreta um número 6. A dicotómica, apesar de obter uma precisão abaixo da OVO e *Softmax*, demora metade do tempo deste último.

Para o teste de 4 números pouco semelhantes, o OVO obtêm uma precisão menos elevada do que o OVA mas demora menos 0.19 segundos. Este classifica incorretamente 3 números diferentes enquanto que o OVA apenas errou ao classificar um 1. É de referenciar os 100% de precisão obtidos pelo *Softmax* em apenas 0.06 segundos. Comparando com a dicotómica, por menos 0.02 segundos perde-se aproximadamente 6% de precisão. A dicotómica apresenta a mesma dificuldade do que a OVO na classificação do número 2.

Para o teste com todos os 10 números presentes no *dataset*, o OVO foi o que obteve maior precisão (96.67%) seguido da OVA e *Softmax* com resultados muito semelhantes e por último a dicotómica (93.33%). Neste caso e contrastando com os anteriores, o *Softmax* é o mais rápido com uma diferença de aproximadamente 3 segundos para a segunda técnica mais rápida (dicotómica).Em todas as técnica, o número em que houve mais classificações erradas foi o 8.

Para concluir, as técnicas OVO e OVA, à medida que é aumentada a quantidade de *labels* aumenta consideravelmente o tempo de execução. Isto deve-se aos testes das combinações dos valores da classe para a escolha dos classificadores. Já a abordagem dicotómica aumenta o tempo numa taxa menor de crescimento por não verificar todas as combinações testadas pelo OVO e OVA. A dicotómica seria a técnica mais vantajosa a usar para 4 números sequenciais ou semelhantes pois obtêm uma precisão muito bom em aproximadamente metade do tempo do *Softmax*. No caso de 10 números, o *Softmax* é o mais vantajoso pois apresenta uma precisão muito boa com um tempo de computação baixissimo. Naturalmente é de se esperar que o uso do *Softmax* seja a primeira escolha para problemas multiclasses/multinomiais mas, a utilização de classificadores logísticos com agrupamentos adequados (tamanho da partições vs combinações de labels) em *datasets* com um elevado número de amostras e de labels, pode trazer resultados satisfatórios tendo em conta o reduzido tempo computacional.

Como trabalho futuro, estas técnicas poderiam ser testadas *datasets* de maior dimensão, tanto em número de amostras como de labels, com a finalidade de avaliar em que circunstâncias é melhor usar um *Multiclass Logistic* em detrimento do *Softmax* de forma a melhorar os resultados de precisão mas também de custo computacional.

	Accuracy (%)			
Label	OVA	OVO	Softmax	Dicotomica (Tree)
4 números sequenciais (0 1 2 3)	98,82	98,82	98,81	98,22
4 números semelhantes (0 6 8 9)	96,20	99,46	99,46	98,91
4 números pouco semelhantes (0 1 5 2)	99,42	98,26	100,0	94,93
10 números (0 a 9)	95,78	96,67	95,11	93,33

Tabela 1: Tabela com as percentagens de precisão das várias técnicas para todos os testes realizados.

	Time (seconds)			
Label	OVA	OVO	Softmax	Dicotomica (Tree)
4 números sequenciais (0 1 2 3)	1.82	1.82	0.0625	0.03
4 números semelhantes (0 6 8 9)	1.90	1.90	0.0625	0.04
4 números pouco semelhantes (0 1 5 2)	1.71	1.90	0.06	0.04
10 números (0 a 9)	13.15	8.13	0.13	3.79

Tabela 2: Tabela com os tempos das várias técnicas para todos os testes realizados.

5 Anexo

5.1 *One vs All*

- **4 números sequenciais:** Os números utilizados foram o 0, 1, 2 e 3. Foram utilizadas 551 entradas para treino e 169 para teste. Esta função demorou 1.90 segundos e obteve uma precisão de aproximadamente 0.99. A matriz de confusão está representada na figura 5.

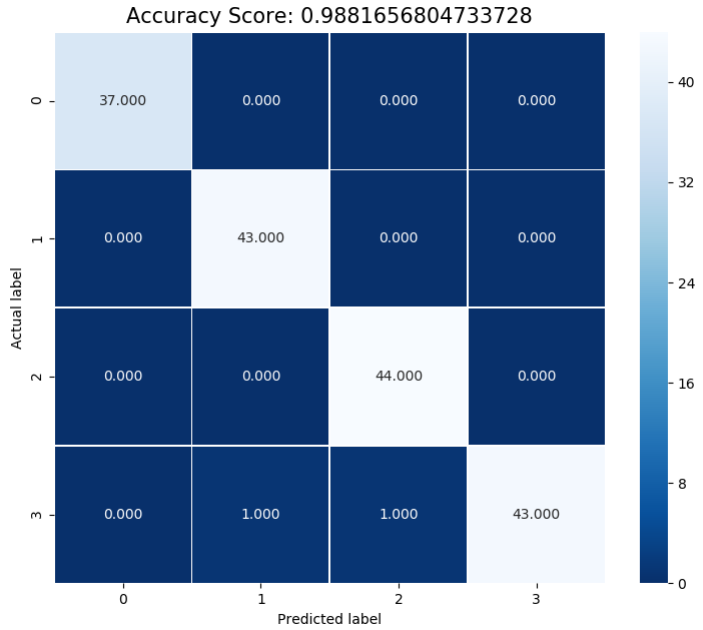

Figura 5: Matriz de confusão *One vs All* para os dígitos 0,1,2 e 3.

Através da tabela de confusão podemos verificar que esta função classificou de forma incorreta o número 3, 2 vezes.

- **4 números semelhantes:** Os números escolhidos foram o 0, 6, 8 e 9. Foram utilizadas 529 entradas para treino e 184 para teste. Esta função demorou 1.82 segundos e obteve uma precisão de 0.96. Na figura 6 está representada a matriz de confusão.

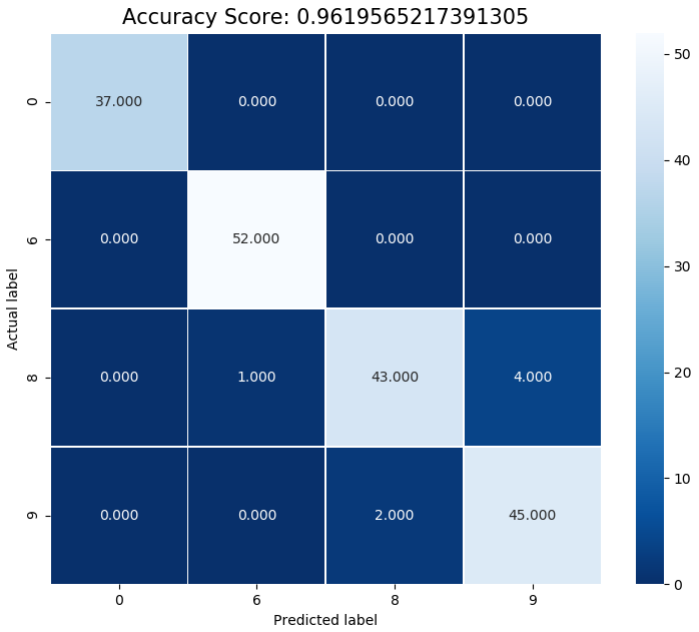

Figura 6: Matriz de confusão *One vs All* para os dígitos 0,6,8 e 9.

Através da figura 6 podemos verificar que esta função apresenta maior dificuldade na classificação do número 8, sendo que este é classificado 4 vezes como sendo um 9 e 1 vez como sendo um 8. Também o número 9 é classificado de forma incorreta 2 vezes como sendo um 8.

- **4 números pouco semelhantes:** Os números escolhidos foram o 0, 1, 5 e 2. Foram utilizadas 547 entradas para treino e 172 para teste. Esta função demorou 1.90 segundos e obteve uma precisão de aproximadamente 0.99. Na figura 7 está representada a matriz de confusão.

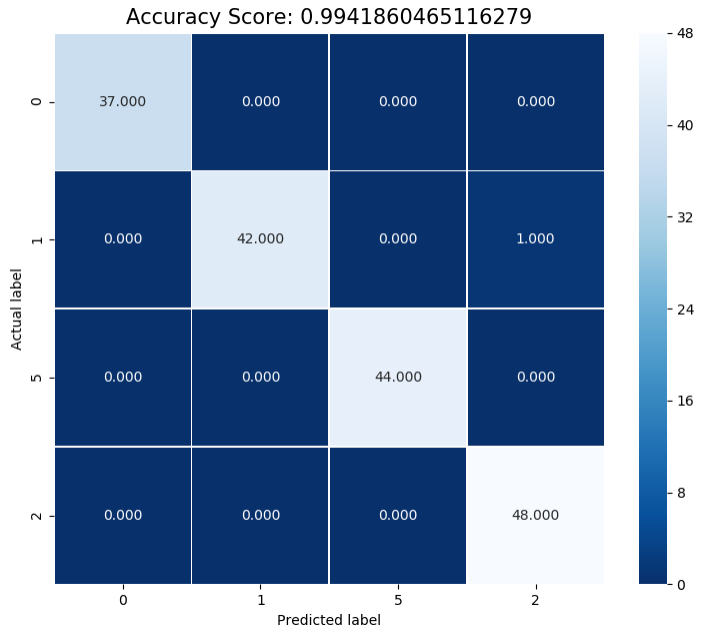

Figura 7: Matriz de confusão *One vs All* para os dígitos 0, 1, 5, 2.

Através da tabela de confusão podemos verificar que a *One vs All* apenas errou uma vez, sendo que classificou um 1 como um 2.

5.2 *One vs One*

- **4 números sequenciais:** Os números utilizados foram o 0, 1, 2 e 3. Foram utilizadas 551 entradas para treino e 169 para teste. Esta função demorou 1.90 segundos e obteve uma precisão de aproximadamente 0.99. A matriz de confusão está representada na figura 8.

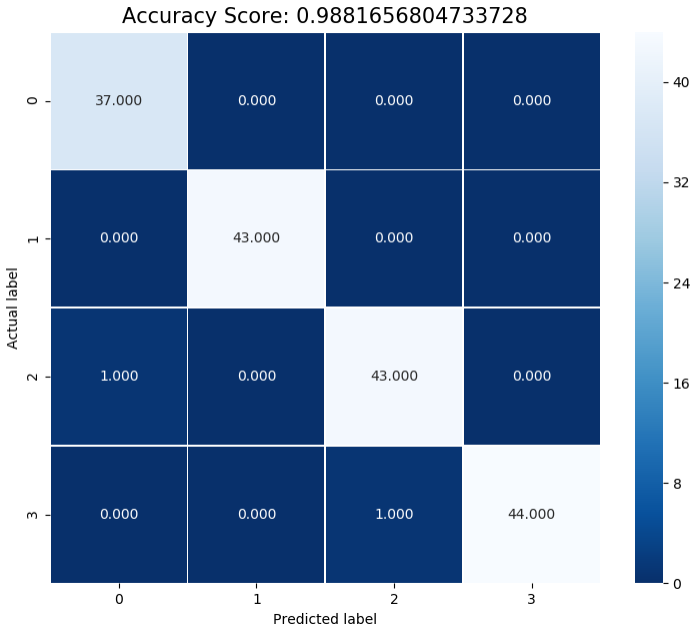

Figura 8: Matriz de confusão *One vs One* para os dígitos 0,1,2 e 3.

Através da tabela de confusão podemos verificar que esta função classificou de forma incorreta um número 2 como 0 e um número 3 como 2.

- **4 números semelhantes:** Os números escolhidos foram o 0, 6, 8 e 9. Foram utilizadas 529 entradas para treino e 184 para teste. Esta função demorou 1.82 segundos e obteve uma precisão de 0.99. Na figura 9 está representada a matriz de confusão.

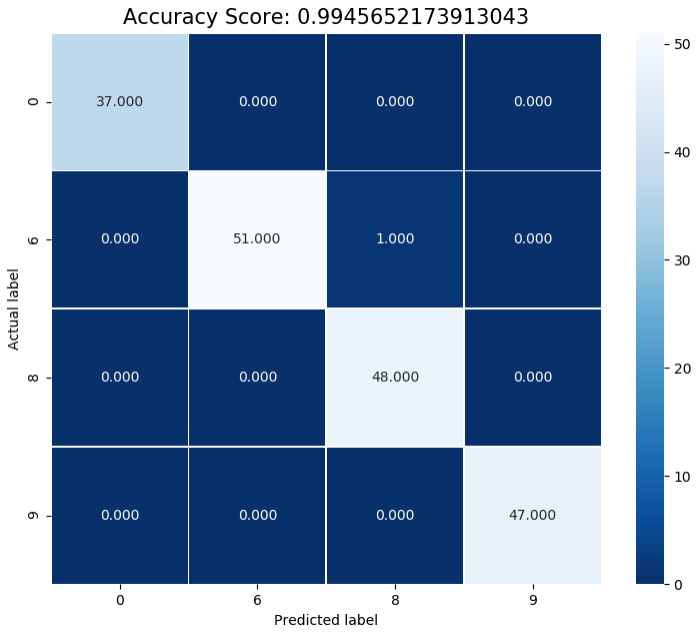

Figura 9: Matriz de confusão *One vs One* para os dígitos 0,6,8 e 9.

Através da figura 9 podemos verificar que esta função apenas obteve uma classificação incorreta, sendo que classificou um 6 como um 8.

- **4 números pouco semelhantes:** Os números escolhidos foram o 0, 1, 5 e 2. Foram utilizadas 547 entradas para treino e 172 para teste. Esta função demorou 1.72 segundos e obteve uma precisão de aproximadamente 0.98. Na figura 10 está representada a matriz de confusão.

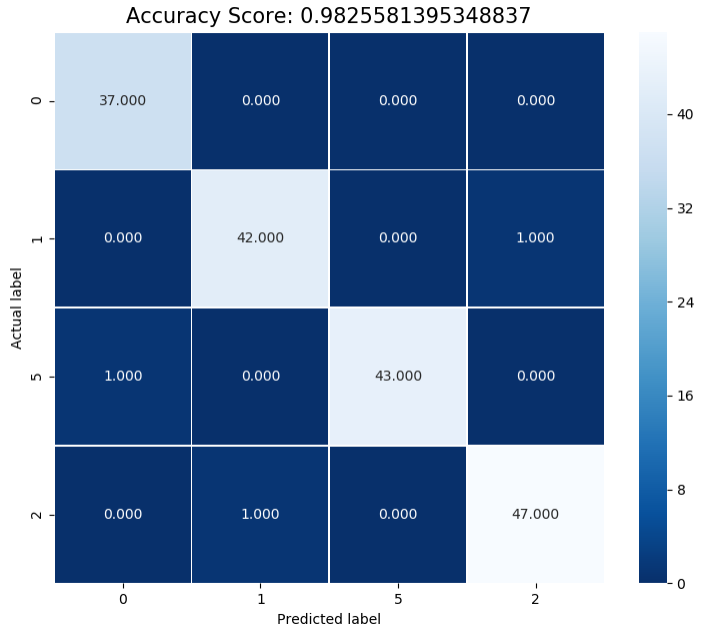

Figura 10: Matriz de confusão *One vs All* para os dígitos 0, 1, 5, 2.

Através da tabela de confusão podemos verificar que a *One vs One* classificou de forma incorreta um 1 como um 2, um 5 como um 0 e um 2 como um 1.

5.3 Dicotómica

- **4 números sequenciais:** Os números utilizados foram o 0, 1, 2 e 3. Foram utilizadas 551 entradas para treino e 169 para teste. Esta função demorou 0.04 segundos e obteve uma precisão de aproximadamente 0.98. A matriz de confusão está representada na figura 11.

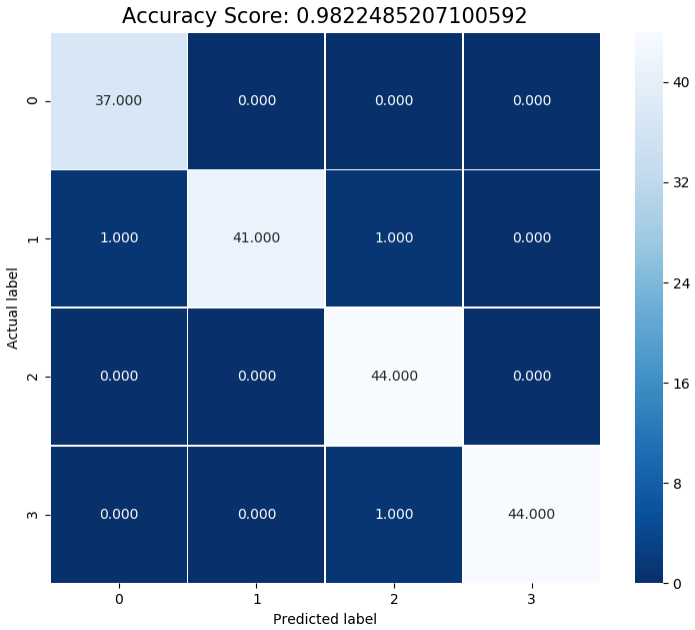

Figura 11: Matriz de confusão *Dicotómica* para os dígitos 0,1,2 e 3.

Através da tabela de confusão podemos verificar que esta função classificou de forma incorreta um número 1 como sendo um 2 e um 1 como sendo 0.

- **4 números semelhantes:** Os números escolhidos foram o 0, 6, 8 e 9. Foram utilizadas 562 entradas para treino e 149 para teste. Esta função demorou 0.03 segundos e obteve uma precisão de aproximadamente 0.99. Na figura 12 está representada a matriz de confusão.

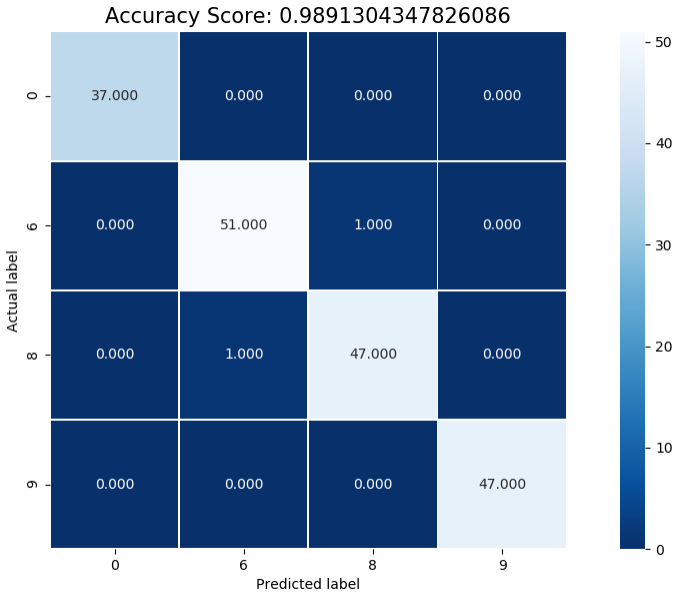

Figura 12: Matriz de confusão *Dicotómica* para os dígitos 0,6,8 e 9.

Através da figura 12 podemos verificar que esta função apenas obteve uma classificação incorreta, sendo que classificou um 8 como um 6 e um 6 como um 8.

- **4 números pouco semelhantes:** Os números escolhidos foram o 0, 1, 5 e 2. Foram utilizadas 581 entradas para treino e 132 para teste. Esta função demorou 0.04 segundos e obteve uma precisão de 0.94. Na figura 13 está representada a matriz de confusão.

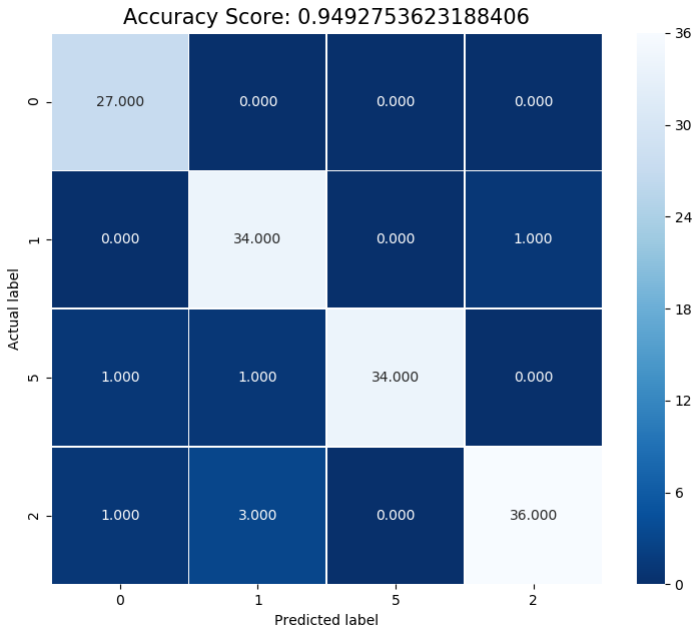

Figura 13: Matriz de confusão *Dicotómica* para os dígitos 0, 1, 5, 2.

Através da tabela de confusão podemos verificar que a *Dicotómica* teve maior dificuldade na classificação do número 2.

5.4 *Softmax*

- **4 números sequenciais:** Os números utilizados foram o 0, 1, 2 e 3. Foram utilizadas 551 entradas para treino e 169 para teste. Esta função demorou 0.0625 segundos e obteve uma precisão de aproximadamente 0.99. A matriz de confusão está representada na figura 14.

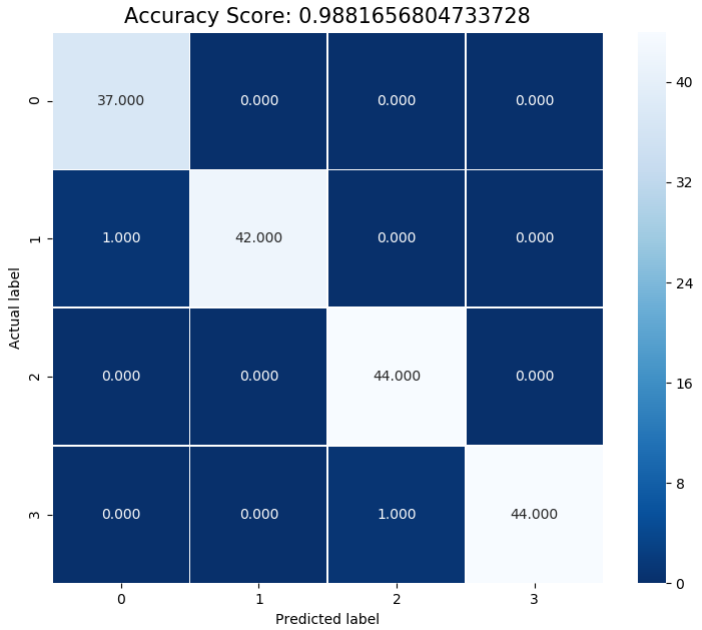

Figura 14: Matriz de confusão *Softmax* para os dígitos 0,1,2 e 3.

Através da tabela de confusão podemos verificar que esta função classificou de forma incorreta um número 3 como 2 e um número 1 como 0.

- **4 números semelhantes:** Os números escolhidos foram o 0, 6, 8 e 9. Foram utilizadas 529 entradas para treino e 184 para teste. Esta função demorou 0.065 segundos e obteve uma precisão de 0.99. Na figura 15 está representada a matriz de confusão.

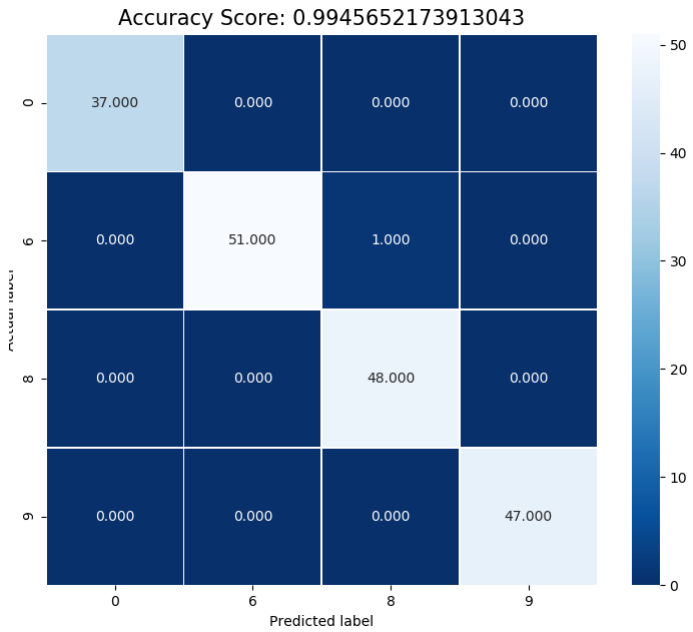

Figura 15: Matriz de confusão *Softmax* para os dígitos 0,6,8 e 9.

Através da figura 15 podemos verificar que esta função apenas obteve uma classificação incorreta, sendo que classificou um 6 como um 8.

- **4 números pouco semelhantes:** Os números escolhidos foram o 0, 1, 5 e 2. Foram utilizadas 547 entradas para treino e 172 para teste. Esta função demorou 0.06 segundos e obteve uma precisão de 1. Na figura 16 está representada a matriz de confusão.

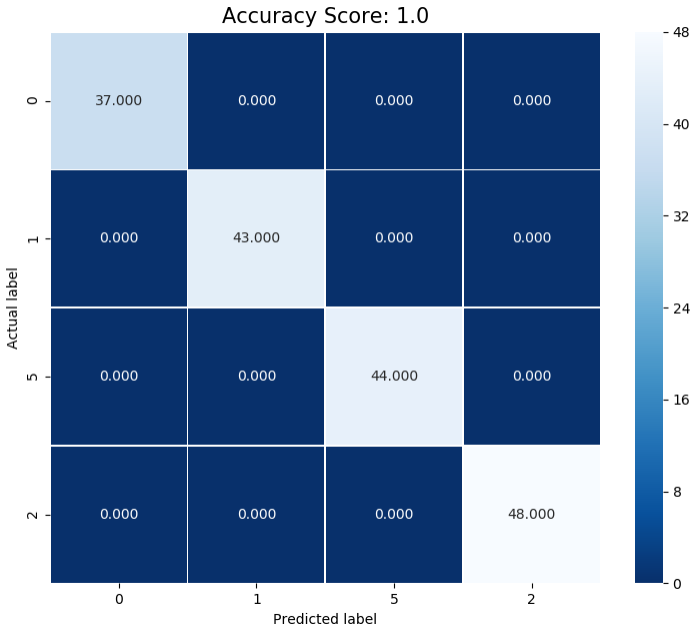

Figura 16: Matriz de confusão *Softmax* para os dígitos 0, 1, 5, 2.

Através da tabela de confusão podemos verificar que a *Softmax* classificou de forma correta todos os dígitos presentes no *dataset*, o que é impressionante dado a rapidez da computação.

Referências

[1] Cho-jui Hsieh. ECS289 : Scalable Machine Learning. 2015.

[2] Andrew Trask. *Grokking Deep Learning*. Manning Publications, 2019.