Trabalho 2 - Técnicas de Classificação

Paulo Mulotto
RA 175541
p175541@dac.unicamp.br

Vitor Aoki
RA 178474
v178474@dac.unicamp.br

I. Introdução

O trabalho realizado teve como intuito o teste de Técnicas de Classificação. Técnicas de Classificação são utilizadas com o intuito de identificar de forma automática, que tipo de objeto é, o analisado.

Para o trabalho, o conjunto de dados analisado foi composto por imagens de vestimentas que são classificadas entre 10 classes (0 a 9). As técnicas utilizadas foram de Regressão Logística e Redes Neurais.

II. Método

Os métodos utilizados para a realização do trabalho foram o de Regresão Logística e Redes Neurais

Dentro da Regressão Logística foram utilizadas duas técnicas. A primeira foi a "One-vs-All" onde é utilizada uma predição binária para classificação de cada uma das classes individualmente e posterior escolha da predição que possuir o maior valor de probabilidade. A segunda foi a "Softmax". Essa função é uma generalização da Regressão Logística binária. Com ela, automaticamente são geradas probabilidades que o input seja de cada classe. Ao final, a esolha da predição é igual a da "One-vs-All"

Dentro das Redes Neurais foram testadas redes com 1 e 2 camadas internas. Em ambas o método de predição é o mesmo, sendo gerado nos neurônios de output, 10 predições que são as probabilidades do input pertencer a cada classe. Para a ativação dos neurônios foram utilizadas as funções "Sigmoid", "Tanh" e "ReLu". Ao final a escolha da predição final é escolhendo o que tiver maior probabilidade.

Em cada um dos métodos utilizados, a atualização dos thetas é realizada por meio do método "Gradient Descent".

III. Soluções Propostas

Do conjunto de treino disponibilizado, decidimos por utilizar 80% do conjunto para o treinamento e 20% do conjunto para validação.

Para cada método foram realizados diversos testes, com diferentes valores de learning rate e iterações (épocas). Após cada teste realizado, escolhemos o nosso melhor modelo "INSIRA AQUI O MELHOR MODELO" que foi utilizado no conjunto de teste e por fim foi criada as matrizes de confusão dos métodos.

IV. Experimentos e Discussões

- Regressão Logística (One-vs-All):

O prmeiro experimento realizado foi com a implementação da Regressão Logística "One-vs-All". Para essa implementação, os valores inicias de theta foram zero. O método utilizado nos primeiros testes foi o de Batch Gradient Descent. Inicialmente utilizamos um baixo valor de número de iterações (5) e um alto valor de learning rate (0.1). Com esses valores obtivemos os seguintes resultados: 25.4% de acertos no conjunto de treino e 24.9% de acertos no conjunto de validação.

Com esses baixos valores, decidimos aumentar o número de iterações em dez vezes (50 iterações) e manter o alto valor de learning rate (0.1). Com esses valores, tivemos os seguintes resultados: 68.54% de acertos no conjunto de treino e 68.59% de acertos no conjunto de validação. Mostrando assim um aumento significante no número de acerto, mostrando que a função estava convergindo. Como o tempo de funcionamento da função estava muito alto, e o programa necessitava de mais iterações para convergir, resolvemos utilizar o método de Mini-Batch Gradient Descent. Para isso, utilizamos um batch de tamanho 400.

Utilizando agora o mini-batch, testamos o programa com um learning rate de 0.1 e 10 iterações. Com isso, tivemos o seguinte resultado: 71.66% de acertos no conjunto de treino e 70.79% de acertos no conjunto de validação. Com isso, tivemos além de boa uma melhora no tempo de execução do programa, uma melhora no número de acertos.

Em seguida, resolvemos aumentar o número de iterações, dobrando o valor anterior (20) e diminuímos o learning rate (0.001). Com isso, obtivemos os seguintes resultados: 84.22% de acertos no conjunto de treino e 83.39% de acertos no conjunto de validação. Com esse aumento significativo no número de acertos, decidimos por manter o learning rate e aumentar o número de iterações para 50.

Com esse aumento no número de iterações, obtivemos os seguintes resultados: 84.76% de acertos no conjunto de treino e 83.4% de acertos no conjunto de validação. Com isso, percebemos que o programa estava próximo da convergência , pois, mesmo com um bom aumento no número de iterações, o programa não aumentou muito a quantidade de acertos.

Decidimos alterar o valor do learning rate e de iterações. Diminuímos o primeiro para 0.0001 e dobramos o valor do segundo para 100. Obtivemos os resultados de 84.81% de acertos no conjunto de treino e 84.2% de acertos no

conjunto de validação. Com isso, vimos que tinhamos atingido a convergência.

Aumentamos, então, o número de iterações para 300 e mantivemos o valor do learning rate, e com isso obtivemos então os valores de 85.85% de acertos no conjunto de treino e 84.81% de acertos no conjunto de validação. Como, mesmo aumentando bastante o número de iterações, os valores não modificaram muito, concluimos que atingimos a convergência. Como uma última tentativa, abaxiamos o learning rate para 0.000001 e mantivemos o número de iterações. Porém, tivemos uma redução nos acertos, inde para 77.5% para o conjunto de treino e 77.41% para o conjunto de validação. Assim, escolhemos o learning rate de 0.0001 e 300 iterações como nosso melhor modelo para "One-vs-All".

Após os experimentos com "One-vs-All", começamos os experimentos com a regressão logística utilizando a função "Softmax". Com os experimentos já realizados anteriormente, decidimos que iríamos utilizar o Mini-Batch Gradient Descent para todos os modelos daqui em diante.

- Regressão Logística (Softmax):

Para a "Softmax" foram utilizados algumas lições aprendidas com "One-vs-All", então começamos com um learning rate de 0.01 e 10 iterações, para testar a corretudo de nosso modelo. Nesse primeiro teste, obtivemos 75.47% de acertos no conjunto de treino e 74.08% de acertos no conjuntos de validação. Com isso, mesmo com uma quantidade baixa de iterações e um learning rate alto, obtivemos um bom número de acertos, estando próximo ao valor que acreditávamos ser de convergência.

Mantendo o mesmo learning rate, decidimos dobrar o número de iterações para 20, e obtivemos, então: 83.96% de acertos no conjunto de treino e 82.92% de acertos no conjunto de validação. Com esse pequeno aumento no número de iterações, vimos que o modelo convergia muito rápido.

Decidimos, então extrapolar os valores e testamos com alguns dos mesmos utilizados para "One-vs-All". Testamos um learning rate de 0.0001 e 100 iterações e obtivemos 85.78% de acertos no conjunto de treino e 84.79% de acertos no conjunto de validação. Com isso, chegamos muito próximos aos valores obtidos no método anterior, porém vimos que o valor estava convergindo já para o valor de convergência, já que com uma mudança significante nos valores dos parâmetros, não tivemos uma grande mudança.

Posteriormente, testamos com os valores que escolhemos como os melhores para "One-vs-All", 0.0001 de learning rate e 300 iterações. Com isso, chegamos em 86.78% de acertos no conjunto de treino e 85.12% de acertos no conjunto de validação. Ainda uma mudança pouco significativa. Testamos então com o mesmo learning rate, porém com 400 iterações. Os resultados, foram: 86.94% de acertos no conjunto de treino e 85.08% de acertos no conjunto de validação. Então vimos que estavámos próximos da convergência.

No penúltimo teste realizado, subimos as iterações para 500 e mantivemos o learning rate. Chegamos em, 87.09% de acertos no conjunto de treino e 85.10% de acertos no conjunto de validação. Então, com um aumento de 0.15% e 0.02% no número de acertos, chegamos a conclusão que o modelo havia convergido, chegando ao nosso modelo com 0.0001 de learning rate e 500 iterações. Com um último teste, assim como "One-vs-All", utilizamos 300 iterações e 0.000001 de learning rate. Obtivemos, 79.52% de acertos no conjunto de treino e 79.25% de acertos no conjunto de validação. Fortalecendo assim, nossa escolha de modelo.

Após todos os testes com Regressão Logística, começamos os experimentos com as Redes Neurais. Para nossas redes, decidimos utilizar um valor de 600 neurônios para cada camada interna. Com as duas redes implementadas, realizamos alguns testes, utilizando o mesmo número de iterações e learning rates já utilizados anteriormente. Porém, nesses testes, sempre obtínhamos 10% de acerto tanto no conjunto de testes, quanto no conjunto de validação. Após várias análises no código, descobrimos, analisando os valores dos thetas, que eles praticamente não estavam sendo atualizados. Isso nos mostrava que a rede não estava atualizando os valores dos neurônios. Acabamos por descobrir que a função randomica para geração dos thetas que estávamos utilizando estava nos fornecendo valores iniciais muito baixos, quase próximos de zero, o que estava inutilizando as redes. Após a troca da função de random, a rede começou a ter resultados melhores.

- Redes Neurais (1 camada interna):

Os primeiros testes foram, utilizando a função de ativação "Sigmoid" e apenas uma camada interna. Ele feito foi com learning rate de 0.001 e 10 iterações. Chegamos em, 44.51% de acertos no conjunto de treino e 45.13% de acertos no conjunto de validação. Mostrando um aumento significativo nos acertos e que a rede estava realmente funcionando.

Aumentamos então o número de iterações para 20, mantendo o learning rate. Os resultados foram: 47.20% de acertos no conjunto de treino e 47.44% de acertos no conjunto de validação. Com esse baixo aumento nos acertos, decidimos então elevar mais o número de iterações mantendo o learning rate. Chegamos em resultados não tão satisfatórios de 31.56% de acertos no conjunto de treino e 32.12% de acertos no conjunto de validação. Vimos então que o learning rate estava muito alto. E que o programa estava demorando muito a rodar com esse número de iterações.

Decidimos então, utilizar um learning rate de 0.0001 e 20 iterações. Chegamos então em 83.05% de acertos no conjunto de treino e 82.35% de acertos no conjunto de validação. Esses valores estavam bem próximos do que acreditávamos ser o valor de convergência. Tentamos então manter esse learning rate, e aumentar o número de iterações para 30. Chegamos então, em 83.73% de acertos no conjunto de treino e 82.88% de acertos no conjunto de validação. Vimos que não foi um aumento muito significativo, dado que aumentamos 10 iterações, então testamos com 50 iterações mantendo o learning rate. Chegamos em 83.44% de acertos no conjunto de treino e 82.37% de acertos no conjunto de validação.

Como os valores pouco se alteraram novamente, escolhemos esse modelo com 30 iterações e learning rate de 0.0001 como nosso melhor para a rede neural com 1 camada interna e função de ativação ”Sigmoid”.

- Redes Neurais (2 camadas internas):

Testamos então, nossa rede neural com 2 camadas internas. Utilizamos ainda a função de ativação ”Sigmoid” para ativação dos neurônios. No primeiro teste utilizamos 10 iterações e 0.001 de learning rate. Isso nos deu 74.42% de acertos no conjunto de treino e 74.18% de acertos no conjunto de validação. Para um primeiro teste foram valores muito bons, estando próximos dos melhores valores que obtivemos.

Utilizamos então o mesmo learning rate, porém com 20 iterações. Chegamos em: 73.98% de acertos no conjunto de treino e 73.52% de acertos no conjunto de validação. Aparentemente, nosso modelo havia convergido, então decidimos alterar o learning rate, para ver se convergia para um outro mínimo. Utilizamos 20 iterações e 0.0001 de learning rate. Os resultados foram: 83.47% de acertos no conjunto de treino e 82.53% de acertos no conjunto de validação. Agora vimos que nosso modelo estava convergindo possivelmente para o mínimo global.

Testamos com 50 iterações e o mesmo learning rate. Obtivemos: 85.37% de acertos no conjunto de treino e 84.17% de acertos no conjunto de validação. Com essa variação, concluímos que chegamos então próximo ao mínimo global e escolhemos esse modelo com 50 iterações e 0.0001 de learning rate como nosso melhor para a rede neural com 2 camadas internas, utilizando a função de ativação ”Sigmoid”.

- Teste com Funções de ativação Tanh e ReLu:

Após todos os testes com as redes neurais, experimentamos outras funções de ativação como, ”Tanh” e ”ReLu”. Decidimos por experimentá-las com os valores de learning rate e iterações que nos deram os melhores resultados com as redes. Para 1 camada interna, utilizando ”Tanh”, chegamos em, 83.11% de acertos no conjunto de treino e 82.42% de acertos no conjunto de validação. São valores bem próximos ao obtido com ”Sigmoid”, e um tempo de execução similar a ”Sigmoid”. Considerando resultados e tempo, ”Sigmoid” possui uma ligeira vantagem. Para ”ReLu”, tivemos 10.07% de acertos no conjunto de treino e 9.7% de acertos no conjunto de validação, o que são valores muito baixos. Testamos então com 30 iterações e 0.0000001 de learning rate e tivemos 69.91% de acertos no conjunto de treino e 70.28% de acertos no conjunto de validação. Porém, utilizando os mesmos parâmetros da ”Sigmoid”, não obtivemos bons resultados, mesmo a função ”ReLu” tendo um tempo computacional melhor.

Para a rede de duas camadas internas, efetuamos o mesmo processo de aplicar os parâmetros do melhor modelo nas duas funções. Para ”Tanh”, obtivemos: 82.52% de acertos no conjunto de treino e 82.08% de acertos no conjunto de validação. Esses valores se mostraram próximos ao obtido com ”Sigmoid”. Porém, o tempo de computação foi muito alto, fazendo assim com que ”Sigmoid” tivesse vantagem. Para ”ReLu”, tivemos 9.99% de acertos no conjunto de treino e 10.06% de acertos no conjunto de validação, o que são valores muito baixos. Assim como para uma camada interna, testamos então com 30 iterações e 0.0000001 de learning rate e tivemos 35.11% de acertos no conjunto de treino e 34.89% de acertos no conjunto de validação. Esse valores continuam ainda baixos, o que torna ”Sigmoid” mais vantajosa em termos de resultados. Assim, a função que melhor nos deu resultados nas redes neurais foi a ”Sigmoid”.

Vimos com os experimentos, que no geral a função de ativação com a melhor performance foi a ”ReLu”, que demorava menos tempos para rodar. Porém, os resultados obtidos com ela não forma tão bons quanto com ”Sigmoid” e ”Tanh”. Para se obter resultados parecidos, seriam necessárias mais iterações, o que faria com que a ”ReLu” aumentasse seu custo computacional, ficando próximo ou até maior comparado com as outras duas funções de ativação.

Dentre todos os métodos testados, o que nos deu melhores resultados foi a Regressão Logística utilizando ”Softmax”. Com isso, utilizamos o conjunto de teste no nosso modelo e chegamos em 85.21% de acertos.

		Predicts									
		0	1	2	3	4	5	6	7	8	9
Targets	0	822	7	12	33	8	0	102	0	15	1
	1	2	972	4	16	0	2	4	0	0	0
	2	17	3	727	9	134	0	102	0	8	0
	3	30	25	12	874	34	0	22	0	3	0
	4	1	1	63	30	814	0	88	0	3	0
	5	1	1	0	0	0	895	1	58	10	34
	6	165	6	86	29	99	0	600	0	15	0
	7	0	0	0	0	0	27	0	926	0	47
	8	4	1	5	4	5	11	20	4	944	2
	9	0	0	0	0	0	14	0	37	2	947

Figure 1. Matriz de Confusão para o conjunto de Testes

Da matriz de confusão podemos ver que o modelo conseguiu predizer com maior precisão quando uma imagem é da classe 1, com 972 acertos. Por outro lado, o que ele mais errou foi quando resultou que a imagem era da classe 0, porém ela era da classe 6, tendo 165 erros. Podemos tirar disso, que a classe 1 é a que mais difere comparada as outras classes, já que pelos resultados obtidos, sua classificação é a mais fácil. Por outro lado, podemos concluir que a classe 6 é parecida com a classe 0, já que o programa em alguns casos confundiu ambas, pois, além de confundir 0 quando na verdade a imagem era da classe 6, uma boa quantidade de vezes ele confundiu 6, quando na verdade era da classe 0.

Comparada a Regressão Logística ”One-vs-All”, a Regressão Logística ”Softmax” teve uma acurácia maior e também uma performance maior. Isso se deve muito ao fato dela calcular de uma única vez a probabilidade da imagem ser de cada classe, enquanto ”One-vs-All” precisa calcular essas probabilidades separadamente.

V. Conclusões e Trabalhos Futuros

Após o uso de todos esses modelos, podemos ver que por mais que suas implementações sejam diferentes, em sua essência, seu funcionamento é o mesmo, utilizando-se de funções probabilísticas para fazer as predições. Dentre todos os métodos utilizados, o que apresentou melhor performance e acurácia foi a Regressão Logística com ”Softmax”. Porém

isso se aplica ao conjunto de dados utilizados para o trabalho, podendo, em outros dados, essa função não ter o resultado desejado.

Como descoberto durante as pesquisas no decorrer do trabalho, as Redes Neurais são as mais utilizadas hoje em dia para problemas de classificação. Isso se deve muito à sua versatilidade, podendo ser ”facilmente” modificada alterando os neurônios e camadas internas, assim, como as funções de ativação.

Diferente da Regressão Linear, os métodos vistos no trabalho são muito mais versáteis e permitem uma maior gama de possibilidades para tentar otimizar e melhorar tanto performance quanto resultados desejados. Como visto na matriz de confusão e nos resultados de acurácia obtidos, os modelos não conseguem predizer 100% dos dados, porém é perceptível o potencial que tais modelos tem de simular o que o cérebro é capaz de fazer.

REFERENCES

[1] https://www.coursera.org/learn/machine-learning
[2] http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/
[3] http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression
[4] https://www.coursera.org/lecture/ml-classification/multiclass-classification-with-1-versus-all-N7QA6
[5] https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/
[6] https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0
[7] https://theclevermachine.wordpress.com/tag/tanh-function/
[8] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn & TensorFlow*