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 utilizado um classificador binário para classificação de cada uma das classes individualmente, onde os elementos da classe analisada são positivos enquanto os das outras classes são negativos. Após a classificação dos elementos a escolha da predição é feita através do resultado 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 escolha da predição é igual a da One-vs-All.

Dentro das Redes Neurais foram testadas redes com 1 e 2 camadas internas. Em ambas as redes neurais, o método de predição utilizado foi o mesmo. Para a ativação dos neurônios foram utilizadas as funções "Sigmoid", "Tanh" e "ReLu". Há um neurônio de output, que gera uma predição, para cada classe, obtendo as probabilidades do input pertencer a cada uma das classes. A escolha da predição da classe do input é escolhendo o que tiver maior probabilidade nos neurônios de output.

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, que foi utilizado no conjunto de teste e por fim foi criada a matriz de confusão para esse modelo aplicado ao teste.

IV. Experimentos e Discussões

Após a implementação de todas as funções, iniciamos os testes do programa. Porém, ao chegarmos nos testes com redes neurais, começamos a ter problemas com overflow no cálculo da função Sigmoid, na computação do expoente. Após vários testes, como a normalização do expoente, por meio da subtração do maior valor da matriz z, ainda estávamos tendo o problema com overflow, mas mesmo assim, ainda conseguíamos resultados satisfatórios, como 85% de acertos com as redes neurais. Nossa útlima tentativa de sanar os problemas com overflow, decidimos por normalizar os dados de entrada, dividindo os valores dos pixels de entrada, pelo máximo valor de intensidade de um píxel (255). Essa normalização resolveu os problemas com overflow, e então realizamos os experimentos, cujos resultados estão a seguir. Em todos os métodos, a função de atualização dos thetas foi a Gradient Descent.

A. Regressão Logística (One-vs-All)

Nossos primeiros testes foram com a Regressão Logística One-vs-All. Para esse método, os valores iniciais de theta foram zeros. Iniciamos com um valor baixo de learning rate, de 0.0001 e 100 iterações. Obtivemos 41.66% de acertos no conjunto de treino e 41.02% de acertos no conjunto de validação. Com essa baixa porcentagem de acertos, decidimos então manter o learning rate e aumentar o número de iterações para 300. Com isso, obtivemos 51.42% de acertos no conjunto de treino e 51.26% de acertos no conjunto de validação. Para tentar diminuir o número de iterações, decidimos por aumentar o learning rate para 0.001 e utilizar 100 iterações. Com isso, chegamos em 60.65% de acertos no conjunto de treino e 60.44% de acertos no conjunto de validação. Aumentamos então o número de iterações para 300, mantendo o mesmo learning rate. Porém, tivemos somente 64.74% de acertos no conjunto de treino e 64.92% de acertos no conjunto de validação.

Como a porcentagem de acertos não aumentou de forma significativa, aumentamos novamente o learning rate, dessa vez para 0.01 e testamos dessa vez com 500 iterações. Chegamos em 70.14% de acertos no conjunto de treino e 70.48% de acertos no conjunto de validação. Aumentamos novamente o learning rate, para 0.1 e utilizamos 100 iterações. Chegamos ao resultado de 74.3% de acertos no conjunto de treino e 74.08% de acertos no conjunto de validação. Acreditávamos então que nosso modelo estava chegando ao valor de convergência. Então, aumentamos o número de iterações para 500 com o

mesmo learning rate. Obtivemos 80.58% de acertos no conjunto de treino e 80.19% de acertos no conjunto de validação. Dobramos então o valor de iterações para 1000 mantendo o mesmo learning rate, e então chegamos a 82.03% de acertos no conjunto de treino e 81.79% de acertos no conjunto de validação. Como, com esse alto valor de learning rate e dobrando o número de iterações, não tivemos um aumento substancial na porcentagem de acertos, vimos que tínhamos obtido a convergência de nosso modelo. Assim, o modelo escolhido para a Regressão Logística One-vs-All possui 0.1 de learning rate e 1000 iterações. Para esse método não plotamos os gráficos do número de acertos por iterações, pois, como temos um modelo para cada classe, teríamos 10 modelos, e portante 10 gráficos, que não seriam viáveis de adicionar ao relatório.

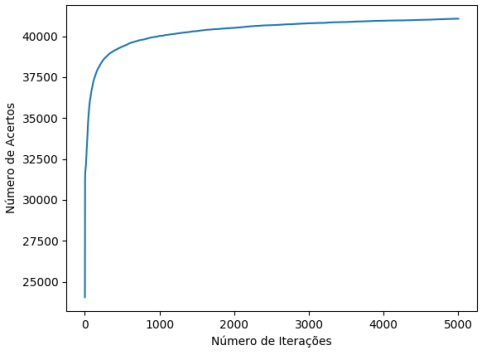

Figure 1. Gráfico 1: Número de acertos por Iteração - Regressão Logística Softmax

B. Regressão Logística (Softmax)

Para a Regressão Logística Softmax, inicializamos nossos thetas com zeros também. Depois dos experimentos com a Regressão Logística One-vs-All, realizamos os experimentos com a Regressão Logística Softmax. Para um primeiro teste, utilizamos um learning rate de 0.000001 e 20 iterações. Com esses dados, tivemos os seguintes resultados: 31.76% de acertos no conjunto de treino e 31.62% de acertos no conjunto de validação. Com esses baixos valores de acerto resultantes, aumentamos o learning rate para 0.0001 e 30 iterações. Mesmo com esse aumento, não tivemos um aumento substancial, sendo 33.03% de acertos no conjunto de treino e 32.78% de acertos no conjunto de validação. Aumentamos, então, o número de iterações para 500 e mantivemos o learning rate. Com isso, chegamos em 48.86% de acertos no conjunto de treino e 48.60% de acertos no conjunto de validação.

Mesmo tendo um bom aumento na porcentagem de acertos, ainda estava muito baixo. Decidimos então aumentar o learning rate para 0.01 e mantivemos 500 iterações. Com isso, tivemos um bom aumento na porcentagem de acertos, sendo 73.37% de acertos no conjunto de treino e 76.96% de acertos no conjunto de validação. Decidimos então dobrar o número de iterações para 1000 e o learning rate para 0.1. Com isso, chegamos a 83.35% de acertos no conjunto de treino e 83.08% de acertos no conjunto de validação, mostrando um bom aumento no número de acertos. Tentamos então 3000 iterações e o mesmo learning rate, chegando em 84.99% de acertos no conjunto de treino e 84.22% de acertos no conjunto de validação, mostrando que o modelo estava convergindo. Como último teste, utilizamos 5000 iterações e 0.1 de learning rate, chengando em, 85.58% de acertos no conjunto de treino e 84.63% de acertos no conjunto de validação. Para verificar a convergência do modelo, plotamos um gráfico de número de acertos por iteração e vimos que houve a convergência, como pode-se ver no seguinte gráfico:

C. Redes Neurais (1 camada interna)

No início dos testes com as redes neurais, estávamos tendo problemas com um acerto de apenas 10% nos conjuntos de teste e validão, para quaisquer valores de learning rate e número de iterações. Após alguns testes e prints das matrizes de thetas, verificamos que a nossa rede não estava se alterando com as iterações, ou seja, estava inutilizada. Acabamos por descobrir que o problema estava com a função que gerava automaticamente os valores de theta. Ela estava gerando valores muito baixos, que não estavam ativando os neurônios, fazendo a rede se inutilizar. Ao trocarmos a função, a rede passou a ativá-los e obtivemos como primeiro resultado, utilizando 10 iterações e 0.000001 de learning rate, 46.12% de acertos no conjunto de treino e 46.28% de acertos no conjunto de validação. Para esses resultados indicados a seguir, utilizamos a função de ativação Sigmoid.

Em seguida, mantivemos o valor do learning rate e aumentamos o número de iterações para 50, e obtivemos já 63.61% de acertos no conjunto de treino e 63.82% de acertos no conjunto de validação, um aumento substancial. Decidimos então manter o learning rate e dobrar o número de iterações, obtendo 67.17% de acertos no conjunto de treino e 67.42% de acertos no conjunto de validação. Dobrando o número de iterações, não tivemos um aumento substancial, então decidimos por aumentar o learning rate, utilizando 0.0001 e 30 iterações. Chegamos então em 81.76% de acertos no conjunto de treino e 81.14% de acertos no conjunto de validação, um aumento muito significativo e chegando próximo aos valores obtidos nos modelos anteriores quando convergiram. Aumentamos então o número de iteraçõs para 200 mantendo o mesmo learning rate, e chegamos em 86.9% de acertos no conjunto de treino e 85.98% de acertos no conjunto de validação, porém o modelo demorou muito tempo para rodar, então decidimos aumentar novamente o learning rate. Testamos um learning rate de 0.001 e 30 iterações, e chegamos em 87.34% de acertos no conjunto de treino e 86.18% de acertos no conjunto de validação. Tentamos então, aumentar o número de iterações.

Utilizamos um learning rate de 0.001 e 50 iterações. Com esses valores, chegamos em 88.55% de acertos no conjunto de treino e 86.73% de acertos no conjunto de validação. Escolhemos esse como o melhor modelo para a Rede Neural com 1 camada interna. Ao analisarmos o gráfico para esse modelo, pudemos concluir que nosso modelo havia convergido, como

podemos ver no gráfico a seguir:

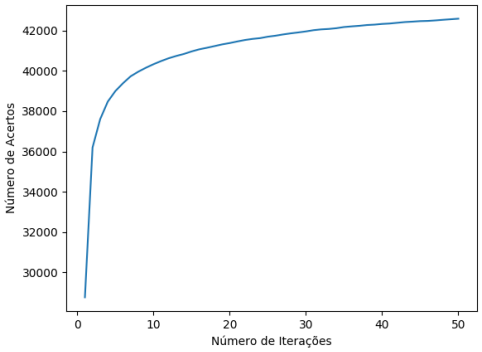

Figure 2. Gráfico 2: Número de acertos por Iteração - Rede Neural (1 Camada Interna - Função de ativação: Sigmoid)

Tentamos utilizar um learning rate de 0.1 e 10 iterações, porém verificamos que a função divergia. Pudemos verificar isso ao encontrarmos um overflow na função Sigmoid e também ao verificar o calculo do valor do neurônio ativado, vimos que o valor estourava para um valor muito baixo (4.36e-239), como verificado no print a seguir que é um trecho dos valores dos neurônios ativados:

```
[4.67246448e-024 1.00000000e+000 1.04618781e-088 ... 1.00000000e+000
 0.00000000e+000 4.36483380e-239]]
```

Figure 3. Figura 1: Modelo divergindo

D. Redes Neurais (2 camadas internas)

Para as Redes Neurais com 2 camadas internas e função de ativação Sigmoid, iniciamos nossos thetas com valores aleatórios também. Já com o problema de overflow resolvido, decidimos por iniciar nossos experimentos com um learning rate de 0.00001 e 50 iterações. Com isso, tivemos 63.32% de acertos no conjunto de treino e 63.46% de acertos no conjunto de validação.

Com esse bom valor inicial obtido, decidimos então, utilizando as experiências feitas com a rede neural com 1 camada interna, aumentar o learning rate substancialmente para 0.001 e 5 iterações. Chegamos então a 78.19% de acertos no conjunto de treino e 77.61% de acertos no conjunto de validação. Percebemos então que estávamos chegando ao valor de convergência obtido pelos outros modelos. Então decidimos aumentar em 10 vezes o número de iterações, utilizando o mesmo learning rate. Chegamos em 87.03% de acertos no conjunto de treino e 85.82% de acertos no conjunto de validação, valores muito próximos ao melhor obtido até agora, que foi com a Rede Neural com 1 camada interna. Mantendo o mesmo learning rate e aumentando o número de iterações para 70, obtivemos 88.18% de acertos no conjunto de treino e 86.54% de acertos no conjunto de validação e escolhemos esse como nosso melhor modelo para a Rede Neural com 2 camadas internas, pois, ao analisarmos o gráfico de número de acertos por iteração, percebemos que nosso modelo havia convergido. Podemos perceber isso, no gráfico a seguir:

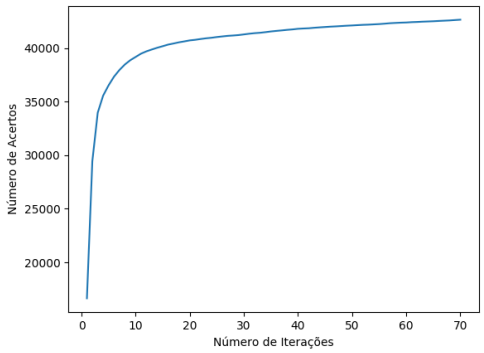

Figure 4. Gráfico 3: Número de acertos por Iteração - Rede Neural (3 Camadas Internas - Função de ativação: Sigmoid)

Após os testes com as redes neurais utilizando a função Sigmoid como função de ativação, realizamos alguns testes com outras funções de ativação, como Tanh, e ReLu. Para uma base de comparação, realizamos testes iguais aos dois últimos testes realizados para a Rede Neural com 1 camada interna e para a Rede Neural com 2 camadas internas. Chegamos então, aos seguintes resultados:

Para a Rede Neural com 1 camada interna e função Tanh: 56.04% de acertos no conjunto de treino e 55.69% de acertos no conjunto de validação, com learning rate 0.001 e 30 iterações; e 15.63% de acertos no conjunto de treino e 15.38% de acertos no conjunto de validação, com learning rate 0.001 e 50 iterações.

Para a Rede Neural com 1 camada interna e função ReLu: 85.46% de acertos no conjunto de treino e 84.72% de acertos no conjunto de validação, com learning rate 0.001 e 30 iterações; e 70.33% de acertos no conjunto de treino e 69.75% de acertos no conjunto de validação, com learning rate 0.001 e 50 iterações.

Para a Rede Neural com 2 camadas internas e função Tanh: 74.55% de acertos no conjunto de treino e 73.43% de acertos no conjunto de validação, com learning rate 0.001 e 50 iterações; e 75.41% de acertos no conjunto de treino e 74.51% de acertos no conjunto de validação, com learning rate 0.001 e 70 iterações.

Para a Rede Neural com 2 camadas internas e função ReLu: 10% de acertos no conjunto de treino e 9.99% de acertos no conjunto de validação, com learning rate 0.001 e 50 iterações; e 84.80% de acertos no conjunto de treino e 83.17% de acertos no conjunto de validação, com learning rate 0.001 e 70 iterações.

O que pudemos observar utilizando essas outras funções de ativação, é que a função de ativação Tanh aumentou o tempo de processamento do modelo, enquanto a ReLu, esteve um pouco mais rápida, porém próxima ao tempo de processamento da Sigmoid. Mesmo assim, nenhum foi capaz de superar os resultados obtidos com a Rede Neural com 1 camada de

ativação e função Sigmoid. Assim, aplicando nosso modelo ao conjunto de testes chegamos a 87.05% de acertos. Obtivemos também a matriz de confusão, do modelo aplicado ao conjunto de testes. Ela pode ser conferida a seguir:

		Predict									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Target	**0**	805	1	8	37	1	3	134	0	10	1
	1	1	972	1	17	0	2	7	0	0	0
	2	11	1	718	14	85	0	166	0	5	0
	3	23	13	4	915	18	1	25	0	1	0
	4	1	0	41	37	783	0	136	0	2	0
	5	2	0	0	1	0	929	1	41	5	21
	6	124	2	34	29	43	1	757	0	10	0
	7	0	0	0	0	0	28	0	917	1	54
	8	4	0	3	3	2	6	17	4	959	2
	9	0	0	0	0	0	11	0	38	1	950

Figure 5. Tabela 1: Matriz de Confusão para o conjunto de Testes

Podemos perceber da tabela, que a classe que o modelo mais acertou o predict foi a classe 1, que representa as calças. A classe que mais causou erros na identificação pelo modelo, foi a classe 2, que representa os pullovers, um tipo de blusa. Em 166 oportunidades, o modelo classificou o pullover como sendo da classe 6, que representa as camisas. Isso é esperado, já que o formato de ambos é bem parecido. Outro erro bastante comum, foi classificar a classe 4, casacos, como pullover. Novamente, o erro é explicado pela proximidade de ambas as vestimentas. Da diagona principal da tabela, podemos tirar a acurácia do modelo, já previamente calculada, resultando em 87%. Como observado na tabela, o modelo esteve próximo de simular o cérebro humano, predizendo bem as classes, a partir dos dados que utilizou para aprender. Os erros que ocorreram são esperados, já que o modelo não é uma cópia perfeita de um cérebro humano.

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*