Modelagem com Aprendizado Simbólico e Redes Neurais

Gabriel M. Conke[1]

[1]Universidade Tecnológica Federal do Paraná(UTFPR)

gabrielconke@alunos.utfpr.edu.br

1. Introdução

O aprendizado de máquina, em suma, é o nome dado ao processo do computador aprender a tomar decisões. Nesse contexto, podemos considerar a seguinte citação:

"Any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population".[Simon, 1983]

Assim, o presente documento tem como objetivo relatar a implementação de três diferentes algoritmos de aprendizado de máquina - ID3, *Forest Random* e Redes Neurais. O propósito é capacitar a máquina a "aprender"e, consequentemente, tomar decisões cada vez mais precisas em resposta às entradas fornecidas.

Portanto, ao utilizar diferentes algoritmos nesse processo, busca-se desenvolver modelagens específicas para cada um, a fim de analisar suas diferenças, vantagens, desvantagens e limites, assim como o algoritmo que apresenta maior acurácia no contexto fornecido. Dessa forma, é possível compreender melhor o desempenho de cada abordagem e suas particularidades.

2. Modelagem

2.1. ID3

O algoritmo ID3, também conhecido como árvore de decisão, tem como principal característica a formação de uma árvore de fácil representação que faz a divisão dos atributos de entrada em diversos nós até alcançar um atributo de saída.

Para a implementação do algoritmo ID3, em primeiro momento, foi necessária realização da modelagem do problema. Primeiramente, foi fornecido um arquivo csv contendo os dados que seriam utilizados para o treinamento e os testes.

Em suma, os dados consistiam de pressão sistólica, pressão diastólica, qualidade da pressão, pulso, respiração, gravidade e rótulo, sendo a qualidade da pressão arterial influenciada pelas pressões sistólica e diastólica. Desta forma, foi definido que os atributos de entrada para o sistema seriam qualidade da pressão, pulso e respiração e o de saída o rótulo de gravidade do caso.

Ademais, os dados fornecidos não estavam divididos em dados de treinamento e dados de testes, sendo necessária ser feita essa divisão de forma manual. Desta forma, para essa divisão foi utilizado o método de validação cruzada "k-fold", isto é, os dados são divididos em k partições, sendo que a cada iteração uma partição é utilizada para os testes e as demais utilizadas para o treinamento [Rabello 2019]. Para o contexto em questão, foram feitas as divisões com diferentes escalas e os resultados entre elas comparadas nas seções subsequentes.

Após os dados para treinamentos e testes obtidos, fora-se necessário particionar cada atributo em classes, tendo em vista que todos eram números reais em intervalos pré-determinados. Para esse processo, em primeiro momento, foi pensado em dividir cada atributo em 4 classes de intervalos uniformes. Todavia, após posteriores análises foi notado que tal abordagem iria gerar muitas incongruências com relação ao treinamento, pois devido ao número pequeno de classes haveriam casos em que dados de treinamento com mesmas classes gerariam rótulos de gravidades diferentes.

Desta forma, criou-se um processo para determinar como os números contínuos iriam ser divididos em classes. Em primeiro lugar, para cada atributo determinou-se possíveis pontos de corte, esses determinados com base nos valores dos dados. Por exemplo, considerando os valores [2.3, 4.7, 6.1], o algoritmo pode testar cortes em valores como 2.3; 4.7 ou 6.1.

Logo após, foi calculada a entropia para o subconjunto a direita do ponto de corte e à esquerda do ponto de corte, assim como o ganho de informação de cada uma das partes, sendo o subconjunto que apresenta maior ganho de informação escolhido. As fórmulas para a entropia e ganho de informação utilizadas em todo o escopo do trabalho podem ser vistas no apêndice A e B, assim como suas respectivas implementações no código.

Ademais, após determinadas as classes de cada atributo e seus limites, foi realizado um processo para determinar como seria feita a geração da árvore de decisão. Durante o próprio algoritmo de cálculo da entropia para cada subconjunto da variável, foi também encontrada qual a variável que possui o subconjunto com maior ganho de informação. Desta forma, era possível afirmar que, para determinado conjunto de dados, aquela *feature* com o maior ganho de informação era a que deveria ficar na raiz.

Desta forma, o cálculo do ganho de informação de cada um dos subconjuntos era utilizado para construir um nó de uma árvore binária. Vamos supor, por exemplo, que o subconjunto que apresenta maior ganho de informação considerando todos os subconjuntos possíveis entre todos os atributos seja a qualidade da pressão arterial maior que 5. Desta forma, a qualidade da pressão arterial seria representada como um nó e suas arestas seriam ”sim”e ”não”, levando o problema a uma nova sub-árvore de decisão e assim sucessivamente.

Assim, após a divisão em grupos e formação das árvores, as folhas representavam as saídas para determinadas classes das entradas, possibilitando a realização dos testes e uma análise do sistema. Os resultados obtidos para diferentes modelagens podem ser visto na seção seguinte.

2.2. Random Forest

O *Random Forest*, em suma, é um algoritmo que se faz da utilização de diversas árvores de decisão para gerar uma saída. Desta forma, ele é capaz de reduzir o *overfitting* gerado nas árvores de decisões, tendo em vista que as saídas de cada árvore são combinadas ao fim para determinar uma saída geral. Desta forma, as árvores com o *overfitting* não atuam diretamente na saída e possuem menos influência nela. [EBAC 2024]

Para a criação do algoritmo *Random Forest* utilizado neste trabalho, em primeiro momento, foi notado que para a floresta produzir árvores de decisões diferentes era necessário que apenas parte do conjunto de atributos e um subconjunto de dados fossem

considerados para cada árvore. Desta forma, para a realização desse processo foi determinado o número máximo de *features* utilizadas em cada árvore, valor esse dado por $srqt(n)$, sendo n a quantidade total de atributos.

Já com relação a escolha do subconjunto de dados utilizados para cada árvore, foram escolhidos dados aleatórios do conjunto geral com reposição, de forma de que dados iguais poderiam ser utilizados mais de uma vez, aumentando a aleatoriedade.

Após todos esses processos de tratamento de dados, foram construídas florestas com diferentes quantidades de árvores para a análise, sendo esses dados detalhados de forma mais clara posteriormente. Em suma, o processo de produção das árvores se da igual ao já citado no algoritmo ID3, a exceção da quantidade de árvores produzidas e como essas saídas são tratadas.

Assim, também foi decidido que todas as árvores da floresta seriam executadas e a classe mais comum entre as saídas seria a escolhida como resposta. Desta forma, procurou-se reduzir a influência de árvores com *overfitting* e treinamento muito específico aos dados de entrada - vale ressaltar que para o treinamento e teste desse algoritmo também foi considerada a validação cruzada k-fold.

2.3. Redes Neurais

As redes neurais, em suma, foram criadas como ”imitações”dos neurônios do nosso cérebro, sendo que cada neurônio possui diversas entradas com seus respectivos pesos e verifica se a soma do produto dos pesos de cada entrada com a entrada são maiores que um valor determinado. [AWS 2023]

As redes neurais possuem característica bem diferentes dos demais algoritmos tratados até aqui. Desta forma, para sua implementação, foi feita em primeiro momento uma normalização dos valores de entrada, de modo a facilitar as conversões das redes neurais. Além disso, as classes de saída foram representadas utilizando-se da codificação *one-hot*, utilizada para converter as classes para números, possibilitando a montagem da rede.

Logo após, se fez necessário definir como seria a estrutura da rede neural. Para tanto, algumas medidas foram tomadas:

- Utilização de uma rede neural com múltiplas camadas: como os dados apresentados para o treinamento não eram extensos, verificou-se a possibilidade computacional de fazer uma estrutura multicamadas; essa que necessita de um maior processamento, mas ao mesmo tempo apresenta uma maior acurácia na resposta. Desta forma, a rede passou a possuir maior capacidade de aprendizado, tendo em vista que com multicamadas ela é capaz de aprender funções não-lineares, diferentemente da rede de única camada [Carvalho 2009]. Para o procedimento em questão, foi decidida a utilização de apenas 1 camada intermediária, pois, de acordo com [Ceccon], essa estrutura já é capaz de aproximar qualquer equação não-linear.

- Número de neurônios: com relação a camada intermediária, segundo [Heaton 2005], existem três abordagens ideais para a definição do número de neurônios para a camada intermediária. Segundo o autor, o número de

neurônios escondidos deve estar entre o tamanho da camada de entrada e o da camada de saída ou o número de neurônios escondidos deve ser 2/3 do tamanho da camada de entrada, mais o tamanho da camada de saída ou o número de neurônios escondidos deve ser menor que o dobro do tamanho da camada de entrada. Considerando essas afirmações, o modelo criado para o caso utilizou-se dessas três abordagens com o objetivo de compará-las posteriormente.

- Modelo sequencial: pelo fato da arquitetura seguir um fluxo linear e não ser necessária tanta complexidade na implementação, tendo em vista que a rede foi criada considerando-se apenas três atributos de entrada, não se viu necessidade da implementação de modelos não sequenciais.

- Função de ativação *"relu"*: escolhida principalmente por possuir um custo computacional menor que as demais funções, sendo utilizada nas camadas escondidas [Data Science Academy, 2022]. Ademais foi utilizada uma função de ativação *"softmax"* na camada final, sendo ela responsável por transformar os valores da entrada em uma distribuição de probabilidades que soma 1 [Data Science Academy, 2022].

- Épocas: também foi necessário determinar a quantidade de épocas para o treinamento da rede neural, isto é, quantas vezes o modelo passa por todo o conjunto de dados de treinamento. Assim, foram feitos testes para diferentes números de épocas, a fim de verificar especialmente o *overfitting* existente com a utilização de muitas delas [Data Science Academy, 2022].

- Batelada: por fim, decidiu-se a quantidade de amostras que seriam passadas pelo treinamento antes da atualização dos pesos. Desta forma, de acordo com [Masters and Luschi 2018], a quantidade de amostras ideais por batelada estariam no intervalo entre 2 e 32, assim como a quantidade de atributos na base de dados era pequena, foram escolhidas 6 amostras por bateladas.

Desta forma, após feitas as definições anteriores, assim como nas demais modelagens, foi utilizada a divisão k-fold dos dados. Tendo em vista que os algoritmos anteriores já apresentaram resultados referentes a quantidade de partições dos dados do k-fold, esse modelo irá tratar, principalmente, da influência das épocas e da quantidade de neurônios na camada intermediária no resultado final.

Assim, em primeiro momento, os dados eram preparados para a rede neural, por meio da validação cruzada k-fold, normalização dos dados e codificação *one-Hot*. Logo após, o modelo foi criado conforme as descrições acima e compilado. Desta forma, os dados da entrada são inseridos na rede de acordo com *feedfoward*, os erros verificados por meio da diferença entre a saída apresentada e a saída prevista e, após uma quantidade de amostras para treinamento, os pesos atualizados com o algoritmo de *backpropagation*, isto é, partindo da saída e indo até os pesos das entradas[Ganesan]. Por fim, os dados de testes eram introduzidos para verificar a acurácia do modelo.

3. Resultados e análises

3.1. ID3

Desta forma, após realizado o algoritmo descrito e modelado nas seções anteriores, foram feitos testes considerando diferentes números de partições dos dados e, consequentemente, diferentes números de dados para treinamentos e testes, considerando que os testes são feitos com $1/k$ dos dados totais. Assim, os resultados obtidos estão localizados no apêndice C.

Com base nos resultados apresentados, pode-se concluir que quanto maior a quantidade de dados utilizados para treinamento, maior a acurácia máxima de uma amostra. No entanto, quando analisado as médias de cada tratamento de dados, nota-se que o padrão não foi o mesmo, tendo k=5 apresentado acurácia maior que k=6 e k=7, o que a princípio foi uma surpresa pois esperava-se que com maior dados para treinamento seria possível gerar árvores com maior poder de generalização.

Assim, depois de uma análise mais detalhada, notou-se que esse fenômeno pode ser decorrente da quantidade relativamente pequena de dados para testes quanto k cresce, desta forma a acurácia média seria maior impactada pelos casos mais específicos de testes, que não foram completamente abrangidos no treinamento.

Não obstante, por mais que existam diferenças entre os resultados para as diferentes partições, essas distinções não chegam a ser significativas; o que retrata a homogeneidade dos dados fornecidos para treinamentos e testes.

3.2. Random Forest

Como já citado anteriormente, o *Random Forest* tem como principal característica a utilização de diversas árvores de decisões para se chegar na resposta. Desta forma, como esperado, o algoritmo apresentou um tempo de execução muito maior que o ID3, especialmente nos casos em que as quantidades de árvores são elevadas.

Ademais, foram feitas execuções com diferentes números de árvores e diferentes quantidades de partições dos dados de acordo com a validação cruzada k-fold. Os resultados desses procedimentos podem ser vistos no apêndice D.

Através dos resultados, nota-se que em todos os casos analisados, quanto maior a quantidade de árvores maior também a acurácia do modelo. Isso acontece principalmente porque quanto maior a quantidade de árvores menor a interferência do *overfitting* na saída. Isso também pode ser observado pelo fato das amostras com menos árvores também serem aquelas que apresentam acurácia mínima para qualquer k inteiro.

Ademais, também se pode observar um pequeno aumento na acurácia em casos de maiores divisões dos dados, de acordo com a validação k-fold. Com isso, essas maiores divisões acabam também por fornecerem uma quantidade maior de dados para treinamento e a árvore é capaz de adquirir maior poder de generalização.

Por fim, se comparado o algoritmo ID3 com *Random Forest*, nota-se que o segundo é capaz de adquirir maior acurácia com relação a modelagem realizada. Um dos possíveis motivos para isso acontecer é a quantidade reduzida de dados, não sendo capaz de fornecer uma grande quantidade de generalização para todas as árvores da floresta, havendo árvores que ”pioram”as estatísticas. Além disso, a reposição dos dados após a

escolha para cada árvore pode gerar situações onde nem todos os dados da partição são utilizados, o que ocasiona menos contextos de treinamento, diferentemente do ID3 que garante que todos os dados da partição são utilizados.

3.3. Redes Neurais

Como descrito na modelagem, a estrutura da rede neural criada possuía apenas uma camada intermediária, sendo medidas as acurácias do modelo utilizando diferentes neurônios na camada escondida (para todos os casos, considerou-se a camada de entrada contendo 16 neurônios e a camada de saída contendo 4 neurônios). Desta forma, considerando as 3 melhores opções afirmadas pelo professor Jeff Heaton, as acurácias obtidas em cada abordagem podem ser verificadas no apêndice E.

Dentre os resultados obtidos, é possível notar que, em geral, o aumento do número de épocas ocasiona também um aumento na acurácia do modelo. Esse fato pode ser comprovado pelo fato de quanto maior a quantidade de épocas, maior também a quantidade de análises feitas pelo modelo. No entanto, com esse aumento no número de análises, há também há possibilidade do modelo se adaptar demais aos dados de treinamento e perder sua capacidade de generalização, como pode ser observado na tabela 6, onde mesmo com a utilização de mais épocas, a linha 6 apresenta acurácia menor que a linha 5.

Além disso, nota-se, para o contexto em questão, que a utilização da terceira abordagem trouxe uma maior acurácia da saída, sendo, nesse caso, a melhor alternativa testada para utilização. Esse fato pode ter ocorrido pela característica homogênea dos dados, sem apresentar muitas possibilidades de *overfitting*. No entanto, vale ressaltar que para diferentes modelos e dados de treinamento, o resultado obtido poderia ser outro.

Por fim, ao comparar a rede neural com os outros algoritmos discutidos anteriormente, como o ID3 e o *Random Forest*, constatou-se que a rede neural apresentou a menor acurácia. É importante destacar que, à medida que a rede neural era aprimorada com um maior número de neurônios e se tornava mais complexa, sua acurácia aumentava. No entanto, essa melhoria de desempenho demandava uma capacidade computacional mais elevada para suportar a expansão da rede.

Por outro lado, as árvores de decisão, mesmo em suas configurações mais simples, já eram capazes de alcançar altos níveis de desempenho. Considerando o contexto em que o número de atributos de entrada era reduzido e a complexidade do cálculo da saída em relação à entrada não era elevada, as árvores de decisão se mostraram a solução mais eficiente e adequada.

4. Limitações

Dentre as principais limitações encontradas para o desenvolvimento, pode-se ressaltar a necessidade computacional necessária para a implementação dos algoritmos, em especial as redes neurais, que ao possuírem uma quantidade de camadas ou épocas muito elevadas, eram incapazes de serem executadas.

Ademais, considerando o terceiro algoritmo utilizado, a função de ativação ’relu”utilizada na camada intermediária pode apresentar problemas se os dados não forem tratados corretamente antes do treinamento, isso porque essa função faz com que o modelo deixe de aprender quando as entradas se aproximam de zero ou são negativas.[Data Science Academy, 2022]

Não obstante, é importante notar que os modelos foram treinados com um conjunto limitados de dados de saúde sem um contexto específico, como acidentes de carro, queda de lugar alto, entre outros. Desta forma, é imprescindível uma análise mais aprofundada do contexto de inserção do modelo, a fim de que possa se verificar a acurácia dele considerando os dados mais específicos.

Por fim, ressalta-se que, por maior a acurácia do modelo, todos eles estão expostos ao *overfitting*. Assim, dependendo do conjunto de dados e especialmente se fornecidos em baixa quantidade, o modelo pode não indicar a saída mais ideal para a situação.

5. Impactos

Ao fim deste processo de implementação e modelagem dos algoritmos, pode-se notar a importância da capacidade computacional para o desenvolvimento de soluções e modelos de aprendizado mais eficazes, sendo que muitos dos processos descritos anteriormente poderiam ter melhor aproveitamento se utilizado, por exemplo, de redes neurais mais complexas.

Desta forma, ressalta-se a importância do desenvolvimento tanto de softwares capazes de reproduzir modelagens com mais agilidade, quanto de hardwares capazes de acompanhar tal desenvolvimento. Por meio desse processo, o aprendizado de máquina, por exemplo, poderia ser utilizado em contextos cruciais da humanidade, como a amostra utilizado neste trabalho.

Ademais, vale ressaltar que, por mais que os modelos criados nesse e em outros trabalho apresentem bom aproveitamento, ainda é necessária uma discussão multidisciplinar para verificar a aplicabilidade deles em casos reais, tendo em vista que, considerando um caso extremo, um treinamento com *overfitting* pode custar a vida de uma pessoa.

Por fim, observa-se que, embora a análise realizada até o momento esteja focada em aprendizado de ”máquina”, a verdadeira utilidade dessas máquinas só é alcançada quando combinadas com a capacidade de modelagem humana e discussões interdisciplinares. Sem a contribuição do conhecimento humano e a integração de diversas perspectivas, as máquinas, por si só, teriam utilidade limitada, sendo a interação entre os sistemas inteligentes e a compreensão humana que realmente potencializa o valor e a aplicação dessas ferramentas em problemas complexos do mundo real.

Referências

AWS (2023). O que é uma rede neural? Technical report, AWS.

Carvalho, A. P. L. F. (2009). Perceptron multi-camadas (mlp). Technical report, USP.

Ceccon, D. Quantas camadas escondidas e quantos neurônios incluir numa rede neural artificial? 2020. Disponível em: https://iaexpert.academy/2020/05/04/quantas-camadas-escondidas-e-quantos-neuronios-incluir-numa-rede-neural-artificial/. Acesso em: 21 de agosto de 2024.

Data Science Academy, 2022. Data Science Academy. Deep Learning Book, 2022. Disponível em: ¡https://www.deeplearningbook.com.br/¿. Acesso em: 21 de agosto de 2024.

EBAC (2024). O que é random forest? Technical report, EBAC.

Ganesan, T. Backpropagation in neural network. 2022. Disponível em: https://www.geeksforgeeks.org/backpropagation-in-neural-network/. Acesso em: 21 de agosto de 2024.

Heaton, J. (2005). *Introduction to Neural Networks for Java*. Heaton Research.

Masters, D. and Luschi, C. (2018). Revisiting small batch training for deep neural networks.

Rabello, E. B. (2019). Cross validation: Avaliando seu modelo de machine learning. *Medium*.

Simon, 1983. WHY SHOULD MACHINES LEARN? In: Machine Learning. [S. l.]: Elsevier, 1983. p. 25–37. E-book. Disponível em: https://doi.org/10.1016/B978-0-08-051054-5.50006-6.

6. Apêndice A - Fórmula para o cálculo de entropia

$$H = -\sum_{i=1}^{\mathrm{n}} p_i \times log_2(p_i)$$

- n é o número de classes na feature.
- p_i é a probabilidade de ocorrência da classe i no conjunto de dados.

```
def entropy(labels):
    label_counts = Counter(labels)
    total_labels = len(labels)
    entropy_value = 0.0
    for label_count in label_counts.values():
        probability = label_count / total_labels
        entropy_value -= probability * math.log2(probability)
    return entropy_value
```

7. Apêndice B - Fórmula para o ganho de informação

$$G = H_0 - (\frac{|E|}{|T|} * H_e + \frac{|D|}{|T|} * H_d)$$

- G corresponde ao ganho de informação.
- H_0 corresponse a entropia inicial.
- E corresponde ao subconjunto à esquerda da separação.
- D corresponde ao subconjunto à direita da separação.

```
def information_gain(parent_labels, left_labels, right_labels):
    parent = entropy(parent_labels)
    left = entropy(left_labels)
    right = entropy(right_labels)

    left_weight = len(left_labels) / len(parent_labels)
    right_weight = len(right_labels) / len(parent_labels)

    return parent - (left_weight * left + right_weight * right)
```

8. Apêndice C - Resultados obtidos com o algoritmo ID3

Tabela 1. Resultados obtidos com ID3 para diferentes números de partições

k	Partição 1	Partição 2	Partição 3	Partição 4	Partição 5	Partição 6	Partição 7	Média
3	89,20	90,80	89,00					89,67
4	92,27	88,80	91,20	90,40				90,67
5	92,00	90,67	88,67	93,00	90,33			90,93
6	92.40	88.80	90.00	87.60	93.20	90.00		90.33
7	92.56	90.70	89.25	90.19	89.72	93.46	89.72	90.80

9. Apêndice D - Resultados obtidos com o algoritmo *Random Forest*

Tabela 2. Resultados obtidos com *Random Forest* para 3 partições

Árvores	Partição 1	Partição 2	Partição 3	Média
2	83,20	79,20	76,20	79,53
4	88.00	82.80	87.40	86.07
8	89.60	83.40	87.00	86.67
16	89.40	88.00	90.00	89.13

Tabela 3. Resultados obtidos com *Random Forest* para 4 partições

Árvores	Partição 1	Partição 2	Partição 3	Partição 4	Média
2	73.07	78.93	78.67	75.20	76.47
4	89.07	83.20	86.40	86.93	86.40
8	89.33	86.13	86.93	89.33	87.93
16	91.20	87.73	88.27	90.67	89.47

Tabela 4. Resultados obtidos com *Random Forest* para 5 partições

Árvores	Partição 1	Partição 2	Partição 3	Partição 4	Partição 5	Média
2	84.00	80.33	77.67	84.00	85.33	82.27
4	83.33	85.67	82.00	82.00	85.67	83.73
8	90.33	88.67	84.33	87.67	89.00	88.00
16	91.33	90.67	87.00	88,33	89.67	89.40

10. Apêndice E - Resultados obtidos com a Rede Neural

Tabela 5. Resultados da *Rede Neural* considerando 10 neurônios escondidos

Épocas	Partição 1	Partição 2	Partição 3	Partição 4	Média
2	64.00	53.87	55.20	59.47	58.13
5	71.47	65.07	64.80	69.07	67.60
16	81.87	73.07	71.47	72.53	74.73
35	85.87	74.93	82.67	81.87	81.33
42	84.80	77.87	83.73	82.40	82.20

Tabela 6. Resultados da *Rede Neural* considerando 14 neurônios escondidos

Épocas	Partição 1	Partição 2	Partição 3	Partição 4	Média
2	62.93	56.27	57.87	62.67	59.93
5	69.33	64.00	65.60	69.60	67.13
16	78.67	74.13	78.13	76.80	76.93
35	86.40	82.40	81.33	83.20	83.33
42	84.53	80.53	85.33	81.33	82.93

Tabela 7. Resultados da *Rede Neural* considerando 24 neurônios escondidos

Épocas	Partição 1	Partição 2	Partição 3	Partição 4	Média
2	64.80	57.33	56.27	62.40	60.20
5	66.93	65.60	67.20	67.20	66.73
16	78.40	71.73	77.60	77.07	76.20
35	85.60	83.20	88.00	83.73	85.13
42	88.27	86.13	86.13	87.73	87.07

11. Apêndice F - Informações adicionais

- Tempo para realização do trabalho: Gabriel Moro Conke - 18 horas.
- Link para as implementações: https://github.com/Conke94/machine-learning