Sistemas Inteligentes - CSI30
Modelagem com Aprendizado de Máquina (Simbólica e Neural)

Gabriel Leão Bernarde[1], **Gustavo Riodi Nakamura**[2]

[1] Curitiba – Paraná - Brazil

[2]Universidade Tecnológica Federal do Paraná (UTFPR)

{gabrielbernarde, gustavonakamura}@alunos.utfpr.edu.br

1. Definição do Problema

Nesta tarefa, o objetivo principal é utilizar sinais vitais já coletados de outros acidentes, juntamente com as respectivas classes de gravidade, para treinar modelos de aprendizado de máquina que possam prever a gravidade do estado de saúde de acidentados. A gravidade do estado de saúde dos acidentados é classificada em quatro categorias, como especificado no enunciado do projeto:

- **Classe 1**: Crítico
- **Classe 2**: Instável
- **Classe 3**: Potencialmente Estável
- **Classe 4**: Estável

A tarefa envolve a aplicação de três diferentes abordagens de aprendizado de máquina:

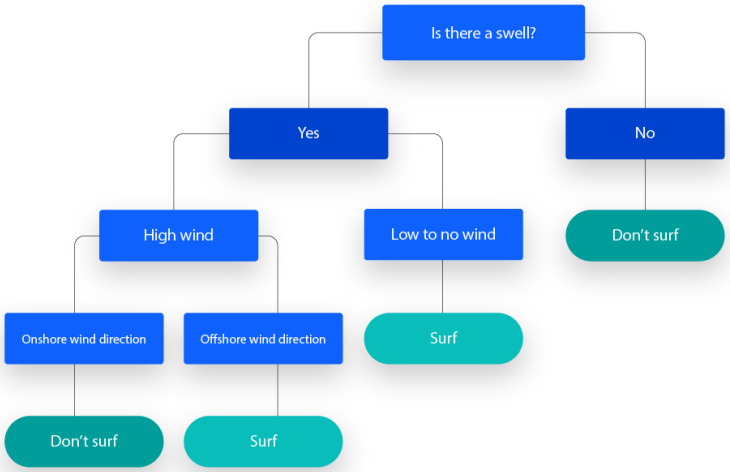
1. **Aprendizagem Simbólica: ID3**,
2. **Aprendizagem Simbólica: Random Forest**,
3. **Redes Neurais**.

Os modelos devem ser treinados com os dados dos sinais vitais existentes (arquivo do excel chamado `treino_sinais_vitais`) para prever a gravidade de novos pacientes, sem a necessidade de se ter a fórmula específica do cálculo. Assim, o objetivo é analisar os modelos e suas diferenças, bem como suas vantagens e limitações além de também discutir os impactos de cada abordagem em cenários que envolvem seres humanos.

2. Árvores de decisão

2.1. Fundamentação teórica

Arvore de decisão é um algoritmo de machine Learning utilizado para tarefas de classificação e regressão. Sua estrutura consiste em um nós, ramos e folhas, os nós representam os atributos ou condições que dividem os dados, os ramos correspondem aos possíveis valores e as folhas as decisões ou resultados finais.[IBM 2024a]Abaixo, um exemplo de árvore de decisão do site da IBM que mostra as regras de decisão para um surfista avaliando se deve entrar no mar



2.1.1. ID3 (Aprendizagem Simbólica)

O algoritmo ID3(Iterative Dichotomiser 3) é um dos algoritmos conhecidos para a construção de árvores de decisão. Desenvolvido em 1986 por Ross Quinlan, ele utiliza conceitos de entropia e ganho de informações para avaliar as divisões das árvores[IBM 2024a]

2.1.2. Entropia

A entropia é uma medida para calcular a impureza dos valores da amostra, seus valores variam entre 0 e 1; ela é usada para calcular o ganho de informação. Quanto mais baixa a entropia, mais organizados e previsíveis estão os dados, facilitando a separação em classes. Já a entropia alta tem dados desordenados e imprevisíveis, o que dificulta a separação em classes. A fórmula é a seguinte:

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

2.1.3. Ganho de informação

O ganho de informação representa a diferença entre a entropia original e depois de uma divisão de um determinado atributo; no ID3, são escolhidos os atributos com maior ganho de informação, pois isso garante a escolha de subconjuntos mais homogêneos e menor incerteza. Podemos contextualizar com a árvore da figura 1; lá o nó raiz pergunta se tem ondulação. Essa pergunta é a mais decisiva para o surfista decidir se irá para o mar ou não, pois sem onda não tem como ele surfar, portanto é a que tem maior ganho de informação, a fórmula do ganho de informação é a seguinte:

$$IG(T, A) = Entropia(T) - \sum_{v \in Valores(A)} \frac{|T_v|}{|T|} \cdot Entropia(T_v)$$

2.1.4. Discretização

Discretização é o processo de converter dados contínuos em dados discretos, dividindo uma faixa de valores em bins, um exemplo seria a divisão de uma temperatua em graus celsius, uma faixa de valores de 0 a 30 poderia ser dividida em baixa para valores de 0 a 15, media para valores de 16 a 23 e alta para valores de 23 a 30

2.2. Metodologia

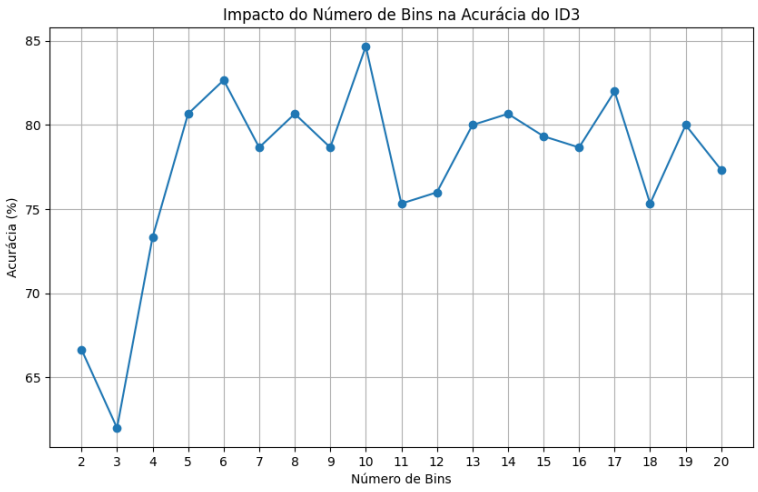
Para a implementação do ID3, foram testados na árvore os parâmetros qPa, pulso e respiração, e foram analisados dois parâmetros ajustaveis principais, são eles a quantidade de bins e a porcentagem de divisão do treino

1. **Quantidade de bins:**
   Foram feitos diversos testes, variando a taxa de bins entre 2 e 20, esses números foram escolhidos pois 2 é o mínimo número que possa gerar uma divisão para a árvore de decisão e 20 pois é o número máximo de divisões para os parâmetros qPa e respiração, os testes foram iniciados com a taxa de treinamento em 90% e teste em 10%, após conseguir a melhor divisão de bins, foi regulada a porcentagem de treino e teste

2. **Porcentagem de treino e teste:**
   Após conseguir extrair a maior acurácia, foram feitos testes com a porcentagem variando de 5% de teste e 95% de treino para 95% de teste e 5% de treino

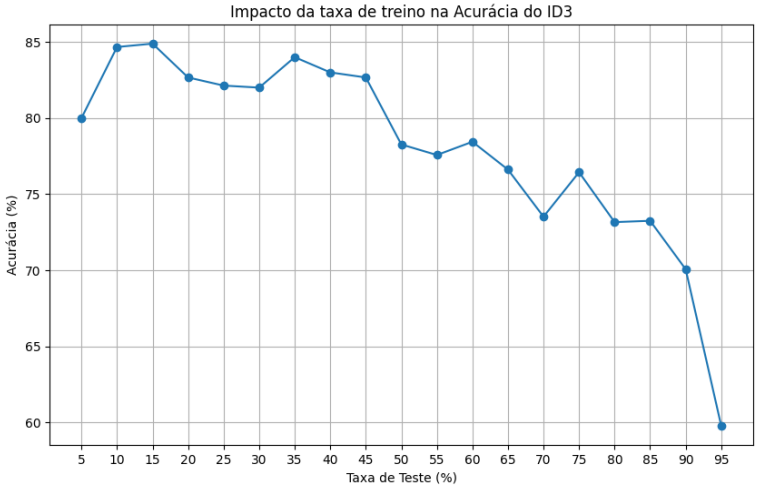
2.3. Resultados

Com a execução do algoritmo com diferentes bins variando de 2 a 20, os resultados foram os do gráfico abaixo:



Podemos ver que um número de bins muito baixo deixou uma acurácia baixa, porém a partir do número 5, gerou uma certa estabilidade, não variando tanto, porém a acurácia mais alta não foi com o maior número de bins e sim com 10 bins.

Após testar os bins, usamos o número de 10 bins para analisar qual a melhor divisão entre treino e teste e o resultado foi o seguinte:



O treino começou com 80 % de acurácia com 5% de teste e 95% de treino, continuou relativamente estável até os 45% de teste e 55% de treino e passou a diminuir após continuar diminuindo a taxa de treino. Isso mostra que com taxas de treinamento muito pequenas, a acurácia tende a ser baixa, porém, a taxa no máximo não necessáriamente trará a maior acurácia possível, portanto se faz necessário sempre fazer uma análise da divisão de forma a encontrar o melhor resultado

3. Random Forest (Aprendizagem Simbólica)

O Random Forest é um algoritmo de aprendizado de máquina que combina múltiplas árvores de decisão independentes para melhorar a precisão das previsões, fazendo com que exista uma diminuição do risco de overfitting e também uma melhor robustez do modelo se comparado com a própria árvore de decisão, uma vez que utiliza a estratégia de variabilidade e generalização.
Segundo a IBM, ”enquanto decision trees consideram todas as possíveis divisões de características, random forests selecionam apenas um subconjunto dessas características” (IBM, 2024). O que, de forma simplificada, significa que ao considerar toda a variabilidade potencial nos dados, é possível reduzir a chance de acontecer o overfitting (propensão do modelo de se ajustar demais aos dados de treinamento) e também o surgimento de problemas com viés nos dados.

3.1. Metodologia

Para a implementação, a Random Forest possui três principais parâmetros, sendo eles: Profundidade máxima da árvore (maxDepth), o Número de árvores (nEstimators) e o Número de características amostradas (maxFeatures).

Variáveis do algoritmo e seus impactos (tradeoffs)

1. **Profundidade máxima da árvore:** Limita o número de nós que a árvore pode ter. Valores muito altos para profundidade podem resultar em overfitting (fica

excelente para os dados de treinamento, mas ruim para dados novos) e valores muito baixos em underfitting (modelo com baixa capacidade de generalização).

2. **Número de árvores a serem treinadas:** Cada árvore é treinada de maneira independente com uma amostra aleatória do conjunto de dados. Assim, quanto mais árvores melhor tende a ser a precisão (menor variância) do modelo, porém o custo computacional, a complexidade e o tempo de processamento pode aumentar significativamente, o que faz necessário encontrar um ponto de equilíbrio entre a melhora da precisão e o custo.

3. **Número de características amostradas:** Controla quantas características são consideradas para cada divisão em uma árvore de decisão. O número de caracterísiticas muito baixo pode fazer com que o modelo não aprenda bem os padrões, e um número muito alto pode aumentar a chance de overfitting.
   Para o trabalho estamos utilizando o default da biblioteca RandomForestClassifier, que é o "sqrt" - número de características consideradas por divisão será a raiz quadrada do número total de características - já que esse valor promove um bom equilíbrio entre a generalização e a eficiência computacional. Outro valor que poderia ser utilizado seria o "log2", que considera o logaritmo de base 2 do número total de características, mas reduz ainda mais o número de características - pode acabar diminuindo a capacidade do modelo de capturar padrões importantes.

3.2. Explicação das escolhas dos valores das variáveis

Para a os testes, utilizamos o método `train_test_split` do pacote `sklearn`, que é usado para dividir um conjunto de dados em dois subconjuntos: dados de treinamento e dados de teste. Assim, considerando que temos 1500 dados no total, realizamos alguns testes para confirmar qual seria a melhor divisão e concluimos que 30% retornou uma acurácia melhor se aproximando mais de 91%, comparado com 40% com uma acurácia de aproximadamente 88%. (valores como 10% fazem com que o algoritmo fique acostumado com os dados de treinamento retornando também valores mais próximos de 85%, e valores onde se tem pouco treinamento fizeram com que a acurácia caísse ainda mais).
Para o algoritmo em si da Random Forest, utilizamos 100 como o número de árvores, uma vez que para os 1050 dados de treinamento não foi preciso um número muito grande, já que valores mais altos não variaram significativamente a acurácia e também acabaram só aumentando o tempo e o uso de recursos computacionais.
Complementando a escolha de 100 árvores, escolhemos a profundidade para cada árvore de 10, visando evitar o underfitting e também o overfitting, uma vez que para valores de profundidade muito baixa como por exemplo 2 (árvores simples, com capacidade limitada de capturar padrões) a acurácia já caia para 60%. Ademais, vale ressaltar que estamos falando de relativamente poucos dados e também poucas colunas.
Para a escolha do valor de `max_features`, tanto `"sqrt"` quanto `"log2"` resultaram em valores extremamente parecidos, o que é justificado pelo fato de estarmos trabalhando com apenas 3 colunas, ou seja, tanto `log2(3)` quanto `sqrt(3)` resultam em calcular apenas 1 único atributo na avaliação de cada divisão.

3.3. Resultados

Mesmo com a alteração de parâmetros e análises, a acurácia se manteve em aproximadamente 90%, que na área da saúde é um valor significativo, mas não faz com que a margem

de erro não gere implicações críticas de falsos positivos, negativos e de ética e justiça. Ademais, a estabilização da precisão em torno dos 90% pode ser explicada pela simplicidade e também pela quantidade dos dados utilizados (1500 no total, sendo 1050 para treinos e 450 para os testes), e analisando de maneira mais técnica, podemos também entender que deva existir algum tipo de ruído nos dados utilizados ou quem sabe características adicionais que não foram consideradas e que poderiam ser relevantes.

4. Redes Neurais

4.1. Fundamentação Teórica

As redes neurais são modelos computacionais inspirados no funcionamento dos neurônios do cérebro humano, nesse modelo, os neurônios são usados para tentar fazer uma imitação do processo de aprendizagem e tomada de decisão

4.2. Metodologia

Para o algoritmo de Redes Neurais, o começo do código é bem parecido com os anteriores (ID3 e Random Forest), uma vez que precisamos tratar os dados e também separar os 1500 exemplos em dados de treinamento e de teste. Para isso, utilizamos a função `train_test_split` da biblioteca `sklearn.model_selection` [Scikit-learn 2024], que possui como parâmetros de entrada os dados, a proporção os dados que serão utilizados para os testes e um estado inicial para ser possível replicar a mesma divisão dos dados posteriormente, caso o valor do random state selecionado seja o mesmo. Ademais, para as redes neurais, também foi necessário a utilização da `LabelEncoder e StandardScaler` para o ajuste dos rótulos para o intervalo iniciando em 0 ao invés de 1 (uma vez que o próprio problema já define os valores no intervalo de 1 a 4) - o que é feito não só para se ter uma certa padronização, mas também devido ao fato de que a biblioteca Keras, utilizada no projeto, espera por padrão rótulos numéricos iniciando em zero - e a normalização/padronização dos dados visando evitar problemas com valores maiores que poderiam acabar dominando a função de custo do modelo e assim afetar a convergência ou a performance do treinamento para os métodos de gradiente descendente.

4.3. Implementação

Para a modelagem do modelo da rede neural, utilizamos a API por camadas do `TensorFlow`, implementada por meio da classe `Sequential`. Optamos por essa abordagem porque o TensorFlow.js oferece duas opções para criação de modelos: a API por camadas ou a API principal com operações de baixo nível. Assim, após análises breves de outros códigos e também pesquisas, concluímos que a modelagem por camadas é mais simples, fácil de utilizar e atende bem às necessidades do projeto.

4.3.1. Camadas do Modelo

Utilizamos 2 camadas densas (fully connected), sendo a primeira uma com 15 neurônios utilizando a função de ativação ”Sigmoid” e a camada de saída com 12 neurônios,

que utiliza a função de ativação ”Softmax” que converte as saídas em probabilidades associadas às classes (neste caso, 12 classes) - para analises posteriores. A escolha da função de ativação da primeira camada como `sigmoid` foi feita considerando que a ativação por `ReLU` (Rectified Linear Unit), apesar de ser considerado por muitas pessoas superior à sigmoid e também à tanh (), não se saiu melhor nos testes, o que pode ser justificado por já estarmos normalizando previamente e também devido ao pequeno número de dados disponíveis para o treinamento. [No Apêndice A está disponível os resultados visuais dos testes entre a diferença entre relu e sigmoid]

ACTIVATION FUNCTION	SIGMOID	TANH	RELU
Range	0 to 1	-1 to 1	0 to Infinity
Vanishing Gradient Problem	Yes	Yes	No
Nature	Non Linear	Non Linear	Linear
Zero Centered Activation Function	No	Yes	No
Symmetric Function	No	Yes	No
Equation	y = 1/(1+e^(-x))	y = tanh(x)	{ xi if x >=0 0 if x <=0 }
Model Accuracy	Good	Very Good	Excellent

4.3.2. Compilação e Treinamento do Modelo

Para a compilação, utilizamos o parâmetro `loss='sparse_categorical_crossentropy'`, que especifica a função de perda a ser utilizada, que foi escolhida por ser adequada a problemas de classificação multiclasse, onde os rótulos de classe são representados por inteiros. Um outro parâmetro muito importante na compilação do modelo é o `optimizer=tf.keras.optimizers.SGD(learning_rate=0.2)`, que define o otimizador a ser utilizado no treinamento do modelo, que neste caso é o *Stochastic Gradient Descent* (SGD) - O SGD é um algoritmo de otimização que ajusta os pesos da rede neural com base no gradiente da função de perda em relação aos pesos. Dessa forma, a taxa de aprendizado (`learning_rate`) foi configurada com o valor de 0.3, o que determina a magnitude dos ajustes feitos nos pesos a cada iteração. Em suma, valores muito altos ou muito baixos da taxa de aprendizado podem impactar significativamente o desempenho da rede neural, uma vez que se a taxa de aprendizado for muito alta, o algoritmo pode aprender rapidamente, mas com um desempenho inferior, devido a ajustes excessivos nos pesos e, por outro lado, uma taxa de aprendizado muito baixa pode fazer com que o modelo aprenda de forma muito lenta, possivelmente não conseguindo aprender direito dentro do número de épocas estabelecido (nossa escolha de 0.3 também foi baseada na quantidade totais de dados disponíveis e testes realizados) - mais informações da análise disponível no Apêndice B.
Para treinar o modelo utilizamos a função `model.fit()` e os parâmetros fornecidos foram, os `epochs=120`, que definem o número de passagens completas pelos dados durante o treinamento, os `batch_size=10` que determinam que os dados serão processados em lotes de 10 amostras por vez e o parâmetro `validation_split=0.1`, que

separa 10% dos dados para validação, permitindo monitorar o desempenho do modelo em dados não vistos. (O `shuffle=True` garante que os dados sejam embaralhados antes de cada época, evitando que o modelo aprenda padrões indesejados)

5. Resultados

Ao analisar os resultados obtidos após o treinamento do modelo, verificamos que o desempenho foi consistente com as expectativas, considerando o tamanho limitado dos dados. A acurácia alcançada foi de aproximadamente 90%, o que indica que o modelo conseguiu identificar corretamente as classes de gravidade em uma boa parte das amostras e uma coisa extremamente relevante de ser comentada é o fato de que a rede neural, apesar de não ter atingido 100%, para os valores onde ela errou, todos eles estiveram muito próximos do valor real, o que significa que para situações como a analisada - relacionada à saúde - o algoritmo se sairia muito bem. [Mais informações referente a isso no Apêndica C]

Um outro ponto importante, relacionado mais à modelagem em si, foi a enorme importância e impacto de cada um dos parâmetros - desde os ajustes nos dados utilizados (normalização e reclassificação) até o treinamento em si. A escolha dos parâmetros, como a taxa de aprendizado, o número de épocas, e o tamanho do lote, tiveram um impacto significativo no desempenho do modelo, o que pode ser observado mais nos resultados visuais do Apêndice. Assim, fica evidente que cada um desses ajustes deve ser cuidadosamente otimizado para garantir que o modelo não apenas aprenda de maneira eficiente, mas também se generalize bem para dados não vistos. Resumidamente, conseguimos observar como uma simples mudança nos parâmetros pode alterar drasticamente o resultado da rede neural, evidenciando a complexidade e a sensibilidade desse processo de modelagem.

6. Impactos de cada Modelagem

1. **Algoritmo ID3**
   O algoritmo ID3 de árvore de decisão teve uma acurácia máxima de 87%. Para chegar nesse nível foi necessário analisar a cada mudança tanto na quantidade de bins, quanto na porcentagem de treino e teste;
2. **Random Forest**
   Para o Random Forest, a maior porcentagem obtida foi de 92%, e foi necessário o ajuste de três principais variáveis: profundidade máxima da árvore, número de árvores a serem treinadas e número de características amostradas;
3. **Redes Neurais**
   Conseguimos uma porcentagem de acurácia de aproximadamente 92%, e para chegar nesse valor foi necessário inúmeros ajustes e análises dos parâmetros das funções, como por exemplo o número de épocas, a taxa de aprendizado, o tamanho do lote, e vários outros.

Os 3 modelos tiveram a acurácia próxima de 90%, um valor bom, porém, é necessário analisar com qual tipo de dado está sendo trabalhado (quantidade e qualidade), no contexto geral, o resultado obtido é um valor bom, pois é uma porcentagem alta, porém, no contexto da saúde, principalmente nesse modelo de dados específicos, que é sobre risco de vida, 10% de chance de erro é um número significativo, portanto, não se deve confiar 100% no sistema, o ideal é usar-lo para auxílio, porém não se deve depender dele os 3 modelos 90é um dado legal mas que 10

7. Papel dos membros

Primeiramente, foi feita uma chamada para ler, entender o enunciado em conjunto e traçar os requisitos. Após isso, ficou definido que o membro Gabriel ficaria responsável pela implementação do algoritmo ID3 e o Gustavo pela implementação do algoritmo de machine learning e redes neurais. Após finalizadas as implementações, foi feita uma nova chamada, na qual cada um explicou detalhadamente para o outro o funcionamento de cada algoritmo e foram feitos testes em conjunto, a quantidade de horas exercidas por cada membro foi de aproximadamente 19.

8. Referências Bibliográficas

[IBM 2024b], [EBAC 2024], [Tech 2024], [Scikit-learn 2024].

References

EBAC, E. (2024). Random forest: O que é, onde é utilizado e por que é importante? Acesso em: 1 dez. 2024.

IBM (2024a). O que é uma árvore de decisão? Acesso em: 10 dez. 2024.

IBM (2024b). Random forest. Acesso em: 1 dez. 2024.

Scikit-learn (2024). sklearn.model$_s election.train_t est_s plit. Acessoem : 10dez.2024.

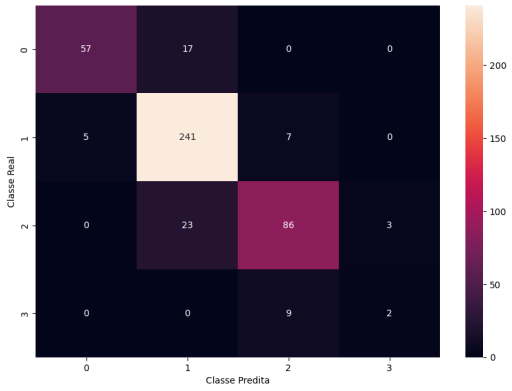
Tech, D. (2024). O que é e como funciona o algoritmo random forest. Acesso em: 1 dez. 2024.

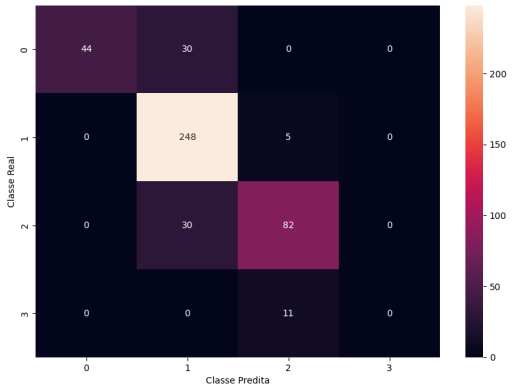
Repositório no GitHub

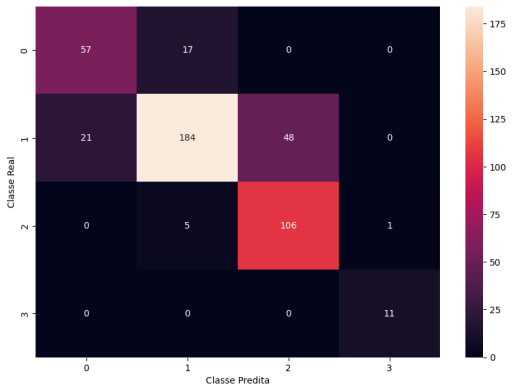
https://github.com/Nkamura/Modelagem_com_Aprendizado_de_Maquina

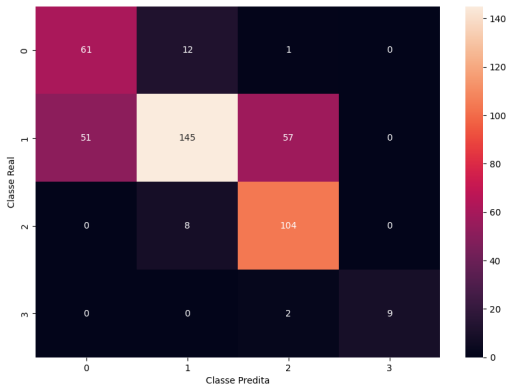
Apêndices

A. Comparação da função de ativação Sigmoid com Relu

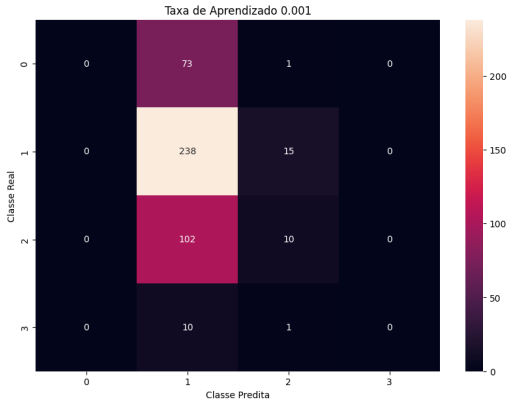


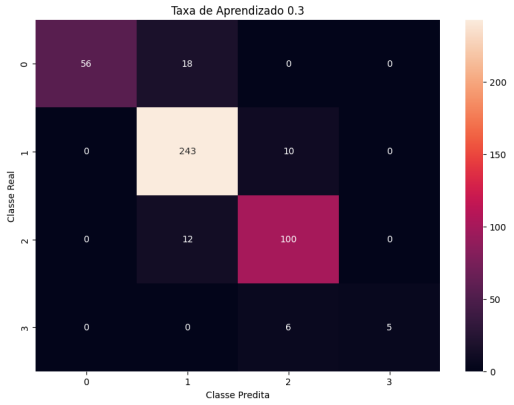


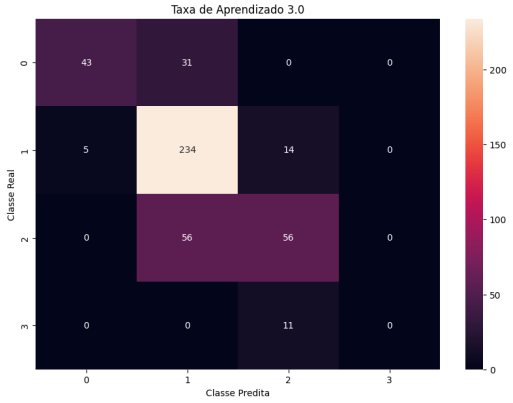


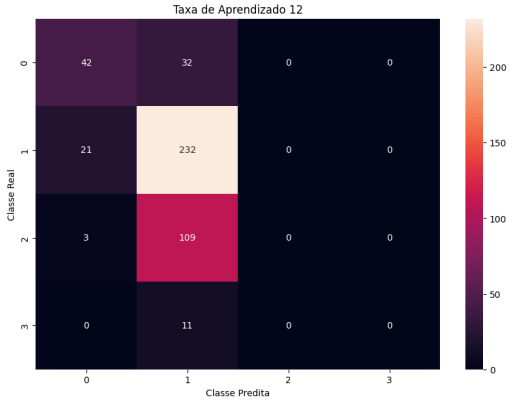


B. Análise de quanto a taxa de aprendizagem afeta o resultado final

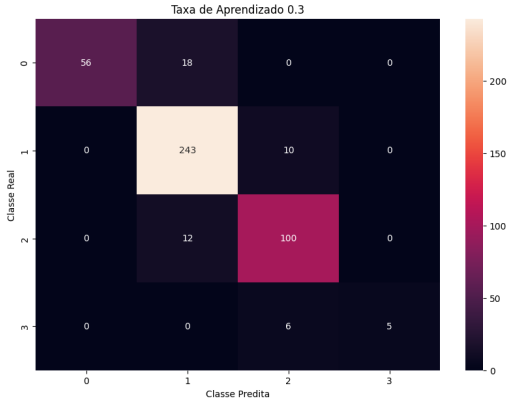








C. Análise do resultado das Redes Neurais



C.1. Conclusão

Dada a imagem, fica bem nítido que o algoritmo realmente aprendeu com os dados, uma vez que para os valores em que ele errou, os valores reais e preditados são próximos, não tendo nenhum caso onde o predito foi 0 e o real 3 ou vice e versa.