FIAP

Pós-Graduação IA para Devs

Diego de Moraes Pereira Sonnenthal

Guilherme Munhoz Lima

Leonardo Fernandes de Oliveira

Otimização de Redes Neurais utilizando Algoritmos Genéticos para Previsão de Sobrevivência no Titanic

Relatório entregue para cumprimento da atividade exigida na Fase 2.

São Paulo

2024

Sumário

I. INTRODUÇÃO

Contexto

O desafio de prever a sobrevivência dos passageiros do Titanic é um clássico problema de classificação na ciência de dados.

Este projeto utiliza algoritmos genéticos para otimizar os hiperparâmetros durante a criação de uma rede neural, com o objetivo de melhorar a acurácia na previsão da sobrevivência dos passageiros do Titanic.

Hiperparâmetros de uma Rede Neural

- Número de unidades (neurônios) em cada camada
- Número de camadas ocultas
- Tipo de otimizador (por exemplo, Adam, SGD, RMSprop)
- Taxa de aprendizado
- Função de ativação
- Batch size
- Número de épocas

Hiperparâmetros Escolhidos para Otimização

- Número de unidades (neurônios) por camada
- Número de camadas ocultas
- Tipo de otimizador

Esses hiperparâmetros foram escolhidos por meio de heurística, pressupondo serem os mais relevantes para o caso em comento, a influenciar na capacidade de aprendizado e generalização do modelo. O número de unidades e camadas ocultas afeta diretamente a complexidade e a capacidade de representação da rede neural, enquanto o tipo de otimizador influencia a eficiência e a rapidez com que o modelo converge durante o treinamento.

Métodos de Otimização

Além dos algoritmos genéticos, já são usados métodos convencionais de otimização como gradiente descendente, busca em *grid* e busca aleatória. Inclusive, heurísticas, como *simulated annealing* e otimização por enxame de partículas, também são aplicadas em contextos similares.

Considerando que se trata de uma atividade voltada ao uso do algoritmo genético e que em termos de conteúdo programático foram explorados classificadores convencionais de aprendizagem de máquina na Fase 1, o objetivo deste relatório demonstrar qual foi o desempenho ao se aplicar algoritmos genéticos para otimizar uma rede neural, comparando seu desempenho com classificadores convencionais: K-Nearest Neighbors (KNN), Random Forest, Support Vector Machine (SVM) e Naive Bayes.

O sucesso do projeto foi medido pela melhoria da acurácia (e validação cruzada) do modelo otimizado por algoritmos genéticos em comparação com os classificadores convencionais.

II. METODOLOGIA

Coleta e Pré-processamento dos Dados

Os dados utilizados neste estudo são provenientes do *dataset* Titanic, disponível no arquivo digital de Stanford, conforme URL no código-fonte.

O pré-processamento dos dados incluiu:

- Tratamento de valores ausentes, por meio da função *dropna;*
- Transformação de variáveis categóricas ("*sex*") em variáveis *dummy*, por meio da função *LabelEnconder*, da biblioteca *scikitlearn;*
- Normalização dos dados numéricos.

Definição da Rede Neural

A rede neural foi definida utilizando a biblioteca *Keras*. A arquitetura da rede inclui camadas densas totalmente conectadas com funções de ativação *ReLU* e uma camada de saída com função de ativação sigmoide para a previsão binária.

As funções de ativação, como *ReLU*, *Sigmoid* e *Tanh*, introduzem não linearidade nas redes neurais, permitindo que elas capturem relações complexas entre as entradas e saídas desde a década de 80. Isso foi um marco significativo, pois possibilitou resolver o problema XOR, um problema clássico na ciência da computação e redes neurais, onde a saída não pode ser separada linearmente em relação às entradas. Redes neurais simples (*perceptrons* de camada única) não conseguem resolver o problema XOR, porque ele não é linearmente separável. Com a introdução de funções de ativação não lineares e camadas ocultas, redes neurais profundas podem aprender representações complexas e abstrações, permitindo a solução de problemas como o XOR.

As funções de ativação e de saída foram escolhidas com base na prevalência vista nos códigos-fontes em geral e a premissa de que são mais eficientes pelo virem sendo usadas ostensivamente.

Classificadores Convencionais

Da mesma maneira, foram definidas funções simples, com base na biblioteca *scikitlearn*, no que se refere aos classificadores convencionais, parâmetros foram implementados conforme a praxe, como 100 *estimators* no Random Forest e 05 vizinhos no *KNN*, não cabendo maior aprofundamento, para que não se incorra em divagação e consequente fuga do objeto do relatório.

Para visualizar o progresso do treinamento do modelo em tempo real, utilizou-se o *call-back DynamicPlotCallback*, pois permite visualizar graficamente como as métricas de desempenho do modelo evoluem durante o treinamento, ajudando a identificar problemas e realizar ajustes no modelo ou nos hiperparâmetros de forma iterativa, com base nos gráficos gerados, facilitando inferir se o modelo está convergindo para uma solução próxima da ótima ou se necessita de mais treinamento; além de fornecer uma maneira visual de documentar o processo de treinamento, o que é útil para apresentações e relatórios, como este.

Algoritmo genético para otimização

Os algoritmos genéticos são métodos de busca heurística, inspirados no processo de seleção natural. Eles são amplamente utilizados para encontrar soluções aproximadas para problemas de otimização e busca.

Os indivíduos são soluções para o problema e os genes do indivíduo são variáveis definidos pela modelagem do problema, que pode ser uma função matemática ou pode se basear em heurísticas, formando o cromossomo.

Estabelece-se uma população inicial, de modo randômico ou não, variando conforme máximas de experiência, regras de negócio etc.

A implementação típica de um algoritmo genético inclui quatro componentes principais: função de aptidão, cruzamento, mutação e seleção.

A função de aptidão avalia o quão boa é uma solução potencial (ou indivíduo) em resolver o problema em questão. Ela atribui uma pontuação a cada indivíduo da população, determinando sua adequação para o problema.

O cruzamento combina duas soluções (pais) para gerar novas soluções (filhos). Ele permite a troca de informações entre indivíduos, promovendo a diversidade genética, permitindo a possibilidade de termos melhores soluções.

A mutação introduz pequenas alterações em indivíduos, ajudando a manter a diversidade genética e permitindo a exploração de novas áreas do espaço de soluções.

A seleção escolhe quais indivíduos serão mantidos na próxima geração com base em suas aptidões. Métodos comuns incluem seleção por torneio e roleta.

Os algoritmos genéticos foram utilizados para otimizar os seguintes hiperparâmetros da rede neural:

- Número de neurônios em cada camada.
- Taxa de aprendizado.
- Número de camadas ocultas.

A implementação do algoritmo genético se valeu da biblioteca DEAP (*Distributed Evolutionary Algorithms in Python*). No caso, foram utilizados o módulo *deap.base, base.Fitness*, que é uma classe, para definir a função de aptidão. É usada para calcular o valor de aptidão dos indivíduos, por meio do método *creator.create,* que recebe FitnessMax, base.Fitness e a atribuição de um peso.

Cumpre acrescentar que *creator.create* é uma função utilizada na criação de novas classes. É utilizada para definir tipos personalizados para indivíduos e aptidões.

Ainda, foi usada *base.Toolbox*, uma classe usada na configuração e no armazenamento das funções genéticas (inicialização, avaliação, cruzamento, mutação, etc.), por meio de *toolbox.register*.

Quanto a *tools.initRepeat*, do módulo *deap.tools*, trata-se de função, que inicializa um contêiner, com elementos repetidos, gerados por uma função, utilizada no caso para criar indivíduos e populações.

Já *tools.cxTwoPoint,* refere-se a uma função, de cruzamento, que aplica o cruzamento de dois pontos, entre dois indivíduos, servindo, no caso, como a função de cruzamento.

Quanto à *tools.mutFlipBit*, opera como a função de mutação, ou seja, inverte os bits de um indivíduo, com uma determinada probabilidade, na toada de conferir estocasticidade ao algoritmo, assim como na natureza, em que, mesmo indivíduos com genética desfavorecida em certos aspectos podem gerar indivíduos de genética perfeita para determinada situação.

Salienta-se que *tools.selTournament* é a função de seleção por torneio, que escolhe indivíduos com base em torneios de tamanho especificado, método que pode ser mais lento, *verbi gratia*, que Seleção por Amostragem Estocástica Universal (*Stochastic Universal Sampling* - SUS), método pelo qual todos os indivíduos são selecionados em uma única rodada, uma variação da roleta. Não há uma razão específica para sua escolha no caso, mas, como não havia problema em escolher um método mais custoso, o do torneio foi escolhido.

Tendo em vista a natureza contínua do algoritmo, com a concatenação de etapas, os dados não podem se perder, o que justifica o uso de *tools.Statistics*. classe para calcular e armazenar estatísticas durante o processo evolutivo. No caso, serviu para armazenar logs do processo evolutivo, facilitando, inclusive, a análise do progresso (ver *tools.Logbook*).

Some-se que *HallOfFame* é um componente importante da biblioteca DEAP, pois mantém um registro dos melhores indivíduos encontrados ao longo do processo evolutivo, na toada de garantir coesão e concatenação, sem que os melhores indivíduos sejam perdidos devido à variabilidade introduzida pelas operações de cruzamento e mutação.

No mais, *algorithms.eaSimple,* módulo de *deap.algorithms*, foi a função usada para aproveitar um algoritmo evolutivo simples, já estruturado. Inclui ciclos de seleção, cruzamento e mutação, e atualiza a população. Aplicou o algoritmo genético à população de 10 indivíduos inicialmente estabelecida.

III. TESTES E AVALIAÇÃO

Salienta-se que a primeira tentativa foi interrompida em razão de não haver tempo hábil ante a necessidade de *hardware* robustos para trabalhar com classificadores de imagens, sendo que os testes iniciais ocorreram na tentativa de classificar números manuscritos, com base no *dataset* MNIST.

Uma vez modificado o escopo e escolhido o classificador de sobrevivência do Titanic, os modelos foram treinados e testados utilizando validação cruzada, e suas acurácias foram comparadas. Os classificadores KNN, Random Forest, SVM e Naive Bayes foram utilizados como referência.

O número de gerações pode ser definido manualmente e diversas configurações podem tornar a busca mais eficiente do ponto de vista de se aproximar da solução ótima, mas também é possível que esbarrem em problemas como *overfitting* ou *underfitting*. Do modo como configurado o algoritmo genético, já na primeira vez em que todas as células rodaram sem apresentação de erros, foi possível visualizar superioridade no desempenho tanto em relação aos classificadores convencionais, como em relação a uma rede neural profunda configurada com hiperparâmetros vistos rotineiramente em diversos códigos-fontes, sem personalização (servindo, pois, como ponto de partida para uma análise comparativa).

Há diversas técnicas de otimização de hiperparâmetros, sendo uma área em constante evolução, não havendo uma técnica que prevalece sobre as demais.

A Busca em Grid, por exemplo, é a exploração exaustiva de um espaço de hiperparâmetros predefinido. Testa todas as combinações possíveis, o que pode ser possível em casos menos complexos, porém a ideia do algoritmo genético é encontrar soluções sub-ótimas, em situações em que a exploração exaustiva é inviável.

E termos de notação Big O, temos tempo de execução da Busca em Grid é $O(k^n)$, então, se temos 3 hiperparâmetros e cada um pode assumir 10 valores diferentes, a Busca em Grid testaria 10^3=1000 combinações possíveis, mas para cada 1000 combinações de função de otimização, ainda há infinitas possibilidades em relação ao número de camadas e o número de neurônios. Se com uso de GPU T4 fornecida pelo Google Collab demorou

cerca de 15 minutos para rodar a célula em questão, seriam 250 horas no total apenas para ajuste do hiperparâmetro de função de otimização. Em resumo, por esta técnica, que garante que todas as combinações serão testadas, encontrando a configuração ótima de apenas um hiperparâmetro, seriam necessários mais de 10 dias para encontra-la, sendo muito caro computacionalmente para grandes espaços de hiperparâmetros e ineficiente em termos de tempo e recursos, quando há muitos hiperparâmetros ou valores possíveis, sendo que há, no caso, infinitas configurações de redes neurais profundas, com diferentes números de camadas e *percepetrons* por camada, além de diversas arquiteturas diferentes.

Comparado aos algoritmos genéticos, em que a ideia é encontrar soluções sub-ótimas, em situações onde a exploração exaustiva é inviável, em termos de notação Big O, a complexidade dos algoritmos genéticos depende do número de gerações, (g), do tamanho da população (p) e do número de avaliações de aptidão necessárias (e); e pode ser ser expressa como $O(g \cdot p \cdot e)$, então, no caso, uma população de 10 indivíduos, rodando por 10 gerações, pressupondo que cada avaliação de aptidão leva tempo constante, tendo em vista que houve pouca variação, conforme *logs* do código-fonte anexado, a complexidade seria O(5000). Apesar de cinco vezes maior, vale lembrar que a notação do parágrafo anterior se refere apenas a um dos hiperparâmetros, que só tem três valores possíveis, pois, no caso de número de camadas e de neurônios, seria infinito.

O Gradiente Descendente, por usa vez, é um método iterativo, usado para encontrar mínimos locais de uma função. Atualiza os parâmetros na direção negativa do gradiente da função de custo. Em termos de Notação Big O, a complexidade do Gradiente Descendente depende do número de iterações e do número de parâmetros. A notação Big O pode ser expressa como O(t·p). Ela é eficiente para problemas de otimização contínua e amplamente utilizado devido à sua simplicidade e eficácia, mas pode ficar presa em mínimos locais e está suscetível à explosão do gradiente e ao desaparecimento do gradiente, especialmente em redes profundas.

A Explosão do Gradiente ocorre quando os gradientes aumentam exponencialmente, resultando em atualizações de peso instáveis, o que é fácil de visualizar ao se ter em mente que são realizadas operações sobre os resultados da camada anterior sucessivamente, ou seja, quanto mais vezes realizam-se operações (quanto mais

camadas houver), dependendo da função matemática escolhida, maior a chance de explosão ou desaparecimento.

O Desaparecimento do Gradiente: Ocorre quando os gradientes se tornam extremamente pequenos, impedindo a atualização efetiva dos pesos, lembrando que as redes neurais são formadas por pesos e vieses.

Também se mostrou eficiente para grandes espaços de solução onde a busca exaustiva (como Busca em Grid) é impraticável, mas a eficiência prática pode variar com base na implementação e no *tuning* dos parâmetros específicos - a Otimização por Enxame de Partículas - inspirada no comportamento social de pássaros e peixes, em que as partículas exploram o espaço de soluções, movendo-se em direção à melhor posição encontrada por elas e por seus vizinhos, inicializado com um conjunto de partículas inicialmente distribuído aleatoriamente no espaço de soluções, sendo que o movimento de cada partícula (ajuste de posição) se dá com base em duas informações principais: Melhor posição individual (melhor posição encontrada pela própria partícula até o momento) e Melhor Posição Global (melhor posição encontrada por qualquer partícula no enxame). A atualização da velocidade de cada partícula leva em conta sua velocidade atual, a distância da melhor posição individual e a distância da melhor posição global; a posição de cada partícula é atualizada com base em sua nova velocidade; A função de aptidão (ou função objetivo) é avaliada nas novas posições, e as melhores posições são atualizadas conforme necessário; o processo é repetido por um número fixo de iterações ou até que um critério de convergência seja atendido.

Por fim, para não se debruçar sobre temática longa e complexa, fugindo do objetivo deste relatório, há, ainda, a Busca Aleatória (seleção aleatória de combinações de hiperparâmetros dentro de um espaço definido, menos exaustiva que a busca em *grid*); o *Simulated Annealing* (técnica probabilística de busca do ótimo global em um espaço de soluções, permitindo soluções menos ótimas, temporárias, para escapar de mínimos locais).

Superioridade dos Algoritmos Genéticos

Os algoritmos genéticos são muitas vezes superiores a métodos convencionais de otimização e seu resultado, quando aplicado a redes neurais profundas, tende a apresentar

desempenho superior ao de classificadores convencionais, especialmente em problemas com espaços de solução grandes e complexos. Eles são inspirados na seleção natural e utilizam operações como seleção, cruzamento e mutação para explorar eficientemente o espaço de hiperparâmetros, aumentando a probabilidade de encontrar uma configuração próxima do ótimo global.

Muitas vezes, testar todas as configurações de arquitetura é inviável em termos de hardware, tempo ou recursos devido ao grande espaço de busca e ao custo computacional associado ao treinamento de múltiplos modelos complexos. O uso de algoritmos genéticos permite uma exploração mais eficiente do espaço de hiperparâmetros, encontrando boas configurações sem a necessidade de testar exaustivamente todas as combinações possíveis.

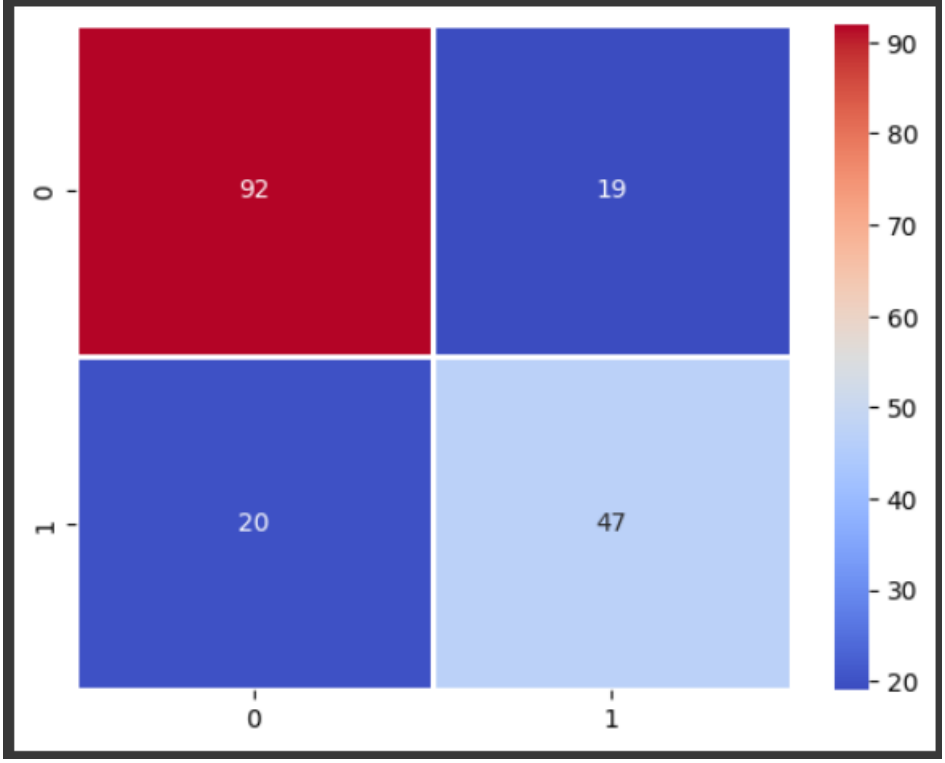
Desempenho dos Classificadores

Os resultados de treinamentos individuais dos classificadores “tradicionais”, pode sem observados conforme imagens abaixo, em que demostra os reports e matrizes de confusão:

```
KNN CV Accuracy: 0.8053541104784736
KNN Test Accuracy: 0.7808988764044944
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.83      0.83       111
           1       0.71      0.70      0.71        67

    accuracy                           0.78       178
   macro avg       0.77      0.77      0.77       178
weighted avg       0.78      0.78      0.78       178
```

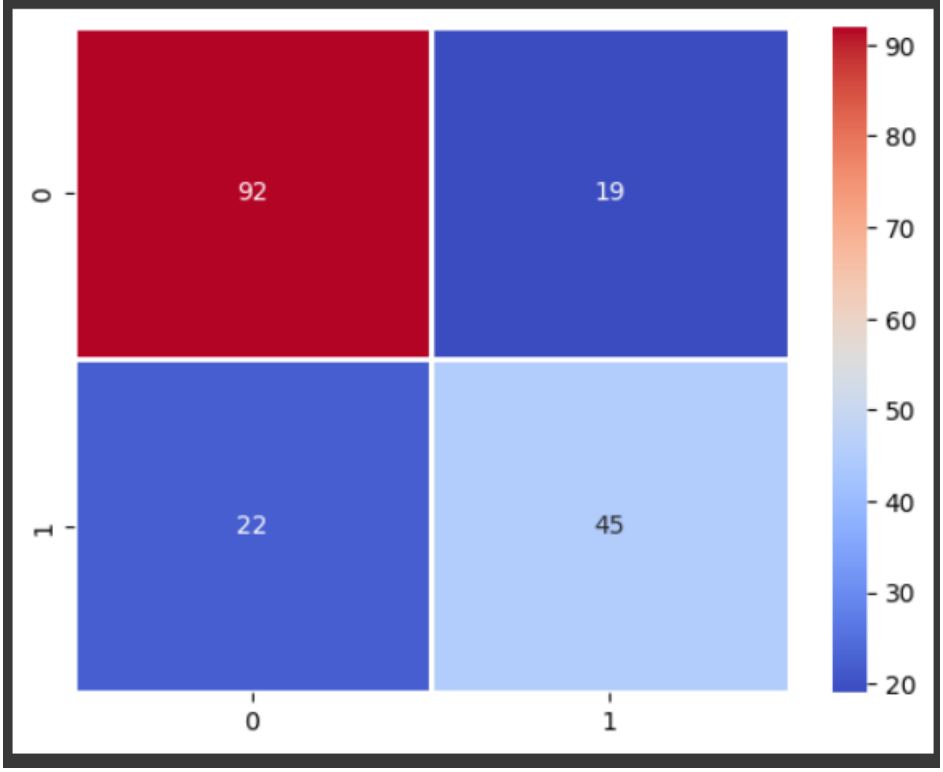


Report e matrix de confusão no modelo de KNN

```
Random Forest CV Accuracy: 0.8152332434322244
Random Forest Test Accuracy: 0.7696629213483146
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.83      0.82       111
           1       0.70      0.67      0.69        67

    accuracy                           0.77       178
   macro avg       0.76      0.75      0.75       178
weighted avg       0.77      0.77      0.77       178
```

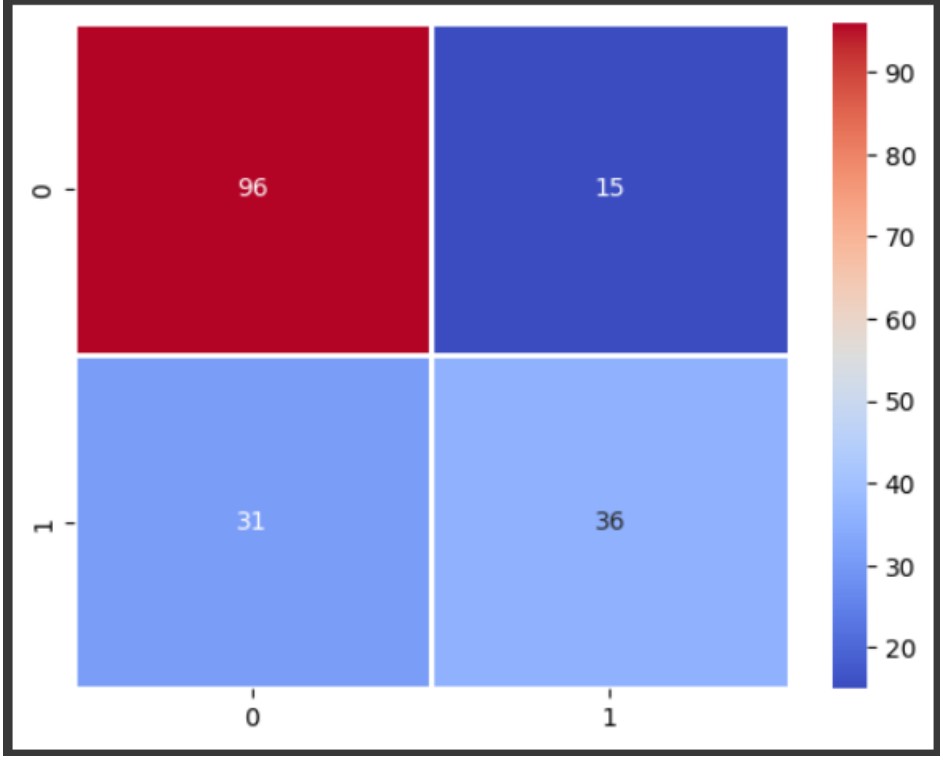


Report e matrix de confusão no modelo de Random Florest

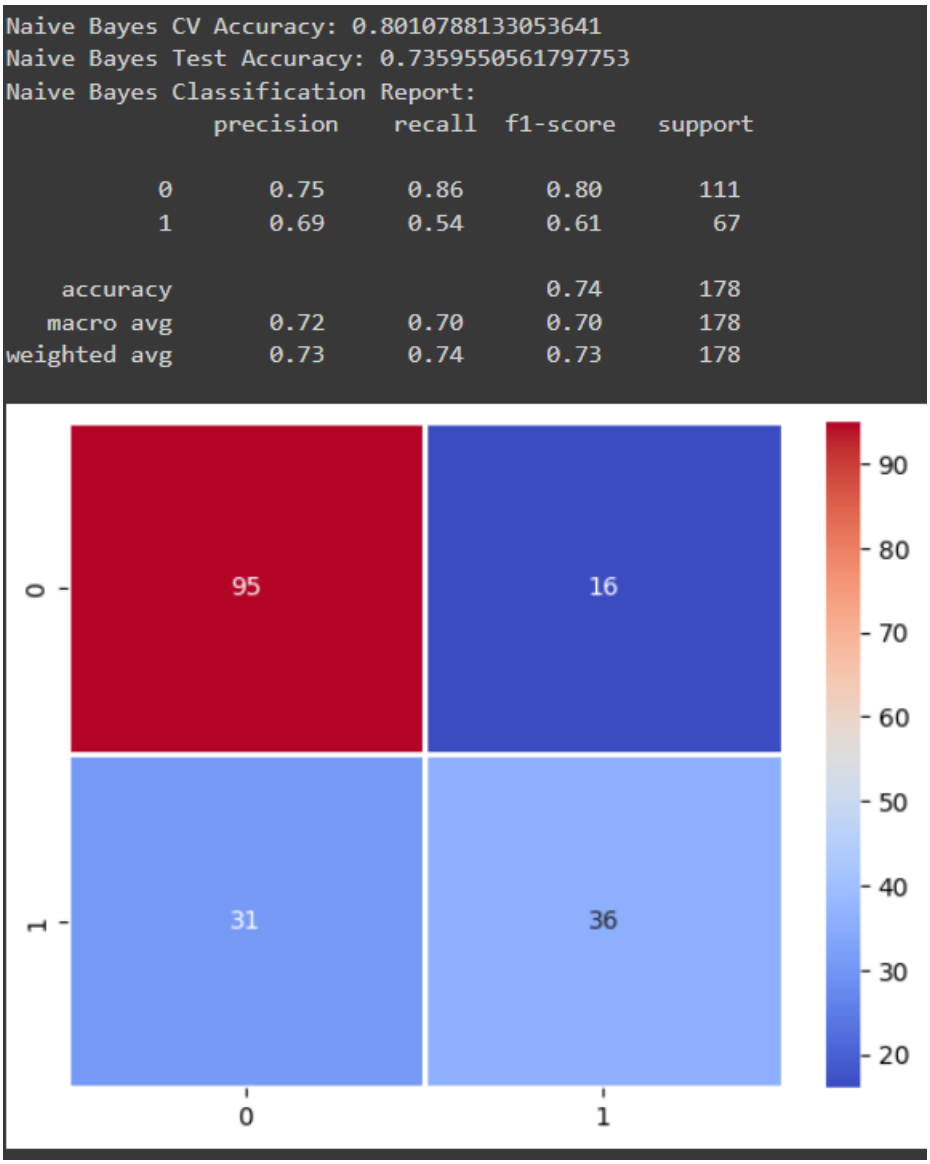
```
SVM CV Accuracy: 0.796923384277295
SVM Test Accuracy: 0.7415730337078652
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.86      0.81       111
           1       0.71      0.54      0.61        67

    accuracy                           0.74       178
   macro avg       0.73      0.70      0.71       178
weighted avg       0.74      0.74      0.73       178
```

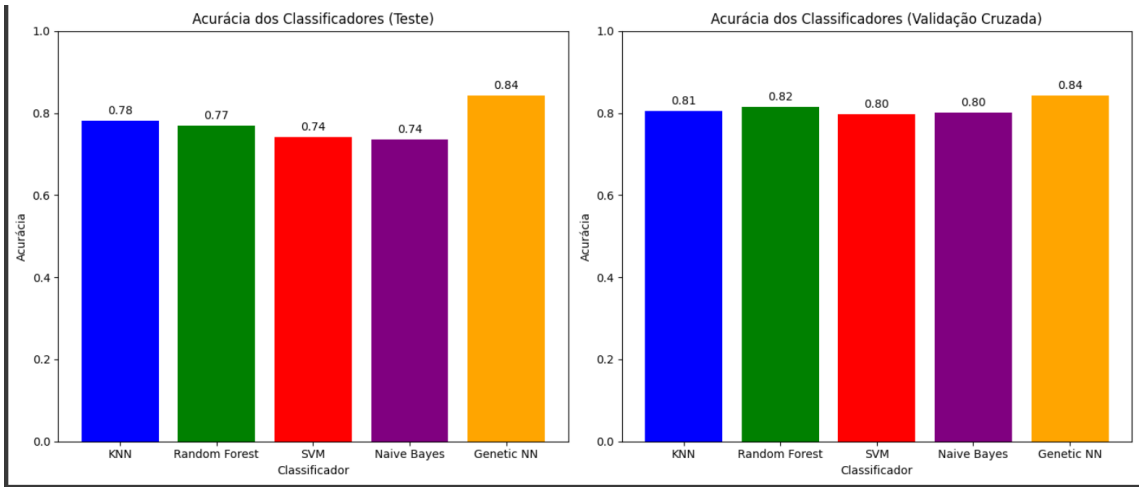


Report e matrix de confusão no modelo de SVM



Report e matrix de confusão no modelo de Naive Bayes

Os resultados das acurácias de teste e validação cruzada para cada classificador foram comparados, conforme os gráficos abaixo:



IV. ANÁLISE DOS RESULTADOS

A rede neural otimizada por algoritmos genéticos apresentou melhoria significativa na acurácia em comparação com os classificadores convencionais. A abordagem de otimização demonstrou ser eficaz na busca por hiperparâmetros ótimos, resultando em um modelo mais robusto.

V. DISCUSSÃO

Ainda que o algoritmo genético tenha capacidade de explorar um espaço de hiperparâmetros mais amplo e que tenha o potencial para alcançar melhores performances em comparação com métodos tradicionais de otimização e em relação ao uso de redes neurais profundas sem otimização, há outras técnicas, como a referida otimização por enxame de partículas – além de seu potencial de ser computacionalmente intensivo, requerendo muito tempo de execução.

Entretanto, a otimização de redes neurais utilizando algoritmos genéticos é aplicável em diversos domínios onde a modelagem preditiva é crucial, como finanças, saúde e engenharia. A capacidade de encontrar combinações ótimas de hiperparâmetros pode resultar em modelos mais precisos e eficientes.

VI. CONCLUSÃO

A implementação do algoritmo genético demonstrou que quanto estes algoritmos são uma ferramenta poderosa para a otimização de redes neurais, resultando em melhorias significativas na acurácia do modelo.

A abordagem se mostrou viável e se mostrou aplicável a problemas complexos, de otimização, no mundo real.

VII. TRABALHOS FUTUROS

Para trabalhos futuros, recomenda-se:

- Explorar outras arquiteturas de redes neurais.
- Investigar a combinação de algoritmos genéticos com outras técnicas de otimização.
- Avaliar a escalabilidade da abordagem em *datasets* maiores e mais complexos.

VIII. REFERÊNCIAS

Stanford Titanic Dataset

Gerón, A., (2021). Mãos à Obra: Aprendizado de Máquina com Scikit-Learn, Keras & TensorFlow.

Bhargava, A., (2017). Entendendo Algoritmos: Um Guia Ilustrado Para Programadores e Outros Curiosos

Documentação DEAP

IX. ANEXOS

Código-fonte: https://github.com/LeonardoFOliveira/ia-para-devs-tech-challenge-fase2/blob/main/titanic1_1_0.ipynb

Vídeo Explicativo: https://youtu.be/Tz78_NYoiNM