Análise de algoritmos de classificação utilizando a base de dados Abalone

1st Maria Eduarda Farias das Chagas
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
mefc@cin.ufpe.br

2nd Ariel Rodrigues Sousa dos Santos
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
arss5@cin.ufpe.br

***Abstract*—O presente artigo tem o objetivo de aplicar e comparar a eficiência de 4 algoritmos de aprendizado supervisionado: Árvores de decisão, K-Nearest Neighbors (KNN), Support Vector Machine (SVM) e Rede Neural (backpropagation). Esta análise é feita utilizando a clássica base de dados Abalone, que consiste em diversas medições de um molusco homônimo.**

***Index Terms*—Abalone, Classification, Decision Tree, KNN, Neural Network, SVM**

I. Introdução

O aprendizado supervisionado é uma das principais áreas da aprendizagem de máquina, onde o objetivo é treinar algoritmos para fazer previsões ou tomar decisões com base em dados rotulados. Esse campo é diverso e inclui vários algoritmos, cada um com suas vantagens e desvantagens.

No presente artigo, exploramos a eficiência dos algoritmos Árvore de Decisão, K-Nearest Neighbors (KNN), Support Vector Machine (SVM) e Redes Neurais (com ênfase no backpropagation) numa base de dados de moluscos Abalone.

Árvore de Decisão é um algoritmo que representa regras de decisão baseadas nos valores dos atributos de um conjunto de dados [1]. Ele é representado em uma estrutura hierárquica de nós, onde cada nó interno representa um teste em um atributo, cada ramo representa o resultado de tal teste e cada folha representa uma classe ou valor de saída (categorias discretas ou valores numéricos)

K-Nearest Neighbor é um método de predição que utiliza da distância entre a amostra atual e seus k vizinhos mais próximos no conjunto de treinamento para definir qual será o resultado de sua predição [2].

Support Vector Machine (SVM) é um algoritmo que busca encontrar um hiperplano ideal que separa as diferentes classes de dados no espaço de características. É eficaz em espaços de alta dimensionalidade e é ótimo para maximizar a margem de separação entre classes, sendo bastante utilizado para problemas de classificação complexos.

Redes Neurais são inspiradas na estrutura e funcionamento do cérebro humano. Compostas por camadas de nós (neurônios) que processam e transmitem informações, essas redes são capazes de modelar relações não lineares complexas e são a base de muitos avanços recentes em aprendizado profundo, pois reconhece padrões escondidos e correlações em dados brutos.

O conjunto de dados Abalone contêm medidas físicas de abalones, que são grandes moluscos gastrópodes comestíveis. Este dataset foi criado em 1995 por Sam Waugh, que o utilizou em seu PhD para comparar diversos algoritmos de classificação e, desde então, é extremamente utilizado em artigos que realizam esse tipo de validação

II. Metodologia

A. *Preparação de Dados*

O dataset escolhido possui 4177 linhas e 9 colunas. As colunas incluem uma variável categórica (sexo), sete variáveis contínuas (Length, Diameter, Height, Whole weight, Shucked weight, Viscera weight, Shell weight) e uma variável inteira (número de aneis). A variável categórica possui 3 valores atribuídos ”Male”, ”Female” e ”Infant”, que são convertidos para os valores 0, 1 e 2 para melhor se adequar aos modelos.

Após esse processo, os dados passam pelo processo de normalização utilizando a função StandartScaler da biblioteca Scikit Learn. Ela transforma cada valor de forma que a média dos dados modificados seja zero e o desvio padrão seja igual a um. Essa transformação garante que eles estejam em uma escala comum, evitando problemas como viés (bias) e prevenir que o modelo se ajuste demais aos dados. não se ajuste aos dados ou não consiga generalizar muito bem para novos dados futuros (falhas de ajuste e sobreajuste).

Foi determinada a separação de dados para treinamento e teste numa proporção de 80% e 20% respectivamente, sendo uma escolha balanceada para o treinamento e para garantir que o modelo vai performar bem com dados não vistos anteriormente.

A função train_test_split retorna quatro conjuntos:

- X_train: Contém 80% dos dados das features, que serão usados para treinar o modelo.
- X_test: Contém 20% dos dados das features, que serão usados para testar o modelo.
- y_train: Contém 80% dos dados dos labels, correspondentes a X_train, que serão usados para treinar o modelo.
- y_test: Contém 20% dos dados dos labels, correspondentes a X_test, que serão usados para testar o modelo.

Então, é realizado o cálculo de três métricas de desempenho, a MAE (Mean Absolute Error), que é a a média dos erros

absolutos entre os valores previstos e os valores reais (y_test e y_pred); a MSE (Mean Squared Errors), que é a média dos quadrados dos erros entre os valores previstos e os valores reais e o $R\hat{2}$, coeficiente de determinação que mede a proporção da variância dos dados que é explicada pelo modelo.

Em alguns dos modelos foi utilizada a técnica de Grid Search para encontrar a melhor combinação de hiperparâmetros para um modelo de aprendizado de máquina. Hiperparâmetros são configurações ajustáveis que não são aprendidas pelo algoritmo de treinamento, mas que influenciam significativamente o desempenho e comportamento do modelo, como por exemplo a taxa de aprendizado em redes neurais, a profundidade em árvores de decisão, o número de vizinhos em algoritmos de KNN ou "C", "Gamma" e "Kernel" no algoritmo de SVM.

O Grid Search funciona testando todas as combinações possíveis de valores para esses hiperparâmetros dentro de um conjunto predefinido de valores, criando uma grade (Grid) de combinações. Para cada combinação de hiperparâmetros, o modelo é treinado e avaliado usando uma métrica de desempenho, como precisão, F1-score ou erro quadrático médio, dependendo do tipo de problema.

O objetivo do Grid Search é identificar a combinação de hiperparâmetros que maximiza o desempenho do modelo em um conjunto de dados de validação ou em um processo de validação cruzada. Isso ajuda a evitar o ajuste excessivo (overfitting) do modelo aos dados de treinamento e a encontrar uma configuração que generalize bem para novos dados, de modo a melhorar o desempenho dos modelos de aprendizado de máquina.

B. Árvore de decisão

A estrutura de uma árvore é composta por nós e ramos. Neste exemplo, o ponto de partida (raíz) representa o conjunto de dados completo. A partir dele são feitas divisões nos nós internos de acordo com as características específicas dos dados, até chegar nos que representam as classes finais ou rótulos de saída (folhas). Cada divisão é determinada por uma regra de decisão que avalia uma característica e um valor de corte, separando os dados em subconjuntos mais homogêneos.

O algoritmo de árvore de decisão realiza tal processo recursivamente, selecionando a melhor característica e o melhor valor de corte para dividir os dados de modo a otimizar a acurácia do modelo, criando novos nós internos e ramos até que sejam atendidos os critérios de parada. Esses critérios podem incluir uma profundidade máxima da árvore, um número mínimo de amostras necessárias para dividir um nó ou um número mínimo de amostras em uma folha.

A recursão termina quando os critérios de parada são atingidos, resultando em folhas que representam a classe prevista. Neste exemplo buscamos encontrar a idade do Abalone (que é definida a partir do número de anéis) a partir das variáveis Sex, Length, Diameter, Height, Whole weight, Shucked weight, Viscera weight e Shell weight.

O código apresentado no notebook treina um modelo de clasificação para Árvore de Decisão utilizando o método "DecisionTreeRegressor" do módulo "tree" do scikit-learn, treinando com os dados de treinamento que foram definidos anteriormente. Após o treinamento, o modelo é utilizado para realizar as predições e apresentou o desempenho a seguir:

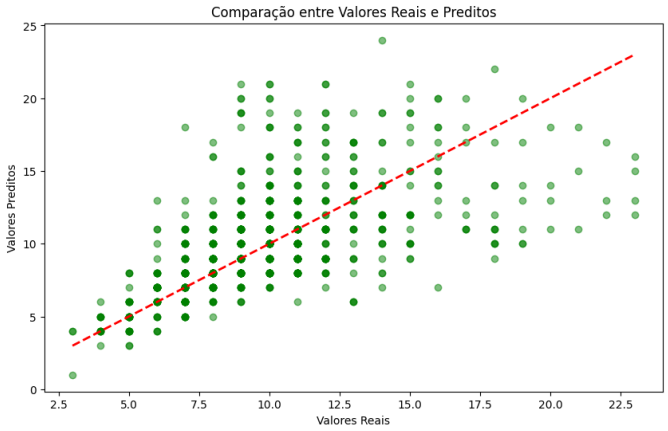

Fig. 1. Decision Tree: Comparação entre valores reais e preditos

- **Mean Absolute Error (MAE):** 2.1830143540669855
- **Mean Squared Error (MSE):** 9.80023923444976
- **R$\hat{2}$ Score:** 0.09468368805725214

Com o objetivo de melhorar as métricas de desempenho que estão apresentadas na na Fig. 1 foi aplicado o algoritmo de GridSearch com os valores de parâmetros sendo max_depth: [None, 5, 10, 15, 20], min_samples_split: [2, 5, 10] e min_samples_leaf: [1, 2, 4, 6], com os resultados apresentados na Fig. 2

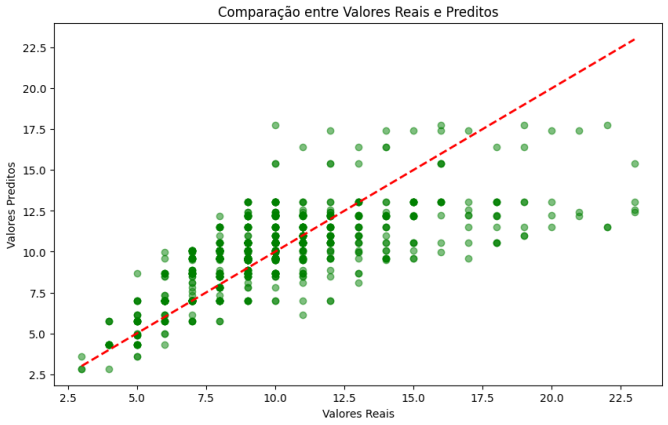

Fig. 2. Decision Tree após Grid Search

Após esse processo, foram encontradas novas métricas e que os melhores parâmetros são max_depth: 5, min_samples_leaf: 6, min_samples_split: 2

- **Mean Absolute Error (MAE):** 1.6245254516433962
- **Mean Squared Error (MSE):** 5.3628119870044815
- **R$\hat{2}$ Score:** 0.5045997292952975

C. *K-Nearest Neighbor*

O algoritmo K-Nearest Neighbors (KNN) tem como ponto de partida o conjunto de dados analizado representado como uma nuvem de pontos no espaço de características, e pressupõe que dados semelhantes tendem a se agrupar. Toda vez que uma nova instância precisa ser classificada, o algoritmo KNN procura pelos K pontos mais próximos dessa instância para poder fazer a previsão. Eles são encontrados usando uma métrica como a distância euclidiana, e assim que eles são identificados o agoritmo considera as classes e atribui a nova instância a classe mais comum.

É importante perceber que o valor de K é um hiperparâmetro que deve ser escolhido de forma adequada, já que um valor muito pequeno pode levar a acontecer o fenômeno de ”overfitting”, no qual o modelo se ajusta bem aos dados de treinamento mas não consegue realizar previsões com acurácia em dados novos. Isso acontece pois com poucos vizinhos para realizar a comparação o algoritmo KNN tende a se tornar muito sensível ao ruído (informações irrelevantes, imprecisas ou aleatórias) nos treinamento ao invés de aprender com os padrões corretos.

Além disso, esse baixo valor de K gera uma instabilidade alta nas fronteiras de decisão (linha, curva ou hiperplano que separa as classes de dados no espaço de características), gerando grandes mudanças na previsão do modelo se houver mais dados de treinamento. Um valor muito alto de K também é negativo, pois suaviza as fronteiras de decisão (modelo muito menos sensível a nuances menos precisão), maior custo computacional e favorecimento maior das classes majoritárias

Dessa forma, é necessário encontrar um equilíbrio no valor de K para melhorar o para resultados mais satisfatórios. O código apresentado no notebook treina um modelo de clasificação para K-Nearest Neighbors (KNN) utilizando o método ”KNeighborsRegressor” do módulo ”neighbors” do scikit-learn, treinando com os dados de treinamento que foram definidas anteriormente. O modelo é testado com diferentes valores de k entre 1 e 25 para encontrar o resultado mais adequado, sendo ele mais adequado o com 15 vizinhos, como demonstrado na Fig. 3

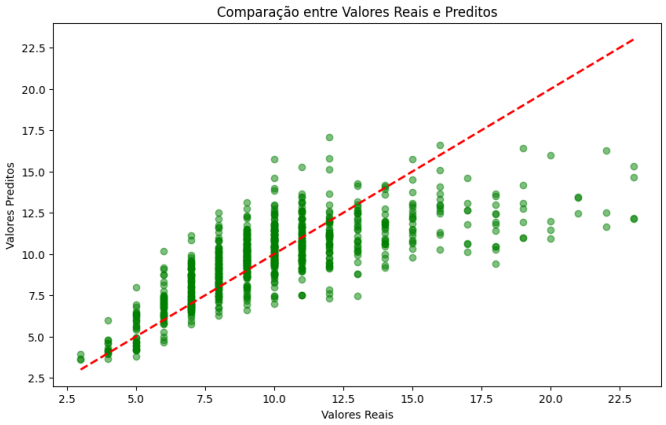

Fig. 3. KNN: Comparação entre valores reais e preditos

- **Mean Absolute Error (MAE):** 1.5618819776714516
- **Mean Squared Error (MSE):** 5.111111111111111
- **R$\hat{2}$ Score:** 0.5278510911473167

Para tentar melhorar o desempenho do algoritmo, foram utilizadas 3 métricas diferentes diferentes da Euclidiana, a Manhattan, a Chebyshev e a Minkowski. A Manhattan calcula a soma das diferenças absolutas entre as coordenadas dos pontos, a Chebyshev calcula a distância máxima entre duas dimensões ao longo de todos os eixos e a Minowski é uma generalização das métricas Euclidiana e Manhattan. No entanto, nenhuma delas foi superior a Euclidiana.

D. *Support Vector Machine*

O algoritmo de Support Vector Machine (SVM) funciona ao encontrar o hiperplano que melhor separa diferentes classes num espaço de alta dimensão. Ele busca identificar um limite de decisão que maxiza a margem, que é distância entre o hiperplano e os pontos de dados mais próximos de cada classe (Support Vectors).

Inicialmente os dados são todos mapeados para um espaço de características de alta dimensão de modo que os pontos de dados possam ser categorizados. Esse processo é feito utilizando uma função ”kernel”, que deixa a informação linearmente separável e facilita a criação do hiperplano mesmo em casos onde isto não era possível. Então, a classificação de novos dados é feita determinando de que lado do hiperplano eles vão estar após esta transformação.

O código apresentado no notebook treina um modelo de clasificação para Support Vector Machine (SVM) utilizando o método SVR do módulo ”SVM” do scikit-learn, treinando com os dados de treinamento que foram definidos anteriormente. Após o treinamento, o modelo é utilizado para obter as métricas e realizar as predições, com o resultado visto na Fig. 4

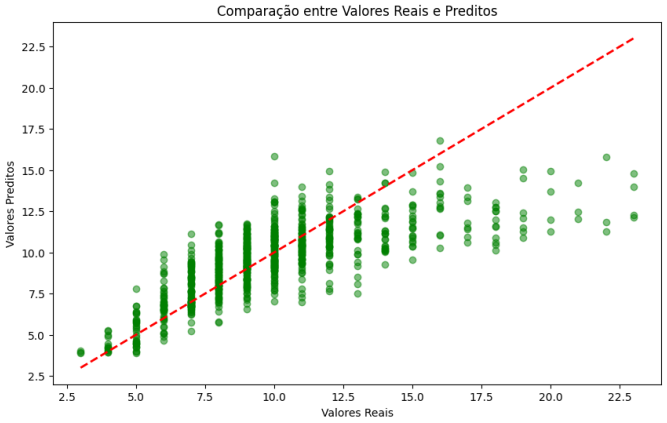

Fig. 4. SVM: Comparação entre valores reais e preditos

- **Mean Absolute Error (MAE):** 1.5243960560678536
- **Mean Squared Error (MSE):** 4.997554954719112
- **R$\hat{2}$ Score:** 0.538341063712662

O algoritmo de Support Vector Machine possui alguns hiperparâmetros que podem ser modificados para aumentar a acurácia do modelo, que são C, Gamma e Kernel.

C é o parâmetro de regularização e controla a tolerância a erros permitida pelo modelo de SVM, um valor baixo permitindo mais erros de classificação no treinamento (fronteiras de decisão menos complexas) e um valor alto penaliza de maneira mais forte os erros (fronteiras de decisão mais precisas).

Gamma é o parâmetro de ajuste do kernel, que define o escopo da influência de um único exemplo de treinamento. Um valor baixo significa que cada exemplo de treinamento possui uma influência de maior alcance (modelo considera mais exemplos no cálculo de fronteira de decisão) e um valor alto limita o alcance a influência de cada exemplo de treinamento, deixando os mais próximos (tornando a fronteira de decisão mais dependente desses pontos).

Kernel é uma função que calcula a similaridade entre dois pontos de dados no espaço de entrada original. A escolha dele é importante , pois determina como os dados serão transformados para tornar a fronteira de decisão mais linear ou não linear. Alguns dos kernels comuns incluem o linear, o polinomial (que mapeia os dados em um espaço de maior dimensionalidade usando funções polinomiais) e o Radial Basis Function (que mapeia os dados em um espaço de dimensionalidade infinita usando funções de base radial).

Com o objetivo de melhorar as métricas de desempenho que estão apresentadas na na Fig. 4 foi aplicado o algoritmo de GridSearch com os valores de parâmetros sendo C: [0.1, 1, 10], gamma: [1, 0.1, 0.01] e kernel: [rbf], com os resultados apresentados na Fig. 5

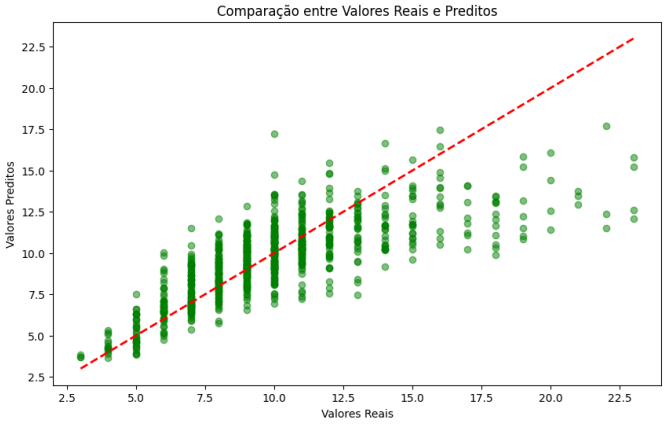

Fig. 5. SVM após Grid Search

Após esse processo, foram encontradas novas métricas e que os melhores parâmetros são C: 10, gamma: 0.1, kernel: rbf

- **Mean Absolute Error (MAE):** 1.496428448815378
- **Mean Squared Error (MSE):** 4.759854636685773
- **R$\hat{2}$ Score:** 0.5602990965852791

E. Redes Neurais

Rede neural é um modelo de aprendizado de máquina inspirado no funcionamento do cérebro humano. Ela consiste em uma coleção de unidades interconectadas, chamadas neurônios, que trabalham em conjunto para processar e aprender. Inicialmente, os dados de entrada são propagados através da rede, passando por várias camadas de neurônios. Cada um deles realiza uma operação de transformação nos dados, aplicando pesos e adicionando um viés. Essa transformação é feita através de uma função de ativação, que determina se o neurônio deve ser ativado ou não com base no resultado da operação.

Durante o treinamento da rede neural, os pesos e viés de cada neurônio são ajustados iterativamente de forma a minimizar uma função de perda, que mede a diferença entre as saídas previstas pela rede e os valores reais. Isso é feito utilizando algoritmos de otimização, como o gradiente descendente, que atualizam os parâmetros da rede na direção que minimiza a função de perda. Assim como o SVM utiliza uma função kernel para mapear os dados para um espaço de características de alta dimensão, as redes neurais podem utilizar diferentes arquiteturas e funções de ativação para aprender representações complexas dos dados, permitindo-as lidar com problemas não-lineares.

O código apresentado no notebook utiliza a biblioteca Scikit-learn para treinar um modelo de regressão usando uma Rede Neural Multicamadas (MLPRegressor). O modelo é configurado com três camadas ocultas contendo 100, 100 e 50 neurônios, e usa a função de ativação ReLU (Unidade Linear Retificada).

O solver 'adam' é usado para otimizar a rede, e a taxa de aprendizado é adaptativa, iniciando em 0.001, com uma regularização L2 (alpha) de 0.3. Após o treinamento, o modelo é utilizado para obter as métricas e realizar as predições, com o resultado visto na Fig. 6.

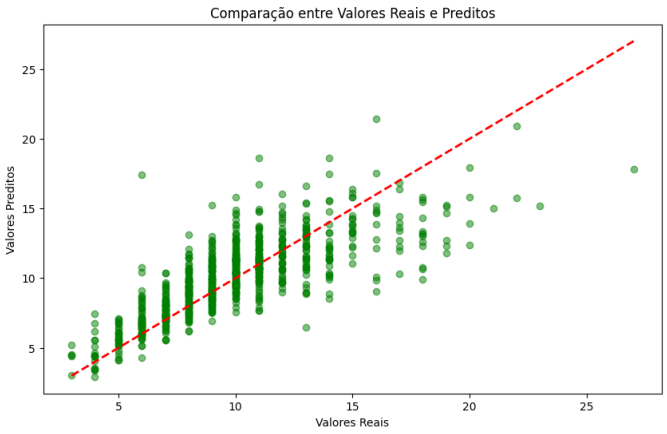

Fig. 6. Neural Network: Comparação entre valores reais e preditos

- **Mean Absolute Error (MAE):** 1.4990553357938265
- **Mean Squared Error (MSE):** 4.59350767266472
- **R$\hat{2}$ Score:** 0.596121355522754

III. Questões

A. Qual classificador/configuração apresenta melhor resultados em termos de acurácia e de tempo? Apresente justificativas para suas respostas.

No presente artigo, foram testados os algoritmos para a criação de modelos de de aprendizado supervisionado Decision Tree, K-Nearest Neighbors, Support Vector Machine e Neural Networks com diferentes métricas e obtido os valores de Mean Absolute Error (MAE), Mean Squared Error(MSE) e R$\hat{2}$ Score, além de criar um gráfico de dispersão para demonstrar visualmente os resultados.

Como se pode observar na Fig. 7 o modelo de Neural Network utilizando Stacking Regressor se mostrou superior aos demais, apresentando um desempenho de MAE de 1.497833, um MSE de 4.541728, um R$\hat{2}$ Score de 0.580449 com um tempo de treino de 12.152027 segundos. Embora outros algoritmos possuam tempos muito menores, a acurácia deles é inferior ao analisar as outras métricas. Mesmo que a disparidade seja pouca, e esse valor aumentaria ainda mais se o conjunto de dados fosse maior e mais complexo, já que as redes neurais conseguem identificar padrões de maneira excepcional.

	MAE	MSE	R^2 Score	Tempo de treino
Ávore de decisão	2.183014	9.800239	0.094684	0.023089
Ávore de decisão tunned	1.624525	5.362812	0.504600	6.332086
KNN	1.558038	5.156576	0.523651	0.015751
SVM	1.524396	4.997555	0.538341	1.724085
SVM tunned	1.496428	4.759855	0.560299	43.259344
Rede Neural	1.499055	4.593508	0.575666	12.152027
SVM + MLP - VR	1.478321	4.590305	0.575962	12.152027
SVM + MLP - SR	1.497833	4.541728	0.580449	12.152027

Fig. 7. Tabela de resultados

B. Avalie os parâmetros: taxa de aprendizado, número de épocas de treinamento, funções de ativação de neurônio, parâmetros de configuração, etc. Caso julgue necessário outros parâmetros podem ser inseridos na análise. Analise e discuta os resultados obtidos.

A taxa de aprendizado é um hiperparâmetro crítico nos algoritmos de treinamento de redes neurais. Ele determina o tamanho dos passos que o algoritmo de otimização dá durante a minimização da função de perda, uma taxa de aprendizagem mais alta resulta em uma convergência mais rápida. Uma taxa de aprendizado moderada (e.g., 0.001) proporcionou um bom equilíbrio entre velocidade de convergência e precisão final. Taxas de aprendizado muito altas (e.g., 0.1) causaram oscilações significativas e piores desempenhos. As funções de ativação determinam a saída dos neurônios e são essenciais para a capacidade de aprendizado de redes neurais, a função ReLU se mostrou eficiente no problema trabalhado. A quantidade de neurônios por camada afeta diretamente a capacidade do modelo de capturar padrões complexos, uma arquitetura com duas ou três camadas com 50 ou 100 neurônios apresentou bons resultados.

C. Quais as características do modelo que influenciam seu desempenho na base de dados?

O Multi-Layer Perceptron (MLP) é um tipo de rede neural artificial que pode capturar padrões complexos e não lineares em conjuntos de dados. O desempenho superior do MLP em uma base de dados específica pode ser influenciado por várias características e fatores. É notável que sua capacidade de se ajustar às relações não-lineares e à flexibilidade proporcionada pelas suas múltiplas camadas e neurônios que utilizam o backpropagation se mostraram características importantes que permitiram um melhor desempenho geral.

D. Como melhorar o desempenho do classificador?

Existem diversas maneiras de melhorar o desempenho de um modelo de aprendizagem de máquina. Neste estudo foram utilizados métodos de ajuste de hiperparâmetros, padronização de dados e técnicas de ensemble. No primeiro caso, o método StandardScaler da biblioteca scikit-learn foi utilizado. Posteriormente, para o ajuste de hiperparâmetros, foi utilizado o método GridSearchCV, que consiste em uma busca exaustiva que retorna a combinação de parâmetros que apresentou melhor desempenho. Já no último caso, três técnicas de ensemble foram utilizadas: Voting Regressor, que usa a média da previsão dos modelos de base para prever o resultado final, Stacking Regressor, que utiliza um meta-modelo para realizar previsões com base nos outputs dos modelos de base, e o Random Forest, que aplica a técnica de Bagging ao algoritmo de árvore de decisão.

Além das técnicas citadas, existem diversas outras abordagens que permitem melhorar o desempenho de modelos. A engenharia de features, por exemplo, permite que as melhores características sejam selecionadas para otimizar a captura de padrões e evitar altas dimensionalidades. As técnicas de regularização, como L1 e L2, buscam evitar o overfitting, penalizando modelos muito complexos. Enquanto isso, o aumento de dados, seja através do aumento sintético ou da coleta de novos dados reais, ajuda os modelos a generalizarem melhor após o processo de treinamento. Esses são apenas alguns exemplos dentre muitos que podem ser implementados. É importante destacar que a escolha de quais técnicas utilizar deve sempre levar em consideração a natureza do problema trabalhado, pois diferentes contextos exigem abordagens distintas para atingir o melhor resultado.

E. Quais técnicas podem ser aplicadas para o pré-processamento ou pós-processamento dos dados?

O pré-processamento é uma etapa crucial para melhorar a qualidade dos dados e, consequentemente, o desempenho dos modelos. A limpeza de dados consiste na remoção de features com muitos valores ausentes ou então na imputação de novos valores a partir da média, mediana, moda ou então através de métodos mais sofisticados que utilizam algoritmos

de aprendizagem de máquina. A transformação dos dados, realizada através de técnicas como normalização e padronização, e a codificação de dados categóricos, através de One-Hot-Encoding ou Label-Encoding, são utilizadas para mudar as características do conjunto e torná-lo mais adequado para os modelos. Feature engineering, balanceamento de dados e redução de dimensionalidade são outras técnicas que podem ser utilizadas nesta etapa.

Por outro lado, o pós-processamento é essencial para melhorar a interpretação, usabilidade e precisão das previsões feitas pelo modelo. Algumas técnicas comuns são: calibração de modelos, ajustes de threshold, combinação de modelos e correção de bias.

F. Os classificadores podem ser combinados? Como? Quais as implicações?

Sim, é possível combinar diferentes modelos. Existem várias formas de combinação e elas podem ser homogêneas, quando usam o mesmo tipo de algoritmo, ou heterogêneas, quando combinam diferentes algoritmos. O Bagging é uma técnica que envolve o treino de diversos modelos e obtém a resposta final através de votação ou cálculo de média, Random Forest é um exemplo deste tipo de estratégia. Já o Boosting treina estimadores de forma sequencial, onde cada novo modelo tenta corrigir os erros do anterior.

A combinação de modelos pode trazer tanto implicações positivas como negativas. Como pontos positivos pode-se destacar a redução de viés e variância e o aumento da flexibilidade, já que combinar diferentes algoritmos pode permitir uma captura mais abrangente dos padrões dos dados. Por outro lado, combinar modelos pode exigir mais recursos computacionais e resultar em um maior tempo de treinamento, além de serem mais difíceis de interpretar, especialmente quando muitos estimadores são utilizados, o que também resulta em uma implementação e manutenção mais complexas.

Neste estudo foram utilizadas três técnicas de ensemble. O Bagging foi utilizado para treinar o Random Forest Regressor, que consiste em uma coleção de árvores de decisão.

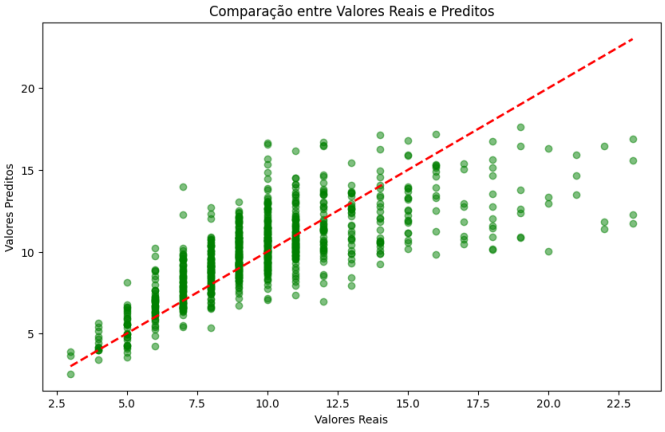

Fig. 8. RFR: Comparação entre valores reais e preditos

- **Mean Absolute Error (MAE):** 1.5916746411483256
- **Mean Squared Error (MSE):** 5.094096889952153
- **R$\hat{2}$ Score:** 0.5294228131820258

É possível observar que os resultados obtidos pelo Random Forest foram consideravelmente superiores ao primeiro modelo de árvore de decisão e levemente superiores aos resultados da árvore de decisão após o ajuste de hiperparâmetros.

Ademais, as técnicas de Voting Regressor e Stacking Regressor foram utilizadas para combinar os modelos Support Vector Machine e Rede Neural.

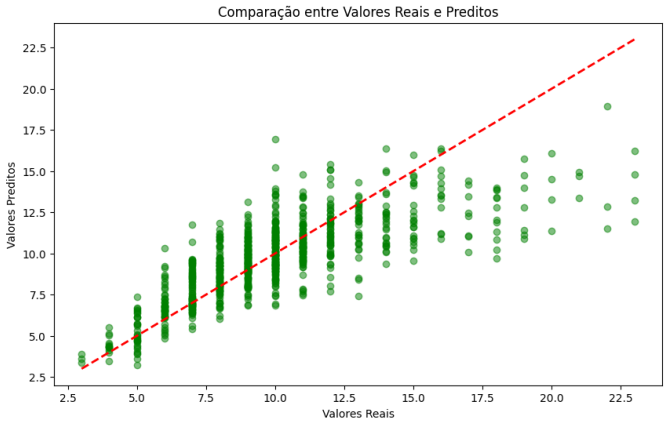

Fig. 9. SVM + MLP - VR: Comparação entre valores reais e preditos

- **Mean Absolute Error (MAE):** 1.478321201495115
- **Mean Squared Error (MSE):** 4.590305238544629
- **R$\hat{2}$ Score:** 0.5759615546279038

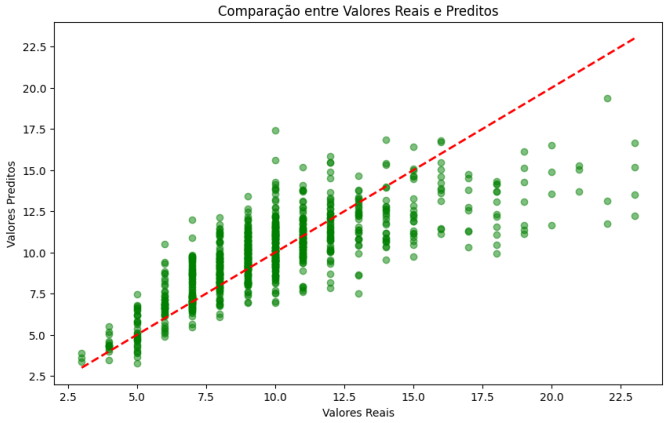

Fig. 10. SVM + MLP - SR: Comparação entre valores reais e preditos

- **Mean Absolute Error (MAE):** 1.4978328152072984
- **Mean Squared Error (MSE):** 4.541728105194604
- **R$\hat{2}$ Score:** 0.5804489625530701

Foi possível observar uma pequena melhora de desempenho em relação aos modelos base. Em especial a combinação através do Stacking Regressor apresentou os melhores resultados nas métricas MSE e R2 em relação a todos os outros modelos, enquanto que a combinação através do Voting Regressor apresentou o melhor resultado para o MAE.

IV. Conclusão

A realização deste estudo nos permitiu aprofundar nossos conhecimentos nos algoritmos de aprendizagem de máquina bem como nas técnicas de aperfeiçoamento de modelos. Ao longo do desenvolvimento, pudemos verificar as características de cada algoritmo e suas vantagens e desvantagens no contexto de um problema de regressão.

Os resultados demonstraram que a Rede Neural (MLP) apresentou a melhor performance geral, o que demonstra seu poder em capturar padrões complexos nos dados. Essa superioridade pode ser atribuída à sua capacidade de se ajustar às relações não-lineares e à flexibilidade proporcionada pelas suas múltiplas camadas e neurônios que utilizam o backpropagation. O Support Vector Machine (SVM) também apresentou uma performance satisfatória, especialmente após o ajuste de hiperparâmetros, o que evidencia a importância de conhecer diferentes técnicas que permitam aperfeiçoamento dos modelos. Tal fato também pôde ser observado na Árvore de decisão, que teve uma melhora considerável após a realização do Grid Search. Além disso, o KNN, apesar de ser um algoritmo simples, se mostrou consideravelmente eficiente na tarefa mesmo sem sofrer ajustes relevantes.

Adicionalmente, o uso da estratégia de ensemble se mostrou uma ferramenta poderosa que permite reunir as vantagens de diferentes modelos visando alcançar resultados ainda melhores. O uso do Bagging no caso da Random Forest se mostrou bastante eficiente para melhorar o desempenho da Árvore de Decisão, permitindo um modelo mais robusto mesmo com um algoritmo simples. Já a combinação de SVM com MLP se mostrou bastante promissora, aproveitando a capacidade do SVM de capturar padrões específicos em espaços de alta dimensionalidade, enquanto o MLP fornece a flexibilidade e o poder de modelar relações complexas. Essa combinação não só potencializa a precisão das previsões, mas também aumenta a robustez do modelo final.

Em suma, este estudo destaca a importância de compreender bem as características do conjunto de dados e de cada algoritmo de aprendizagem de máquina para entender qual estratégia aplicar em diferentes contextos.

References

[1] M. S. Lauretto, “Árvores de Decisão”, EACH-USP, Novembro 2010.

[2] F. M. Mariz, ”Avaliação e Comparação de Versões Modificadas do Algoritmo KNN”, Centro de Informática - UFPE, Dez 2017