AVALIAÇÃO – APRENDIZADO DE MÁQUINA

Rodrigo Barroso Rodrigues \_ 219031139

**Índice**

1. [Introdução 4](#_TOC_250040)
   1. [Descrição da Avaliação 4](#_TOC_250039)
      1. [Dataset 4](#_TOC_250038)
      2. [Sobre os dados de treino e teste 4](#_TOC_250037)
      3. [Sobre o tratamento dos dados](#_TOC_250036) 4
      4. [Sobre as features do dataset](#_TOC_250035) 5
      5. [Introdução dos modelos treinados 6](#_TOC_250034)
2. [Avaliações dos Modelos 7](#_TOC_250032)
   1. [Introdução 7](#_TOC_250031)
   2. [Árvore de Decisão (Decision Tree) 8](#_TOC_250030)
   3. Random Forest 10
   4. Multilayer Perceptron (MLP) 12
3. [Resultados 11](#_TOC_250029)
   1. [Caminho percorrido pelo HillClimb: 11](#_TOC_250028)
   2. [Caminho percorrido pelo Best First: 12](#_TOC_250027)
   3. [Caminho percorrido pelo Branch and Bound: 13](#_TOC_250026)
   4. [Caminho percorrido pelo A\*: 15](#_TOC_250025)
4. [Dificuldades: 17](#_TOC_250024)
   1. [Limitações: 17](#_TOC_250023)
      1. [Branch and Bound: 17](#_TOC_250022)
   2. [Dificuldades e como superamos: 17](#_TOC_250021)
      1. [Swish.SWI-Prolog: 17](#_TOC_250020)
      2. [Problema na implementação: 17](#_TOC_250019)
5. [Comparação dos algoritmos: 18](#_TOC_250018)
   1. [Gráfico de Posições Percorridas: 18](#_TOC_250017)
      1. [Custo dos movimentos: 18](#_TOC_250016)
   2. [Gráfico do Tempo de Execução: 19](#_TOC_250015)
      1. [Tempo de execução: 19](#_TOC_250014)
   3. [Conclusão: 20](#_TOC_250013)
6. [Desenvolvimento das Atividades: 21](#_TOC_250012)
   1. [Divisão de trabalho: 21](#_TOC_250011)
      1. [Programação: 21](#_TOC_250010)
      2. [Relatório: 22](#_TOC_250009)
      3. [Apresentação: 24](#_TOC_250008)
   2. [Ferramentas: 25](#_TOC_250007)
      1. [Comunicação: 25](#_TOC_250006)
      2. [Programação: 25](#_TOC_250005)
      3. [Relatório e Apresentação: 26](#_TOC_250004)
   3. [Contribuição individual: 26](#_TOC_250003)
7. [Links e Referências Bibliográficas: 28](#_TOC_250002)
   1. [Links: 28](#_TOC_250001)
      1. [Links para auxílio: 28](#_TOC_250000)

# Introdução

## Descrição da Avaliação

### Dataset

O dataset utilizado para a seguinte avaliação de aprendizado de máquina foi obtido na plataforma Kaggle através do seguinte [link](https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification). Ele consiste em uma série de dados referentes a especificações de aparelhos de telefonia móvel (smartphones), e a partir desses dados, foi inferido uma coluna chamada de “price\_range”, que guarda o valor de 4 classes, com valores inteiros de 0 a 3, onde cada valor inteiro representa um intervalo de preços para cada aparelho celular baseado em suas específicações, sendo 0 o mais barato até 3 sendo os aparelhos mais caros.

### Sobre os dados de treino e teste

O dataset “Mobile Price Classification” contém 21 features, sendo 1 delas, “price\_range”, que será utilizado como o target para o aprendizado de máquina. Apesar desse conjunto apresentar um arquivo CSV tanto para treino quanto para teste, o arquivo de teste não conta com a coluna “price\_range”, o que faz com que não seja possível utilizá-lo em um aprendizado supervisionado.

A solução que optei por utilizar foi utilizar apenas o arquivo CSV de treino, porém dividindo-o através do código Python para que 70% dele fosse utilizado para treino e os 30% restantes como teste.

### Sobre o tratamento dos dados

Todas as features presentes no dataset utilizado estão representadas como valores do tipo int64 ou float64 sem precisar fazer nenhum tratamento prévio como a aplicação de Label Encoding ou One-Hot Encoding para termos apenas valores númericos.

Além disso, todas as linhas do conjunto estão preenchidas, ou seja, nenhuma coluna apresentou valores em branco ou nulo, esses que também precisariam ser tratados antes de serem aplicados aos modelos de aprendizado de máquina.

### Sobre as features do dataset:

O dataset em questão apresenta 21 features:

- “battery\_power”, do tipo int64, contendo a quantidade em miliampere-hora da bateria do aparelho celular.

- “blue”, do tipo int64, contendo apenas valores binários (0 ou 1) para a existência (1) ou não (0) de adaptador bluetooth no aparelho.

- “clock\_speed”, do tipo float64, contendo a velocidade na qual o microprocessador do celular em questão executa instruções.

- “dual\_sim”, do tipo int64, contendo apenas valores binários (0 ou 1) para a existência (1) ou não (0) de 2 slots para chips de operadoras de telefonia móvel no aparelho.

- “fc”, do tipo int64, contendo a quantidade de megapixels da câmera frontal do aparelho.

- “four\_g”, do tipo int64, contendo apenas valores binários (0 ou 1) para a existência (1) ou não (0) da tecnologia 4G no aparelho.

- “int\_memory”, do tipo int64, contendo a quantidade de memória interna em gigabytes do aparelho.

- “m\_dep”, do tipo float64, contendo a quantidade em centímetros da espessura lateral do aparelho celular.

- “mobile\_wt”, do tipo int64, contendo o peso em gramas do aparelho celular.

- “n\_cores”, do tipo int64, contendo a quantidade de núcleos do processador do aparelho.

- “pc”, do tipo int64, contendo a quantidade de megapixels da câmera primária do aparelho.

- “px\_height”, do tipo int64, contendo a resolução em pixels da altura do aparelho.

- “px\_width”, do tipo int64, contendo a resolução em pixels da largura do aparelho.

- “ram”, do tipo int64, contendo a quantidade de memória RAM em megabytes do celular.

- “sc\_h”, do tipo int64, contendo a altura da tela do aparelho em centímetros.

- “sc\_w”, do tipo int64, contendo a largura da tela do aparelho em centímetros.

- “talk\_time”, do tipo int64, contendo a quantidade em horas que uma carga completa na bateria consegue manter o telefone ligado durante uma chamada telefônica.

- “touch\_screen”, do tipo int64, contendo apenas valores binários (0 ou 1) para a existência (1) ou não (0) da tecnologia touch screen no aparelho.

- “wifi”, do tipo int64, contendo apenas valores binários (0 ou 1) para a existência (1) ou não (0) da tecnologia WiFi no aparelho.

- “price\_range”, do tipo int64, sendo a target variable, contém os valores de 0 a 4, com 0 representando um aparelho celular de baixo custo, 1 sendo de médio custo, 2 de alto custo e 3 de altíssimo custo.

### Introdução dos modelos treinados:

Os seguintes modelos de aprendizado de máquina foram utilizados para o dataset utilizado nessa avaliação: Árvores de Decisão, Random Forest e MLP (Multilayer Perceptron).

- Árvores de Decisão: É um modelo baseado em árvores binárias para classificação ou regressão. Cada nó da árvore representa uma decisão baseada em uma condição nos dados. É um modelo de fácil interpretação, e não requer normalização dos dados, entretanto, é mais propício a overfitting, principalmente em árvores mais profundas.

- Random Forest: São um ensemble de árvores de decisão, onde múltiplas árvores são treinadas em subconjuntos dos dados com substituição (bootstrapping) e combina os resultados por votação (classificação) ou média (regressão). É um modelo que reduz o overfitting, melhorando a precisão, porém que pode ser bastante lento em cenários com muitas árvores.

- Multilayer Perceptron (MLP): É um modelo de rede neural feedforward que utiliza camadas densamente conectadas (fully connected). É adequado para resolver problemas não lineares. É vantajoso pois permite uma alta flexibilidade na captura de relações complexas nos dados , entretanto, requer mais dados e poder computacional, é mais sensível em relação aos parâmetros setados como a a taxa de aprendizado e a quantidade de neurônios utilizada e pode levar tempos de treino maiores.

# Avaliações dos modelos

### Introdução

Para treinar os modelos em questão citados acima, optei por utilizar a ferramenta ‘GridSearchCV’ do Scikit-learn, para que pudesse ser feita busca matricial (grid search), com o objetivo de fazer uma validação cruzada de valores para os parâmetros dos modelos. Ao encontrar a melhor combinação de hiperparâmetros para cada modelo, utilizei-os para o fit e o predict de cada modelo.

Além disso, também optei por testar o tuning dos modelos com a utilização da biblioteca RandomizedSearchCV, que através do treinamento com combinações aleatoriamente atribuídas a partir da mesma matriz que utilizei para o “grid search” para os hiperparâmetros dos modelos, reduziu de forma bastante significativa o tempo de treino de alguns dos modelos, principalmente o Random Forest. O uso da busca aleatória atualmente consegue reproduzir resultados muito próximos do ótimo encontrado através do GridSearchCV, entretanto, apesar do grande período de tempo que foi despendido para testar várias combinações com a abordagem GridSearch, optei por utilizar o mesmo.

Além das ferramentas GridSearchCV e RandomizedSearchCV do pacote Scikit-learn, também foram utilizadas as bibliotecas “time”, para poder calcular o tempo de treino e teste dos modelos, a biblioteca “numpy” para calcular a distribuição de classes da predição de cada modelo, a biblioteca “pandas”, utilizada para transformar o CSV do dataset em um DataFrame para ser utilizado para o treino e teste dos modelos.

Em relação as ferramentas do Scikit-learn, foram utilizadas:

- “sklearn.model\_selection.train\_test\_split”, para dividir o dataset na proporção 70% para treino, 30% para teste, com o random\_state = 42, com o intuito de garantir a reprodutividade dos resultados.

- “sklearn.metrics.accuracy\_score”, para calcular a acurácia da predição do modelo nos testes após o treino.

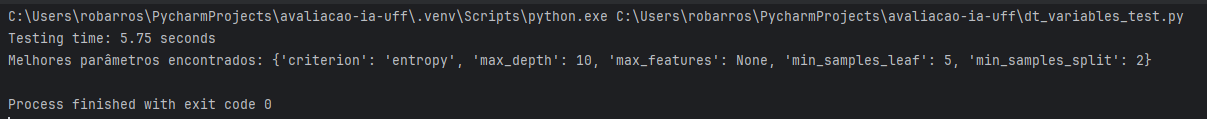
- “sklearn.metrics.f1\_score”, para calcular o F1 score da predição do modelo nos testes após o treino.

- “sklearn.metrics.precision\_score”, para calcular a pontuação da precisão da predição do modelo nos testes após o treino.

- “sklearn.metrics.recall\_score”, para calcular a pontuação de recall da predição do modelo nos testes após o treino.

- “sklearn.metrics.confusion\_matrix”, para gerar a matriz de confusão da predição do modelo nos testes após o treino.

As pontuações de acurácia, F1, precisão, recall e matriz de confusão foram utilizados para avaliar a performance de cada modelo utilizado nessa avaliação.



### Árvore de Decisão (Decision Tree)

No modelo Decision Tree, utilizei a biblioteca DecisionTreeClassifier do pacote sklearn.tree do Scikit-learn.

Após testar uma série de combinações possíveis de parametrização através de validações cruzadas com a biblioteca GridSearchCV, o resultado foi que a melhor combinação para o treino do modelo de árvore de decisão seria:

- Para a lógica de divisão, com o objetivo de medir a qualidade da divisão, foi utilizado o parâmetro “criterion” como “entropy”. Foi escolhido ao invés dos valores “gini” ou “log\_loss”. O impacto dessa decisão influencia como a árvore vai dividir os nós na hora do treino.

- Para a lógica de limite do tamanho da árvore, com o intuito de ajudar a prevenir o overfitting , foram usados os parâmetros:

- “max\_depth”: 10, que mantém fixo a profundidade máxima da árvore em 10 níveis

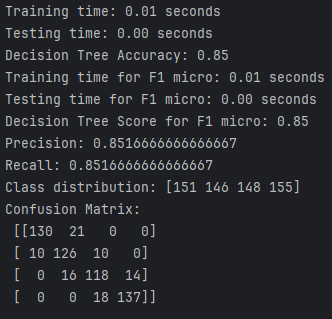
- “min\_samples\_split”: 2, sendo o número mínimo de amostras do conjunto de dados necessária para dividir um nó interno. Por padrão, esse valor já é 2.

- “min\_samples\_leaf”: 5, sendo o número mínimo de amostras necessárias nos nós folha.

- “max\_features”: None, sendo o número de features usadas na melhor divisão dos nós. Nesse caso, sendo None, todas as features do dataset foram consideradas.

- E para a randomização, foi utilizado o “random\_score” = 42, pois utilizar um valor inteiro fixo é importante para garantir a reprodutividade dos resultados.

Após atribuir os valores aos parâmetros estabelecidos acima, realizei o treino e o teste do modelo. Os resultados obtidos foram:



É possível ver acima que tanto o tempo de treino e teste para a acurácia da árvore de decisão e seu F1 Score (utilizando a abordagem micro, que combina os resultados de todas as classes, considerando cada amostra individualmente, através da soma dos verdadeiros positivos (TP), falsos positivos (FP), e falsos negativos (FN) para todas as classes antes de calcular precisão e recall) foram iguais. Além disso, a pontuação para as 2 abordagens também foi a mesma, com 85% de assertividade.

Além disso, podemos ver que através da classe de distribuição, temos 4 valores muito próximos de cada um, o que demonstra que o conjunto de dados está bem balanceado.

Por fim, através da matriz de confusão, podemos ver que a quantidade de elementos fora da diagonal (elementos que foram erronamente preditos) foi baixa em relação ao total de

predições, o que também demonstra que o modelo conseguiu fazer uma boa predição dos dados.

### Random Forest

No modelo Random Forest, utilizei a biblioteca RandomForestClassifier do pacote sklearn.ensemble do Scikit-learn.

Após testar uma série de combinações possíveis de parametrização através de validações cruzadas com a biblioteca GridSearchCV, o resultado foi que a melhor combinação para o treino do modelo de árvore de decisão seria:

### 

### 

- Para controlar a amostragem com repetição, utilizei o parâmetro “bootstrap” como verdadeiro. O processo de bootstraping trata-se de uma amostragem com repetição, ou seja, ao criar cada árvore da Random Forest, o conjunto de treinamento é amostrado aleatoriamente, permitindo que um mesmo ponto de dados seja selecionado mais de uma vez. Isso é importante pois previne o overfitting (performa muito bem no treino, porém apresenta uma má generalização nos testes) e garante que cada árvore veja um conjunto de dados ligeiramente diferente.

- A quantidade de estimadores (número de árvores de decisão da floresta) calculada foi de 100. Através dos testes, foi testada quantidades maiores, que, no entanto, não surtiram em um ganho de performance e ainda resultaram em um custo computacional maior.

- Os pesos de classe (class\_weight) foram setados como ‘balanced’, que faz com que esses pesos sejam ajustados de forma inversamente proporcional às frequências das classes. Esse parâmetro é importante para ajudar o modelo a lidar com desbalanceamentos do conjunto de dados dando maior importância as classes minoritárias.

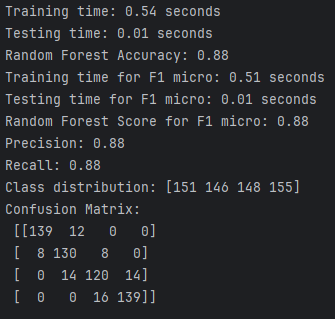
- “max\_depth”: 10, que mantém fixo a profundidade máxima da árvore em 10 níveis

- “min\_samples\_split”: 2, sendo o número mínimo de amostras do conjunto de dados necessária para dividir um nó interno. Por padrão, esse valor já é 2.

- “min\_samples\_leaf”: 5, sendo o número mínimo de amostras necessárias nos nós folha.

- “max\_features”: None, sendo o número de features usadas na melhor divisão dos nós. Nesse caso, sendo None, todas as features do dataset foram consideradas.

Após a atribuição dos parâmetros acima, obtive os seguintes resultados pós treino e teste:



Em relação ao modelo de árvore de decisão, o modelo Random Forest apresentou tempos superiores de treino e teste, porém, que ainda são extremamente baixos, praticamente apresentando pouca diferença. Olhando para a pontuação em acurácia. F1, precisão e recall, todos tiveram um crescimento de 3 pontos percentuais, agora com 88% de assertividade.

A distribuição de classes continua com os mesmos valores da árvore de decisão, com os mesmos 4 valores, bem próximos entre si, mostrando que o conjunto está bem balanceado.

Por fim, a matriz de confusão apresentou melhora nos acertos da predição em relação ao modelo anterior, prevendo menos resultados errados em relação aos reais, tendo a maior diferença no elemento da primeira linha com a segunda coluna (previu a classe B mas o real era a classe A), errando 21 elementos na árvore de decisão e agora 12 no Random Forest.

### Multilayer Perceptron (MLP)

No modelo MLP, utilizei a biblioteca MLPClassifier do pacote sklearn.neural\_network do Scikit-learn.

Antes de treinar o modelo MLP, apliquei um escalonador (Scaler), que é altamente recomendado pois modelos de redes neurais são sensíveis à escala dos dados. Esse escalonamento é importante uma vez que esses modelos usam gradientes que ajustam os pesos dos neurônios, e se as variáveis de entrada não estiverem na mesma escala, o cálculo desses gradientes pode ficar desbalanceado. Além disso, o modelo MLP usa funções de ativação que apresentam respostas não lineares e também são sensíveis a escala dos dados. Por fim, escalonar também se faz importante pois ajudam na convergência do modelo, uma vez que quando os dados estão escalonados, a convergência é mais rápida e o modelo tende a encontrar uma solução mais eficaz.

Portanto, optei por utilizar o escalonador padrão “StandardScaler”, que subtrai a média e divide pelo desvio padrão, de modo que os dados de cada característica tenham média 0 e desvio padrão 1, o que faz com que todas as características contribuam de forma equilibrada para o aprendizado do modelo.

Após testar uma série de combinações possíveis de parametrização através de validações cruzadas com a biblioteca GridSearchCV, o resultado foi que a melhor combinação para o treino do modelo de árvore de decisão seria:



- A função de ativação tahn (‘activation’: ‘tahn’) é usada nas camadas ocultas da rede neural, e é uma função hiperbólica tangente, ou seja, ela mapeia os valores entre -1 e 1, o que é interessante quando se busca valores próximos de 0.

- O parâmetro ‘alpha’ é importante para controlar a regularização L2, que também é chamada de “Ridge regularization”. Sua importância se dá pelo fato de ajudar a evitar overfitting, pois esse parâmetro penaliza pesos muito grandes, balanceando-os. É necessário obter um valor bom para ‘alpha’ pois um valor alto aumenta a penalização de pesos grandes e pode reduzir a complexidade do modelo, o que é bom, porém pode gerar um grande trade-off de redução de poder de aprendizado. Como o valor utilizado será 0.0001, a regularização será relativamente fraca.

- A “learning\_rate”, que é a métrica que define qual estratégia será utilizada para atualizar a taxa de aprendizado durante o treinamento foi definida como “constant”, ou seja, a taxa de aprendizado se manterá fixa durante o aprendizado. Ela também poderia ser definida de forma a diminuir ao longo do tempo de treino ou se ajustar automaticamente com base em caso a acurácia não melhorasse a cada interação, reduzindo-a.

- O parâmetro “hidden\_layer\_sizes” define o número de neurônios em cada camada oculta da rede neural. No caso do meu conjunto de dados, foi definido uma tupla, logo, esse modelo utilizará 2 camadas ocultas, cada uma com 100 neurônios, logo, (100, 100). Conforme fossem adicionadas mais camadas ou neurônios, poderia aumentar a capacidade do modelo aprender padrões mais complexos, porém aumentaria o tempo despendido pelo modelo.

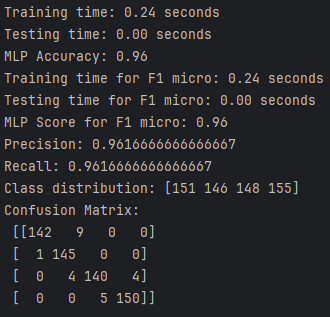
- “max\_iter” é a definição do número máximo de iterações (epochs) para o algoritmo de otimização, em outras palavras, quantas vezes o modelo irá passar pelos dados durante seu treino. Com o máximo de iterações = 1000, quer dizer que ele será treinado com 1000 iterações no máximo, e caso o modelo venha a convergir antes de alcançar todas essas iterações, o treino é interrompido antes.

- Para evitar o overfitting, foi atribuído também o recurso de “early\_stopping", que garante que o treinamento será interrompido caso a performance do conjunto de validação não melhore após um número definido de iterações.

- A taxa de aprendizado inicial (learning\_rate\_init) foi definida como 0.0001, o que significa que no início do treinamento, o modelo fará pequenas atualizações nas primeiras iterações.

- Por fim, o algoritmo de otimização escolhido foi o “lbfgs”, que é um método baseado na otimização numérica, que é adequado a redes neurais com poucas camadas e dados pequenos, ele é baseado em gradiente e é mais preciso, porém pode ser mais lento.

Após a atribuição dos parâmetros acima, realizei o treino e teste do modelo MLP e obtive os seguintes resultados:



Em comparação com os modelos anteriores, o MLP apresentou um tempo de treino tanto para acurácia e F1 score, maior que o da Árvore de Decisão, porém menor que o Random Forest. Além disso, teve um tempo de teste similar à Árvore de Decisão.

Em relação a suas pontuações, foi o que mais se aproximou do perfeito, com a acurácia, F1 micro score, precisão e recall com 96% de assertividade, contra 88% da Random Forest e 85% da Árvore de Decisão.

Assim como nos 2 modelos anteriores, a distribuição de classe ficou igual, com valores muito próximos, mostrando balanceamento no conjunto de dados.

Por fim, ao olhar a matriz de confusão, vemos uma grande melhora em relação aos 2 modelos anteriores, zerando um valor fora da diagonal, que apresentava uma predição errada, quase zerando o erro na predição da classe A para B e diminuindo também outros erros de predição.