Questão 2 -

O primeiro passo para resolver essa questão foi a importação e instalação das bibliotecas necessárias para a visualização e execução do código (no código estão detalhadas as características específicas de cada biblioteca). O próximo passo foi estabelecer o estilo dos gráficos e carregar os dados, onde X recebe as características e y a variável alvo, o método .squeeze() permite simplificar e preparar os dados, transformando-os em 1D.

A função using_knn foi criada a fim de treinar, avaliar e definir o melhor k, além de permitir a plotagem dos gráficos para melhor visualização dos dados. Para isso, foram criadas as listas que armazenam os valores de cada k, possibilitando o cálculo posterior da média de cada execução. Um loop foi criado, permitindo que essa parte do código seja repetida para todos os 5 valores de k. O tempo é devidamente calculado para posterior plotagem e o classificador KNN é determinado pela métrica de distância euclidiana.

Os dados foram divididos da seguinte forma, 6000 amostras de treino selecionadas aleatoriamente a cada execução e o restante para teste, e para a determinação do melhor k, essas amostras de treino foram subdivididas na metade, uma para treino e outra para validação.

Dessa forma, o modelo foi treinado com os dados de treino e posteriormente validado pelo conjunto de validação. Conjunto este que tem a acurácia avaliada para a determinação do melhor k.

Após o loop descrito acima, o modelo do melhor k é aplicado nos dados de teste.

Os gráficos plotados representam os valores de acurácia, precisão e recall de cada k, além de marcar a média dos valores da execução.

Para finalizar, o loop ‘for i in range(5):’, permite que o treinamento e teste seja executado 5 vezes, retornando a acurácia, recall e precisão média para cada algoritmo, conforme solicitado no enunciado da questão.

Os resultados, tempo de execução e número de protótipos foram plotados para cada k usado no código, podendo ser, dessa forma, comparados. Vale ressaltar que o número de protótipos usados em cada análise e execução é fixo e determinado na divisão do dataset.

Considerando o gráfico 1, retirado da base de dados, podemos verificar a concentração dos dados em uma só classe, sendo desbalanceada. Dessa forma, a avaliação apenas pela acurácia pode representar um panorama geral, mas não uma visão completa do modelo. Para uma melhor visualização deve-se considerar outras métricas, como precisão e recall, garantindo que o modelo seja eficaz para todas as classes de dados e não apenas para a majoritária. Métodos como o “stratify” usado na divisão do dataset, tentam amenizar a tendência provocada pelo desbalanceamento dos dados, pois ele permite que a proporção de classes no conjunto de treinamento e no conjunto de teste seja a mesma que no conjunto de dados original.

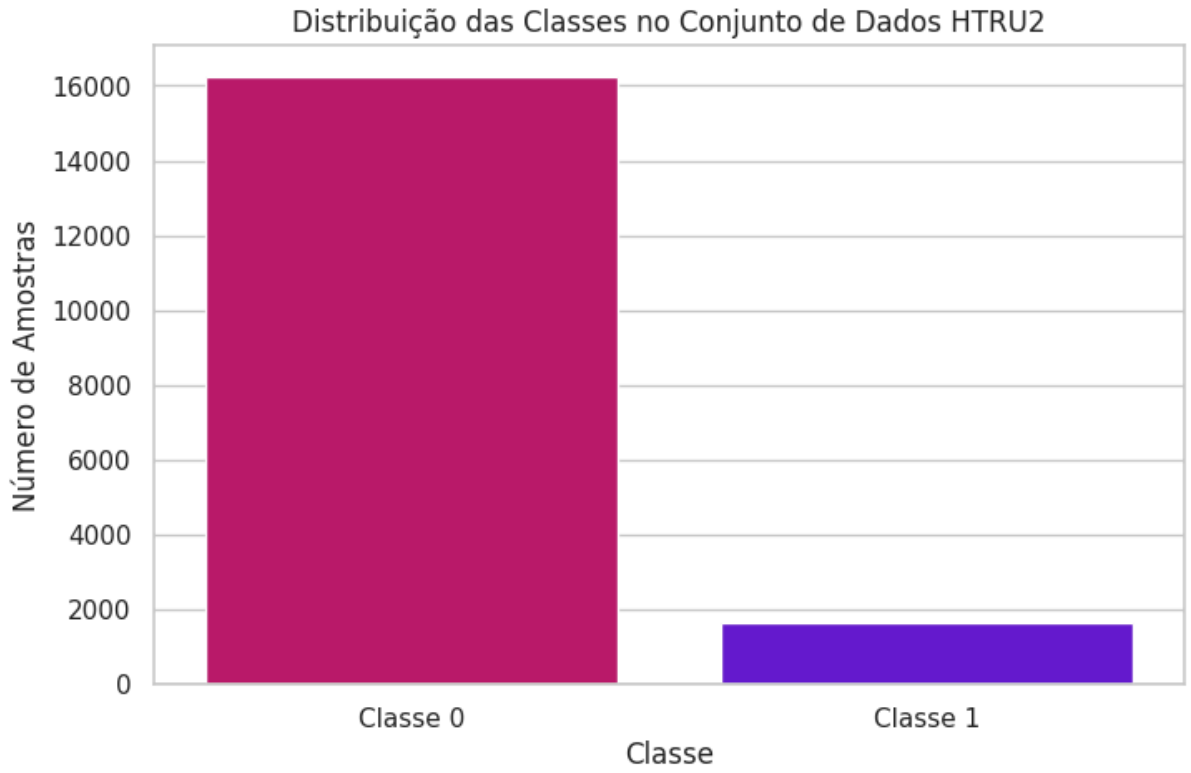

Gráfico 1. Distribuição das classes na base de dados. (Do Autor)

Questão 3 -

Essa base de dados contém informações relacionadas à admissão de crianças em creches e possui diversas características que indicam critérios importantes para esta classificação. Para essa análise foi solicitado uma árvore de decisão, implementada com os passos descritos abaixo.

Primeiramente, foi realizada a instalação da base de dados sugerida, assim como a importação das bibliotecas necessárias para a execução e visualização das questões propostas. Foi necessário transformar as variáveis categóricas em númericas, para que fosse possível o código da árvore de decisão processar os dados, para isso foi usado o método de one-hot encoding (X_encoded = pd.get_dummies(X)).

O próximo passo adotado foi a divisão da base de dados, conforme o enunciado, em 10000 amostras aleatórias dos dados para treinamento do modelo e as demais para avaliação. Acredito ser importante ressaltar as instâncias random_state=42 e stratify=y, para esse exercício, foi acrescentado o random_state a fim de que os dados separados sejam aleatórios, mas reproduzíveis, para que mesmo em outras execuções, tenham o mesmo resultado, enquanto o stratify auxilia em possíveis desbalanceamentos de dados.

A árvore de decisão foi criada com base no critério de Ganho de Informação, indicada no código através do comando: criterion='entropy'. A profundidade de dois níveis de nós de decisão foi determinada considerando que os nós de decisão são os nós que obtemos após dividir os nós raiz, fato determinado no código por: max_depth=2. Em seguida, o modelo foi treinado com os dados de treino e plotados usando a função tree.plot_tree da biblioteca sklearn, onde foi determinado os nomes das características e das classes, os nós foram preenchidos com cores baseadas na classe, as bordas foram formatadas como arredondadas e o tamanho da fonte definido em 10.

Para a letra b, a acurácia foi calculada aplicando o modelo nos dados de validação, através do método decision_tree.score.
Enquanto que para responder a letra c do problema, as regras de decisão da árvore foram extraídas pelo comando tree.export_text, as três linhas seguintes do código foram tratativas dessa saída das regras, a fim de melhorar a visualização desses dados, plotados na última linha do código.