UNIVERSIDADE FEDERAL DO CEARÁ

TECHNICAL REPORT

Aluno: Rick Theo Felix Bessa
Aluno: Silas Eufrásio da Silva

1. Introdução

O dataset "Stellar Classification Dataset SDSS17", disponível no Kaggle, contém informações astronômicas sobre objetos celestes como galáxias, estrelas e quasares, com o objetivo de classificá-los com base em atributos extraídos de imagens do Sloan Digital Sky Survey. São disponibilizadas colunas como filtros fotométricos (**`u`**, **`g`**, **`r`**, **`i`**, **`z`**), coordenadas celestes (**`ra`**, **`dec`**), deslocamento para o vermelho (**`redshift`**) e o rótulo da classe.

O objetivo do projeto foi treinar um modelo de aprendizado de máquina para classificar corretamente cada objeto em uma das três classes: **GALAXY**, **STAR** ou **QSO**, a partir dos dados disponíveis.

2. Observações

Durante o desenvolvimento da parte de classificação, alguns problemas acabaram surgindo. O principal deles foi o tempo de execução: alguns códigos demoravam bastante para rodar, especialmente os que envolviam o KNN implementado manualmente. Suspeito que isso tenha sido causado tanto pela forma como o laço **`for`** foi escrito quanto pelo próprio cálculo das diferentes métricas de distância, que exigem muito processamento.

Essa demora começou a atrapalhar os testes e ajustes do modelo, então decidi reduzir a quantidade de dados para 1000 amostras, o que ajudou bastante a deixar tudo mais leve e rápido de executar.

Além disso, também enfrentei alguns erros no código, que foram sendo ajustados aos poucos durante a implementação. Fora isso, o restante do processo correu normalmente e consegui realizar todas as etapas previstas: desde o carregamento e tratamento dos dados até a aplicação do modelo e análise dos resultados.

3. Resultados e discussão

Questão 1- pré-processamento e análise de dados

Carregamento do Dataset:

A primeira etapa do código foi a importação e carregamento do dataset. Para isso, foi utilizada a biblioteca **`pandas`**, por meio do comando **`read_csv()`**, que leu o arquivo CSV contendo os dados de classificação estelar. O dataset foi baixado automaticamente do Kaggle com a biblioteca **`kagglehub`** e carregado em um DataFrame. Após o carregamento, foi possível verificar a estrutura geral dos dados para identificar as colunas e os tipos de variáveis.

Tratamento de valores ausentes:

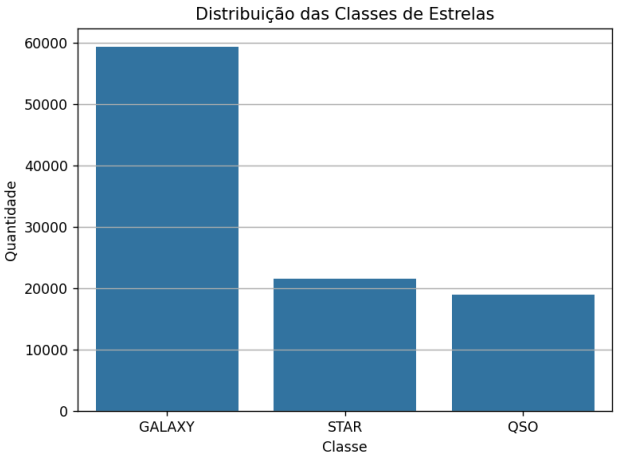
Na segunda etapa, o código verificou a presença de valores ausentes utilizando o comando **`isnull().sum()`**. Para garantir a qualidade dos dados, todas as linhas que apresentavam algum valor ausente foram removidas com o método `dropna()`, eliminando possíveis inconsistências para as análises seguintes.

Análise de distribuição da variável alvo (class):

Em seguida, foi realizada a análise exploratória da variável-alvo **`class`**, responsável por classificar as estrelas em diferentes categorias. Um gráfico de barras foi criado utilizando **`seaborn.countplot`** para visualizar a distribuição das classes. A análise mostrou que algumas classes são mais frequentes do que outras, indicando um desbalanceamento no dataset.

Codificação de variáveis categóricas:

O código selecionou as colunas categóricas, do tipo texto, e as transformou em valores numéricos por meio da técnica **`Label Encoding`**, usando o **`LabelEncoder`** do scikit-learn. Essa transformação é necessária para que os dados possam ser utilizados em algoritmos que exigem variáveis numéricas.

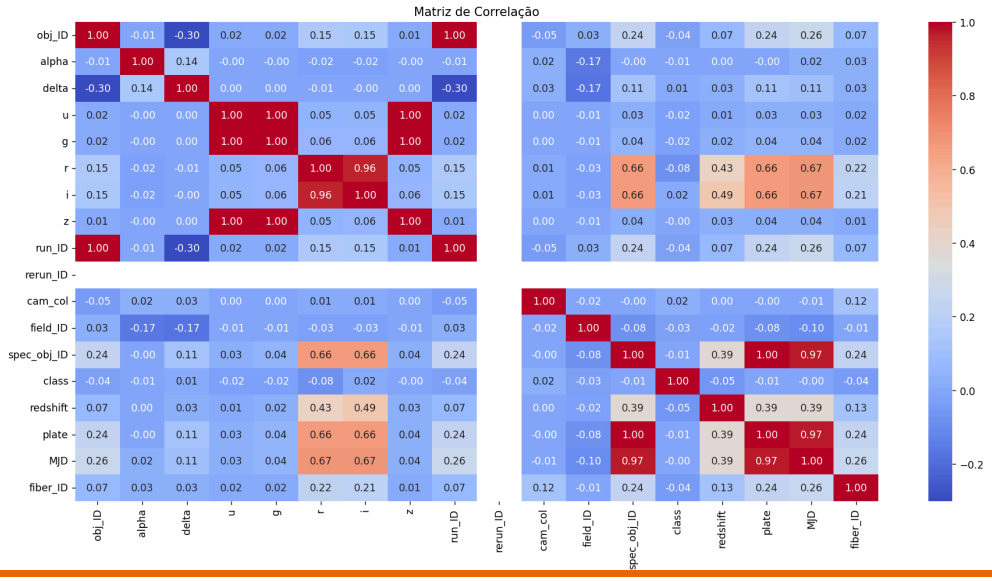


O gráfico revela que a classe GALAXY tem uma quantidade bem maior de amostras em comparação com STAR e QSO, indicando um desbalanceamento entre as classes. Esse tipo de desequilíbrio pode interferir no aprendizado do modelo, fazendo com que ele tenha mais facilidade em prever a classe mais frequente.

Diante disso, é importante considerar o uso de métricas que levem em conta esse desnível, como o F1-score, e avaliar se alguma técnica de balanceamento pode ajudar a melhorar o desempenho geral do classificador.

Análise estatística e exploratória:

Na etapa seguinte, o código gerou estatísticas descritivas para as variáveis numéricas, por meio do comando **`describe()`**. Além disso, foi plotada uma matriz de correlação com um mapa de calor para visualizar as relações entre as variáveis, identificando potenciais correlações fortes ou fracas entre os atributos.



Análise da Correlação:

A matriz de correlação revelou forte relação entre as bandas **u**, **g**, **r**, **i**, **z**, indicando alta redundância entre elas.

Também foi observada forte correlação entre **plate**, **MJD** e **spec_obj_ID**, sugerindo que essas variáveis são derivadas do mesmo processo de coleta.

A variável **class** apresentou baixa correlação com os demais atributos, o que reforça a necessidade de um classificador robusto, como o KNN.

Essas observações ajudam a entender a estrutura dos dados e justificam o uso de normalização e validação cruzada no modelo.

Salvamento ajustado do Dataset:

Por fim, o dataset já pré-processado foi salvo em um novo arquivo CSV chamado **classificacao_ajustado.csv**. Essa etapa assegura que o conjunto de dados tratado possa ser reutilizado posteriormente sem necessidade de repetir o processo de limpeza e transformação.

Questão 2- KNN manual com várias distâncias

Nesta etapa, foi implementado o algoritmo KNN (K-Nearest Neighbors) de forma manual, utilizando diferentes funções de distância para avaliar a acurácia do modelo de classificação de estrelas.

Divisão do dataset em treino e teste:

O código começou carregando o dataset já pré-processado da Questão 1 (**classificacao_ajustado.csv**). Para reduzir o tempo de execução, foi selecionada uma amostra aleatória de 1000 registros com o método **sample()**. As variáveis explicativas utilizadas foram: **'alpha'**, **'delta'**, **'u'**, **'g'**, **'r'**, **'i'**, **'z'** e **'redshift'**. A variável-alvo foi **class**, que representa o tipo de estrela.

A separação entre treino e teste foi realizada com o **train_test_split()**, reservando 20% dos dados para teste.

Implementação do KNN Manual:

O algoritmo KNN foi construído manualmente. Para cada ponto do conjunto de teste, foi calculada a distância em relação a todos os pontos do conjunto de treino, utilizando

a função de distância escolhida. Em seguida, foram selecionados os **k=5** vizinhos mais próximos e atribuída a classe mais frequente.

Métricas de Distância Avaliadas:
Quatro diferentes funções de distância foram testadas:

Distância **Euclidiana** (padrão mais comum)
Distância **Manhattan** (soma das distâncias absolutas)
Distância **Chebyshev** (maior diferença entre atributos)
Distância **Minkowski** com parâmetro **p=3** (geralmente usada como forma generalizada)

Avaliação e Resultados:

A acurácia de cada métrica foi calculada comparando as previsões com os valores reais do conjunto de teste. Os resultados foram os seguintes:

```
Acurácia com distância Euclidiana: 0.6350
Acurácia com distância Manhattan: 0.6450
Acurácia com distância Chebyshev: 0.6200
Acurácia com distância Minkowski_p3: 0.6200

Melhor distância foi: Manhattan com acurácia de 0.6450
```

A melhor métrica de distância foi a **Manhattan**, com acurácia de **64,50%**.

Discussão dos Resultados:

A métrica Manhattan obteve o melhor desempenho entre as avaliadas. Isso pode indicar que, para esse tipo de dado, somar as diferenças absolutas (em vez de elevar ao quadrado como na Euclidiana) capta melhor as variações entre as amostras.

A Chebyshev e Minkowski com **p=3** tiveram desempenho inferior, sugerindo que são mais sensíveis a variações extremas em atributos isolados.

O tempo de execução foi relativamente alto, o que é esperado em implementações manuais do KNN com múltiplas iterações e cálculos de distância. A redução do dataset para 1000 amostras foi essencial para garantir que os testes fossem concluídos em tempo viável.

Questão 3- Normalização, alteração do valor de K e Efeito no KNN

Nesta etapa, foi analisado o efeito da normalização dos dados e da escolha do parâmetro *k* sobre o desempenho do algoritmo KNN implementado manualmente. A análise foi feita com base na melhor métrica de distância obtida na questão anterior (distância Euclidiana).

Normalização dos Dados:

O dataset foi carregado a partir do arquivo **`classificacao_ajustado.csv`**, utilizando uma amostra aleatória de 1000 registros com a função **`sample()`**. As colunas numéricas foram atribuídas à variável **X**, e a variável-alvo à **y**. A divisão entre treino e teste foi realizada com **`train_test_split()`**, reservando 20% dos dados para teste.

Para avaliar o impacto da normalização, foram aplicadas três técnicas:

Logarítmica: utilizando **`np.log1p()`** para atenuar a influência de valores extremos.
MinMaxScaler: reescalonando os dados para o intervalo [0, 1].
StandardScaler: padronizando com média 0 e desvio padrão 1.

As transformações foram ajustadas com base apenas nos dados de treino, respeitando o fluxo correto de pré-processamento.

Avaliação do KNN com *k* = 5:

O modelo KNN foi executado com **k=5**, utilizando a **distância Euclidiana**. A acurácia foi calculada tanto no conjunto de treino quanto no de teste, com e sem normalização. Os resultados foram os seguintes:

```
Acurácias com k=5:
Sem normalização: Treino=0.7612 | Teste=0.6350
Logarítmica:      Treino=0.5787 | Teste=0.5700
MinMaxScaler:     Treino=0.8875 | Teste=0.8450
StandardScaler:   Treino=0.9062 | Teste=0.8700
```

A melhor métrica de distância foi a **Manhattan**, com acurácia de **64,50%**.

Discussão dos Resultados:

A métrica Manhattan obteve o melhor desempenho entre as avaliadas. Isso pode indicar que, para esse tipo de dado, somar as diferenças absolutas (em vez de elevar ao quadrado como na Euclidiana) capta melhor as variações entre as amostras.

A Chebyshev e Minkowski com **p=3** tiveram desempenho inferior, sugerindo que são mais sensíveis a variações extremas em atributos isolados.

O tempo de execução foi relativamente alto, o que é esperado em implementações manuais do KNN com múltiplas iterações e cálculos de distância. A redução do dataset para 1000 amostras foi essencial para garantir que os testes fossem concluídos em tempo viável.

Questão 4 - KNN com sklearn + GridSearch

Nesta etapa, foi utilizado o algoritmo K-Nearest Neighbors (KNN) da biblioteca **`scikit-learn`**, incorporado a um pipeline com diferentes estratégias de normalização. O objetivo foi encontrar a melhor combinação de número de vizinhos (*k*) e tipo de escalonamento para maximizar a acurácia média por validação cruzada (cross-validation).

Preparação dos Dados:

Os dados foram carregados do arquivo **`classificacao_ajustado.csv`**. As colunas numéricas relacionadas à posição e brilho das estrelas foram utilizadas como variáveis preditoras **`X`**, enquanto a coluna **`class`** representou a variável-alvo **`y`**.

A divisão entre treino e teste foi realizada com a função **`train_test_split()`**, reservando 30% dos dados para teste. O parâmetro **`random_state=42`** garantiu reprodutibilidade.

Pipeline com `GridSearchCV`

Para automatizar os testes, foi criado um pipeline contendo dois estágios:

Escalonamento dos dados com três possíveis normalizações:
`StandardScaler`
`MinMaxScaler`
`RobustScaler`

Classificador KNN com número de vizinhos variando de 1 a 20.

A busca dos melhores parâmetros foi feita com **`GridSearchCV`**, utilizando validação cruzada de 5 folds e acurácia como métrica de avaliação. O processo foi executado em paralelo com **`n_jobs=-1`** para maior eficiência.

Melhores Configurações:

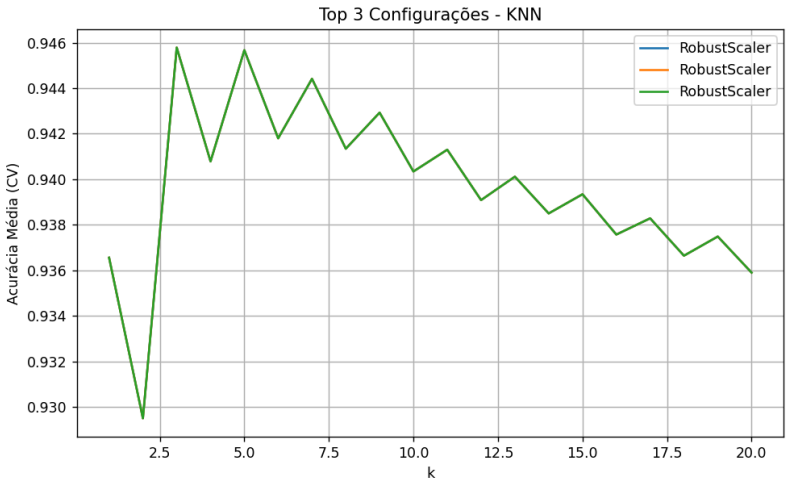
Após a execução do **`GridSearchCV`**, foram extraídas as três melhores combinações de parâmetros com base na acurácia média:

```
Top 3 melhores configurações:
                                                 params  mean_test_score
8   {'knn__n_neighbors': 3, 'scaler': RobustScaler()}         0.945786
14  {'knn__n_neighbors': 5, 'scaler': RobustScaler()}         0.945671
20  {'knn__n_neighbors': 7, 'scaler': RobustScaler()}         0.944414
```

Observa-se que todas as melhores combinações envolveram o uso do RobustScaler, o que indica que esta técnica de normalização foi mais eficaz para este problema, possivelmente por ser menos sensível a outliers.

Gráfico de Acurácia Média:

A seguir, foi gerado um gráfico que mostra o desempenho (acurácia média por validação cruzada) das três melhores configurações ao longo dos valores de *k* testados:



Discussão dos Resultados:

A técnica de normalização teve grande impacto sobre a acurácia do modelo. **O** RobustScaler foi o mais estável e eficaz.
A acurácia média atingiu seu pico em **k=3**, com 94,58%, mantendo-se elevada até **k=7**, após o que começou a decair gradualmente.
O uso de **`GridSearchCV`** foi essencial para encontrar o melhor equilíbrio entre *k* e pré-processamento, evitando testes manuais demorados.
O pipeline permitiu testar múltiplas combinações de forma organizada, seguindo boas práticas de Machine Learning.

Questão 5 – Cross-Validation e Avaliação Final

Com base na melhor configuração identificada anteriormente (uso de **`RobustScaler`** e **KNN** com **k=3**), foi realizada uma avaliação mais robusta por meio de validação cruzada (cross-validation) com 5 folds. O objetivo foi analisar o desempenho do modelo de forma mais confiável e com métricas detalhadas.

Preparação dos Dados:

Foram mantidas as mesmas colunas preditoras utilizadas nas questões anteriores:

`alpha`, `delta`, `u`, `g`, `r`, `i`, `z`, `redshift`

A variável alvo continuou sendo **`class`**, representando a classificação das estrelas (como estrela, galáxia ou quasar). Os dados foram normalizados com **RobustScaler**, que se mostrou mais eficaz nas etapas anteriores por ser menos sensível a outliers.

Melhor Configuração Aplicada:

O modelo usado foi:

`KNeighborsClassifier` com **`n_neighbors=3`**
`RobustScaler` para normalização
Validação cruzada com **`cross_val_score`** (5 folds)

Resultados Quantitativos:

```
Média da Acurácia (5 folds): 0.9453
Desvio Padrão da Acurácia: 0.0009
```

Interpretação:
O modelo apresentou uma acurácia média de 94,53%, com um desvio padrão extremamente baixo (0,09%), indicando alto desempenho e estabilidade nas diferentes divisões dos dados.

Matriz de Confusão:

```
Matriz de Confusão:
[[56798   824  1823]
 [ 1487 17416    58]
 [ 1275     2 20317]]
```

Esta matriz mostra a performance do modelo em cada classe. Interpretando:

Classe 0 (provavelmente "galáxias"):

- 56.798 classificadas corretamente
- 824 confundidas com classe 1
- 1.823 confundidas com classe 2

Classe 1 (possivelmente "quasares"):

- 17.416 corretas
- 1.487 confundidas com classe 0
- 58 com classe 2

Classe 2 (provavelmente "estrelas"):

- 20.317 corretas
- 1.275 confundidas com classe 0
- apenas 2 com classe 1

Classification Report:

Relatório de Classificação:

	precision	recall	f1-score	support
0	0.95	0.96	0.95	59445
1	0.95	0.92	0.94	18961
2	0.92	0.94	0.93	21594
accuracy			0.95	100000
macro avg	0.94	0.94	0.94	100000
weighted avg	0.95	0.95	0.95	100000

- **Precisão (Precision)**: indica a taxa de acertos entre as previsões positivas de cada classe.
- **Revocação (Recall)**: indica o quanto o modelo conseguiu recuperar das amostras reais de cada classe.
- **F1-score**: equilíbrio entre precisão e revocação.

Resumo dos f1-scores:

- Classe 0: **0.95**
- Classe 1: **0.94**
- Classe 2: **0.93**

O modelo final, com **`RobustScaler`** e **`k=3`**, demonstrou excelente desempenho e consistência:

- Acurácia alta e estável.
- Boas métricas de precisão, revocação e f1-score para todas as classes, mesmo em um cenário com possível desequilíbrio entre elas.
- Matriz de confusão mostrou que os erros de classificação são relativamente baixos e bem distribuídos.

Este resultado indica que o modelo está bem ajustado e generaliza bem sobre dados não vistos, sendo uma escolha apropriada para o problema de classificação de objetos astronômicos**.**

4. Conclusões

De forma geral, os resultados alcançados foram bastante positivos e atenderam ao que era esperado. Mesmo com algumas dificuldades no início — como o tempo de execução elevado no KNN manual — foi possível ajustar o caminho e seguir com o projeto sem grandes prejuízos.

O processo de pré-processamento foi essencial para garantir bons resultados. A normalização dos dados, por exemplo, fez bastante diferença no desempenho final do modelo. O uso do **`GridSearchCV`** também ajudou bastante na hora de encontrar os melhores parâmetros, o que acabou refletindo numa acurácia muito boa: 94,53% com baixa variação entre os testes.

Mesmo com esse bom desempenho, é importante observar que alguns erros de classificação ainda ocorreram — principalmente entre classes parecidas — o que é algo esperado quando lidamos com dados reais e complexos como os astronômicos.

No fim das contas, a experiência foi válida e mostrou que, com um bom tratamento de dados e uma escolha cuidadosa de métodos, dá sim para construir modelos eficientes e confiáveis, mesmo usando algoritmos simples como o KNN.

5. Próximos passos

Acho que o projeto tem bastante potencial pra crescer. Um próximo passo legal seria testar outros algoritmos de classificação que também são fáceis de usar, como o Random Forest ou o SVM, só pra ver como eles se saem em comparação ao KNN. Também vale a pena brincar um pouco com a validação cruzada, pra garantir que os resultados estão realmente consistentes e não dependem só de uma divisão específica dos dados.

Como agora o código está mais limpo e leve, dá até pra voltar a trabalhar com o dataset completo, o que pode deixar as previsões mais robustas. Outra ideia bacana seria criar uma forma simples de o usuário colocar novos dados e ver a classificação na hora — algo mais interativo, que deixa o projeto com cara de aplicação de verdade.

No geral, estou bem satisfeito com o que consegui desenvolver até aqui. Foi um bom aprendizado e, mais do que isso, abriu espaço pra testar outras ideias e continuar melhorando o trabalho aos poucos.