Classificação de Objetos Celestes

February 24, 2025

1 Introdução

As características espectrais são fundamentais para diferenciar distintos tipos de objetos celestes, como **estrelas**, **galáxias** e **quasares**. As estrelas emitem radiação eletromagnética de maneira semelhante a um corpo negro, exibindo linhas de absorção específicas. No caso das galáxias, o seu espectro é resultado da soma dos espectros de todas as estrelas e de demais materiais radiantes que as compõem. Os quasares por outro lado, apresentam linhas de emissão marcantes, frequentemente acompanhadas por um desvio expressivo para o vermelho (redshift).

2 Motivação e Contexto do DataSet

O **Sloan Digital Sky Survey (SDSS)** é um projeto de pesquisa astronômica baseado em imagens, conduzido com um telescópio grande angular de 2,5 metros localizado no Observatório de Apache Point, no Novo México (Estados Unidos). O objetivo do projeto é mapear um quarto do céu visível, obter observações de aproximadamente 100 milhões de objetos e o espectro de um milhão de objetos. O levantamento inclui informações espectrais e fotométricas de todos os objetos astronômicos detectados, incluindo **estrelas**, **galáxias** e **quasares**.

Os dados são publicados periodicamente e disponibilizados publicamente. Este conjunto de dados corresponde aos dados da Data Release 17 (DR17) [1]. .

2.1 Importância do Estudo

Categorizar objetos astronômicos com base em características espectrais e fotométricas é essencial para entender a evolução estelar e a formação de galáxias. As análises espectrais revelam a composição e temperatura das estrelas, enquanto as características fotométricas fornecem dados sobre luminosidade e distância. Juntas, essas informações ajudam a mapear a história do universo e os processos que moldam sua estrutura.

3 Dados

- **obj_ID:** Identificador do objeto, um valor único que identifica o objeto no catálogo de imagens utilizado pelo arquivo de dados do Sloan Digital Sky Survey (SDSS).
- **alpha:** Ângulo de Ascensão Reta (em graus) na época J2000, um sistema de coordenadas amplamente utilizado em astronomia para descrever a posição de objetos celestes no céu.
- **delta:** Ângulo de Declinação (em graus) na época J2000, outro sistema de coordenadas comumente usado em astronomia para localizar objetos celestes.

- **u, g, r, i, z:** Filtros fotométricos usados no sistema SDSS para medir a quantidade de luz emitida pelos objetos em diferentes faixas de comprimento de onda. Cada filtro corresponde a uma cor específica da luz: u no ultravioleta, g no verde, r no vermelho, i no infravermelho próximo e z no infravermelho.
- **run_ID:** Número de execução que identifica uma varredura específica do céu realizada pelo SDSS. Cada varredura cobre uma região determinada do céu e recebe um número único.
- **rerun_ID:** Número de reprocessamento que indica como a imagem foi processada, incluindo a versão do software ou o método de calibração utilizado na criação da imagem.
- **cam_col:** Coluna da câmera, usada para identificar a linha de varredura dentro da execução. Cada varredura é dividida em múltiplas colunas da câmera para cobrir uma área maior do céu.
- **field_ID:** Número do campo, utilizado para identificar cada campo dentro da varredura. O campo representa uma região menor dentro da coluna da câmera.
- **spec_obj_ID:** Identificador único para objetos espectroscópicos ópticos. Isso significa que duas observações diferentes com o mesmo spec_obj_ID devem pertencer à mesma classe de objeto, que pode ser galáxia, estrela ou quasar.
- **class:** Classe do objeto, a classificação atribuída ao objeto com base em suas propriedades espectrais. Pode ser uma galáxia, uma estrela ou um quasar.
- **redshift:** Valor do desvio para o vermelho, baseado no aumento do comprimento de onda da luz emitida pelo objeto devido ao seu movimento em relação ao observador. Esse valor mede a expansão do universo desde o momento em que a luz foi emitida pelo objeto.
- **plate:** Identificador da placa utilizada na pesquisa espectroscópica do SDSS. Cada placa contém múltiplas fibras que coletam a luz de diferentes objetos.
- **MJD:** Data Juliana Modificada, utilizada para indicar quando um determinado conjunto de dados do SDSS foi coletado. É uma versão modificada da Data Juliana, um sistema padronizado para representar datas e horários em astronomia.
- **fiber_ID:** Identificador da fibra óptica que direcionou a luz para o plano focal durante cada observação. Cada fibra coleta a luz de um objeto diferente, permitindo que o SDSS observe vários objetos simultaneamente.

Mais informações sobre os atributos deste DataSet podem ser encontradas aqui [2].

4 Atributo alvo: Classe

4.1 Classes do Problema: Estrelas, Galáxias e Quasares

4.1.1 Estrelas

As **estrelas** são enormes esferas de gás quente, compostas principalmente de hidrogênio e hélio. Elas se formam a partir do colapso de nuvens moleculares frias e densas, onde a gravidade reúne matéria até criar uma protoestrela, que eventualmente inicia a fusão nuclear.

Durante a fase principal da vida de uma estrela, a fusão do hidrogênio em hélio mantém seu equilíbrio contra a gravidade. Estrelas de baixa massa brilham por trilhões de anos, enquanto as massivas

consomem seu combustível rapidamente e vivem apenas milhões de anos.

No final da vida, o esgotamento do hidrogênio faz a estrela se expandir. Estrelas de baixa massa se tornam gigantes vermelhas e terminam como anãs brancas, cercadas por nebulosas planetárias. Já estrelas massivas sofrem fusão até formar ferro, colapsam e explodem em supernovas, deixando para trás estrelas de nêutrons ou buracos negros. O material expelido por essas explosões enriquece futuras gerações de estrelas. [3]

4.1.2 Galáxias

As **galáxias** são estruturas gigantescas compostas por estrelas, planetas e vastas nuvens de gás e poeira, todos mantidos juntos pela gravidade. Elas variam em tamanho, desde pequenas, com algumas milhares de estrelas, até gigantes que podem conter trilhões de estrelas e medir mais de um milhão de anos-luz. A maioria das grandes galáxias abriga buracos negros supermassivos em seus centros.

Elas apresentam diferentes formas, sendo as mais comuns as espirais e elípticas, além das irregulares, que possuem aparências menos organizadas. A maioria das galáxias tem entre 10 bilhões e 13,6 bilhões de anos, com algumas sendo quase tão antigas quanto o próprio universo. A galáxia mais jovem conhecida formou-se há cerca de 500 milhões de anos.

As galáxias podem se organizar em grupos de até 100 membros, mantidos unidos pela gravidade. Estruturas maiores, chamadas aglomerados, podem conter milhares de galáxias. Esses, por sua vez, podem formar superaglomerados, que não são gravitacionalmente ligados. No conjunto, superaglomerados, vazios cósmicos e grandes estruturas formam a teia cósmica do universo. [4]

4.1.3 Quasares

Os **quasares** são os núcleos extremamente brilhantes de galáxias ativas, alimentados por buracos negros supermassivos que consomem grandes quantidades de matéria. A matéria forma um disco de acreção ao redor do buraco negro, onde a fricção aquece o gás a milhões de graus, emitindo intensa radiação. Parte do material é ejetada em jatos colimados por campos magnéticos.

No universo primitivo, fluxos de gás cósmico alimentavam os buracos negros centrais. Mais tarde, colisões entre galáxias direcionaram gás para esses buracos negros, ativando os quasares. Quasares podem brilhar até 100.000 vezes mais que a Via Láctea. Apesar disso, seu disco de acreção tem apenas algumas centenas a milhares de unidades astronômicas, comparado aos 100.000 anos-luz da Via Láctea. [5]

5 Atributos Físicos

5.1 Filtros Fotométricos

Os **filtros fotométricos** do Sloan Digital Sky Survey (SDSS) são elementos ópticos que selecionam faixas específicas do espectro eletromagnético, permitindo a observação de objetos celestes em diferentes comprimentos de onda. Eles são fundamentais para caracterizar a composição, temperatura e idade de estrelas, galáxias e outros corpos celestes.

Eles são colocados na frente dos detectores CCD da câmera do SDSS, transmitindo apenas a luz em uma faixa específica de comprimentos de onda enquanto bloqueiam outras frequências. A eficiência

de transmissão depende do material e do revestimento dos filtros, sendo calibrada para garantir medições consistentes e comparáveis entre diferentes observações. [6][7]

Cada filtro captura características físicas específicas dos objetos astronômicos:

5.1.1 Filtro u (ultravioleta próximo, ~355 nm)

Detecta estrelas jovens e quentes, pois elas emitem mais fortemente no UV. Útil para estudar a formação estelar e a presença de gás ionizado em galáxias.

5.1.2 Filtro g (verde-azulado, ~469 nm)

Sensível à emissão de estrelas intermediárias e contém a linha de absorção do cálcio (Ca II H&K). Importante para classificar estrelas segundo temperatura e idade.

5.1.3 Filtro r (vermelho, ~617 nm)

Inclui a linha de emissão do hidrogênio ionizado (Hα, 656.3 nm), um traçador da formação estelar em galáxias. Muito usado para diferenciar populações estelares jovens e antigas.

5.1.4 Filtro i (infravermelho próximo, ~748 nm)

Capta estrelas frias e regiões ricas em poeira interestelar. Essencial para estudar a estrutura de galáxias espirais e populações estelares evoluídas.

5.1.5 Filtro z (infravermelho, ~893 nm)

Permite a detecção de objetos a altos redshifts, pois a luz visível de galáxias distantes é deslocada para o infravermelho. Útil para estudar a evolução cósmica das galáxias. Esses filtros são projetados para cobrir diferentes partes do espectro visível e infravermelho próximo, otimizando a fotometria em larga escala para estudos astrofísicos.

5.2 Redshift

O **redshift** é um parâmetro fundamental na astronomia e cosmologia, que mede o deslocamento das linhas espectrais de um objeto para comprimentos de onda maiores devido à expansão do Universo. Ele é definido como

$$z = \frac{\lambda_{observada} - \lambda_{emitida}}{\lambda_{emitida}} \tag{1}$$

onde $\lambda_{emitida}$ é o comprimento de onda original da luz e $\lambda_{observada}$ é o comprimento de onda medido no espectro do objeto.

O redshift está diretamente relacionado à distância dos objetos devido à **Lei de Hubble**, que estabelece que a velocidade de recessão de uma galáxia é proporcional à sua distância

$$v = H_0 d \tag{2}$$

onde H_0 é a constante de Hubble, v a velocidade de recessão do objeto e d sua distância. v e z se relacionam pela **Fórmula do Efeito Doppler Relativístico**

$$1 + z = \sqrt{\frac{1 + v/c}{1 - v/c}} \tag{3}$$

tal que c é a velocidade da luz no vácuo. Portanto, quanto maior o redshift, mais distante e mais antiga é a luz que observamos do objeto, permitindo estudar diferentes épocas da evolução do Universo. [8]

5.3 Coordenadas Alpha e Delta

Representam as coordenadas equatoriais dos objetos observados. Eles são usados para indicar a posição dos objetos no céu, assim como as coordenadas de latitude e longitude em mapas terrestres.

A **ascenssão reta**, RA (right ascension) ou α, é a coordenada equivalente à longitude em mapas terrestres, mas no céu. Ela mede a posição do objeto ao longo da linha do equador celeste variando de 0° a 360°.

A **declinação**, Dec ou δ é a coordenada equivalente à latitude no sistema de coordenadas terrestres. Ela indica a posição do objeto ao longo do eixo perpendicular à linha do equador celeste, ou seja, a distância do objeto em relação ao equador celestial variando de +90° a -90°. [9]

6 Objetivo do Estudo

O objetivo deste projeto é a classificação de objetos celestes em estrelas (**STAR**), galáxias (**GALAXY**) ou quasares (**QSO**), utilizando diferentes modelos de classificação supervisionados e métricas de avaliação.

7 Importação de Bibliotecas e Pacotes

[1]:
```
# Versão do Python utilizada
!python --version
```

```
Python 3.10.9
```

[2]:
```
# Importa dependências
import pandas as pd
import numpy as np
import sklearn
import time
import astropy.units as u

import matplotlib.pyplot as plt
plt.rcParams['font.family'] = 'serif'
plt.rcParams['axes.labelsize'] = 12

import seaborn as sns
sns.set_style("darkgrid")
sns.set_palette(["#FF5733", "#33FF57", "#3357FF"])

import warnings
warnings.filterwarnings("ignore")
```

```
# Importa classes e funções específicas
from xgboost import XGBClassifier
from sklearn.ensemble import (
    RandomForestClassifier,
    GradientBoostingClassifier
)
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

from sklearn.preprocessing import (
    LabelEncoder,
    StandardScaler
)
from sklearn.model_selection import (
    train_test_split,
    RandomizedSearchCV,
    learning_curve,
    KFold,
    cross_validate
)
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    precision_score,
    recall_score,
    f1_score,
    roc_curve,
    auc,
    precision_recall_curve
)
from sklearn.pipeline import Pipeline
from astropy.coordinates import SkyCoord
from sklearn.DataSets import make_classification
from sklearn.feature_selection import mutual_info_classif
from sklearn.decomposition import PCA

%matplotlib inline
```

8 Análise Exploratória de Dados (EDA)

8.1 Carregamento dos Dados

[3]:
```
# Carrega o DataSet
df = pd.read_csv('star_classification.csv')
```

[4]:
```
# Visualiza as 5 primeiras linhas do DataFrame
df.head()
```

[4]:

	obj_ID	alpha	delta	u	g	r	\
0	1.237661e+18	135.689107	32.494632	23.87882	22.27530	20.39501	
1	1.237665e+18	144.826101	31.274185	24.77759	22.83188	22.58444	
2	1.237661e+18	142.188790	35.582444	25.26307	22.66389	20.60976	
3	1.237663e+18	338.741038	-0.402828	22.13682	23.77656	21.61162	
4	1.237680e+18	345.282593	21.183866	19.43718	17.58028	16.49747	

```
          i         z  run_ID  rerun_ID  cam_col  field_ID   spec_obj_ID  \
0  19.16573  18.79371    3606       301        2        79  6.543777e+18
1  21.16812  21.61427    4518       301        5       119  1.176014e+19
2  19.34857  18.94827    3606       301        2       120  5.152200e+18
3  20.50454  19.25010    4192       301        3       214  1.030107e+19
4  15.97711  15.54461    8102       301        3       137  6.891865e+18

    class  redshift  plate    MJD  fiber_ID
0  GALAXY  0.634794   5812  56354       171
1  GALAXY  0.779136  10445  58158       427
2  GALAXY  0.644195   4576  55592       299
3  GALAXY  0.932346   9149  58039       775
4  GALAXY  0.116123   6121  56187       842
```

Primeiramente, podemos notar que cerca de 8 atributos do DataSet são apenas identificadores únicos, além da data da coleta, atributos fotométricos, o valor do redshift, suas coordenadas equatoriais e a classe do objeto em questão a ser classificado. a A relevância de cada atributo deverá ser estudada, em especial os atributos de indentificação.

[5]:
```
# Analisa as dimensões do DataFrame
df.shape
```

[5]:
```
(100000, 18)
```

O DataSet possui 100000 amostras distribuídas entre 3 classes de objetos astronômicos, com cada objeto no catálogo sendo descrito por 17 atributos (features), além de 1 coluna que identifica sua respectiva categoria.

[6]:
```
# Destaca o tipo de cada atributo
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 18 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   obj_ID       100000 non-null  float64
 1   alpha        100000 non-null  float64
 2   delta        100000 non-null  float64
 3   u            100000 non-null  float64
 4   g            100000 non-null  float64
 5   r            100000 non-null  float64
 6   i            100000 non-null  float64
 7   z            100000 non-null  float64
 8   run_ID       100000 non-null  int64
 9   rerun_ID     100000 non-null  int64
 10  cam_col      100000 non-null  int64
 11  field_ID     100000 non-null  int64
 12  spec_obj_ID  100000 non-null  float64
 13  class        100000 non-null  object
 14  redshift     100000 non-null  float64
 15  plate        100000 non-null  int64
 16  MJD          100000 non-null  int64
```

```
 17  fiber_ID     100000 non-null  int64
dtypes: float64(10), int64(7), object(1)
memory usage: 13.7+ MB
```

Dentre os 17 atributos apenas o atributo **class** é do tipo object, sendo o restante do tipo int64 e float64. A codificação desse atributo será tratada posteriormente.

[7]:
```
# Determina a presença de valores ausentes
df.isnull().sum()
```

[7]:
```
obj_ID         0
alpha          0
delta          0
u              0
g              0
r              0
i              0
z              0
run_ID         0
rerun_ID       0
cam_col        0
field_ID       0
spec_obj_ID    0
class          0
redshift       0
plate          0
MJD            0
fiber_ID       0
dtype: int64
```

Não há valores ausentes neste DataSet e portanto nenhuma técnica de preenchimento ou remoção de dados será necessária.

8.2 Análise Descritiva

[8]:
```
# Descreve estatísticas resumidas
df.describe()
```

[8]:

	obj_ID	alpha	delta	u	\
count	1.000000e+05	100000.000000	100000.000000	100000.000000	
mean	1.237665e+18	177.629117	24.135305	21.980468	
std	8.438560e+12	96.502241	19.644665	31.769291	
min	1.237646e+18	0.005528	-18.785328	-9999.000000	
25%	1.237659e+18	127.518222	5.146771	20.352353	
50%	1.237663e+18	180.900700	23.645922	22.179135	
75%	1.237668e+18	233.895005	39.901550	23.687440	
max	1.237681e+18	359.999810	83.000519	32.781390	

	g	r	i	z	\
count	100000.000000	100000.000000	100000.000000	100000.000000	
mean	20.531387	19.645762	19.084854	18.668810	
std	31.750292	1.854760	1.757895	31.728152	
min	-9999.000000	9.822070	9.469903	-9999.000000	
25%	18.965230	18.135828	17.732285	17.460677	
50%	21.099835	20.125290	19.405145	19.004595	
75%	22.123767	21.044785	20.396495	19.921120	
max	31.602240	29.571860	32.141470	29.383740	

```
              run_ID  rerun_ID        cam_col       field_ID   spec_obj_ID  \
count  100000.000000  100000.0  100000.000000  100000.000000  1.000000e+05
mean     4481.366060     301.0       3.511610     186.130520  5.783882e+18
std      1964.764593       0.0       1.586912     149.011073  3.324016e+18
min       109.000000     301.0       1.000000      11.000000  2.995191e+17
25%      3187.000000     301.0       2.000000      82.000000  2.844138e+18
50%      4188.000000     301.0       4.000000     146.000000  5.614883e+18
75%      5326.000000     301.0       5.000000     241.000000  8.332144e+18
max      8162.000000     301.0       6.000000     989.000000  1.412694e+19

            redshift          plate            MJD       fiber_ID
count  100000.000000  100000.000000  100000.000000  100000.000000
mean        0.576661    5137.009660   55588.647500     449.312740
std         0.730707    2952.303351    1808.484233     272.498404
min        -0.009971     266.000000   51608.000000       1.000000
25%         0.054517    2526.000000   54234.000000     221.000000
50%         0.424173    4987.000000   55868.500000     433.000000
75%         0.704154    7400.250000   56777.000000     645.000000
max         7.011245   12547.000000   58932.000000    1000.000000
```

Com base nas estatísticas resumidas, podemos tirar alguns insights:

1. A ascensão reta **alpha** varia de 0.0055° a 359.999°, o que faz sentido, pois cobre toda a faixa possível no céu. Também, possui média próxima de 177.6°, sugerindo que os dados podem estar mais concentrados em certas regiões.
2. A declinação **delta** vai de -18.78° a 83.00°, então cobre uma grande parte do céu, mas não completamente. Também, sua média próxima de 24.14°, o que indica mais objetos no hemisfério norte celeste.
3. Em relação aos filtros **u**, **g**, **r**, **i**, **z**, algumas magnitudes possuem valores anômalos (-9999.000) e podem se tratar de *placeholders* para dados ausentes. Devemos localizá-los imediatamente e descartá-los para que não afetem as análises posteriores. Também os filtros seguem um valor médio muito próximo. Magnitudes mínimas e máximas mostram objetos brilhantes e fracos, cobrindo uma grande faixa de brilho.
4. **redshift** possui um média próxima de 0.576, o que indica que a maior parte dos objetos estão relativamente distantes, com valores variando de -0.009 a 7.011. O mínimo negativo pode ser erro de medição ou objetos com movimento peculiar e o máximo de 7.011 é altíssimo e sugere a presença de quasares extremamente distantes.
5. **MJD** indica que as observações variam dentro de uma grande faixa de datas.
6. **rerun_ID** possui um desvio padrão nulo e todas outras estátisticas iguais a 301.0. Isso evidencia que todas as 100000 amostras para este atributo possui o mesmo valor e portanto não pode conter informação relevante para este estudo.

Vamos verificar a consistência física dos dados e indentificar valores que não são fisicamente possíveis neste DataSet.

[9]:
```python
# Cria dicionário com condições físicas admitidas
condicoes = {
    "- 'alpha'": (df['alpha'] < 0) | (df['alpha'] > 360),
    "- 'delta'": (df['delta'] < -90) | (df['delta'] > 90),
    "- 'u'": df['u'] < 0,
```

```
    "- 'g'": df['g'] < 0,
    "- 'r'": df['r'] < 0,
    "- 'i'": df['i'] < 0,
    "- 'z'": df['z'] < 0
}

# Valida a presença das condições e retorna o índice correspondente
for coluna, condicao in condicoes.items():
    indices_invalidos = df.index[condicao].tolist()
    if indices_invalidos:
        print(
            f"{coluna} possui {len(indices_invalidos)} valores inválidos nos índices: "
            f"{', '.join(map(str, indices_invalidos))}"
        )
    else:
        print(f"{coluna} está dentro dos limites físicos.")
```

```
- 'alpha' está dentro dos limites físicos.
- 'delta' está dentro dos limites físicos.
- 'u' possui 1 valores inválidos nos índices: 79543
- 'g' possui 1 valores inválidos nos índices: 79543
- 'r' está dentro dos limites físicos.
- 'i' está dentro dos limites físicos.
- 'z' possui 1 valores inválidos nos índices: 79543
```

Como havíamos previsto, trata-se do *placeholder* ou erro de medição dos atributos **u**, **g** e **z** na linha 79543. Devemos descartar esta linha imediatamente.

[10]:
```
# Decarta valores do índice 7943 e reindexa os dados
df = df.drop(index=79543)
df = df.reset_index(drop=True)
```

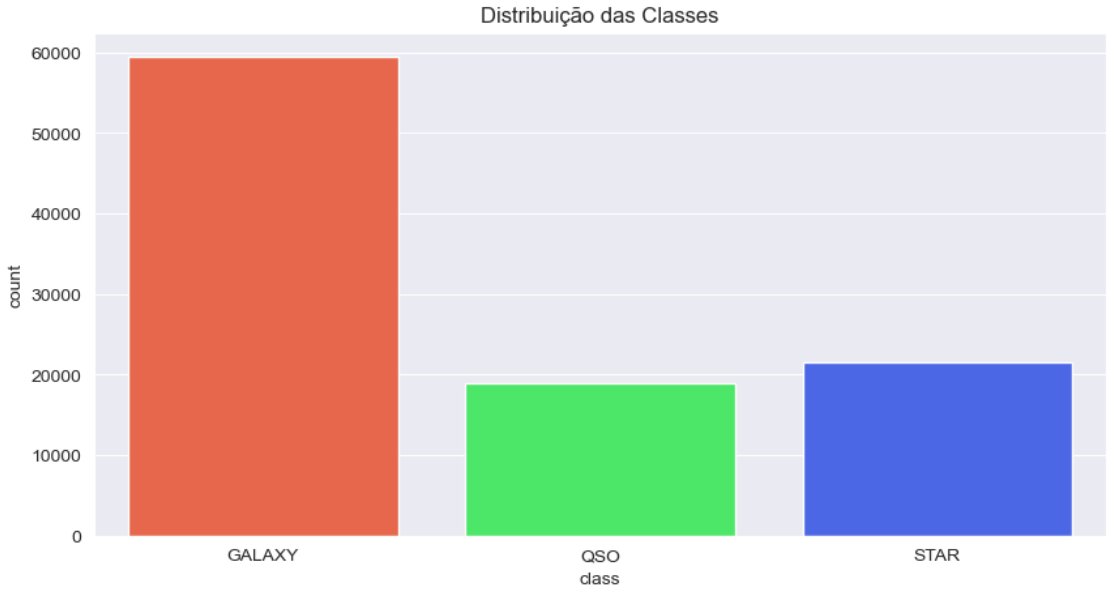
8.3 Visualização dos Dados

8.3.1 Distribuição das Classes

Vejamos a distribuição da contagem de amostras para cada classe.

[11]:
```
# Plot da distribuição da contagem de amostras por classe
plt.figure(figsize=(10,5))
sns.countplot(x=df['class'])
plt.title('Distribuição das Classes')
```

[11]:
```
Text(0.5, 1.0, 'Distribuição das Classes')
```



[12]:
```
# Contagem de amostras por classe
df['class'].value_counts()
```

[12]:
```
GALAXY    59445
STAR      21593
QSO       18961
Name: class, dtype: int64
```

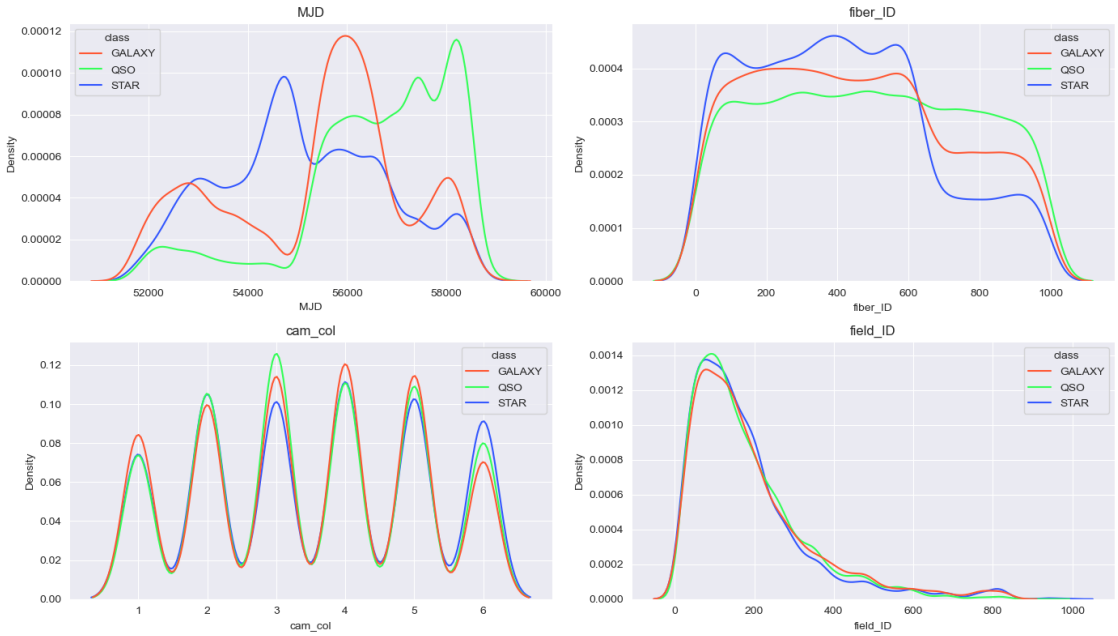
Podemos ver que o DataSet está desbalanceado, com cerca de três vezes mais galáxias do que estrelas e quasares. Para lidar com isso, iremos utilizar as técnicas de *oversampling* e *undersampling*. A técnica de *oversampling* envolve aleatoriamente selecionar exemplos da classe minoritária, substituir e adicionar esses exemplos ao DataSet de treino. Por outro lado, o *undersampling* consiste em remover aleatoriamente exemplos da classe majoritária para equilibrar a distribuição das classes. Essas abordagens ajudam a melhorar o desempenho dos modelos ao lidar com DataSets desbalanceados.

Vejamos as distribuições dos atributos em um gráfico de densidade para cada classe da variável-alvo.

[13]:
```
# Cria objeto para receber o valor de amostras que a menor classe possui
min_amostras = df['class'].value_counts().min()

# Cria um DataFrame balanceado agrupado por 'class'
df_balanceado = df.groupby('class').sample(n=min_amostras, random_state=42)

# Recebe uma lista de atributos e o número de linhas e colunas para os subplots
# e retorna subplots para estimar a função densidade de probabilidade
def kdeplot_balanceado(atributos, linha, col):
    fig, axes = plt.subplots(linha, col, figsize=(14, 8))
    axes = axes.flatten()

    for i, atributo in enumerate(atributos):
        sns.kdeplot(
```

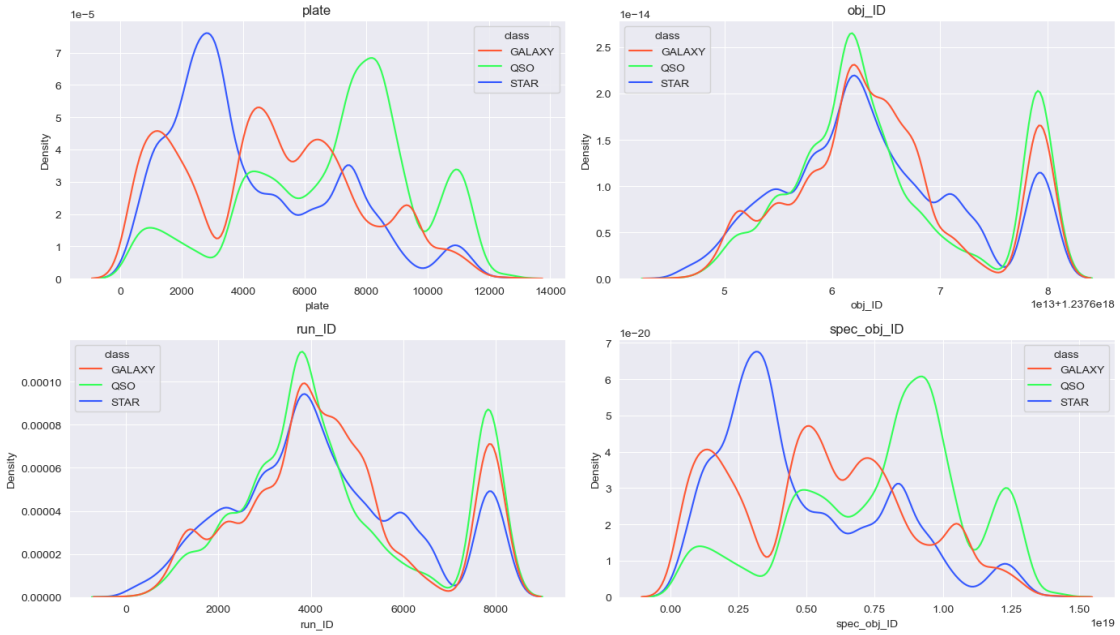
```
            data=df_balanceado,
            x=atributo,
            hue='class',
            fill=False,
            ax=axes[i]
            )
        axes[i].set_title(atributo)

    plt.tight_layout()
```

[14]:
```
# Cria lista para os atributos de identificação
atributos_1 = ['MJD', 'fiber_ID', 'cam_col', 'field_ID']

# Chama a função kdeplot_balanceado
kdeplot_balanceado(atributos_1, 2, 2)
```



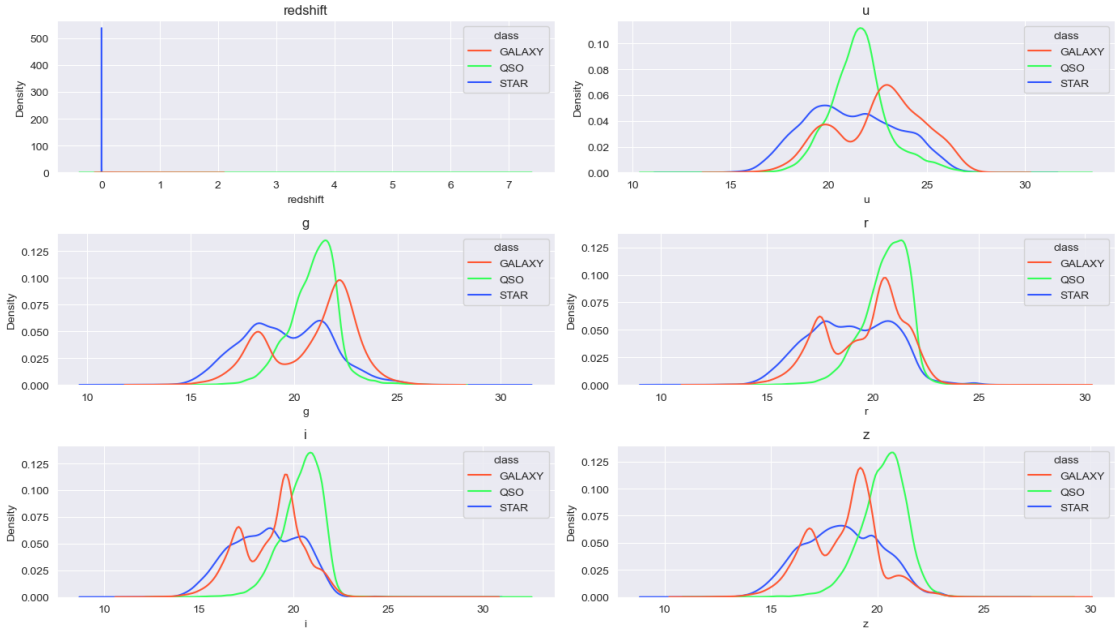
Como podemos ver, a distribuição dos atributos **cam_col** e **field_ID** não possui comportamento suficientemente distinguível por classe, com o primeiro tendo uma distribuição uniforme e periódica e o segundo uma assimétrica, tendo os picos em comum entre as três classes.

[15]:
```
# Cria outra lista para os atributos de identificação restantes
atributos_2 = ['plate', 'obj_ID', 'run_ID', 'spec_obj_ID']

# Chama a função kdeplot_balanceado
kdeplot_balanceado(atributos_2, 2, 2)
```



Todos os atributos parecem possuir uma relação com a classificação das classes, em que cada distribuição se comporta de maneira característica. Nota-se também a semelhança entre as distribuições dos atributos **obj_ID** e **run_ID**, bem como **spec_obj_ID** e **plate**. Pelo comportamento mostrado acima, parecem conter as mesmas informações um do outro. Isso será melhor evidenciado ao estudar a correlação entre eles.

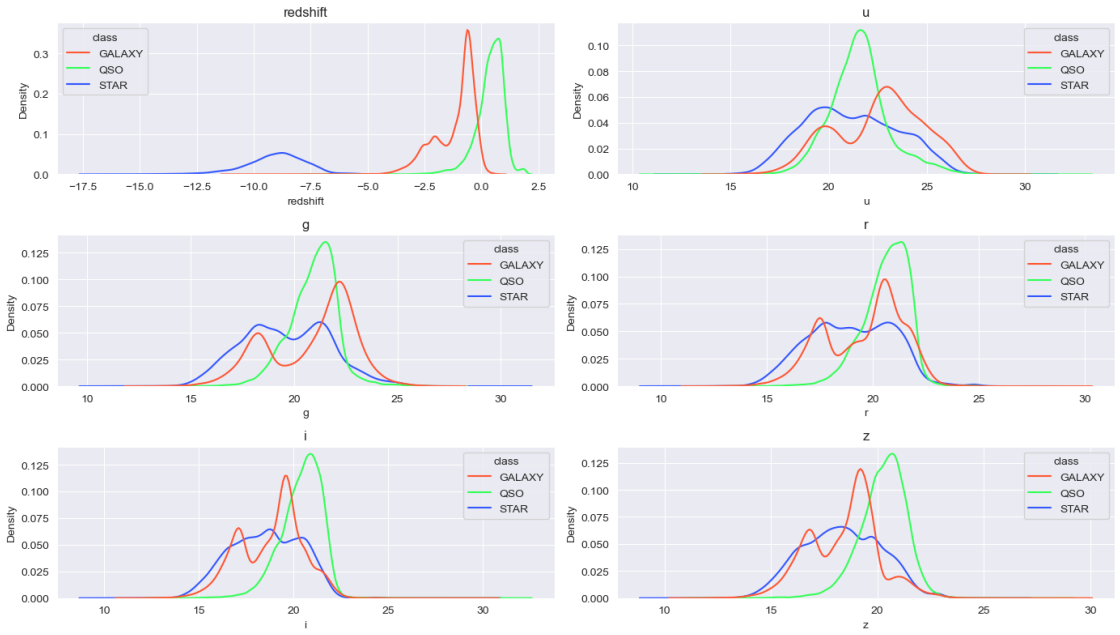
[16]:
```
# Cria lista para os atributos de natureza física
atributos_3 = ['redshift', 'u', 'g', 'r', 'i', 'z']

# Chama a função kdeplot_balanceado
kdeplot_balanceado(atributos_3, 3, 2)
```



Devido aos valores altos de **redshift** e mais baixos de **u**, **g** e **z**, iremos transformá-los para a escala logarítmica para melhor visualização das suas distribuições.

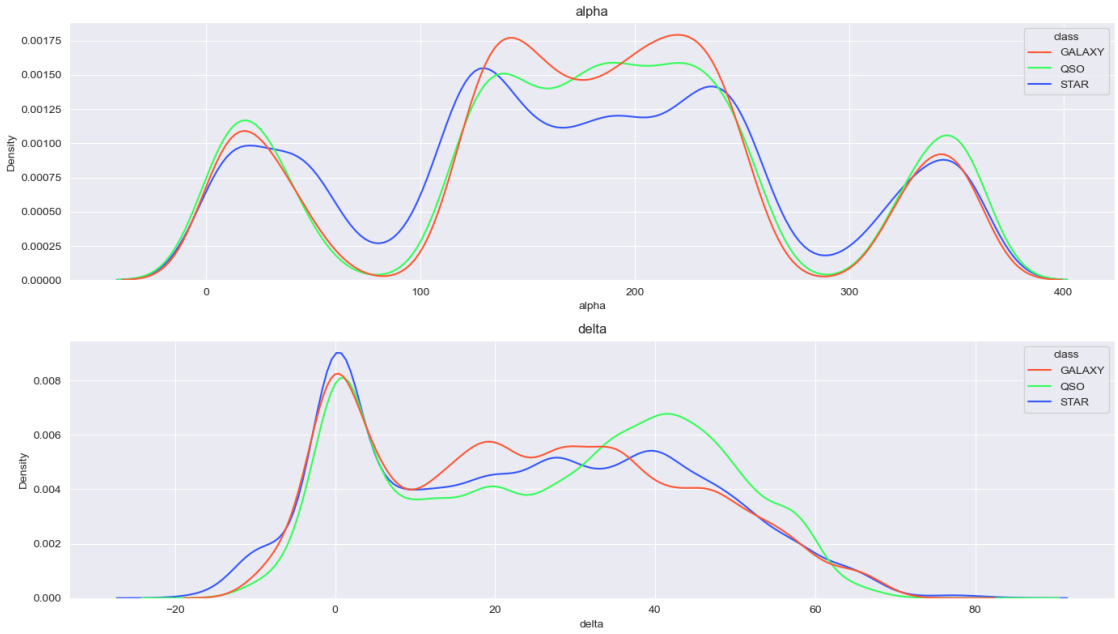
[17]:
```
# Transforma as amostras do atributo 'redshift' para escala logarítimica
df_balanceado['redshift'] = np.log(df['redshift'])
```

[18]:
```
# Chama a função kdeplot_balanceado
kdeplot_balanceado(atributos_3, 3, 2)
```



Podemos notar que todos os filtros fotométricos possuem distribuições características e distinguíveis por classe. Destaque para **resdhift** que possui seus picos de densidade em regiões diferentes para cada classe, o que mostra que se trata de um atributo com grande potencial para a classificação de objetos estelares.

[19]:
```
# Cria lista para as coordenadas equatoriais
atributos_4 = ['alpha', 'delta']

# Chama a função kdeplot_balanceado
kdeplot_balanceado(atributos_4, 2, 1)
```



As coordenadas equatoriais não parecem ter comportamentos muito distinguíveis por classe, com regiões de maior e menor densidade semelhantes entre si.

8.4 Análise de Correlação

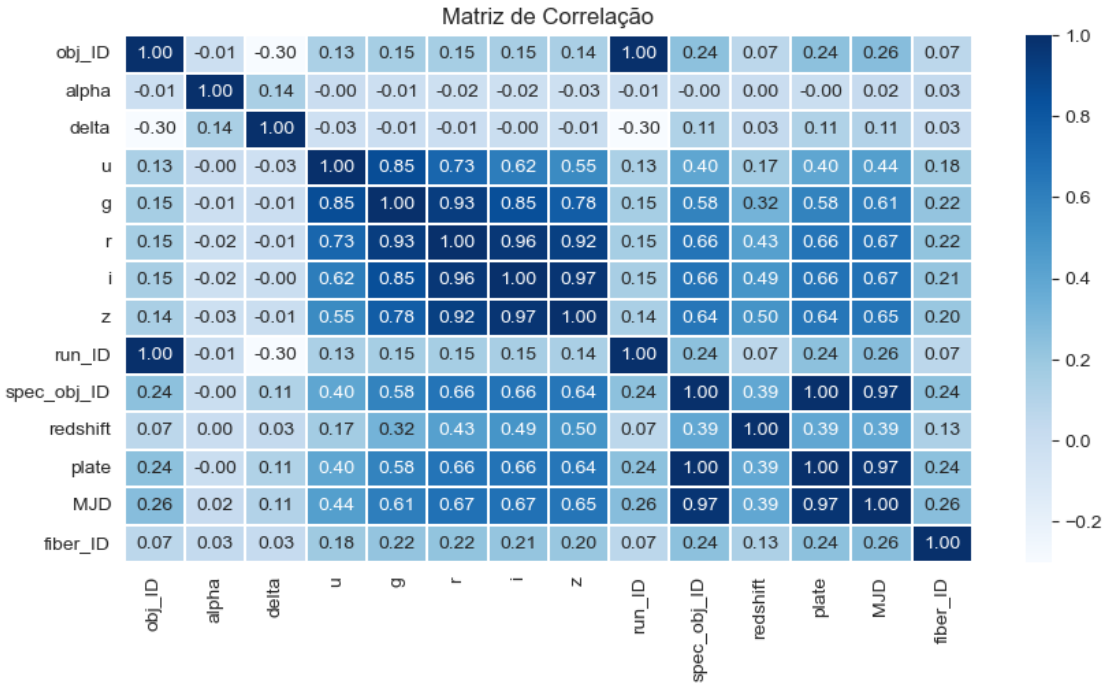
8.4.1 Correlação de Pearson

A fim de determinar a necessidade de reduzir a dimensionalidade do conjunto de atributos, iremos reproduzir um mapa de calor de **Correlação de Pearson** e entender como os atributos se correlacionam linearmente entre si.

A matriz é interpretada da seguinte forma: - 1 indica uma **máxima dependência linear positiva**. Isso significa que, à medida que uma variável aumenta, a outra também aumenta de forma linear. - -1 indica uma **máxima dependência linear negativa**. Isso significa que, à medida que uma variável aumenta, a outra diminui de forma linear. - 0 indica uma **dependência linear nula**. Isso significa que não há uma relação linear entre as duas variáveis. No entanto, é importante destacar

que isso não significa que não exista qualquer tipo de relação entre as variáveis. Elas podem ter uma correlação não linear.

[20]:
```
# Matriz correlação descartando a variável-alvo e 3 identificadores pouco relevantes
plt.figure(figsize=(10,5))
corr = df.drop(columns=['class','rerun_ID', 'cam_col', 'field_ID']).corr()
sns.heatmap(corr, annot=True, cmap='Blues', fmt=".2f", linewidths=0.1)
plt.title('Matriz de Correlação')
plt.show()
```



Como mencionado anteriormente com as curvas de distribuição de densidade, a correlação linear positiva igual a 1 entre **plate** e **spec_obj_ID**, bem como entre **obj_ID** e **run_ID**, reforça a **redundância total de informação** que eles possuem entre si.

Também, podemos notar alta correlação linear positiva entre os filtros fotométricos, com destaque para as bandas **z** e **i**, **z** e **r**, **i** e **r**, e **r** e **g**, assim como entre **plate** e **MJD**. Os atributos de coordenadas equatoriais possuem correlação baixa, próxima a 0 com todos os outros atributos, assim como o redshift que possui correlação positiva em torno de 0,30 a 0,50 com os filtros fotométricos **plate** e **MJD**.

8.4.2 Verificação de Atributos Idênticos

Vejamos se **plate** e **spec_obj_ID**, bem como **obj_ID** e **run_ID** possuem de fato valores idênticos.

[21]:
```
# Verifica se as amostras são identificamente igual
# Retorna True se verdadeiro e False se falso
df['plate'].equals(df['spec_obj_ID'])
```

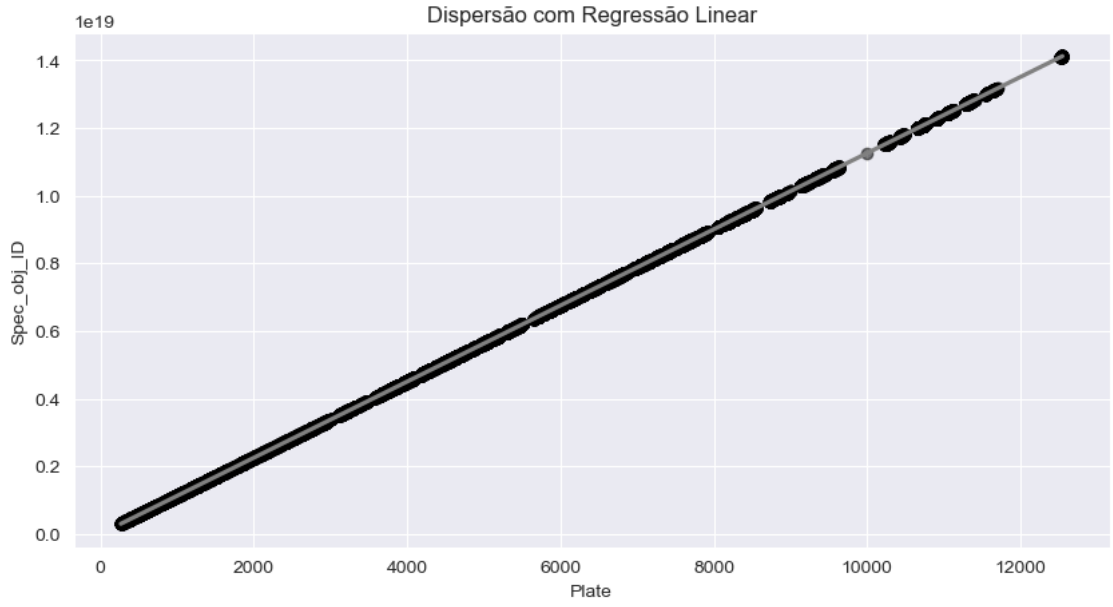
[21]:
```
False
```

[22]:
```
# Verifica se as amostras são identificamente igual
# Retorna True se verdadeiro e False se falso
df['obj_ID'].equals(df['run_ID'])
```

[22]:
```
False
```

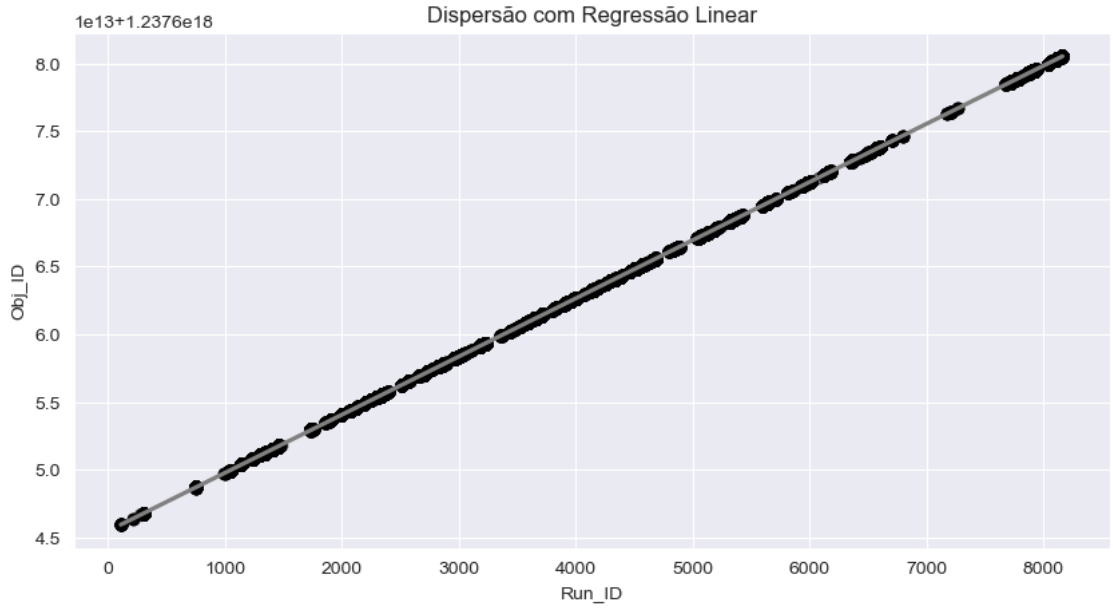
Como podemos ver, eles não possuem valores idênticos. É possível que uma variável seja uma versão transformada da outra, por exemplo, uma pode estar em logaritmo, normalizada ou padronizada. Neste caso, iremos realizar um gráfico de dispersão com regressão linear para comparar a relação entre elas. Se o ajuste linear coincidir com a dispersão dos dados, elas estão apenas escaladas e, portanto, possuem informação idêntica.

[23]:
```
# Ajuste linear com a dispersão 'plate' e 'spec_obj_ID'
plt.figure(figsize=(10,5))

sns.regplot(
    data=df,
    x="plate",
    y="spec_obj_ID",
    scatter_kws={'alpha': 0.5, 'color': 'black'},
    line_kws={'color': 'gray'}
)

plt.xlabel("Plate")
plt.ylabel("Spec_obj_ID")
plt.title("Dispersão com Regressão Linear")
plt.show()
```



Os atributos **plate** e **spec_obj_ID** possuem **informação idêntica**.

[24]:
```
# Ajuste linear com a dispersão entre 'run_ID` e `obj_ID'
plt.figure(figsize=(10,5))
sns.regplot(
    data=df,
    x="run_ID",
    y="obj_ID",
    scatter_kws={'alpha': 0.5, 'color': 'black'},
    line_kws={'color': 'gray'}
)

plt.xlabel("Run_ID")
plt.ylabel("Obj_ID")
plt.title("Dispersão com Regressão Linear")
plt.show()
```



Os atributos **obj_ID** e **run_ID** também possuem **informação idêntica**.

Vamos estudar quais as melhores decisões podem ser tomadas quanto a essa redundância, já que manter ambas as variáveis não adiciona informação útil ao modelo e apenas aumenta a dimensionalidade sem benefício.

8.4.3 Informação Mútua

Para ir além e capturar tanto relações lineares quanto não lineares, iremos utilizar o método da **Informação Mútua (Mutual Information - MI)** que consiste em quantificar a dependência entre duas variáveis ao saber quanto o valor de uma variável reduz a incerteza sobre a outra, ou seja, ela quantifica a quantidade de informação que uma variável contém sobre outra. Essa medida pode nos fazer entender mais a relação entre os atributos e a variável-alvo, por isso será útil para o nosso estudo.

[25]:
```
# Calcula a Informação Mútua entre cada atributo e a variável-alvo
mi = mutual_info_classif(
    df.drop(columns=['class']),
    df['class'],
    discrete_features=False
)

# Crian DataFrame ordenado com os resultados
mi_df = pd.DataFrame({'Feature': df.drop(columns=['class']).columns, 'MI': mi})
mi_df = mi_df.sort_values(by='MI', ascending=False)

mi_df
```

[25]:

	Feature	MI
13	redshift	0.801846
12	spec_obj_ID	0.304013
0	obj_ID	0.296990
14	plate	0.275763
15	MJD	0.192070
8	run_ID	0.145690
7	z	0.145515
4	g	0.120623
6	i	0.109458
3	u	0.100173
5	r	0.075523
16	fiber_ID	0.048798
2	delta	0.044376
1	alpha	0.040454
9	rerun_ID	0.010855
11	field_ID	0.006596
10	cam_col	0.002060

Como podemos ver, é notável a alta relação entre o atributo **redshift** e a variável-alvo. Destaque também para os atributos **spec_obj_ID**, **obj_ID**, **plate** e **MJD**. Enquanto para os atributos restantes nota-se baixa relação e relevância para a determinação da variável-alvo, destaque para os filtros fotométricos e as coordenadas equatoriais. Contudo, iremos mantê-los para análises posteriores.

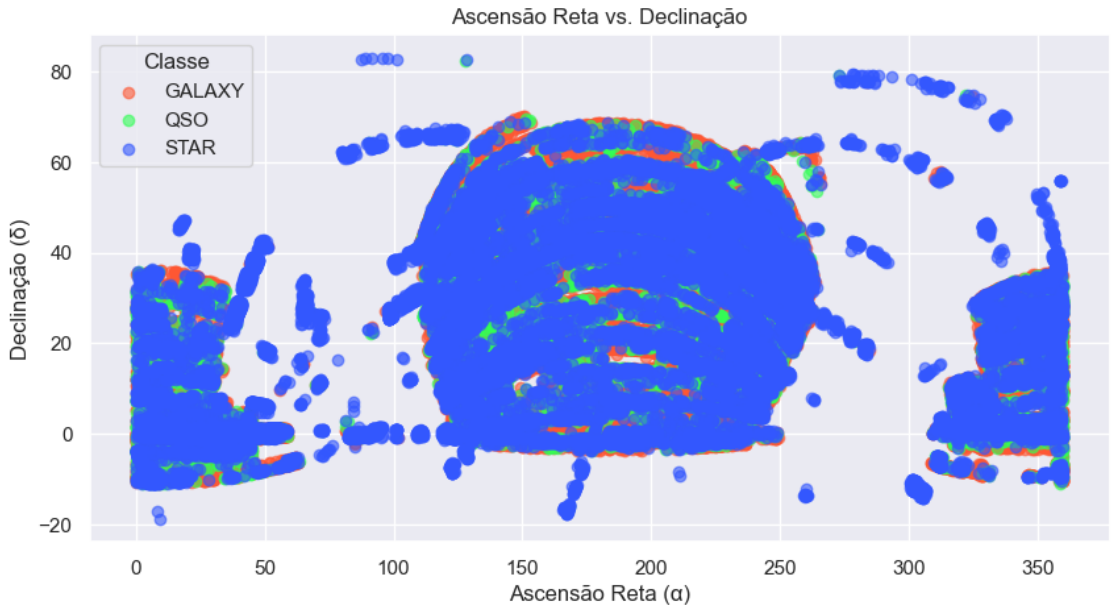
Optaremos por descartar os atributos **rerun_ID**, **field_ID** e **cam_col** devido à baixa relevância que eles demonstraram ter para o estudo.

8.4.4 Correlação Entre Coordenadas Equatoriais e a Variável Alvo

Como foi obtido um valor baixo para a correlação das coordenadas equatoriais e a variável-alvo, vamos realizar um gráfico de dispersão dos objetos por classe em função de ascensão reta e declinação e observar as regiões do céu em que as estrelas, quasares e galáxias foram observadas pelo telescópio.

[26]:
```
cores = ["#FF5733", "#33FF57", "#3357FF"]

classes = df['class'].unique()

# Configura o estilo do Seaborn
sns.set(style='darkgrid')

# Cria a figura e os eixos
plt.figure(figsize=(10, 5))
```

```
# Plot da dispersão dos objetos
for i, classe in enumerate(classes):
    subset = df[df['class'] == classe]
    plt.scatter(
        subset['alpha'],
        subset['delta'],
        color=cores[i],
        label=classe,
        alpha=0.6  # Transparência para melhor visualização
    )

# Adiciona rótulos e título
plt.xlabel('Ascensão Reta (α)')
plt.ylabel('Declinação (δ)')
plt.title('Ascensão Reta vs. Declinação')
plt.legend(title='Classe')
plt.grid(True)

# Mostra o gráfico
plt.show()
```

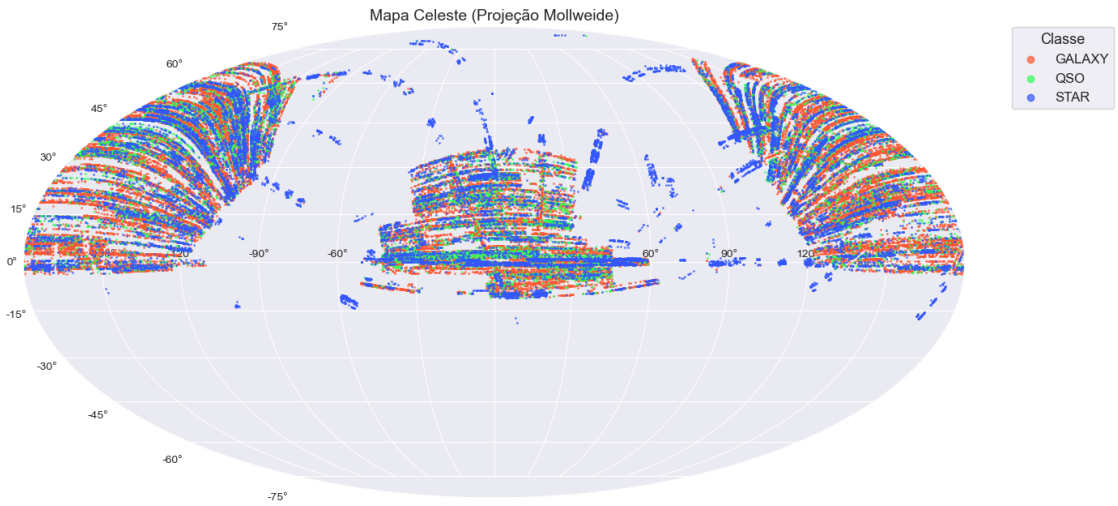


Pensando na ascensão reta como o ângulo que traça o equador terrestre, pode-se notar a presença de duas regiões com alta concentração de objetos. Também, é evidenciado que a coordenada geográfica onde determinado objeto da amostra foi catalogado tem pouca relevância para a sua classe. Isso é demonstrado pela região central com distribuição homogênea entre galáxias e quasares e uma presença maior de aglomerados de estrelas em certas regiões.

Abaixo, uma representação da distribuição dos objetos utilizando astropy para melhor visualização

[27]:
```
# Cria a figura e o subplot Mollweide
classe_unica = df['class'].unique()

fig = plt.figure(figsize=(15, 15))
ax = fig.add_subplot(111, projection='mollweide')

for classe_label in classe_unica:
    subset = df[df['class'] == classe_label]
    coords = SkyCoord(
        ra=subset['alpha'] * u.degree,
        dec=subset['delta'] * u.degree,
        frame='icrs'
    )

    ax.scatter(
        coords.ra.wrap_at(180 * u.degree).radian,
        coords.dec.radian,
        s=1,
        label=classe_label,
        alpha=0.7
    )

ax.grid(True)
ax.set_title(
    'Mapa Celeste (Projeção Mollweide)',
    fontsize=14
)
ax.legend(
    loc='upper left',
    fontsize=12,
    bbox_to_anchor=(1.05, 1),
    borderaxespad=0.,
    title="Classe",
    title_fontsize='13',
    markerscale=6
)
```

[27]:
```
<matplotlib.legend.Legend at 0x1f783282fb0>
```

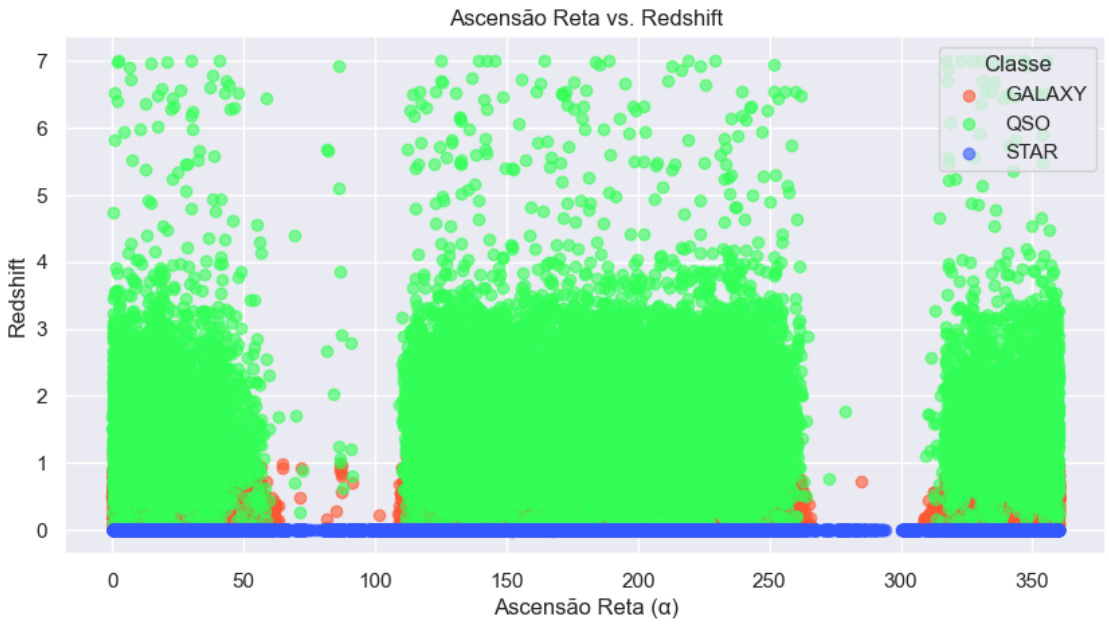


Como já observado, há uma distribuição homogênea em duas regiões principais, com certos aglomerados de estrelas espaçadas entre essas duas regiões. Ainda que pouco evidente a relação das coordenadas equatoriais com a classificação dos objetos.

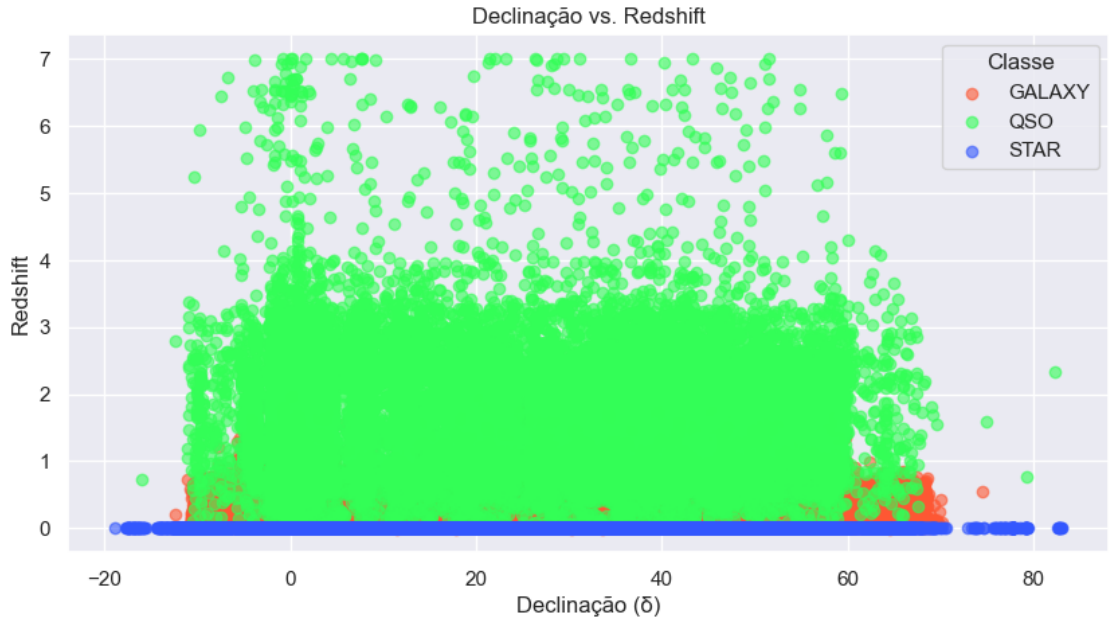
8.4.5 Correlação Entre Redshift e a Variável Alvo

Podemos visualizar sua relação também com o redshift, atributo que irá dar profundidade a esses objetos no céu.

[28]:
```
plt.figure(figsize=(10, 5))

# Plot da dispersão dos objetos
for i, classe in enumerate(classes):
    subset = df[df['class'] == classe]
    plt.scatter(
        subset['alpha'],
        subset['redshift'],
        color=cores[i],
        label=classe,
        alpha=0.6
    )

plt.xlabel('Ascensão Reta (α)')
plt.ylabel('Redshift')
plt.title('Ascensão Reta vs. Redshift')
plt.legend(title='Classe')
plt.grid(True)

plt.show()
```



[29]:
```
plt.figure(figsize=(10, 5))

# Plot da dispersão dos objetos
for i, classe in enumerate(classes):
    subset = df[df['class'] == classe]
    plt.scatter(
        subset['delta'],
        subset['redshift'],
        color=cores[i],
        label=classe,
        alpha=0.6
    )

plt.xlabel('Declinação (δ)')
plt.ylabel('Redshift')
plt.title('Declinação vs. Redshift')
plt.legend(title='Classe')
plt.grid(True)

plt.show()
```



É notável como o atributo **redshift** possui forte relevância para a classificação das classes. Nota-se uma distribuição bem determinada para cada classe em função da sua distância ao telescópio, com grande concentração de quasares em redshifts altos e intermediários, galáxias com redshifts intermediários e estrelas com redshifts próximos a 0. É possível notar que neste DataSet existe uma grande quantidade de estrelas próximas que são jovens e brilhantes, bem como quasares distantes que são antigos e mais ofuscados.

8.5 Identificação de Outliers

Vejamos se há outliers nesta amostra. Como a escala de valores dos atributos é muito diferente entre si, realizar um boxplot pode não ser muito interessante para identificar a presença de outliers. Portanto, vamos realizar a seguinte abordagem.

[30]:
```python
# Recebe uma coluna de atributos e retorna valores abaixo
# do limite inferior e acima do limite superior
# Utiliza 1° e 3° quartil (Q1 e Q3) e intervalo interquartil (IQR)
def detecta_outliers(coluna):
    Q1 = coluna.quantile(0.25)
    Q3 = coluna.quantile(0.75)
    IQR = Q3 - Q1
    lim_inferior = Q1 - 1.5 * IQR
    lim_superior = Q3 + 1.5 * IQR

    outliers_inferior = coluna[coluna < lim_inferior]
    outliers_superior = coluna[coluna > lim_superior]

    return outliers_inferior, outliers_superior
```

[31]:
```python
# Dicionário para armazenar resultados de outliers
dados_outlier = {}

# Itera pelas colunas do DataFrame (exceto 'class')
for col in df.drop(columns=['class']):
    inferior, superior = detecta_outliers(df[col])

    # Contag total de outliers
    total_outliers = len(inferior) + len(superior)

    # Conta outliers por classe para ambos limites
    classe_inferior_count = df.loc[inferior.index,
                                   'class'].value_counts()
    classe_superior_count = df.loc[superior.index,
                                   'class'].value_counts()

    # Armazena dados no dicionário
    dados_outlier[col] = [
        total_outliers,
        len(inferior),
        classe_inferior_count.get('GALAXY', 0),
        classe_inferior_count.get('STAR', 0),
        classe_inferior_count.get('QSO', 0),
        len(superior),
        classe_superior_count.get('GALAXY', 0),
        classe_superior_count.get('STAR', 0),
        classe_superior_count.get('QSO', 0)
    ]

# Cria DataFrame a partir do dicionário
df_outlier = pd.DataFrame.from_dict(
    dados_outlier, orient='index',
    columns=[
        'Outliers Total',
        'Lower Bound', 'Galáxia', 'Estrela', 'Quasar',
        'Upper Bound', 'Galáxia', 'Estrela', 'Quasar'
    ]
```

```
)

# Transpõe o DataFrame para facilitar a visualização
df_outlier = df_outlier.T

# Exibe DataFrame de outliers
df_outlier
```

[31]:

	obj_ID	alpha	delta	u	g	r	i	z	run_ID	rerun_ID	\
Outliers Total	0	0	0	55	98	132	198	319	0	0	
Lower Bound	0	0	0	44	76	118	164	256	0	0	
Galáxia	0	0	0	16	52	87	138	209	0	0	
Estrela	0	0	0	27	22	29	25	45	0	0	
Quasar	0	0	0	1	2	2	1	2	0	0	
Upper Bound	0	0	0	11	22	14	34	63	0	0	
Galáxia	0	0	0	4	15	5	13	38	0	0	
Estrela	0	0	0	5	4	7	8	10	0	0	
Quasar	0	0	0	2	3	2	13	15	0	0	

	cam_col	field_ID	spec_obj_ID	redshift	plate	MJD	fiber_ID
Outliers Total	0	5390	0	8989	0	0	0
Lower Bound	0	0	0	0	0	0	0
Galáxia	0	0	0	0	0	0	0
Estrela	0	0	0	0	0	0	0
Quasar	0	0	0	0	0	0	0
Upper Bound	0	5390	0	8989	0	0	0
Galáxia	0	3548	0	34	0	0	0
Estrela	0	1102	0	0	0	0	0
Quasar	0	740	0	8955	0	0	0

Podemos notar que o DataSet possui outliers para os atributos de filtros fotométricos, para o **redshift** e para **field_ID**.

Para os filtros fotométricos podemos perceber um valor progressivo no número de outliers das bandas mais baixas (filtros mais baixos) até as mais altas, sendo cerca de 80% a 90% correspondentes a outliers abaixo do limite inferior. Esses valores não devem ser descartados; podem-se tratar de valores físicos reais (como estrelas próximas ou galáxias brilhantes, o que pode justificar a baixa concentração desses outliers classificados como quasares, já que se tratam de objetos mais distantes).

Para **redshift** nota-se um valor expressivo de outliers, cerca de 10% de toda a amostra, em que quase a totalidade, com exceção de 34 amostras provenientes de galáxias, são quasares e estão acima do limite superior. Trata-se de um fenômeno físico real, como havíamos previsto o grande número de quasares com altos redshifts, e, portanto, também serão importantes para o problema.

Para **field_ID**, como já observamos pela relação de informação mútua, este atributo possui pouquíssima relevância para prever a variável-alvo e, portanto, devemos descartá-la.

9 Pré-processamento dos Dados

9.1 Seleção de Atributos

Queremos selecionar os principais atributos e descartar aqueles que são pouco relevantes para classificar as classes. Como havíamos analisado, os atributos **rerun_ID**, **cam_col** e **field_ID** se mostraram pouco característicos nos gráficos de distribuição por classe, bem como na análise de informação mútua.

[32]:
```
# Descarta as colunas indesejadas
df.drop(columns=['rerun_ID', 'cam_col', 'field_ID'], inplace=True)
```

9.2 Transformação dos Dados

Devido às diferentes propriedades e unidades físicas entre os atributos, iremos normalizar os dados para que todos estejam na mesma escala. Neste caso, o intervalo utilizado será [-1, 1], tal que o valor médio entre as amostras é nulo e o desvio padrão igual a 1.

[33]:
```
# Seleciona apenas colunas numéricas
df_numeric = df.drop(columns=['class'])

# Normaliza os dados
scaler = StandardScaler()
df_normalized = scaler.fit_transform(df_numeric)

# Converte de volta para DataFrame mantendo os nomes das colunas originais
df_normalized = pd.DataFrame(
    df_normalized,
    columns=df_numeric.columns,
    index=df.index
)

# Recupera a coluna da variável-alvo
df_normalized['class'] = df['class']
```

9.3 Redução de Dimensionalidade

Para tratar das variáveis redundantes, vamos utilizar **PCA (Análise de Componentes Principais)**, que consiste em transformar variáveis correlacionadas em um conjunto menor de variáveis não correlacionadas, reduzindo a dimensionalidade sem perder muita informação.

[34]:
```
# Seleciona as variáveis redundantes
variaveis_redundantes_1 = df_normalized[['spec_obj_ID', 'plate']]
variaveis_redundantes_2 = df_normalized[['obj_ID', 'run_ID']]

# Cria objeto pca para reduzir a uma única dimensão
pca = PCA(n_components=1)

# Aplica PCA nas variáveis redundantes
pca_1_resultado = pca.fit_transform(variaveis_redundantes_1)
pca_2_resultado = pca.fit_transform(variaveis_redundantes_2)

# Cria DataFrames com os resultados do PCA, preservando os índices originais
df_pca_1 = pd.DataFrame(pca_1_resultado, columns=['PCA_1'], index=df.index)
df_pca_2 = pd.DataFrame(pca_2_resultado, columns=['PCA_2'], index=df.index)

# Concatena os resultados ao DataFrame original e remove colunas redundantes
df_reduced = pd.concat(
    [df_normalized, df_pca_1, df_pca_2], axis=1
).drop(
    columns=['spec_obj_ID', 'plate', 'obj_ID', 'run_ID']
)

df_reduced.head()
```

[34]:
```
      alpha     delta         u         g         r         i         z  \
0 -0.434597  0.425517  0.798798  0.806782  0.403953  0.046001  0.013999
1 -0.339915  0.363391  1.198064  1.079967  1.584395  1.185087  1.611170
2 -0.367244  0.582702  1.413732  0.997513  0.519736  0.150012  0.101520
3  1.669522 -1.249122  0.024940  1.543642  1.059894  0.807601  0.272435
4  1.737308 -0.150255 -1.174337 -1.497665 -1.697426 -1.767888 -1.825836

   redshift       MJD  fiber_ID   class     PCA_1     PCA_2
0  0.079549  0.423198 -1.021353  GALAXY  0.323309  0.630186
1  0.277088  1.420719 -0.081893  GALAXY  2.542624 -0.026409
2  0.092415  0.001850 -0.551623  GALAXY -0.268752  0.630186
3  0.486761  1.354918  1.195186  GALAXY  1.921834  0.208336
4 -0.630273  0.330855  1.441060  GALAXY  0.471365 -2.606091
```

9.4 Codificação de Variáveis Categóricas

Precisamos realizar a divisão dos dados de treino e de teste. Antes, vamos codificar o atributo de saída para o tipo numérico.

[35]:
```
# Codifica a coluna 'class'
label_encoder = LabelEncoder()
df_reduced['class'] = label_encoder.fit_transform(df_reduced['class'])

# Verifica o mapeamento
print("Mapeamento das classes:")
for i, class_name in enumerate(label_encoder.classes_):
    print(f"{class_name} → {i}")
```

```
Mapeamento das classes:
GALAXY → 0
QSO → 1
STAR → 2
```

9.5 Divisão dos Dados

Realizando a separação entre dados de teste e de treino.

[36]:
```
# Separa as variáveis X da variável-alvo y
X = df_reduced.drop(columns=['class'])
y = df_reduced['class']

# Divide em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

9.6 Balanceamento dos Dados

9.6.1 Oversampling

Aplicando *oversampling* e *undersampling* para lidar com o desbalanceamento das classes.

[37]:
```
# Aplica oversampling nos dados de treino
sm = SMOTE(random_state=42)
X_train_oversampled, y_train_oversampled = sm.fit_resample(X_train, y_train)

# Conta o novo n° de amostras para cada classe
y_train_oversampled.value_counts()
```

[37]:
```
2    41629
0    41629
1    41629
Name: class, dtype: int64
```

9.6.2 Undersampling

[38]:
```
# Aplica undersampling nos dados de treino
rus = RandomUnderSampler(random_state=42)
X_train_undersampled, y_train_undersampled = rus.fit_resample(X_train, y_train)

# Conta o novo nº de amostras para cada classe
y_train_undersampled.value_counts()
```

[38]:
```
0    13219
1    13219
2    13219
Name: class, dtype: int64
```

10 Aprendizado de máquina

10.1 Modelos de Aprendizado de Máquina

Considerando a complexidade do problema, o tamanho do conjunto de dados e a grande presença de outliers em certos atributos, é necessária a escolha de modelos mais robustos a outliers, maior capacidade de generalização e melhor eficiência computacional.

10.1.1 Random Forest

O **Random Forest** é um modelo baseado em múltiplas árvores de decisão que usa a técnica de Bagging (Bootstrap Aggregating). Ele treina várias árvores de forma independente em subconjuntos aleatórios dos dados e combina suas previsões. Isso reduz o *overfittinge* melhora a estabilidade do modelo.

10.1.2 XGBoost

O **XGBoost (Extreme Gradient Boosting)** é uma versão otimizada do Gradient Boosting, projetada para alta eficiência e desempenho. Ele constrói árvores sequencialmente, corrigindo os erros da anterior, e usa técnicas como regularização L_1/L_2 e poda de árvores para evitar *overfitting*. Seu paralelismo e otimizações tornam-no muito eficiente em grandes DataSets.

10.1.3 LightGBM

O **LightGBM (Light Gradient Boosting Machine)** é uma alternativa ao XGBoost, mas com foco em maior velocidade e eficiência. Ele utiliza a técnica Histogram-based Learning, onde os dados são agrupados em bins antes do treinamento, acelerando a construção das árvores e reduzindo o uso de memória. É ideal para grandes volumes de dados.

10.1.4 CatBoost

O **CatBoost** é uma versão do Gradient Boosting desenvolvida pela Yandex, com melhorias no tratamento de variáveis categóricas. Ele usa técnicas como Ordered Boosting, que evita *overfitting*

ao ordenar os dados de forma específica, e permite capturar melhor interdependências entre variáveis sem necessidade de pré-processamento complexo.

10.1.5 Justificativa da Escolha dos Modelos

Estes modelos foram escolhidos devido às suas seguintes características:

- **Robustez a outliers e dados correlacionados:** Esses modelos conseguem lidar bem com os valores extremos de redshift e minimizar o impacto de variáveis correlacionadas, reduzindo o risco de overfitting.
- **Eficiência computacional e escalabilidade:** Também, possuem otimizações que aceleram o treinamento, tornando-os ideais para conjuntos de dados grandes, como o que temos.
- **Alta capacidade de generalização:** Ao corrigirem iterativamente os erros das árvores anteriores e aplicarem regularizações, esses modelos encontram um equilíbrio entre viés e variância, melhorando a precisão da classificação.
- **Versatilidade e desempenho superior:** O uso de diferentes técnicas de boosting garante alta acurácia, permitindo capturar padrões complexos e relações não lineares nos dados astronômicos.

10.2 Métricas de avaliação

As principais métricas de avaliação de aprendizado de máquina para classificação de várias classes e as que iremos utilizar são as seguintes:

10.2.1 Matriz de Confusão

A **matriz de confusão** é uma tabela que descreve o desempenho de um modelo de classificação, mostrando a distribuição das previsões feitas em comparação com as classes reais. Ela ajuda a entender quais erros o modelo está cometendo.

Para um problema de classificação multiclasse, a matriz de confusão tem a seguinte estrutura:

	Predito: Classe 1	**Predito: Classe 2**	**Predito: Classe 3**
Real: Classe 1	C_{11}	C_{12}	C_{13}
Real: Classe 2	C_{21}	C_{22}	C_{23}
Real: Classe 3	C_{31}	C_{32}	C_{33}

Onde: - C_{ij} representa o número de exemplos da **classe real** $ i $ que foram classificados como **classe predita** j. - Os valores da diagonal principal $C_{11}, C_{22}, C_{33} $ representam as classificações corretas (**acertos**). - Os valores fora da diagonal representam erros de classificação (**falsas previsões**).

Essa matriz permite analisar quais classes o modelo confunde mais e pode ser utilizada para calcular métricas como acurácia, precisão e revocação.

10.2.2 Accuracy (Acurácia)

A **acurácia** é uma das métricas de avaliação mais simples e amplamente usadas para modelos de classificação. Ela mede a proporção de previsões corretas feitas pelo modelo em relação ao total de amostras.

A acurácia é dada por:

$$\text{Accuracy} = \frac{\text{Número de previsões corretas}}{\text{Número total de amostras}} \tag{4}$$

A acurácia para o caso multiclasse pode ser expressa matematicamente como:

$$\text{Accuracy} = \frac{\sum_i C_{ii}}{\sum_i \sum_j C_{ij}} \tag{5}$$

onde C_{ii} representa os valores da **diagonal principal** da matriz de confusão, ou seja, as previsões corretas para cada classe e $\sum_i \sum_j C_{ij}$ é o número total de amostras classificadas.

10.2.3 Precision (Precisão)

A **precisão** é uma métrica que avalia a exatidão das previsões feitas pelo modelo. Ela calcula a proporção de exemplos classificados como pertencentes a uma determinada classe que realmente pertencem a essa classe.

Para problemas de **classificação multiclasse**, a precisão para cada classe i é calculada como:

$$\text{Precision}_i = \frac{C_{ii}}{\sum_j C_{ji}} \tag{6}$$

onde $\sum_j C_{ji}$ é o total de previsões que o modelo fez para a classe i.

Para obter uma métrica geral (**precisão macro**, média das precisões por classe), utilizamos:

$$\text{Macro Precision} = \frac{1}{N} \sum_i \text{Precision}_i \tag{7}$$

10.2.4 Recall (Revocação)

A **revocação** mede a capacidade do modelo de identificar corretamente todas as instâncias de uma determinada classe. Ela calcula a proporção de exemplos de uma classe que foram corretamente identificados.

Para problemas de **classificação multiclasse**, a revocação para cada classe i é calculada como:

$$\text{recall}_i = \frac{C_{ii}}{\sum_j C_{ij}} \tag{8}$$

onde $\sum_j C_{ij}$ é o total de amostras que pertencem à classe i.

Para obter uma métrica geral (**revocação macro**), utilizamos:

$$\text{Macro Recall} = \frac{1}{N} \sum_i \text{recall}_i \tag{9}$$

10.2.5 F1-Score

O **F1-score** é uma métrica que combina precisão e recall em uma única medida, fornecendo um equilíbrio entre as duas. O F1-score é especialmente útil quando há uma distribuição desigual entre as classes, ou seja, em problemas com classes desbalanceadas.

o F1-score é dado por

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{recall}}{\text{Precision} + \text{recall}}$$

Ou seja, é a média harmônica entre precisão e recall.

Para obter uma métrica geral (**F1-Score macro**), utilizamos:

$$\text{Macro F-1 Score} = \frac{1}{N} \sum_i \text{F-1 Score}_i \tag{10}$$

10.2.6 AUC (Area Under the Curve)

A **AUC (Área Sob a Curva)** é uma métrica que quantifica o desempenho de um modelo de classificação, especialmente em problemas binários. Ela mede a capacidade do modelo em distinguir entre as classes positiva e negativa em diferentes limiares de decisão.

A AUC está relacionada à curva **ROC (Receiver Operating Characteristic)**, que traça a taxa de verdadeiros positivos (TP) em relação à taxa de falsos positivos (FP). O valor da AUC varia de 0 a 1.

Para obter uma métrica geral (**AUC macro**) utilizamos a técnica **One-vc-rest**, que consiste em calcular o valor de AUC para cada classe, considerando-a como positiva em relação às outras classes, e em seguida, pode-se calcular uma AUC média para avaliar o desempenho geral em todas as classes.

10.3 Treinamento dos Modelos

10.3.1 Configuração dos Modelos

[39]:
```
# Modelos escolhidos
modelos = [
    ('RandomForest', RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)),
    ('XGBoost', XGBClassifier(n_estimators=100, random_state=42, n_jobs=-1)),
    ('LightGBM', LGBMClassifier(n_estimators=100, random_state=42, verbose=0, num_threads=-1)),
    ('CatBoost', CatBoostClassifier(n_estimators=100, random_state=42, verbose=0))
]
```

10.3.2 Validação Cruzada com *Oversampling*

Primeiro, treinando os modelos com as variáveis nas quais realizamos *oversampling*

[40]:
```
# Cria o objeto de validação cruzada
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Lista para armazenar resultados com oversampling e métricas a serem utilizadas
resultados_oversampled = []
scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']
```

```
# Aplica validação cruzada para cada modelo e coleta as métricas
for nome, modelo in modelos:
    scores_oversampled = cross_validate(
        modelo,
        X_train_oversampled,
        y_train_oversampled,
        cv=cv,
        scoring=scoring,
        return_train_score=False
    )

    resultados_oversampled.append((
        nome,
        scores_oversampled['test_accuracy'].mean(),
        scores_oversampled['test_precision_macro'].mean(),
        scores_oversampled['test_recall_macro'].mean(),
        scores_oversampled['test_f1_macro'].mean(),
        scores_oversampled['fit_time'].mean()
        ))
```

[41]:
```
# Cria um DataFrame para organizar os resultados
df_resultados_oversampled = pd.DataFrame(
    resultados_oversampled,
    columns=[
        'Modelo',
        'Acurácia Média',
        'Precisão Média',
        'Revocação Média',
        'F1 Média',
        'Tempo de Treinamento (s) / fold'
    ]
)

# Exibe os resultados ordenados pela Acurácia Média
df_resultados_oversampled.sort_values(by='Acurácia Média', ascending=False)
```

[41]:

	Modelo	Acurácia Média	Precisão Média	Revocação Média	F1 Média	\
0	RandomForest	0.982192	0.982216	0.982191	0.982178	
1	XGBoost	0.980590	0.980646	0.980587	0.980586	
2	LightGBM	0.978461	0.978567	0.978457	0.978455	
3	CatBoost	0.976939	0.977034	0.976932	0.976931	

	Tempo de Treinamento (s) / fold
0	23.341700
1	2.839957
2	1.668357
3	4.500238

Acima podemos ver a performance média de cada modelo treinado e seu correspondente tempo médio de treinamento para cada iteração da validação cruzada. Isso pode ser útil para comparar a eficiência dos modelos. Os modelos performaram bem para todas as métricas e realizaram em um tempo relativamente curto.

10.3.3 Validação Cruzada com *Undersampling*

E agora treinando os modelos com os dados nos quais foram aplicados *undersampling*.

[45]:
```
# Lista para armazenar resultados com undersampling e métricas a serem utilizadas
resultados_undersampled = []

# Aplica validação cruzada para cada modelo e coletando métricas
for nome, modelo in modelos:
    scores_undersampled = cross_validate(
        modelo,
        X_train_undersampled,
        y_train_undersampled,
        cv=cv,
        scoring=scoring,
        return_train_score=False
    )

    resultados_undersampled.append((
        nome,
        scores_undersampled['test_accuracy'].mean(),
        scores_undersampled['test_precision_macro'].mean(),
        scores_undersampled['test_recall_macro'].mean(),
        scores_undersampled['test_f1_macro'].mean(),
        scores_undersampled['fit_time'].mean()
    ))
```

[46]:
```
# Cria DataFrame para organizar os resultados
df_resultados_undersampled = pd.DataFrame(
    resultados_undersampled,
    columns=[
        'Modelo',
        'Acurácia Média',
        'Precisão Média',
        'Revocação Média',
        'F1 Média',
        'Tempo de Treinamento Médio (s) / Iteração'
    ]
)

# Exibe os resultados
df_resultados_undersampled.sort_values(by='Acurácia Média', ascending=False)
```

[46]:

	Modelo	Acurácia Média	Precisão Média	Revocação Média	F1 Média	Tempo de Treinamento Médio (s) / Iteração
2	LightGBM	0.973750	0.973896	0.973754	0.973731	0.759198
0	RandomForest	0.973195	0.973256	0.973214	0.973156	5.593778
1	XGBoost	0.973119	0.973297	0.973131	0.973111	1.319102
3	CatBoost	0.971733	0.971844	0.971741	0.971703	1.921897

10.3.4 Comparação de Desempenho Entre as Técnicas

Como podemos ver, obtivemos uma performance média melhor em todos os modelos utilizando a técnica de *oversampling*, com destaque para os modelos **Random Forest** e **XGBoost** que obtiveram 98,22% e 98,06% de acurácia média, 98,22 e 98,06, também, de F1-score médio, respectivamente, e com o tempo de execução de cada validação sendo 8 vezes menor para o **XGBoost** em relação

ao **Random Forest**. Destaque também para os modelos **LightGBM** e **CatBoost** que obtiveram performances pouco abaixo, mas um tempo de treinamento muito bom, oferecendo um baixo custo computacional.

11 Otimização de Hiperparâmetros

A **otimização de hiperparâmetros** é essencial para melhorar o desempenho do modelo e garantir que ele generalize bem para novos dados. Modelos de aprendizado de máquina possuem parâmetros internos ajustáveis que influenciam diretamente sua capacidade de aprender padrões e evitar problemas como *overfitting* (excesso de ajuste) ou *underfitting* (subajuste), além de que modelos mais bem ajustados convergem mais rapidamente e podem consumir menos recursos computacionais.

11.1 Configuração dos Hiperparâmetros

Iremos realizar esta etapa para os dois modelos com as melhores performances, o **Random Forest** e o **XGBoost**. Os dados de treino utilizados serão os quais implementamos a técnica *oversampling*, dada a sua performance superior em relação ao *undersampling*.

[47]:
```
# Hiperparâmetros para Random Forest
rf_params = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False],
    'max_features': ['sqrt', 'log2'],
    'criterion': ['gini', 'entropy']
}

# Hiperparâmetros para XGBoost
xgb_params = {
    'n_estimators': [100, 200, 500]  ,
    'learning_rate': [0.01, 0.03, 0.1, 0.2],
    'max_depth': [3, 6, 9],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0],
    'gamma': [0, 0.1, 0.2, 0.5],
    'reg_lambda': [0, 1, 2, 5],
    'reg_alpha': [0, 0.1, 0.5, 1]
}
```

11.2 Aplicação de Técnica para Otimização

A técnica de otimização de hiperparâmetros que iremos utilizar é a **RandomizedSearchCV**, ou Busca Aleatória, que é uma otimização eficiente que amostra aleatoriamente um número fixo de combinações, diferentemente do **GridSearchCV** que testa todas as combinações possíveis dentro de um grid pré-definido.

Ele é uma excelente alternativa para encontrar boas combinações, economizando tempo e recursos computacionais.

[48]:
```python
# Dicionário de modelos e parâmetros
modelos_otimizados = {
    'Random Forest': (RandomForestClassifier(random_state=42, n_jobs=-1), rf_params),
    'XGBoost': (XGBClassifier(random_state=42, n_jobs=-1), xgb_params)
}

# Dicionário para armazenar os melhores modelos
melhores_modelos = {}

# Loop para treinar e otimizar cada modelo
for nome, (modelo, parametros) in modelos_otimizados.items():
    start_time = time.time()  # Inicia o cronômetro

    # Configuração do RandomizedSearchCV
    search = RandomizedSearchCV(
        modelo,
        parametros,
        cv=3,
        n_iter=20,
        scoring='accuracy',
        n_jobs=-1,
        random_state=42
    )

    # Treinamento do modelo
    search.fit(X_train_oversampled, y_train_oversampled)

    # Armazenar o melhor modelo no dicionário
    melhores_modelos[nome] = search.best_estimator_

    # Exibir resultados parciais
    print(f"Modelo: {nome}")
    print(f"Melhores Parâmetros: {search.best_params_}")
    print(f"Tempo de Treinamento: {time.time() - start_time:.2f} segundos")
    print("-" * 127)
```

```
Modelo: Random Forest
Melhores Parâmetros: {'n_estimators': 100, 'min_samples_split': 2,
'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': None, 'criterion':
'entropy', 'bootstrap': False}
Tempo de Treinamento: 1679.02 segundos
-------------------------------------------------------------------------------
Modelo: XGBoost
Melhores Parâmetros: {'subsample': 1.0, 'reg_lambda': 1, 'reg_alpha': 0,
'n_estimators': 500, 'max_depth': 9, 'learning_rate': 0.2, 'gamma': 0,
'colsample_bytree': 0.8}
Tempo de Treinamento: 381.66 segundos
-------------------------------------------------------------------------------
```

É possível notar a diferença no tempo total de treinamento entre ambos os modelos, em que o treinamento do modelo **Random Forest** levou cerca de 4 vezes mais do que o modelo **XGBoost**, totalizando 1679,02s ou cerca de 28min no total.

11.3 Avaliação dos Modelos XGBoost e Random Forest Otimizados

Vamos testar os modelos otimizados avaliando primeiramente o valor médio das métricas **Acurácia**, **Precisão**, **Revocação** e **F1-score**. Posteriormente iremos avaliar os modelos com **Matriz de Confusão** e **AUC**.

11.4 Aplicação dos Modelos aos Dados de Teste

[49]:
```
# Lista para armazenar os resultados
resultados_avaliacao = []

# Loop para avaliar cada modelo no conjunto de teste
for nome, modelo in melhores_modelos.items():
    # Fazer previsões no conjunto de teste
    y_pred = modelo.predict(X_test)

    # Calcular métricas de avaliação
    resultados_avaliacao.append({
        'Modelo': nome,
        'Acurácia': accuracy_score(y_test, y_pred),
        'Precisão Média': precision_score(y_test, y_pred, average='macro'),
        'Revocação Média': recall_score(y_test, y_pred, average='macro'),
        'F1 Média': f1_score(y_test, y_pred, average='macro')
    })
```

11.5 Desempenho dos Modelos com Métricas Iniciais

[50]:
```
# Cria DataFrame para os resultados de avaliação
df_avaliacao = pd.DataFrame(resultados_avaliacao)
df_avaliacao = df_avaliacao.sort_values(by='Acurácia', ascending=False)

# Exibe o DataFrame com os resultados
df_avaliacao
```

[50]:

	Modelo	Acurácia	Precisão Média	Revocação Média	F1 Média
0	Random Forest	0.977567	0.974816	0.973743	0.974258
1	XGBoost	0.976600	0.974039	0.972363	0.973182

Para todas as métricas o modelo **Random Forest** teve uma performance média superior ao **XGBoost** com uma diferença menor que 0,1%, levando um tempo tempo de treinamento total aproximadamente 4 vezes maior.

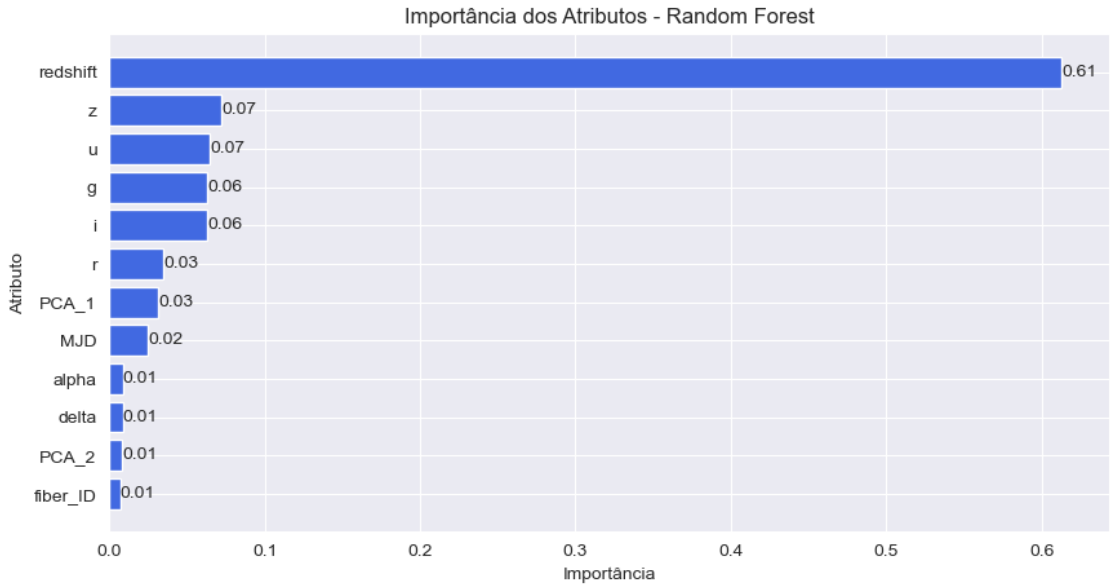
11.6 Importância dos Atributos

Podemos visualizar a importância de cada atributo para a classsifcação dos objetos.

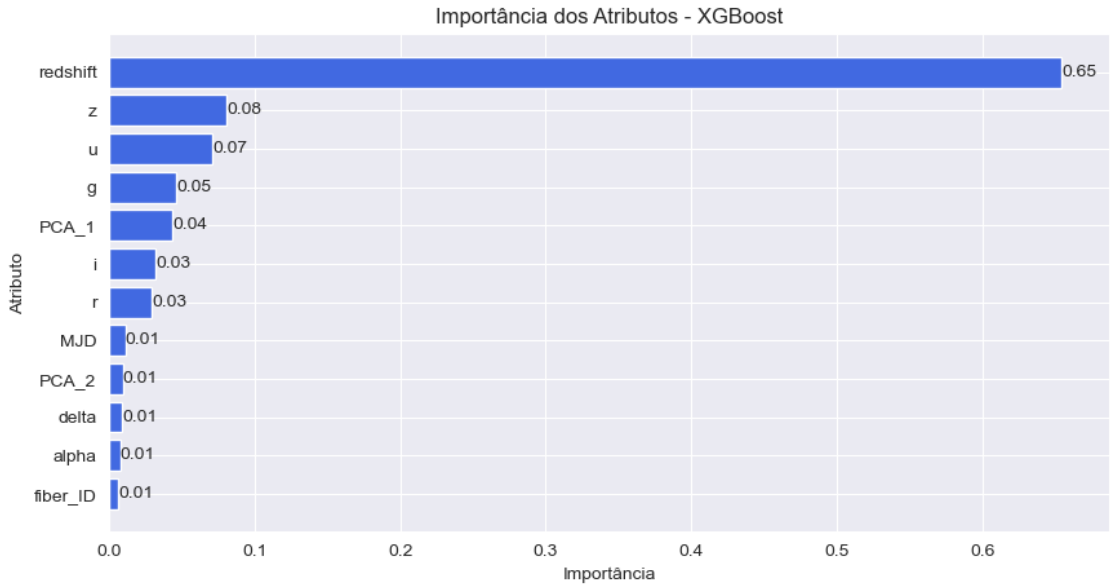
[51]:
```
# Importância dso atributos e matriz de confusão para cada modelo
for nome, modelo in melhores_modelos.items():
    y_pred = modelo.predict(X_test)
    print(f"Modelo: {nome}")

    # Importância das atributos
    feature_importance = modelo.feature_importances_
    importancia_df = pd.DataFrame({
        'Feature': X_train_oversampled.columns,
        'Importância': feature_importance
```

```
    })
    importancia_df = importancia_df.sort_values(by='Importância',
                                                ascending=False)

    plt.figure(figsize=(10, 5))
    bars = plt.barh(
        importancia_df['Feature'],
        importancia_df['Importância'],
        color='royalblue'
    )
    plt.xlabel('Importância')
    plt.ylabel('Atributo')
    plt.title(f'Importância dos Atributos - {nome}')
    plt.gca().invert_yaxis()
    for bar in bars:
        plt.text(
            bar.get_width(),
            bar.get_y() + bar.get_height()/2,
            f'{bar.get_width():.2f}', va='center')
    plt.show()
```

Modelo: Random Forest



Modelo: XGBoost

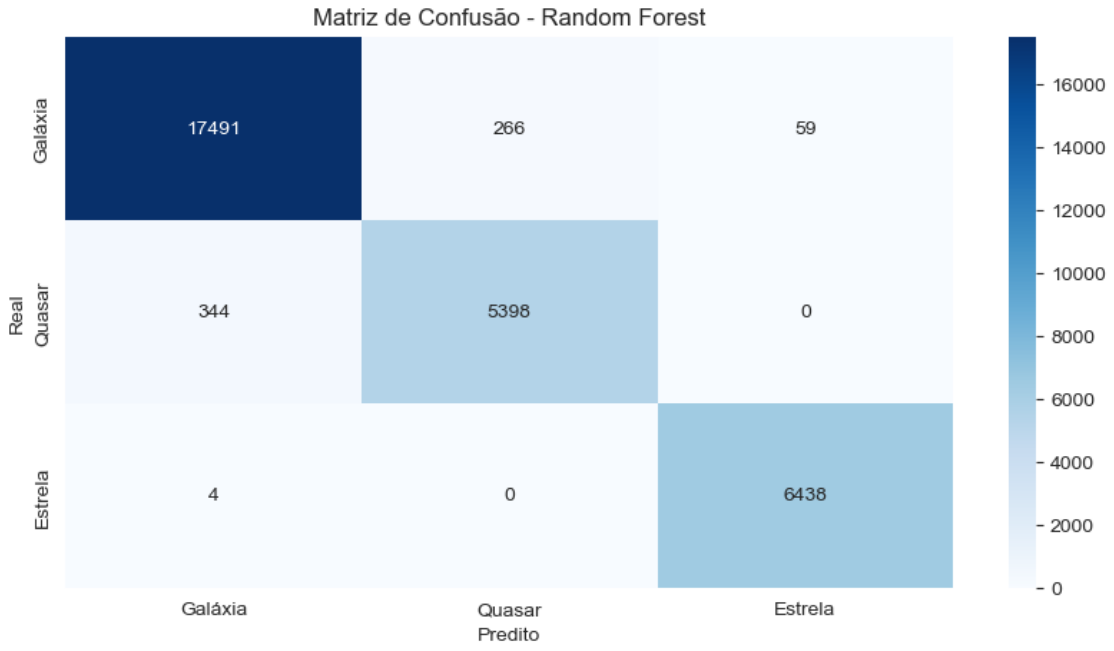


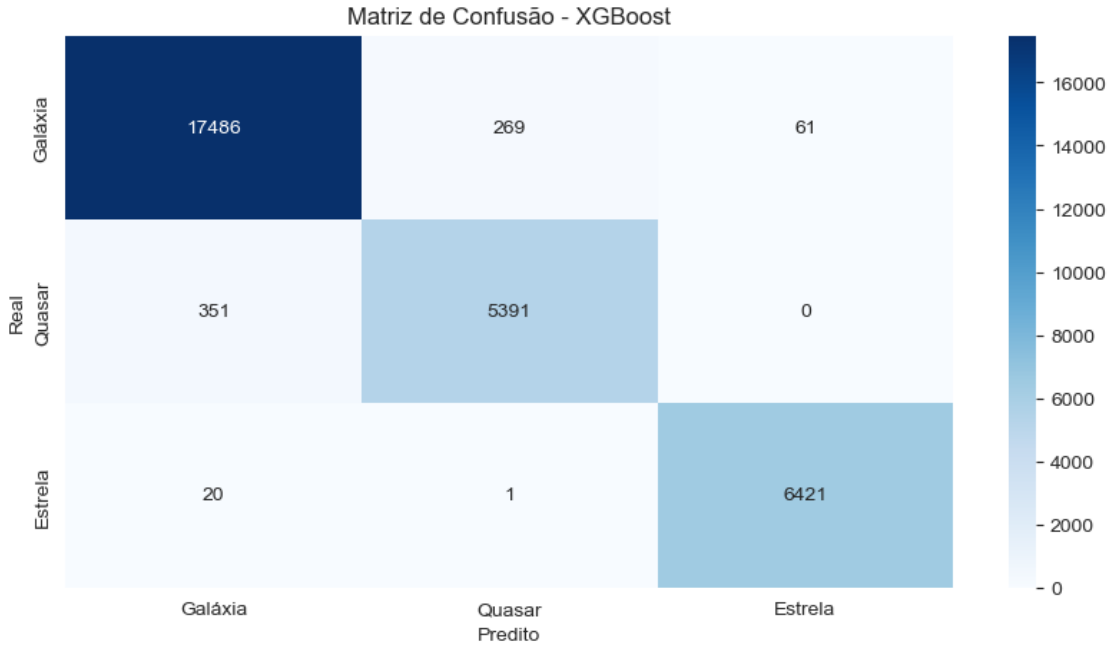
Tanto para o **XGBoost** quanto para o **Random Forest**, nota-se a grande superioridade e relevância do atributo **redshift** para a classficação estelar, com uma pontação superior a 0,60. Por outro lado, todas os outros atrbutos possuem uma pontuação próxima a 0, ou seja, possuem baixa relevância para o problema.

11.7 Desempenho dos Modelos com Matriz de Confusão

Podemos visualizar uma **matriz de confusão** e verificar o desempenho da previsão para cada classe.

[52]:
```
# Plot de matriz de confusão para cada modelo
for nome, modelo in melhores_modelos.items():
    y_pred = modelo.predict(X_test)

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 5))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=['Galáxia', 'Quasar', 'Estrela'],
        yticklabels=['Galáxia', 'Quasar', 'Estrela']
    )
    plt.xlabel('Predito')
    plt.ylabel('Real')
    plt.title(f'Matriz de Confusão - {nome}')
    plt.show()
```





O **modelo Random Forest otimizado** apresenta um bom desempenho na identificação de objetos.

Para as **galáxias**, foram registradas 17491 classificações corretas, com cerca de 1,5% das amostras classificadas como **quasares** e 0,33% classificadas como **estrelas**. No caso dos **quasares**, 5398 amostras foram corretamente classificadas, enquanto não houve classificações incorretas como classe

estrelas. Entretanto, aproximadamente 5,99% das amostras foram incorretamente classificadas como **galáxias**. Quanto às **estrelas**, 6438 classificações foram corretas, com apenas 4 amostras sendo incorretamente classificadas como **galáxias** e nenhuma amostra classificada erroneamente como **quasares**.

O **modelo XGBoost otimizado** apresenta um bom desempenho na identificação de objetos.

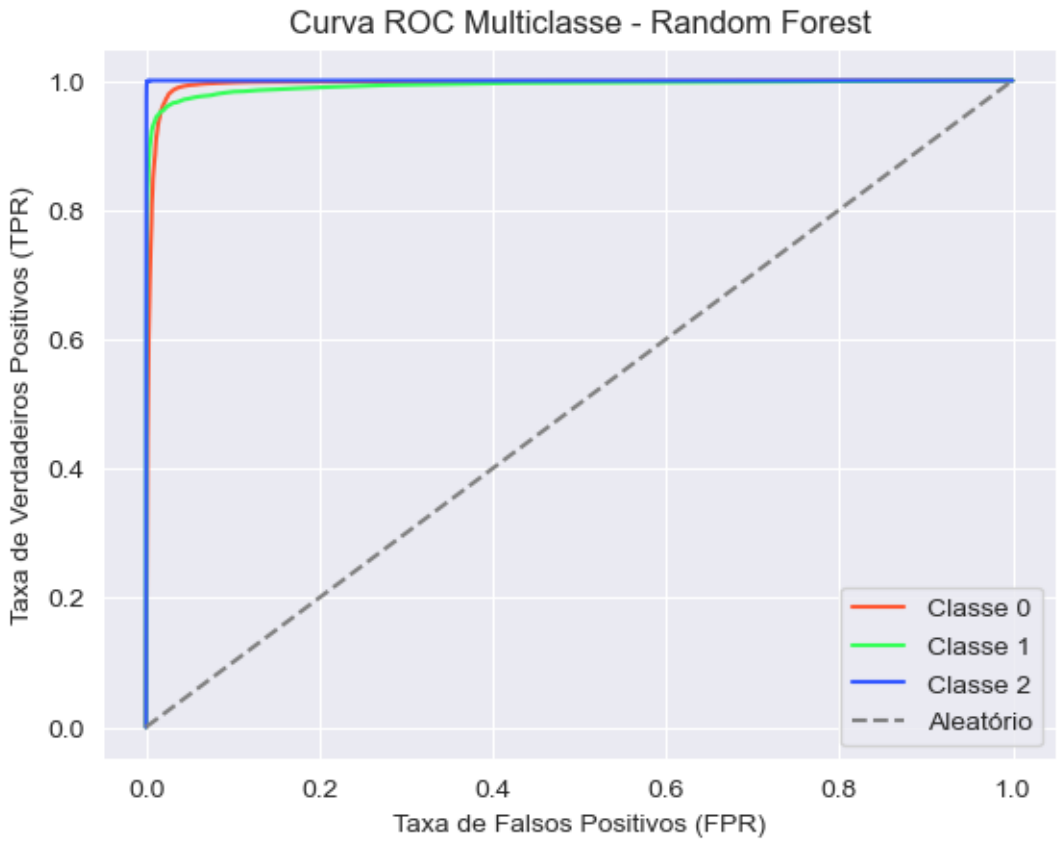
Para as **galáxias**, foram registradas 17486 classificações corretas, com cerca de 1,5% das amostras classificadas como **quasares** e 0,34% classificadas como **estrelas**. No caso dos **quasares**, 5391 amostras foram corretamente classificadas, enquanto não houve classificações incorretas como classe **estrelas**. Entretanto, aproximadamente 6,11% das amostras de **quasares** foram incorretamente classificadas como galáxias. Quanto às **estrelas**, 6.421 classificações foram corretas, com 20 amostras sendo incorretamente classificadas como **galáxias** e 1 amostra classificada erroneamente como **quasares**.

11.8 Desempenho dos Modelos com ROC e AUC

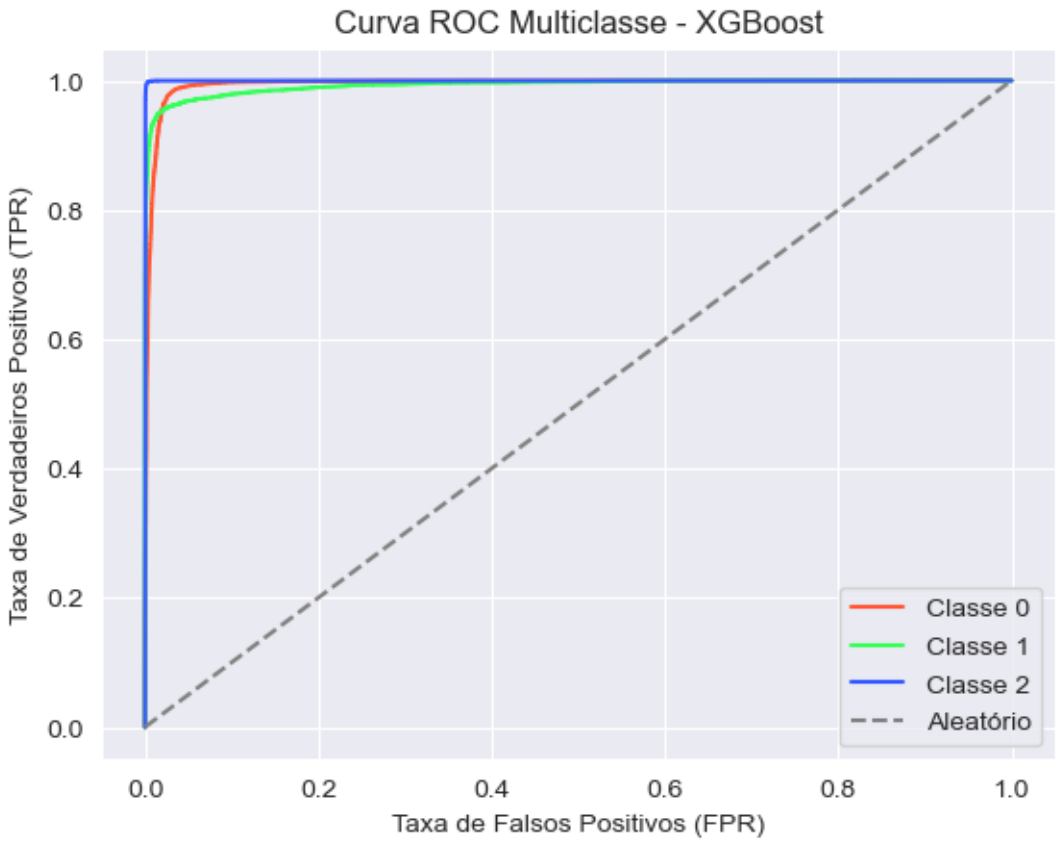
Por fim, utilizaremos a última métrica, a **AUC** com a técnica **One-vs-Rest**, possibilitando que **AUC** média seja calculada e avaliada em todas as classes.

[53]:
```
# Plot da curva ROC e AUC para cada modelo
for nome, modelo in melhores_modelos.items():

    y_pred = modelo.predict(X_test)
    y_proba = modelo.predict_proba(X_test)

    auc_mean = roc_auc_score(y_test, y_proba, multi_class='ovr', average='macro')
    print(f"AUC média (One-vs-Rest) para {nome}: {auc_mean:.4f}")

    for i in range(y_proba.shape[1]):
        fpr, tpr, _ = roc_curve(y_test == i, y_proba[:, i])
        plt.plot(fpr, tpr, label=f'Classe {i}')

    plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Aleatório')
    plt.xlabel('Taxa de Falsos Positivos (FPR)')
    plt.ylabel('Taxa de Verdadeiros Positivos (TPR)')
    plt.title(f'Curva ROC Multiclasse - {nome}')
    plt.legend()
    plt.show()
```

```
AUC média (One-vs-Rest) para Random Forest: 0.9953
```




```
AUC média (One-vs-Rest) para XGBoost: 0.9956
```



Como podemos ver, o valor médio obtido para cada modelo representa uma excelente capacidade de discriminação entre as classes, com ambos superando 0,99. Pelo gráfico, é evidenciado que ambos os modelos se saíram muito melhor do que uma classificação aleatória, não se configurando também como classificações por acaso. Os modelos aprenderam bem e se mostraram capazes de determinar as classes com alto desempenho.

12 Conclusão

O **Random Forest** otimizado apresentou o melhor desempenho geral, com uma acurácia de 97,76% nos dados de teste, superando o **XGBoost** (97,66%). No entanto, o tempo de treinamento do **Random Forest** foi aproximadamente quatro vezes maior, o que pode ser um fator limitante em cenários onde a eficiência computacional é crítica.

Também, o atributo **redshift** mostrou-se fundamental para a classificação, sendo o atributo mais relevante entre todos os atributos analisados.

A avaliação robusta do modelo, utilizando múltiplas métricas como **Acurácia**, **Precisão**, **Revocação**, **F1-score**, **Matriz de Confusão** e **AUC**, garantiu que o modelo não sofreu *overfitting* e apresentou um desempenho consistente. No entanto, observamos que o modelo ainda pode ser aprimorado, especialmente na classificação de quasares, onde houve uma pequena taxa de erro.

Para futuras melhorias, sugerimos a aplicação de técnicas como **PCA** nos filtros espectrométricos

para reduzir a dimensionalidade e viabilizar otimizações mais avançadas, como **GridSearch** ou **Optuna**. Além disso, o *deploy* do modelo em um ambiente de produção permitiria sua utilização em tempo real, facilitando a classificação de objetos celestes em observações astronômicas.