INF01017 - Aprendizado de Máquina

2022/2

Profª Mariana Recamonde Mendoza

GRUPO 10
Alice Santin Varella - 00259915 - asvarella@inf.ufrgs.br
Leonardo de Souza Augusto - 00278998 - lsaugusto@inf.ufrgs.br
Ronald de Souza Maciel - 00281987 - rsmaciel@inf.ufrgs.br

TRABALHO PRÁTICO 1 - CLASSIFICAÇÃO DE OBJETOS ESTELARES

Código-fonte disponível em:
https://colab.research.google.com/drive/1DTRM1zBxfK36PtETjHfnUnC-grljPjwy

INTRODUÇÃO

Pesquisas e estudos na área da astronomia sempre lidam com volumes massivos de dados. Com telescópios e ferramentas cada vez mais modernas, um número incontável de objetos estelares pode ser observado e detalhadamente estudado no espaço profundo. Grande parte dos estudos astronômicos trabalham em campos mais amplos do que a luz visível, usando espectroscopia para obter dados diversos acerca de radiação eletromagnética.

Espectroscopia é o estudo dos fenômenos físico-químicos que ocorrem durante a interação de radiação eletromagnética com matéria. A observação das variações nos níveis de energia de moléculas ou átomos é capaz de fornecer informações cruciais sobre a natureza do objeto observado, seja ele um corpo estelar de dimensões inimagináveis ou uma substância em um pequeno tubo de ensaio de laboratório.

É estimado que o *Sloan Digital Sky Survey* (SDSS) já tenha catalogado mais de 5 milhões de dados espectrais e 0.4 bilhões de fontes fotométricas provenientes do espaço desde 2000. Os dados coletados pelo conjunto de telescópios podem ser analisados sob diferentes critérios e, dessa forma, é possível classificar a qual categoria o objeto estelar observado pertence: estrela, galáxia ou quasar.

A partir de uma base de dados com 100 mil instâncias de observações e classificações feitas pelo SDSS, o grupo fez uso de modelos preditivos de aprendizado de máquina para criar um classificador de objetos estelares capaz de avaliar se um determinado corpo celeste é uma estrela, uma galáxia ou um quasar. Para isso, foram utilizados algoritmos de aprendizado supervisionado. A avaliação foi feita com o uso de *K-Fold Cross Validation* e de métricas de avaliação de modelos. A linguagem de programação escolhida foi Python, com auxílio de bibliotecas e ferramentas de aprendizado de máquina pré-existentes na linguagem como Panda, Sklearn, Imblearn, Numpy, MatPlotLib, entre outras. Este relatório documenta a metodologia utilizada para o estudo proposto, bem como as estratégias de pré-processamento, algoritmos de aprendizado de máquina escolhidos e análise dos resultados obtidos.

METODOLOGIA

1. DATASET

A base de dados selecionada pelo grupo consiste em 100000 instâncias de objetos estelares com 17 atributos que descrevem características espectrais e ferramental utilizado para a captação dos dados. Além dos 17 atributos, a tabela também apresenta a coluna com a classe final de cada objeto observado, onde constam três possíveis valores: galáxia (*'GALAXY'*), quasar (*'QSO'*) e estrela (*'STAR'*). Em suma, é uma tabela com 100 mil linhas e 18 colunas.

1.1. DESCRIÇÃO DOS ATRIBUTOS

Os dados dos atributos são valores numéricos do tipo *float* ou *int*. Cada uma das instâncias é completamente distinta das outras, visto que cada linha da tabela corresponde a um corpo estelar distinto no espaço e os resultados da espectrometria e da fotometria são completamente únicos para cada objeto. A tabela abaixo apresenta uma descrição breve do significado de cada um dos 17 atributos da base de dados:

RÓTULO	TIPO	DESCRIÇÃO
obj_ID	*float*	*Object Identifier*. Identificador exclusivo para cada objeto do catálogo de imagens
alpha	*float*	Ascensão reta. Unidade de coordenada celeste de um astro
delta	*float*	*Declination angle*. Outra unidade de coordenada celeste de um astro
u	*float*	Filtro ultravioleta no sistema fotométrico
g	*float*	Filtro verde no sistema fotométrico
r	*float*	Filtro vermelho no sistema fotométrico
i	*float*	Filtro próximo ao infravermelho no sistema fotométrico
z	*float*	Filtro infravermelho no sistema fotométrico
run_ID	*int*	*Run Number*. Usado para identificar a verificação específica
rereun_ID	*int*	*Rerun Number*. Usado para especificar como a imagem foi processada

cam_col	*int*	*Camera Column.* Identifica a linha de varredura dentro da execução
field_ID	*int*	*Field Number.* Identificador de campo
spec_obj_ID	*float*	ID exclusivo para objetos espectroscópicos ópticos. Duas diferentes observações para um mesmo spec_obj devem ter a mesma classe de saída
redshift	*float*	Valor de *redshift* baseado no aumento do comprimento de onda
plate	*int*	Identificador único para cada placa do SDSS
MJD	*int*	*Modified Julian Date.* Data de quando o dado foi coletado pelo SDSS
fiber_ID	*int*	Identificador da fibra que apontou para o plano focal em cada observação

2. ETAPA DE PRÉ-PROCESSAMENTO

2.1. PUREZA DOS DADOS: REMOÇÃO DE DUPLICATAS E ELEMENTOS NULOS

Antes de dar início às etapas de treinamento e avaliação, são executadas diversas análises para garantir a qualidade da base de dados que será usada para o treinamento. Com um número significativo de instâncias, faz-se pertinente a busca por atributos nulos e instâncias duplicadas, bem como a remoção de quaisquer ocorrências mencionadas.

Devido à natureza da base de dados e suas características, não foram encontrados atributos nulos, conforme ilustra a Figura 1 obtida como saída de um trecho de código:

```
 #   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
```

Figura 1 - info() listando atributos, classe de saída, tipagem e contagem de elementos não-nulos.

Ao executar o trecho de código que remove duplicatas, o número de instâncias também permaneceu inalterado.

2.2. BALANCEAMENTO DAS CLASSES DE SAÍDA

Quando analisamos as imagens processadas do espaço profundo obtidas pelo telescópio James Webb, por exemplo, é possível detectar um alto número de galáxias para cada estrela presente nas imagens. Esse tipo de discrepância é recorrente nas observações astronômicas.

Portanto, o grupo já esperava encontrar um alto desbalanceamento nas classes de saída, tendo em vista o tipo de dados sob estudo. A tabela abaixo lista valores brutos de incidência de galáxias, quasares e estrelas para todas as 100000 instâncias, enquanto a Figura 2 ilustra um histograma e um gráfico em pizza mostrando as proporções das saídas sob duas diferentes perspectivas:

# GALÁXIAS	# QUASARES	# ESTRELAS
59445	18961	21594

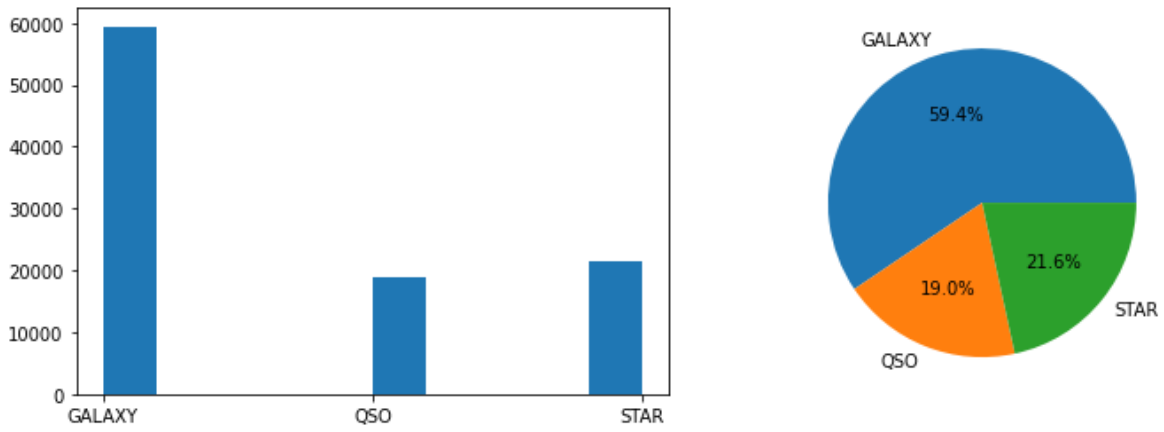

Figura 2 - Histograma e gráfico em pizza com proporções percentuais da incidência de cada uma das classes de saída do *dataset.*

A distribuição das classificações dos dados, inicialmente, está muito longe do ideal. Para o treinamento de um modelo preditivo, por exemplo, se todos os dados fossem classificados erroneamente como *galáxia,* a taxa de acerto ainda estaria próxima dos 60%.

Uma base de dados ideal para uso em aprendizado de máquina possui equilíbrio no número de incidências das classes de saída. Tendo em vista este panorama, o grupo passou a discutir soluções para o balanceamento das informações do *dataset*.

2.3. USO DE UNDERSAMPLE E SMOTE

O grupo optou por combinar *Undersample* e SMOTE para o balanceamento dos dados como uma estratégia moderada para maximizar o número de instâncias disponíveis para o treinamento de modelos. *Oversampling* puro é altamente ineficiente e pode ser muito mais prejudicial do que benéfico para a base de dados.

A classificação estrela (‘*STAR*’) foi usada como uma espécie de “âncora” no balanceamento dos dados; isso significa que o número de ocorrências da classe galáxia (‘*GALAXY’*) foi reduzida até igualar-se ao número de estrelas com uso de *undersampling*. Da mesma forma, a classe dos quasares (‘*QSO’*), que representa a menor incidência no conjunto dos dados, foi expandida até igualar-se ao número de estrelas com uso de SMOTE. Tal processo foi definido pelo grupo como uma boa estratégia devido à baixa diferença no número de quasares e estrelas, fazendo com que poucas instâncias sintéticas fossem adicionadas ao conjunto. Por consequência, essa estratégia também reduziu o número de galáxias que precisaram ser excluídas.

Ao final do balanceamento, cada valor possível para a classe de saída apresenta 21594 incidências no conjunto de dados, totalizando 64782 instâncias no *dataset* atualizado.

A Figura 3 ilustra os mesmos gráficos da Figura 2, agora atualizados após o balanceamento da classe de saída:

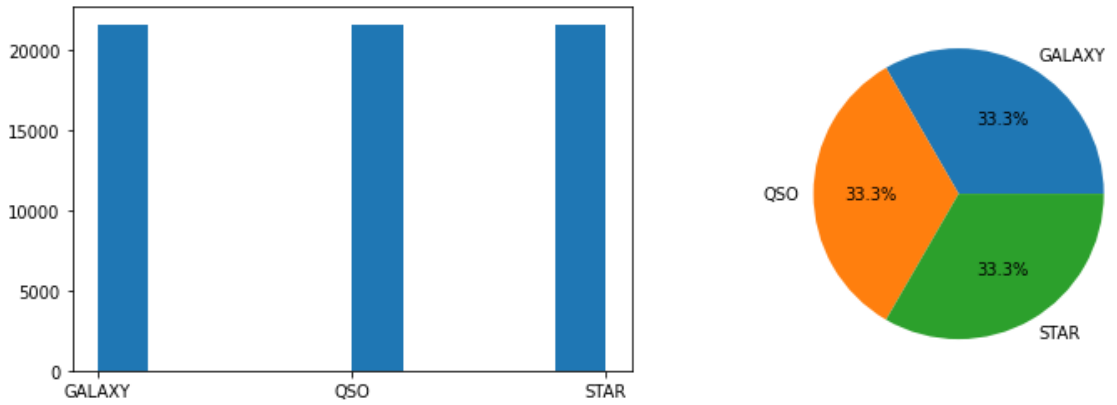

Figura 3: gráficos após o uso de *undersampling* e SMOTE combinados para balanceamento dos dados.

2.4. SELEÇÃO DE ATRIBUTOS

Para a seleção de atributos, foi utilizado o algoritmo de seleção das árvores de decisão. Após a obtenção dos *scores,* foi realizada a remoção manual dos atributos selecionados pela árvore como não relevantes. Grande parte dos atributos removidos eram identificadores de ferramental, portanto, a retirada de muitos deles era previsível, visto que identificadores têm influência mínima nos resultados do estudo. A Figura 4 ilustra a lista de atributos após a execução do algoritmo:

```
Data columns (total 5 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   u         100000 non-null  float64
 1   g         100000 non-null  float64
 2   z         100000 non-null  float64
 3   class     100000 non-null  object
 4   redshift  100000 non-null  float64
dtypes: float64(4), object(1)
```

Figura 4 - Lista de atributos que foram mantidos após o processamento realizado pela árvore de decisão.

3. MATRIZ DE CONFUSÃO E K-FOLD ESTRATIFICADO

Finalizada a etapa de pré-processamento, inicia-se a etapa na qual os modelos são treinados e avaliados a partir de métricas de desempenho e estratégias de avaliação. Os modelos escolhidos pelo grupo para execução de treinamentos foram Árvores de Decisão, Redes Neurais e Floresta Aleatória. Para a avaliação de desempenho dos modelos treinados com o *dataset,* foram criadas implementações próprias de Matriz de Confusão e validação cruzada com *K-Fold* estratificado.

3.1. IMPLEMENTAÇÃO DA MATRIZ DE CONFUSÃO PRÓPRIA

A base de dados escolhida pelo grupo configura um conjunto multiclasse. A fim de fazer uma implementação mais simples, o grupo concordou em trabalhar com os cálculos de macro médias e acurácia global conforme visto em aula. Com isso, para cada modelo treinado, a implementação gera uma matriz de confusão para cada classe individualmente e calcula índices de verdadeiros positivos (VP), verdadeiros negativos (VN), falsos positivos (FP) e falsos negativos (FN) para aquela classe. A Figura 5 ilustra um exemplo do formato da matriz de confusão feita para a classe '*GALAXY*'. As colunas representam a classe predita e as linhas representam a classe verdadeira:

```
Matriz de confusão para  GALAXY

                    predição:
             verdadeiro   falso
|----------|----------|----------|
|verdadeiro|  4142       155
|falso     |  264        8396
```

Figura 5: Matriz de confusão para a classe 'GALAXY'.

3.1.1. FUNÇÕES AUXILIARES DA MATRIZ DE CONFUSÃO

A implementação foi dividida em pequenas funções que executam partes diferentes das etapas da obtenção da matriz de confusão e das métricas de desempenho. Abaixo, a lista com os nomes das funções e uma descrição breve de sua função:

```
getClassIndex()   # A matriz predicted[][] resultante da predição possui um índice para cada classe. A partir do rótulo (‘GALAXY’, ‘STAR’, ‘QSO’), retorna o índice da matriz onde está o resultado para a classe solicitada.

confusionMatrixValues() # Calcula VP, VN, FP e FN para a classe enviada como parâmetro. Retorna esses valores.

printConfusionMatrix() # Utiliza os valores calculados por confusionMatrixValues() para imprimir uma matriz de confusão da classe parâmetro.

calculateMetrics() # Recebe os valores de confusionMatrixValues() para calcular dados de Recall, Precisão e F1 da classe parâmetro. Retorna esses valores.

confusionMatrix() # Função a ser chamada para os treinamentos. Utiliza todas as funções sequencialmente para obter a matriz de confusão de cada classe, imprimir, calcular métricas, atualizar acumuladores para os Stats globais e enviá-los para o cálculo do desempenho final.

globalAndMacroStats() # Recebe acumuladores com valores brutos, calcula acurácia global e macro-médias e imprime resultados.
```

A função mais importante do processo é a `confusionMatrixValues()`, tendo em vista que é dentro dela que a comparação entre classe verdadeira e classe predita acontece, bem como a contagem de verdadeiros positivos e outros valores cruciais para todos os outros processos da matriz de confusão. Abaixo, uma descrição em pseudocódigo a respeito de seu funcionamento:

```
confusionMatrixValues()
        # captura indice da classe na lista predicted
       indice_classe = getClassIndex()
       # Armazena um "gabarito" de quantos positivos e negativos existem
       total_positivos = positiveCount(conjunto_teste[classe][1])
        total_negativos = negativeCount(conjunto_teste[classe][0])
        # Lista de resultados esperados:
        esperado = convertToList(conjunto_teste[classe])
        # Contadores para vp e vn:
        vp, vn = 0

        para i até range(len(esperado):
              Se predicted é VERDADEIRO e esperado é VERDADEIRO:
                    vp ++
              Se predicted é FALSO e esperado é FALSO:
                    vn ++
       # utiliza valores encontrados para calcular fp e fn:
       fp = total_negativos - vn
        fn = total_positivos - vp
```

3.2. IMPLEMENTAÇÃO DO K-FOLD ESTRATIFICADO

Ao final do balanceamento dos dados, o grupo dividiu o *dataset* resultante em 3 diferentes partições, sendo uma para cada classe de saída. Cada uma das três partições foi divida em *folds* de igual tamanho. Na sequência, os *folds* foram concatenados, "reunindo" a classe de saída e resultando em seções que mantêm a proporção de 33% de elementos para cada classe de saída, configurando estratificação.

Abaixo, o pseudocódigo da implementação do *K-fold* estratificado:

```
attributes: Matriz dos dados somente com os atributos
classes: Matriz dos dados somente com as classes
folds: Numero de folds
classesIds: Vetor com os nomes das classes (GALAXY, QSO e STAR, nesse caso)
custom_kfold (attributes, classes, folds, classesIds)
    concatenar as matrizes de atributos e classes
    inicializar o array dos resultados
    inicializar o dicionário das classes que servirá para separar os dados em
para cada classe

        popula o dicionário com todas as instâncias da classe 'classId'
        # com isso temos uma partição contendo todas as instâncias daquela
classe, para cada classe, ou seja, 3 partições

    para cada classe
        divide a partição com as instâncias da classe pelo número de folds
recebido
        para cada fold
            concatena a partição (dividida) com as instâncias da classe,
ordenadamente, em cada fold resultante
            # com isso vamos preenchendo os folds por classe, ou seja, inserimos
todas as instâncias da classe atual em cada fold, e assim por diante, para
mantermos a proporção de classes em cada fold

    retorna os folds
```

3.3. FUNÇÕES AUXILIARES

```
kfold_validate (kfold, classifier) # Recebe os folds criado pela função
custom_kfold e o classificador. Treina os dados dos folds no classificador, depois
ele prediz os dados de teste, e então é chamado a função confusionMatrix com esses
dados preditos, para fazer o cálculo das métricas do classificador

split_dataset (dataset, classes) # Divide o dataset/matriz de instâncias em duas
matrizes de atributos e classes, respectivamente

concat_panda_elements (array) # Concatena os dados do tipo DataFrame (matriz de
dados) da biblioteca pandas em um DataFrame só

calculate_macro_mean (metrics) # Faz a macro média das métricas obtidas por cada
fold do classificador
```

4. TREINAMENTO DOS MODELOS E MÉTRICAS DE DESEMPENHO

O grupo deu continuidade ao estudo fazendo o treinamento dos três modelos escolhidos para a proposta do trabalho. Ao final do treinamento, foram feitas as predições com os conjuntos de teste e as avaliações com *K-fold* estratificado e matriz de confusão, fazendo uso das implementações próprias. A profundidade máxima escolhida para a árvore de decisão e para a floresta aleatória foi igual a 5.

A tabela abaixo demonstra os valores numéricos de desempenho médio e desvio padrão obtidos para cada um dos modelos treinados:

	Árvore de Decisão	**Floresta Aleatória**	**Rede Neural**
Desempenho médio	0.9648	0.9676	0.9637
Desvio Padrão	0.000065	0.000309	0.000288
Tempo de execução	2.2 segundos	53.89 segundos	181.26 segundos

A Figura 6 ilustra o gráfico *boxplot* gerado a partir do desempenho dos modelos:

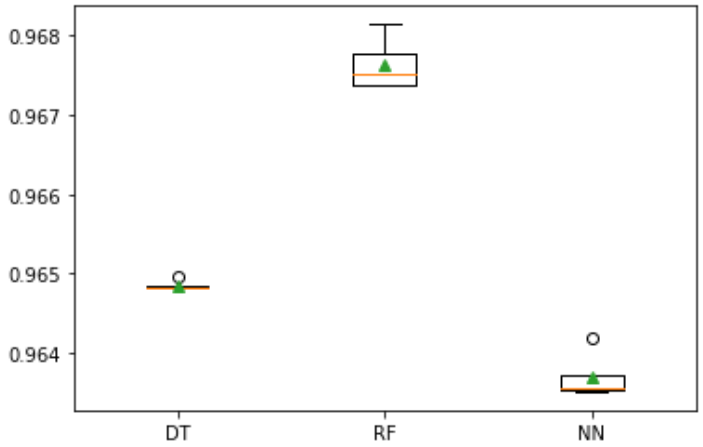

Figura 6 - Gráfico em *boxplot* que ilustra a performance dos modelos escolhidos.

5. CONSIDERAÇÕES FINAIS

Pelos valores médios analisados na tabela de desempenho da seção 4, os três modelos aparentam ter desempenho tecnicamente equivalente. O gráfico apresentado na Figura 6 ajuda a concluir que, em certo momento, os melhores valores de performance foram encontrados nas predições realizadas pela Floresta Aleatória. Isso se dá porque a Floresta Aleatória é um modelo de aprendizado *ensemble*, que faz uso do conceito de sabedoria das multidões para fazer uma análise descentralizada e agregar diferentes resultados.

Os valores encontrados para o desvio padrão e o tempo de execução do modelo de árvores de decisão foram significativamente mais baixos do que o dos outros dois modelos. Isso provavelmente se deve aos dados do conjunto terem sido pré-processados de modo a

ficarem balanceados, visto que a árvore de decisão é altamente sensível a dados desbalanceados.

A etapa de pré-processamento é de extrema importância para o desempenho dos modelos. Garantir que o treinamento será feito fazendo uso de um *dataset* organizado e com boas proporções entre as diferentes classes de saída é crucial para bons resultados. Tendo em vista que a base de dados original contava com 100 mil instâncias, foi possível reduzir incidências excessivas de uma classe de saída sem comprometer criticamente o número final de instâncias no conjunto de dados. A decisão de utilizar SMOTE para igualar o número de quasares ao número de estrelas, em conjunto com o *undersampling* massivo aplicado nas galáxias foi considerada, pelo grupo, uma boa tomada de decisão.

A etapa mais desafiadora foi a implementação das funções de avaliação de desempenho. Ao fazer uso de diversas bibliotecas para manipulação de dados e treinamento de modelos de aprendizado supervisionado, frequentemente as estruturas de dados não eram compatíveis entre si ou dependiam de métodos específicos de suas bibliotecas para serem interpretadas. Por consequência, integrar as implementações à avaliação de desempenho dos modelos treinados também demandou um tempo significativo de trabalho.

REFERÊNCIAS

ABOUT the SDSS. New Mexico: Sloan Digital Sky Survey, 2010. Disponível em: https://skyserver.sdss.org/dr7/en/sdss/. Acesso em: 1 mar. 2023.

THE SDSS Telescopes. New Mexico: Sloan Digital Sky Survey, 2003. Disponível em: https://skyserver.sdss.org/dr1/en/sdss/telescope/telescope.asp. Acesso em: 1 mar. 2023.

ESPECTROSCOPIA. [S. l.]: Wikipédia, a Enciclopédia livre, [20--]. Disponível em: https://pt.wikipedia.org/wiki/Espectroscopia. Acesso em: 1 mar. 2023.

STELLAR Classification. [S. l.]: Wikipedia, the free Encyclopedia, [20--]. Disponível em: https://en.wikipedia.org/wiki/Stellar_classification. Acesso em: 1 mar. 2023.

STELLAR Classification: Astronomy. [S. l.]: Encyclopaedia Britannica, 2022. Disponível em: https://www.britannica.com/science/stellar-classification. Acesso em: 1 mar. 2023.

RIGHT Ascension and Declination. [S. l.]: American Institute of Physics, [20--]. Disponível em: https://www.aip.org/sites/default/files/history/teaching-guides/follow-drinking-gourd/Follow%20the%20Drinking%20Gourd_RA%20Dec%20Handout.pdf. Acesso em: 3 mar. 2023.

UNDERSAMPLING and Oversampling imbalanced data. [S. l.]: Kaggle, 2018. Disponível em: https://www.kaggle.com/code/residentmario/undersampling-and-oversampling-imbalanced-data. Acesso em: 1 mar. 2023.

RANDOM Oversampling and Undersampling for imbalanced classification. [S. l.]: Machine Learning Mastery, 15 jan. 2020. Disponível em: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/. Acesso em: 1 mar. 2023.