Apis Melífera Honeybee na Penísula Ibérica

Afonso de Paula
Universidade da Beira Interior (UBI)
Departamento de Informática
Covilhã, Portugal
afonso.paula@ubi.pt

I. Introdução

As abelhas melíferas (*Apis Melífera Honeybee*) desempenham um papel importante no equilíbrio do ecossistema e na produtividade agrícola, onde são as principais responsáveis na polinização de inúmeras culturas. Na Península Ibérica, a distribuição geográfica e o comportamento desses insetos são afetados a partir de uma série de fatores, onde as variáveis climáticas possuem um papel crucial nesse cenário.

Este trabalho tem como objetivo explorar e comparar algoritmos de aprendizagem supervisionada para investigar a relação existente entre os locais de ocorrência da *Apis Melífera Honeybee* na Península Ibérica com um conjunto de variáveis climáticas que possam determinar a sua presença.

Para atingir os objetivos propostos, serão explorados e comparados diversos algoritmos de aprendizagem supervisionada. Entre eles, destacam-se: *Decision Trees*, *Random Forest*, *Nearest Neightbour*, *Support Vectors Machine* (SVM), *Multi-Layer Perceptron* (MLP) e *Logistic Regression*.

Relativamente à estrutura deste relatório, inicialmente, é apresentado um capítulo detalhado com informações relevantes sobre o *dataset* utilizado, destacando como este se encontra organizado e estruturado. Em seguida, a divisão do conjunto de treino e de teste é realizada. O trabalho prossegue com a abordagem de técnicas *oversampling* e *undersampling*, com a iniciativa de resolver o problema do desequilíbrio dos dados, onde é escolhida a técnica mais adequada para a situação específica do presente *dataset*. Posteriormente, ocorre o treino dos algoritmos selecionados e, consequentemente, a sua avaliação para mensurar o desempenho e a capacidade preditiva de cada um. A análise é enriquecida com um capítulo dedicado às curvas ROC e AUC, onde é fornecido uma avaliação mais aprofundada do desempenho dos modelos. Por fim, este artigo é concluído com uma secção de conclusão, em que os principais resultados obtidos são discutidos e as conclusões relacionadas com a análise são apresentadas.

II. Informações do Dataset

Para a elaboração deste projeto, o docente da unidade curricular de *Aprendizagem Automática* disponibilizou um conjunto de dados fundamental para o treino dos algoritmos que serão abordados ao longo deste trabalho. O *dataset* fornecido pode ser representado como uma matriz, sendo este composto por sete colunas, cada uma contento informações pertinentes para a previsão dos dados. Entre estas colunas, quatro delas correspondem a variáveis climáticas, que serão utilizadas como preditores para o processo de aprendizagem dos algoritmos. As variáveis climáticas disponíveis são as seguintes:

1) ***tann*** - Temperatura Média Anual;
2) ***rfseas*** - Sazonalidade da Chuva;
3) ***mxtwm*** - Temperatura Máxima do Mês mais Quente;
4) ***mntcm*** - Temperatura Mínima do Mês mais Frio.

Além das variáveis climáticas mencionadas, o *dataset* também contém informações sobre a **longitude** e **latitude** da região estudada. Estes dados espaciais são cruciais para vincular as observações climáticas a uma localização geográfica específica, permitindo a análise de como a distribuição das espécies pode estar relacionada com os fatores geográficos.

Por fim, a última coluna dos dados contém a resposta alvo, representada pela variável `y`. Neste projeto, a variável `y` representa os dados de ocorrência da espécie em questão. Estes dados serão essenciais para a predição, pois servem como base para treinar os modelos e avaliar as suas capacidades de fazerem previsões precisas sobre a ocorrência da espécie *Apis Melífera Honeybee* em diferentes regiões geográficas.

mntcm	mxtwm	rfseas	tann	latitude	longitude	y
6.0	23.0	36.0	14.0	17.0	51.0	1.0
6.0	23.0	35.0	13.0	17.0	52.0	0.0
5.0	22.0	35.0	13.0	17.0	53.0	0.0
5.0	23.0	35.0	13.0	17.0	54.0	0.0
6.0	23.0	34.0	14.0	17.0	56.0	0.0
...	...	...	...	...	...	...
7.0	27.0	76.0	17.0	108.0	77.0	0.0
7.0	27.0	76.0	16.0	108.0	78.0	0.0
5.0	28.0	76.0	15.0	108.0	79.0	0.0
8.0	27.0	76.0	17.0	108.0	80.0	0.0
7.0	27.0	78.0	16.0	109.0	79.0	0.0

Fig. 1. *Dataset*.

Após a primeira visualização do *dataset* (Figura 1), tornou-se necessário inserir os nomes das colunas, de modo a organizar melhor as informações e a possibilitar uma interpretação melhor dos dados. O conjunto de dados é composto por **9435** linhas, o que indica que foram registadas **9435** regiões diferentes na Península Ibérica onde a presença (ou pseudo-

presença) do ser vivo em questão foi observada. Neste contexto, o valor **1** indica que a espécie foi observada no local, enquanto o valor **0** indica que a espécie não foi observada.

A visualização dos dados é uma etapa importante para a análise de qualquer conjunto de dados, pois permite uma compreensão inicial das características e distribuição das informações presentes. A partir da biblioteca *Matplotlib*, gráficos e outras formas de representação visual foram criados, fornecendo uma visão mais clara das variações nas ocorrências da espécie em relação às variáveis climáticas e à localização geográfica.

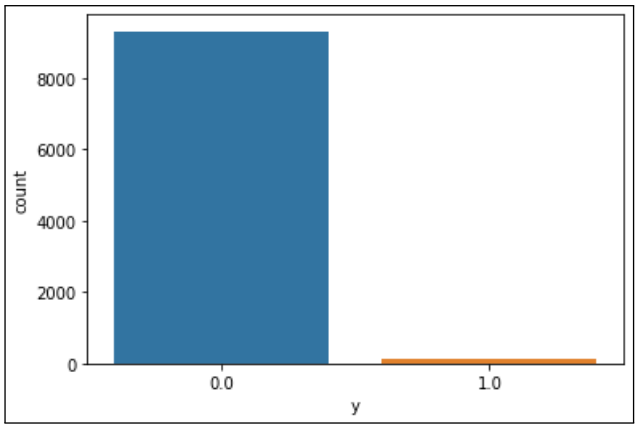

Fig. 2. Dados de ocorrência da *Apis Melífera Honeybee* na Península Ibérica.

Com a Figura 2, a análise inicial do *dataset* revelou uma disparidade significativa entre a quantidade de ocorrências de zeros e uns. Este desequilíbrio de classes sugere que se está a lidar com dados desequilibrados, onde a classe minoritária possui uma ocorrência rara em comparação com a classe majoritária, que representa a ausência ou a pseudo-presença da espécie.

A presença de dados desequilibrados é uma ocorrência comum em diversos cenários da vida real, incluindo a análise de dados biológicos e ecológicos, como é o caso deste estudo sobre a ocorrência das abelhas melíferas na Península Ibéria. No entanto, este desequilíbrio pode apresentar alguns desafios durante a análise e a modelagem dos dados. Uma das principais consequências, em relação ao desequilíbrio de classes, é que os algoritmos de aprendizagem tendem a ser enviesados em direção à classe majoritária, resultando num baixo desempenho relativamente à classificação da classe minoritária. Isto acontece porque o modelo pode priorizar a *accurary* geral, acabando por não conseguir capturar corretamente os padrões presentes na classe minoritária devido à falta de dados para treino.

Para mitigar este efeito, o próximo passo importante é abordar como dividiremos os dados em conjuntos de treino e teste. A correta divisão dos dados é fundamental para que os modelos sejam treinados a partir de um conjunto representativo e que sejam capazes de generalizar de forma adequada, permitindo obter previsões precisas e confiáveis sobre a ocorrência da *Apis Melífera Honeybee*. Nesta próxima etapa, irá ser aplicado a divisão dos dados, levando em consideração o desequilíbrio das classes, de modo a assegurar um processo de treino robusto e uma avaliação precisa do desempenho dos modelos.

III. Divisão em conjuntos de teste e treino

Inicialmente, como podemos visualizar no Figura 3, atribuímos as características ambientais cruciais para a sobrevivência da espécie (que serão consideradas como o conjunto `X`) e os resultados obtidos das observações do ser vivo (a classe que pretendemos determinar – variável dependente, representada pelo conjunto `Y`). Em seguida, aplicaremos a normalização para o intervalo unitário, utilizando o método `MinMaxScaler` da biblioteca *sklearn.preprocessing*. Esta etapa é fundamental para tratar este tipo de dados, pois possibilita reduzir a redundância dos dados, aumentar a integridade e o desempenho geral das análises.

Após a normalização, procedemos à divisão dos dados em dois conjuntos: o conjunto de treino, correspondente a **70%** do *dataset*, e o conjunto de teste, com **30%** dos dados. Para realizar esta divisão, utilizamos a classe `train_test_split` da biblioteca *sklearn.model_selection*, disponibilizada pela linguagem Python. Esta biblioteca oferece recursos valiosos para facilitar a tarefa de separação de dados numa maneira aleatória e representativa.

A divisão dos dados em conjuntos de treino e teste é uma prática essencial para problemas de aprendizagem automática, uma vez que permite avaliar o desempenho do modelo em dados não vistos previamente. O conjunto de treino é utilizado para treinar o modelo, enquanto o conjunto de teste é reservado para avaliar a capacidade de generalização do modelo para novas observações. Estes passos são essenciais para a construção de qualquer modelo preciso e eficiente, proporcionando uma base sólida para a análise das abelhas na Península Ibérica. Com os dados normalizados e a divisão adequada em conjuntos de treino e teste, estamos preparados para prosseguir com a abordagem de técnicas para tratar os dados desequilibrados.

```
X = bees.values[:,0:4]
Y = bees.values[:,6]

scaler = MinMaxScaler()
Xscaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(Xscaled, Y, test_size=0.3,
                                                    random_state=0)
print('Número de dados para treino: ', X_train.shape[0])
print('Número de dados para teste:  ', X_test.shape[0])S

Número de dados para treino:  6604
Número de dados para teste:   2831
```

Fig. 3. Normalização e divisão dos dados.

IV. Oversampling VS Undersampling

Neste caso, observamos a partir da Figura 2 que temos **126** locais onde a espécie foi observada e **9309** locais onde não foi. É importante ressaltar que a ausência das observações em

determinados locais não indica que espécie não possa viver e prosperar nessas regiões, mas sim não foram registadas observações até ao momento. Condições climáticas semelhantes podem existir em locais onde a espécie foi observada. Portanto, o desenvolvimento de um modelo sem considerar esta desproporcionalidade dos dados pode levar ao paradoxo da *accuracy*, onde o modelo pode obter uma alta precisão, mas não consegue fazer distinção entre a classe minoritária e majoritária, resultando em resultados falsos.

Para lidar com este desequilíbrio entre classes, utilizaremos as técnicas *undersampling* e *oversampling* da biblioteca `imblearn` do Python. Nesta secção, abordaremos quatro métodos para ajustar classes desequilibradas em modelos de aprendizagem. Estes métodos serão comparados com os resultados do modelo *Random Forest*, utilizado como *baseline* – um modelo sem *oversampling* ou *undersampling* permite dar uma ideia base para comparar o desempenho do modelo.

A. Oversampling

Esta técnica consiste em aumentar o número de exemplos da classe minoritária, que, no nosso caso, corresponde à classe `1`. Existem vários métodos que realizam esse aumento de formas diferentes. Neste trabalho, foram utilizados dois métodos específicos:

- ***Random Oversampling***: Neste método, exemplos da classe minoritária são selecionados aleatoriamente, com substituição, e adicionados ao conjunto de dados de treino;
- **SMOTE *Oversampling***: Este método gera novos exemplos da classe minoritária através da interpolação entre os pontos mais próximos (k vizinhos mais próximos).

Com a aplicação destes métodos, é possível equilibrar a proporção entre as classes. No conjunto de treino, o número de amostras da classe minoritária (`y=1`) é igualado ao número de amostras da classe majoritária (`y=0`), totalizando **6515** amostras para cada classe.

B. Undersampling

Enquanto o *oversampling* aumenta a quantidade de exemplos da classe minoritária, o *undersampling* faz o oposto: reduz a quantidade de exemplos da classe majoritária. Neste trabalho, foram abordados dois métodos *undersampling*:

- ***Random Undersampling***: Neste método, um conjunto de dados é removido aleatoriamente da classe majoritária, de forma a equilibrar as proporções entre classes;
- ***NearMiss***: Este algoritmo possui três versões diferentes de clusterização. Para este projeto, foi utilizada a versão 3 do algoritmo, que é dividido em dois passos. Primeiro, para cada amostra negativa, os seus M vizinhos mais próximos são mantidos. Em seguida, as amostras positivas selecionadas são aquelas cuja distância média aos N vizinhos mais próximos é maior.

Com a aplicação destes métodos, é possível equilibrar a proporção entre as classes no conjunto de treino. No resultado final, o número de amostras da classe majoritária é igualado ao número de amostras da classe minoritária, totalizando **89** amostras para cada classe.

C. Resultados Obtidos

Após aplicar os métodos *oversampling* e *undersampling* para tratar o desequilíbrio de classes no conjunto de treino, registamos os seguintes resultados:

Atributos	**Baseline**	**RO**	**SMOTE**	**RU**	**NM**
nºdados (*1*)	89	6515	6515	89	89
nºdados (*0*)	6515	6515	6515	89	89
precision (*1*)	0.07	0.03	0.09	0.03	0.01
recall (*1*)	0.03	0.08	0.08	0.62	0.62
accuracy (%)	98.23	95.23	97.73	68.88	36.98

TABLE I
RESULTADOS OBTIDOS – TÉCNICAS OVERSAMPLING/UNDERSAMPLING.

Com base nos resultados alcançados (Tabela I), podemos concluir que a técnica SMOTE apresenta uma melhor performance em relação aos demais algoritmos, com uma *accuracy* de **97,73%**. Portanto, os dados gerados pelo SMOTE serão utilizados para treinar os modelos apresentados na próxima secção. É importante salientar que estes resultados são extremamente úteis para determinar a técnica mais adequada para tratar o desequilíbrio de classes no conjunto de dados e são fundamentais para garantir a precisão e a eficácia dos modelos de aprendizagem.

V. TREINO E AVALIAÇÃO DE ALGORITMOS

A aprendizagem automática, também conhecida como *machine learning*, tem como objetivo criar programas que melhorem a performance de máquinas em tarefas específicas, baseando-se no conceito de ”experiência”, adquirida a partir dos dados. Esta abordagem tem se mostrado extremamente útil em diversos domínios, sendo especialmente importante na resolução de problemas de mineração de dados, onde grandes volumes de informações podem conter implicitamente dados valiosos a serem descobertos automaticamente.

Neste trabalho, é apresentado uma breve introdução aos diferentes algoritmos de aprendizagem automática utilizados. Cada um deles representa uma abordagem única para analisar os dados e aprender com eles, permitindo realizar determinadas tarefas, como classificação, regressão, agrupamento e muito mais. Ao explorar estes métodos, o objetivo passa por identificar a abordagem mais adequada para a previsão da ocorrência da espécie na península. Por meio desta análise, espera-se obter informações úteis que contribuam para uma melhor compreensão da distribuição geográfica da espécie e que possibilitem a implementação de medidas adequadas para a sua preservação e conservação.

A. *Árvores de Decisão (Decision Trees)*

As árvores de decisão são algoritmos de aprendizagem supervisionada amplamente utilizados para tarefas de classificação e regressão. Semelhantes a fluxogramas, as árvores de decisão possuem uma estrutura hierárquica composta por nós (*decision nodes*) interconectados. Estes nós acabam por ser representados pelas variáveis ambientais, onde guiam o fluxo do algoritmo.

O nó-raiz (*root node*) é o ponto de partida da árvore e representa o atributo mais importante para a tarefa de classificação/regressão. A partir dele, os nós subsequentes são gerados, cada um correspondente a uma decisão com base em uma das variáveis disponíveis. Este processo continua até chegarmos aos nós-folha (*leaf nodes*), que representam os resultados finais da classificação/regressão.

No contexto de AA, os nós-folha são responsáveis por atribuir uma classe ou um valor específico como resposta à tarefa em questão. Neste caso de previsão de ocorrência da espécie de abelhas nas regiões da Península Ibérica, os nós-folha podem ser representados por valores binários (0 ou 1), indicando a presença ou ausência da espécie em uma região geográfica.

	Precision	Recall	F1-Score	Support
0	0.99	0.99	0.99	2794
1	0.07	0.08	0.08	37
Accuracy			0.97	2831
Macro AVG	0.53	0.53	0.53	2831
Weighted AVG	0.98	0.97	0.98	2831

TABLE II
RESULTADOS OBTIDOS – *Decision Trees*.

B. *Floresta Aleatória (Random Forest)*

O modelo *Random Forest* é uma extensão do conceito de árvores de decisão, e recebe esse nome porque cria uma "floresta" de várias árvores de decisão. Em vez de depender de apenas uma única árvore de decisão, o *Random Forest* combina as predições de várias árvores individuais para obter resultados mais precisos.

Cada árvore de decisão individual é construída com uma amostra aleatória dos dados disponíveis. Esta aleatoriedade garante que cada árvore tenha informações diferentes e independentes, o que contribui para a diversidade e generalização do modelo. Durante o processo de construção da floresta, os dados são "misturados" para assegurar que cada árvore tenha uma visão única do conjunto de dados.

A principal vantagem do modelo *Random Forest* é a sua capacidade de lidar bem com problemas complexos e de alta dimensionalidade, onde a diversidade de árvores e a combinação das suas previsões são as chaves para o sucesso. Em vez de fornecer apenas um resultado e uma grama estreita de grupos como ocorre numa única árvore de decisão, este algoritmo é capaz de gerar um resultado mais preciso e abrangente, com uma maior variedade de grupos e decisões possíveis.

	Precision	Recall	F1-Score	Support
0	0.99	0.99	0.99	2794
1	0.08	0.08	0.08	37
Accuracy			0.98	2831
Macro AVG	0.53	0.53	0.53	2831
Weighted AVG	0.98	0.98	0.98	2831

TABLE III
RESULTADOS OBTIDOS – *Random Forest*.

C. *Vizinhos mais próximos (Nearest Neighbour)*

O método *K-Nearest Neightbors* (K-NN) é um algoritmo de aprendizagem supervisionada também utilizado para tarefas de classificação e regressão. A classificação ocorre com base na distância geométrica entre o ponto avaliado e os k vizinhos mais próximos pertencentes ao conjunto de treino.

O valor de k é um parâmetro crucial para o desempenho do algoritmo. Quando `k=1` é utilizado, o algoritmo procurará, dentro do conjunto de treino, o ponto mais próximo daquele que está a ser avaliado e atribuirá, posteriormente, o mesmo *label* a esse ponto. Por outro lado, quando k é maior, o algoritmo considerá um grupo de k vizinhos mais próximos para realizar a classificação.

Neste projeto, foi adotado o valor de `k=9` como parâmetro para o K-NN. Isto significa que o algoritmo irá considerar os nove pontos mais próximos do ponto avaliado para realizar a classificação. Esta escolha pode variar de acordo com o conjunto de dados e o problema específico em questão.

O K-NN é um método simples e fácil de implementar, mas a seleção adequada do valor de k é fundamental para obter um bom desempenho. Um valor baixo de k pode levar a um modelo sensível ao ruído, enquanto um valor muito alto pode tornar o modelo excessivamente generalizado.

	Precision	Recall	F1-Score	Support
0	0.99	0.89	0.94	2794
1	0.05	0.46	0.09	37
Accuracy			0.89	2831
Macro AVG	0.52	0.68	0.52	2831
Weighted AVG	0.98	0.89	0.93	2831

TABLE IV
RESULTADOS OBTIDOS – K-NN.

D. *Máquina de Vetores de Suporte (Support Vectors Machine*

O *Support Vector Machine* (SVM) é um algoritmo de aprendizagem supervisionada largamente utilizado para a classificação/regressão. O objetivo deste algoritmo é encontrar a melhor fronteira de separação entre as classes presentes num determinado conjunto de dados, em que sejam linearmente separáveis. Esta fronteira é intitulada por "hiperplano".

Para entender melhor o conceito de hiperplano, imagine-se um conjunto de dados bidimensional com duas classes distintas – como para este projeto (0 ou 1). O SVM procura

encontrar a linha que melhor separa estas duas classes, de modo que todas as amostras de uma classe fiquem de um lado da linha e as amostras da outra classe fiquem do outro lado. Esta linha é o hiperplano de separação e pode ser generalizado para espaços de dimensões mais altas.

Para casos em que os dados não sejam linearmente separáveis, o SVM utiliza uma técnica chamada "*kernel trick*" para mapear os dados para um espaço de maior dimensão, onde a separação entre as classes pode ser realizada por um hiperplano. Isto permite que o SVM trabalhe efetivamente mesmo em conjuntos de dados complexos e não linearmente separáveis.

	Precision	Recall	F1-Score	Support
0	0.99	0.60	0.75	2794
1	0.02	0.62	0.04	37
Accuracy			0.60	2831
Macro AVG	0.51	0.61	0.39	2831
Weighted AVG	0.98	0.60	0.74	2831

TABLE V
RESULTADOS OBTIDOS – SVM.

E. ***Redes neuronais (Multi-Layer Perceptron)***

O *Multi-Layer Perceptron* (MLP) é um tipo de rede neuronal artificial que se assemelha ao perceptron simples, mas com uma estrutura mais complexa, em que contém mais de uma camada de neurónios. Enquanto o perceptron simples é capaz de resolver apenas problemas linearmente separáveis, o MLP é capaz de lidar com problemas mais complexos e não-linearmente separáveis.

A principal diferença entre o perceptron simples e o MLP é a presença de camadas ocultas no MLP. As camadas ocultas são chamadas assim porque não é possível prever diretamente a saída desejada nesses neurónios intermediários. Cada camada oculta é composta por um número indeterminado de neurónios, e a quantidade de camadas ocultas pode variar dependendo do problema. Esta estrutura de múltiplas camadas intermediárias permite que o MLP crie múltiplas fronteiras de separação não-lineares, tornando-o capaz de resolver problemas de classificação e regressão mais complexos.

	Precision	Recall	F1-Score	Support
0	0.99	0.99	0.99	2794
1	0.06	0.05	0.06	37
Accuracy			0.98	2831
Macro AVG	0.52	0.52	0.52	2831
Weighted AVG	0.98	0.98	0.98	2831

TABLE VI
RESULTADOS OBTIDOS – MLP.

O MLP é treinado utilizando um processo chamado *back-propagation*, que ajusta os pesos das conexões entre os neurónios para minimizar o erro entre as previsões do modelo e os *labels* reais do conjunto de treino. Este processo é realizado iterativamente até que o modelo atinja um nível satisfatório de precisão nas previsões.

Uma das vantagens do MLP é a sua capacidade de aprender e representar relações não-lineares em dados complexos. No entanto, o treino de um MLP pode ser mais lento e requer mais dados de treino do que outros algoritmos, devido à grande quantidade de parâmetros envolvidos nas camadas ocultas.

F. ***Regressão Logísitica (Logistic Regression)***

A Regressão Logística é outro algoritmo amplamente utilizado para problemas de classificação. Este modelo ajusta uma curva sigmoide (curva em forma de `S`) aos dados binários e produz a probabilidade de a variável alvo `Y` pertencer a uma das classes. A curva sigmoide permite, então, que o modelo faça previsões de probabilidade, que podem ser transformadas em previsões de classe com um limiar de decisão (geralmente 0,5).

Uma das vantagens da Regressão Logística é a sua simplicidade e eficiência computacional. É frequentemente aplicada a problemas de classificação binária, onde existem apenas duas classes a serem previstas. No entanto, ela também pode ser estendida para lidar com problemas de classificação com mais de duas classes, usando técnicas como a Regressão Logística Multinomial ou *One-vs-All*.

O treino deste algoritmo envolve a minimização de uma função de perda, como a função de perda logística (conhecida também por função de entropia cruzada), utilizando algoritmos de otimização, como o Gradiente Descendente.

	Precision	Recall	F1-Score	Support
0	0.99	0.64	0.78	2794
1	0.02	0.62	0.04	37
Accuracy			0.64	2831
Macro AVG	0.51	0.63	0.41	2831
Weighted AVG	0.98	0.64	0.77	2831

TABLE VII
RESULTADOS OBTIDOS – RL.

- ***Precision*** – Proporção de previsões positivas corretas em relação ao total de previsões positivas feitas pelo modelo;
- ***Recall*** – Proporção de previsões positivas corretas em relação ao total de amostras da classe presentes no conjunto de dados;
- ***F1-Score*** – Média entre a precisão e o *recall*;
- ***Support*** – Número de amostras pertencentes a cada classe no conjunto de teste;
- ***Accuracy*** – Proporção de previsões corretas em relação ao total de amostras no conjunto de teste;
- ***Macro* AVG** – Média aritmética das métricas (*precision, recall* e *F1-score*) para ambas as classes;
- ***Weighted* AVG** - Média ponderada das métricas (*precision, recall* e *F1-score*) para ambas as classes, tendo em consideração o número de amostras de cada classe.

VI. Curvas ROC e AUC

Na aprendizagem automática, a medição do desempenho é uma tarefa crucial. As métricas de desempenho ROC (*Receiver Operating Characteristics*) e AUC (*Area Under the Curve*) são das mais utilizadas para medir o desempenho de modelos de aprendizagem supervisionada, especialmente em problemas de classificação.

A curva ROC é uma ferramenta gráfica que permite avaliar o quão bem o modelo pode distinguir entre duas classes, neste caso, entre 0 e 1. A curva ROC é construída a partir da Taxa de Verdadeiro Positivo (TPR) no eixo Y e a Taxa de Falso Positivo (FPR) no eixo X. A TPR é a proporção de verdadeiros positivos corretamente identificados em relação ao total de positivos reais, enquanto a FPR é a proporção de falsos positivos incorretamente identificados em relação ao total de negativos reais.

No gráfico da curva ROC, um modelo ideal teria uma curva que atinge o canto superior esquerdo do gráfico, representando uma TPR de 1 e uma FPR de 0. Isto significa que o modelo acerta todas as previsões positivas sem cometer falsos positivos. À medida que o modelo se afasta desse ponto, a sua capacidade de separação diminui.

Relativamente à área sob a curva (AUC), trata-se de uma métrica que resume a curva ROC em um único valor, medindo a capacidade do modelo de separar as classes corretamente. Uma AUC próxima de **1** indica um modelo excelente, com uma taxa alta de previsões corretas e uma boa capacidade de separação entre as classes.

```
Random Forest AUC           -> 0.5344560738261526
Nearest Neighbour AUC       -> 0.6751484842036023
Multi Layer Perceptron AUC  -> 0.6893874905685929
Decision Tree AUC           -> 0.5335612993093308
SVM AUC                     -> 0.6118129582696511
Logistic Regression AUC     -> 0.6293505387993577
```

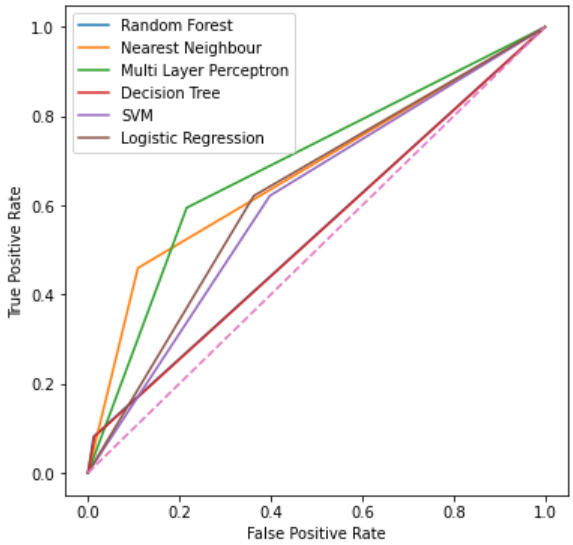

Fig. 4. Resultados da curva ROC e AUC.

Por outro lado, uma AUC próxima de **0** indica um modelo com baixa capacidade de separação e previsões pouco confiáveis. Quando a AUC é igual a **0.5**, o modelo não tem capacidade de distinguir entre as classes e as suas previsões são equivalentes a um palpite aleatório.

A Figura 4 representa as curvas ROC e as respetivas AUC para cada modelo testado, o que ajuda em comparar a selecionar o melhor modelo obtido. Esta análise das curvas ROC e AUC é fundamental para avaliar o desempenho dos modelos de classificação e tomar decisões sobre qual modelo é o mais adequado para o problema em questão.

VII. Conclusões

Após treinar os seis modelos e analisar os resultados obtidos ao longo deste trabalho, conclui-se que o melhor modelo para separar as classes `y=0` e `y=1` é o *Multi-Layer Perceptron*, apresentando uma AUC de **0.69**. Por outro lado, os modelos com pior desempenho foram os *Decision Trees* e *Random Forest*, ambos com valores próximos de **0.53**.

Quanto à métrica de *accuracy*, o algoritmo *Random Forest* obteve a maior precisão, atingindo **97.6**%. É importante salientar que, apesar da alta *accuracy* do *Random Forest*, a baixa AUC sugere que o modelo pode não estar a ser eficiente em fazer previsões corretas, fazendo com que se deva ter em consideração as restantes métricas, como a precisão e o *recall*, para a classe minoritária, de modo a ter uma avaliação mais completa do desempenho do modelo.

Embora os resultados obtidos possam ser considerados bons – mas não excelentes –, existem diversas abordagens para lidar com o problema da baixa precisão dos dados relativos à classe minoritária. Entre elas, destacam-se a inclusão de mais dados de ocorrência da espécie para enriquecer o conjunto de treino, a realização de treinos mais prolongados por parte dos modelos, a exploração de outros algoritmos de aprendizagem automática, a adaptação da arquitetura do classificador e a experimentação com diferentes parametrizações, por exemplo.

Para concluir, este projeto representa uma jornada empolgante na vanguarda da aprendizagem automática, oferecendo um importante salto em direção à compreensão e aplicação de algoritmos para a previsão da ocorrência de espécies em determinadas regiões geográficas. Com os métodos utilizados neste projeto, e com os dados que foram adquiridos previamente, podem, sem dúvida alguma, impulsionar a conservação e a exploração sustentável não só da espécie *Apis Melífera Honeybee*, como também de todas as outras que pertencem ao nosso precioso ambiente natural.

REFERENCES

[1] Oversampling and Undersampling [Online] https://towardsdatascience.com/oversampling-and-undersampling-5e2bbaf56dcf

[2] Understanding The Basics Of SVM With Example And Python Implementation [Online] https://analyticsindiamag.com/understanding-the-basics-of-svm-with-example-and-python-implementation/

[3] Implementação da árvore de decisão usadno python [Online] https://acervolima.com/implementacao-da-arvore-de-decisao-usando-python/

[4] How to Plot a ROC Curve in Python (Step-by-Step) [Online] https://www.statology.org/plot-roc-curve-python/

[5] Entenda o que é AUC e ROC nos modelos de Machine Learning [Online] https://medium.com/bio-data-blog/entenda-o-que-%C3%A9-auc-e-roc-nos-modelos-de-machine-learning-8191fb4df772

[6] Multi-Layer Perceptron Neural Network using Python [Online] https://machinelearninggeek.com/multi-layer-perceptron-neural-network-using-python/

[7] Building A Logistic Regression in Python, Step by Step [Online] https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8

[8] Nearest Neighbors [Online] https://scikit-learn.org/stable/modules/neighbors.html

[9] Scikit-learn [Online] https://scikit-learn.org/stable/modules/neighbors.html

[10] Scikit-learn [Online] https://scikit-learn.org/stable/index.html

[11] Numpy [Online] https://numpy.org/

[12] Matplotlib [Online] https://matplotlib.org/