UNIVERSIDAD FRANCISCO DE VITORIA

ESCUELA POLITECNICA SUPERIOR

Artificial Intelligence 2

Practical work

Perceptron and Multilayer perceptron

GROUP 8

Done by:

Christopher Cobo Piekenbrock, Rodrigo Sáez Escobar, Gonzalo Eusa Silvela, Jaime López de Heredia Delgado and Juan Fernández Cerezo.

Artificial Intelligence 2

Abstract

In this practical project, we explored and compared linear and non-linear machine learning models. We focused on two main exercises to evaluate the performance of the Perceptron and the Multi-Layer Perceptron (MLP).

In the first exercise, we built a binary classifier from scratch using the Perceptron algorithm to classify flying Pokemon based on physical attributes. After cleaning, scaling, and encoding the dataset, we trained the model with different combinations of features for each Pokemon pair. We then evaluated the performance using accuracy scores and visualized the decision boundaries to understand how well each configuration separated the species.

In the second exercise, we used a Multi-Layer Perceptron (MLP) to estimate the survival probability of Game of Thrones characters. After preparing the dataset, we trained the model in Keras using different configurations and balancing techniques. We evaluated its performance with confusion matrices, F1-scores, and error analysis to understand which setup performed best.

Table of contents

Index of Figures

Index of Tables

1 Introduction

All exercises done are influenced by the learnings obtained in class and with the slides provided by our teacher from Unit 5 - Supervised Learning (Linear) [1], Unit 5 - Supervised Learning (Non-Linear) [2].

1.1 Practical work outline

This practical work consists of two exercises focused on both linear and non-linear architectures.

In the first exercise we have done all the process necessary to classify flying Pokemon using a binary linear classifier (Perceptron). The goal was not only to build the model, but also to evaluate and compare it´s performance using different feature combinations.

We have done it in three steps. Firstly, we have prepared the dataset and understanding it in detail. The second step we have done was building a Perceptron from scratch and training it with different feature combinations. And finally, after training the model we have evaluated and compared all the results obtained to answer the questions.

In the second exercise, our objective was to predict the survival of Game of Thrones characters based on various features provided in the dataset.

We have approached this exercise by dividing it into 3 steps. Firstly, we prepared the dataset by splitting the data into training, validation, and test sets. Then, we built and trained several Multi-Layer Perceptron models using Keras. Each model was trained using different hyperparameters (changing the number of hidden layers and neurons), learning rates, and with three different dataset balancing techniques:

No balancing (original data), **SMOTE** and **RUS** (Random Undersampling).

We evaluated each model's performance using metrics like F1-score, accuracy, and confussion matrix with multiple thresholds.

So as you can see, this was done for a main reason: It was done to see which setup worked best, and to learn how different balancing techniques and model hyperparameters (like the number of layers or the learning rate) affected the results. By testing and comparing all these variations, we could clearly understand what makes a model perform better.

2 Exercise 1

2.1 Description of the exercise and their goals.

This exercise focuses on developing a binary classifier (Perceptron) to classify flying Pokemons based on specific features. The goal is to build a model from scratch, test it with different feature combinations, and assess its performance based on the features used.

By implementing the Perceptron, we will explore the process of training a machine learning model using the Perceptron learning rule, which adjusts weights iteratively to reduce classification errors. We also experiment with different input features and evaluate how these affect the model's ability to separate the classes (Pokemons).

2.2 Dataset description

The dataset contains information about various species of flying Pokemon, including their physical attributes, which we will use to classify them. The dataset consists of the following columns:

- **species**: The species of the flying Pokemon.
- **bill_length_mm**: The length of the Pokemon's beak in millimeters.
- **bill_depth_mm**: The depth of the Pokemon's beak in millimeters.
- **wing_length_mm**: The length of the Pokemon's wings in millimeters.
- **body_mass_g**: The mass of the Pokemon in grams.

Each row represents an individual flying Pokemon, with its respective attributes and species.

2.3 Data preparation

Before training the model, we took time to prepare the data properly so everything would work well. This included some steps which were cleaning, transforming, and organizing the dataset before using it. We will explain them in detail below.

1. **Data Cleaning**:

   Missing values in the dataset were handled by removing rows containing any missing data using the dropna() function. This was done to make sure that the dataset used for training the model was complete and didn´t contain any gaps.

2. **Data Transformation**:

   The numerical features ("bill_length_mm", "bill_depth_mm", "wind_length_mm", and "body_mass_g") were scaled using MinMaxScaler. This process transforms the data to a specific range (between 0 and 1), which improved the model's performance.

3. **Data Splitting**:

   We created three datasets, each with a different pair of Pokemon species to compare:

- Pigey vs. Pidgeotto
- Pigey vs. Pidgeot
- Pidgeotto vs. Pidgeot

4. **Data Encoding**:

We converted the species column (text) into numerical values (0 or 1) to use with the perceptron, which only works with numbers.

2.4 Experiments

2.4.1 Experiment definition

For this experiment, the objective was to develop a binary classifier using the Perceptron algorithm in order to distinguish between different species of flying Pokemon based on their physical characteristics. To carry out this experiment, we created different datasets where each one included a pair of Pokemon species:

Pigey vs. Pidgeotto, Pigey vs. Pidgeot, and Pidgeotto vs. Pidgeot

In each case, the perceptron model was trained from scratch using the Perceptron learning rule. The features used to classify the Pokemon were: bill length, bill depth, wing length, and body mass. Our idea consisted of training the perceptron with all the different combinations of these attributes to evaluate their influence on the classification performance.

We started with random weights between -1 and 1, and during each epoch, the weights were updated based on the prediction error. The training stopped when the error reached zero or could not be further minimized. This allowed us to find a hyperplane that separates both Pokemon species.

2.4.2 Experiment evaluation

To evaluate the performance of the perceptron in each classification, we calculated the accuracy obtained after the training phase. The evaluation focused on analysing how well the perceptron was able to correctly classify the Pokemon in each pair.

We also created graphical visualizations showing how the how the data was distributed and where the separating hyperplane was placed. These plots helped us better understand the classification difficulty for each species pair.

2.5 Questions

Analyze the charts created in step 1 and explain which pair of attributes you will use for classifying each pair of flying Pokemon types

The pair plot in step 1 helped us visualize the relationships between all possible pairs of attributes for the three Pokemon species: Pigey, Pidgeotto, and Pidgeot. Each point on the plot represents a Pokemon, and the color indicates its species. The diagonal plots show the distribution of each attribute for each species. The off-diagonal plots show the relationship between two attributes for each species. This can be seen in the figure below.

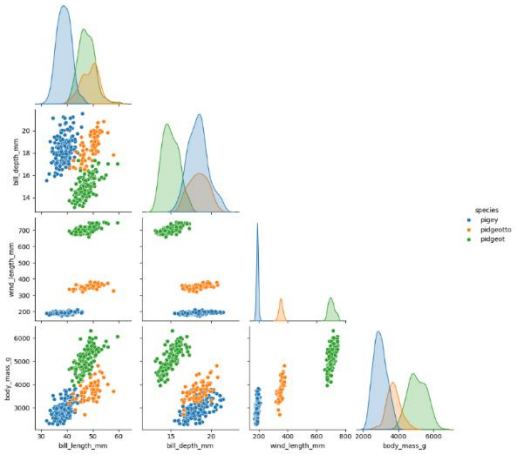


Figure 1. Chart of any pair of attributes representing each of the three flying pokemon

Ideally, we could choose a pair of attributes that shows a clear separation between the species. This will make it easier for our perceptron to learn a decision boundary that can accurately classify the Pokemon.

Pigey vs Pidgeotto	**Pigey vs Pidgeot**	**Pidgeotto vs Pidgeot**
Best Attribute Pair: wind_length_mm and bill_length_mm Pigey (blue) appears in the lower range for both attributes, while Pidgeotto (orange) is higher. The two clusters are visually separable with a straight line.	**Best Attribute Pair:** bill_length_mm and bill_depth_mm Pigey has lower values, and Pidgeot (green) shows higher values in both attributes. This allows for clear linear separation.	**Best Attribute Pair:** wind_length_mm and bill_depth_mm Pidgeotto shows little variation, while Pidgeot spans a wider range. This distinction makes linear classification possible.

Table 1-Pairs of features that are separable with a straight line -Selfmade

In our case, instead of selecting the pair of attributes, we tried all of them because we wanted to see how different pairs of attributes behaved when they couldn't be perfectly separated by a hyperplane. The pair plot allows us to visually assess the relationships between each attribute pair, including those where the classes might not have clear linear separability.

Plot a graph with differently colored points for each type of flying prokemons and draw the hyperplane on it. Write the equation of the hyperplane that has been generated.

For each pair of Pokemon, we generated scatter plots where each species was represented using a different color. We also drew the hyperplane learned by the perceptron after the training process.

The general equation of the hyperplane created by the perceptron is:

$$\textbf{2D:} \qquad \boldsymbol{w1x1 + w2x2 + b = 0}$$

$$\textbf{3D:} \qquad \boldsymbol{w1x1 + w2x2 + w3x3 + \ b = 0}$$

Where:

- **w1, w2** and **w3** are the weights associated with the selected attributes learned by the model,
- **x1, x2** and **x3** are the values of the attributes,
- **b** is the bias.

The exact values of w1,w2, and b varied depending on the Pokemon pair and the attributes used, as the perceptron trained different models for each case.

Also, it´s important to mention that we tried different learning rate values like 0.1, 0.25, and 0.5. After comparing the results, we decided to use 0.1 for all the models, since it gave good accuracy and a stable learning process.

Below, we present the different graphs obtained for each pair of Pokemon, where the points are colored according to their species, and the corresponding hyperplane is drawn to illustrate the model's decision boundary (hyperplane).

For these two examples, we used **pigey vs pidgeot** with the 4 features, to plot this, we did a PCA. For the 2D representation we chose the first 2 and for the 3D representation the first 3.

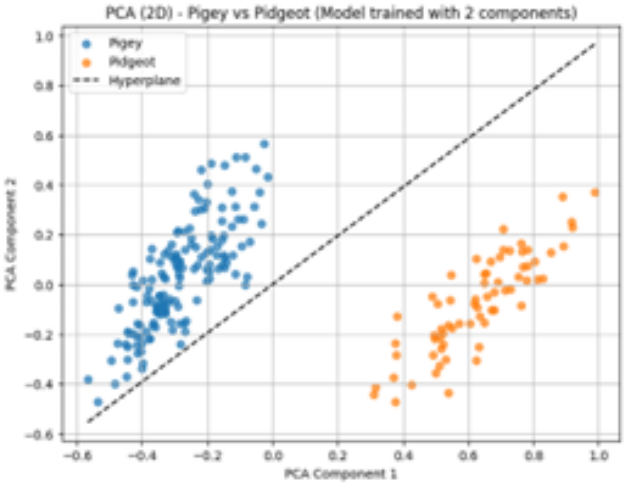


Figure 2-2D visual representation (Pigey vs Pidgeot) – Selfmade

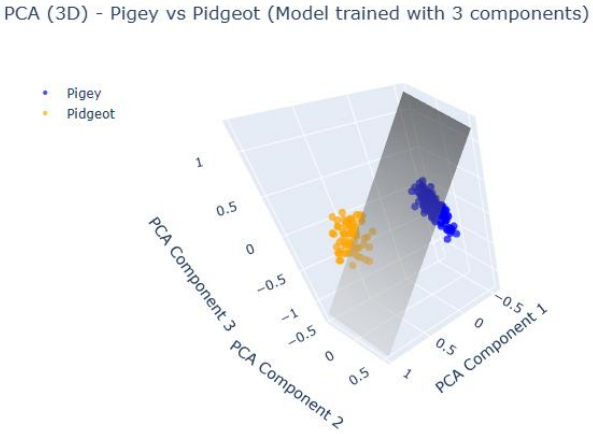


Figure 3-3D visual representation (Pigey vs Pidgeot) – Selfmade

Accuracy (2 components): 1.0000, Error: 0.0000	Accuracy (3 components): 1.0000, Error: 0.0000
Hyperplane equation: 0.5100*x1 + -0.4786*x2 + -0.1000 = 0	Hyperplane equation: 0.8192*x1 + -0.6550*x2 + 0.7989*x3 + -0.2000 = 0

The rest of the visual representations for all the combinations of Pokemons with all the combinations of features are visible in Annex A.

Compare the results obtained in the step 2 and explain which model is the best option.

The models were tested with combinations of two, three, and four features, and their performance was assessed based on accuracy. The results indicate which combinations of features achieve the highest accuracy.

2 Features	3 Features	4 Features
For the 2-feature combinations, trying all	For the 3-feature combinations, trying all	For the 4-feature combinations all models

combinations of features, the range of accuracy was from **0.4384** to **1.0000,** showing that while some pairs of features were highly predictive, others were less reliable.	combinations of features, the range of accuracy was from **0.9817** to **1.0000,** with most combinations performing consistently well. This suggests that adding more features gives the perceptron more information that results in better performance.	trained with the four features achieved **100% accuracy (1.0000)**. This indicates that using all the features together provides the most complete information, allowing the model to perfectly separate the classes in all Pokemon combinations.

Table 2-Comparing the results -Selfmade

To identify the best configuration for classifying each pair of Pokemon, we followed a strategy. We started by testing combinations of 2 features, then moved to 3 features, and finally used all 4 features, as shown in Tables 12 to 20 in Annex B.

For the **Pigey vs Pidgeotto** pair (Table 12) some 2 feature combinations such as bill_length_mm and wind_length_mm achieved 100% accuracy, but others performed poorly. For example, bill_depth_mm and body_mass_g resulted in an accuracy of just 0.4384 showing that 2 features are not always enough to separate the classes well. With 3 features (Table 15), most combinations reached perfect accuracy, except one that had a small error (accuracy 0.9817, error 0.0183). Only when using all 4 features (Table 18), we obtain consistent 100% accuracy and zero error, confirming that all the confirming that using all four physical attributes was the best choice.

For the **Pigey vs Pidgeot** classification was easier. As shown in Table 13, all 2-feature combinations already achieved perfect results, with accuracy 1.0000. This pattern stayed the same for 3 features (Table 16) and 4 features (Table 19). These means that Pigey and Pidgeot are very different in their physical features, and even simple models can classify them correctly.

The **Pidgeotto vs Pidgeot** case was in the middle. In Table 14, most 2 features combinations worked well but one pair (bill_length_mm and bill_depth_mm) only reached 0.9058 accuracy. With 3 features (Table 17) and 4 features (Table 20), all models achieved perfect results, showing that these two species can also be separated, but they need more information to do so.

This strategy we follow (starting with 2 features and progressively adding more) helped us understand how complex each model needed to be. For very different species, like Pigey and Pidgeot, a simple 2 feature model is enough. But for similar species, like Pigey and Pidgeotto, using 3 or 4 features gives better results.

Therefore, we can say that the the model trained with 4 features is the best option. It's not just because it achieve perfect accuracy in all cases (which already show us that is reliable) but also because it performs well no matter which Pokemon pair we were trying to classify.

Another important reason is that when we use all 4 features, we’re giving the model more information about each Pokemon. This helps it creating a better decision boundary that separates the species more clearly. When it was just with 2 or 3 features, we sometimes lose important details, and the model makes more mistakes

So even if some 2 feature combinations worked perfectly in specific cases, only the 4 feature model worked perfectly in all of them. That’s why we think it’s the most complete and trustworthy option overall.

All the accuracy scores and hyperplane equations for the different Pokemon combinations and the features are located in Annex B.

3 Exercise 2

3.1 Description of the exercise and their goals.

The objective of this exercise is to develop a Multi-Layer Perceptron (MLP) model using the Keras framework to predict the probability of death for characters in the Game of Thrones (GoT) universe. The model is trained using features that describe each character, with the final output being the probability that the character is alive (isAlive). The goal is to build, train, and evaluate a neural network capable of generalizing well and identifying the most likely character to die.

3.2 Dataset description

The dataset contains information on 1,946 characters from the Game of Thrones universe, with several features that describe the character's background, relationships, and other relevant factors.

For training the MLP model, the following features were selected:

Gender (male): Indicates the character's gender: 1 for male, 0 for female.

Book1, Book2, Book3, Book4, Book5): Five binary indicators showing whether the character appears in each of the five books.

Maried Status (isMarried): Indicates if the character is married: 1 for married, 0 for not married.

Nobility Status (isNoble): Indicates if the character is noble: 1 for noble, 0 for not noble.

Number of Dead Relations (numDeadRelations): The number of people close to the character who have died.

Popularity (isPopular): Indicates if the character is considered popular: 1 for popular, 0 for not popular.

The target variables is (**isAlive)** which is a binary indicator representing whether the character is alive (1) or dead (0).

All features were normalized using MinMaxScaler to scale the values between 0 and 1, which helps improve the performance of the neural network.

3.3 Data preparation

We started by preparing the dataset to make sure it was ready for training our models. This involved a few simple but important steps:

1. **Data Loading and Exploration:**
   The dataset, containing information about Game of Thrones characters, was loaded from a CSV file (got.csv). Initial exploration (.head(), .info(), .describe()) helped us understand the structure.

2. **Feature Selection:**
   Based on the intructions, features were selected to be used as input for the model(male, book 1 to book 5, isMarried, isNoble, numDeadRelations, isPopular). As we have mentioned before, the target variables was IsAlive.

3. **Data Normalization:**

All selected features were normalized to the [0, 1] range using MinMaxScaler. This helps improve model convergence and ensures no feature dominates due to its scale.

4. **Data Splitting:**
   The dataset was split into training (80%), validation (10% of the training set), and test sets (20%) to evaluate model performance.

5. **Balancing Techniques:**
   To explore how class imbalance affects model performance, we decided to compare three different approaches:
   Using the original dataset without any balancing, applying SMOTE to increase the presence of the minority class and using RUS (Random Undersampling) to reduce the size of the majority class. This helped us to compare the impact each balancing method had on the final performance of the models.

6. **Preparation for Modeling:**
   We separated the features (X) and target (y) for each dataset version and used these consistently across all training configurations.

3.4 Experiments

3.4.1 Experiment definition

In this exercise, our goal was to build a binary classifier using a Multi-Layer Perceptron implemented with Keras, to predict whether a character from Game of Thrones would survive or not.

To do this, we trained several MLP models using different setups. Specifically, we tested multiple model architectures (varying the number of hidden layers and neurons) and combined them with different training hyperparameters, such as the learning rate. This allowed us to analyze how both the structure of the model and the training settings affected the performance.

We also wanted to evaluate the effect of data balancing techniques, so we trained the models using **three different versions seen in the theory** of the training data: the original (unbalanced) dataset, a version balanced using SMOTE and another one balanced using Random Undersampling (RUS).

All models were trained using binary cross-entropy as the loss function and the Adam optimizer. We trained each model for 200 epochs, using early stopping through validation performance monitoring to avoid overfitting.

To evaluate performance, we tracked F1-score, accuracy, and confussion matrices. We also analyzed how different classification thresholds (from 0.3 to 0.7) affected the results.

After training all configurations, we selected the best model for each data balancing technique and compared them to understand which setup achieved the best performance and generalization.

3.4.2 Experiment evaluation

We tested those best models on the test set, checking their accuracy, F1-score, and looking at their confusion matrices for several classification thresholds (from 0.3 to 0.7). This helped us see how well each model handled the balance between precision and recall, and how much the predictions changed depending on the threshold.

To make the results easier to understand, we also created a bar chart that shows the average F1-scores and the variance for each technique across all the model architectures. This gave us a good visual comparison and helped highlight which approach was more stable and effective.

Finally, we used a classification report on the best overall model to check how well it performed on the test set. We noticed that, in some cases, training on the original data with a good architecture gave surprisingly strong results, even better than with SMOTE or RUS.

In the end, this experiment wasn't just about finding the best model. It was about understanding how different things like the architecture, hyperparameters, and data balancing methods affected the results. That comparison gave us real insighst into what worked best and why.

3.5 Questions

Evaluate the different models using the confusion matrix. Choose the best model and determine if the training process was successful and if the model has generalized effectively.

After training our models using three different types of models (Original, Smote, and RUS) with multiple architectures and learning rates we have calculated different metrics such as accuracy, F1-score, and confusion matrices at different thresholds (0.3 to 0.7).

Based on the results, the best model was the one trained on the original dataset with a threshold of 0.3. This configuration achieved an F1-score of 0.86 and an accuracy of 0.76, which was higher than the other two techniques across all thresholds.

We also created a comparison chart showing the average F1-score and variance for each technique. This confirmed that the original dataset not only performed better but was also more stable.

Finally, the model worked well not only during training but also on new data. Its performance stayed strong across different thresholds, as we saw in the confusion matrices and evaluation results.

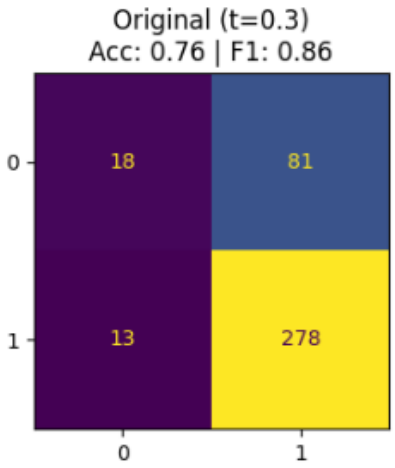


Figure 4. Best Confusion Matrix

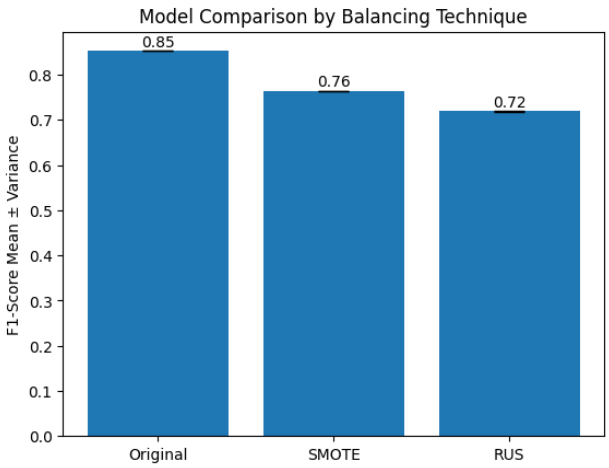


Figure 5. F1 chart comparison

Predict which is the character with more probabilities of death from the test set.

After training and evaluating all models, we used the best-performing configuration to make predictions on the test set. According to the model, the character with the highest probability of dying is **Olene Tyrell**, with a predicted death probability of 0.9985.

We checked her characteristics to justify this prediction and found that she is not popular, not noble, she is female, and only appears in books 3 and 4, which suggests she has limited presence in the story. She is married, but her husband is alive, and she has no dead relatives, meaning she doesn't have strong family connections or an important role in the story that might protect her from being killed.

```
Character most likely to die: Olene Tyrell
Predicted death probability: 0.9985078573226929
```

Figure 6. Result of who is most likely to die next

Draw the network architecture used to generate your best model specifying the optimizer, the number of neurons in the hidden layer, and the activation and error functions used.

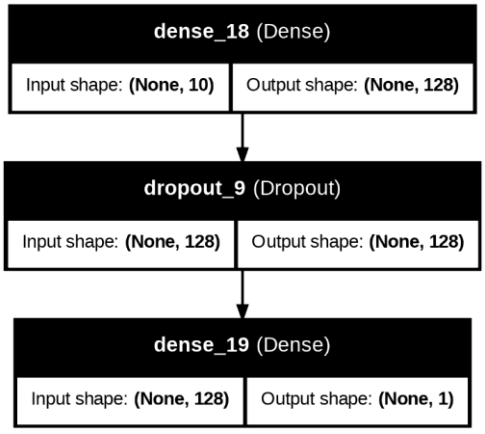


Figure 7. Of the network architecture

```
Best Model Configuration:
Hidden Neurons: 128, 1
Optimizer: Adam
Learning Rate: 0.0024999999441206455
Training Loss: 0.40876272320747375
Activation Function (Hidden Layers): ReLU
Activation Function (Output Layer): Sigmoid
Validation Loss: 0.5085566639900208
Error Function (Loss Function): Binary Cross-entropy
```

Figure 8. Configuration of the best model

Predict the isAlive probability for the characters listed in your test set with the error for the prediction comparing it to the validation error

Identify the character with the highest probability of dying.

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Character most likely to die: Olene Tyrell
Predicted death probability: 0.9985078573226929
```

Figure 9 Result of who is most likely to die

Calculate the error for this prediction.

```
Prediction error for Olene Tyrell: 0.001492142677307129
```

Figure 10. Of the prediction error of Olene Tyrell

Compare it to the validation error: Is it higher or lower? Explain why.

```
Best Model Validation Loss: 0.4823657274246216
```

Figure 11. Result of validation error of the best model

The prediction error for Olene Tyrell is very low, which means the model was very confident that she would die, but she is actually alive. When we compare this prediction error with the validation loss, we can see that this specific prediction had a much smaller error than the average in the validation set.

This makes sense because loss is computed over the whole validation set and includes many other examples where the model might have made bigger errors. In this case, the model made a rare very confident mistake.

Adjust the predictions so that a different character is identified as the one most likely to die:

- **Describe the specific changes you made to the test set. Justify your changes from the perspective of MLP.**
- **Are all input features equally significant? Explain how you can demonstrate their relative importance.**

After changing the features of two characters in the test set, we were able to change the model's prediction about who is most likely to die.

We made Olene Tyrell look safer by giving her more popularity, more nobility, and adding more book appearances. At the same time, we made Aeron Greyjoy look more at risk by removing his popularity and nobility and adding many dead relatives.

In the end, the model picked Jeyne Lydden as the character most likely to die (with a probability of 0.9941). This means that the changes we made affected the prediction, but another character still ended up being the top.

This shows that changing the input values changes the result, and that some features like popularity or book appearances have a big impact on the prediction.

3 Conclusions

3.1 Learnings and Implications of the Experiments

On the first exercise, we have learned how changing things like the features or the learning rate can affect the model's performance. We also saw that it's important to try different things and adjust the model based on the results. Another learning was that visualizing the decision boundary helps us understand what the model is doing and where it might be going wrong. Lastly, we realized that preparing the data well is crucial for getting good results from the model.

For the second exercise, we learned that experimenting with different model configurations and parameters is essential. We realized that there isn't always a single optimal solution, and the key is to try various options, evaluate how they affect the results, and then compare them.

Another important lesson was the significance of choosing the right metrics to evaluate the model. In this case, comparing F1-score, accuracy, and the confusion matrix gave us a more comprehensive view of the model's performance. This allowed us to identify the best configuration, even when the results weren't immediately obvious.

Finally, the process of comparing different class balancing techniques (like SMOTE and RUS) taught us that some approaches have a more significant impact than others, and each model can behave differently depending on the parameters we use. This exercise showed us that adjusting and continuously comparing hyperparameters is a crucial part of the modeling process.

In conclusion, both exercises taught us the importance of testing different approaches to see what works best.

4 Bibliography

[1] M. M. Muñoz, AIII - Unit 5 - Supervised Learning (Linear).

[2] M. M. Muñoz, AIII - Unit 5 - Supervised Learning (Non-Linear).

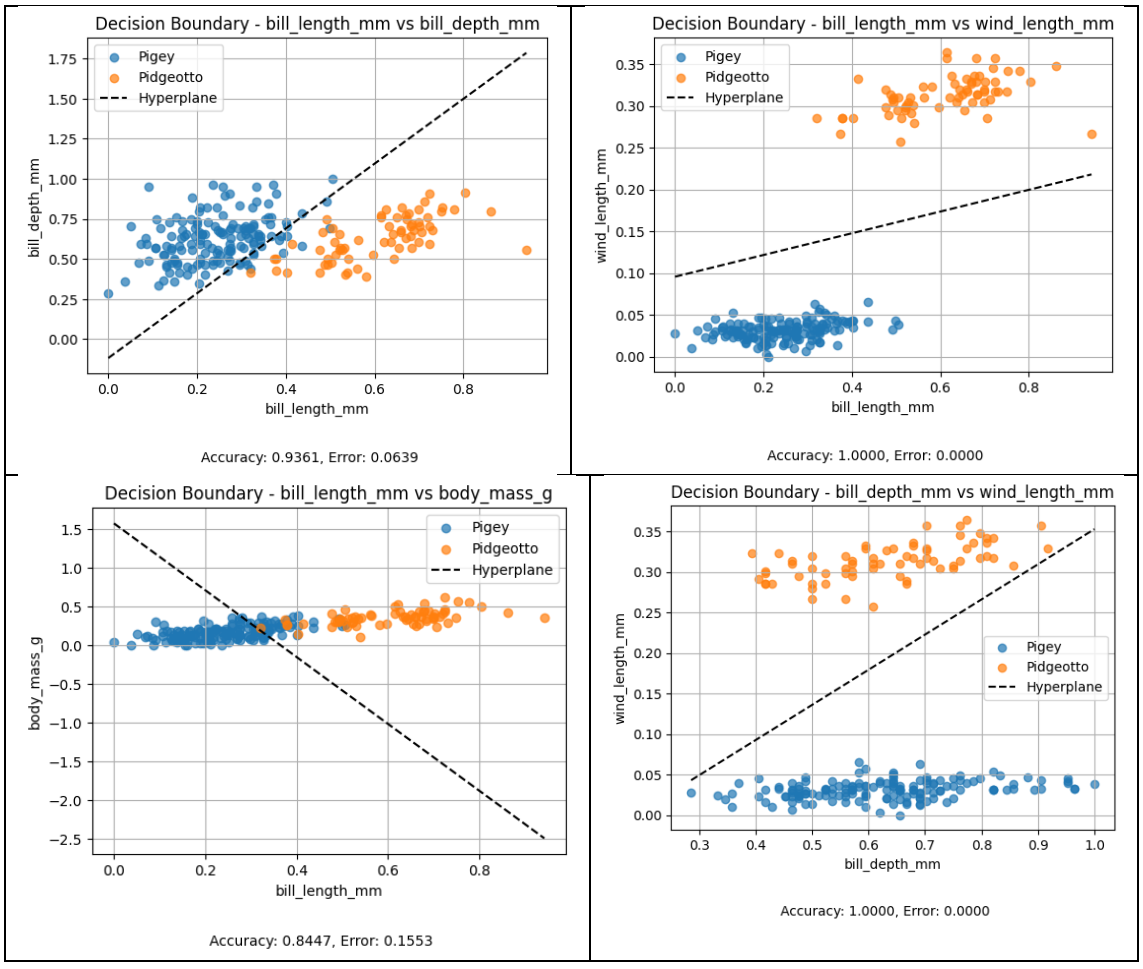
Annex A - Visual representations

In this annex, you will find all the graphs generated during the training and evaluation of the perceptron models. The visualizations are organized by feature combinations (2, 3, and 4 features) and Pokemon pairs. Each plot includes the accuracy and error.

A.1 Graphs with 2 Features

Description: These graphs illustrate the relationships between pairs of features.

A.1.1 Pigey vs Pidgeotto



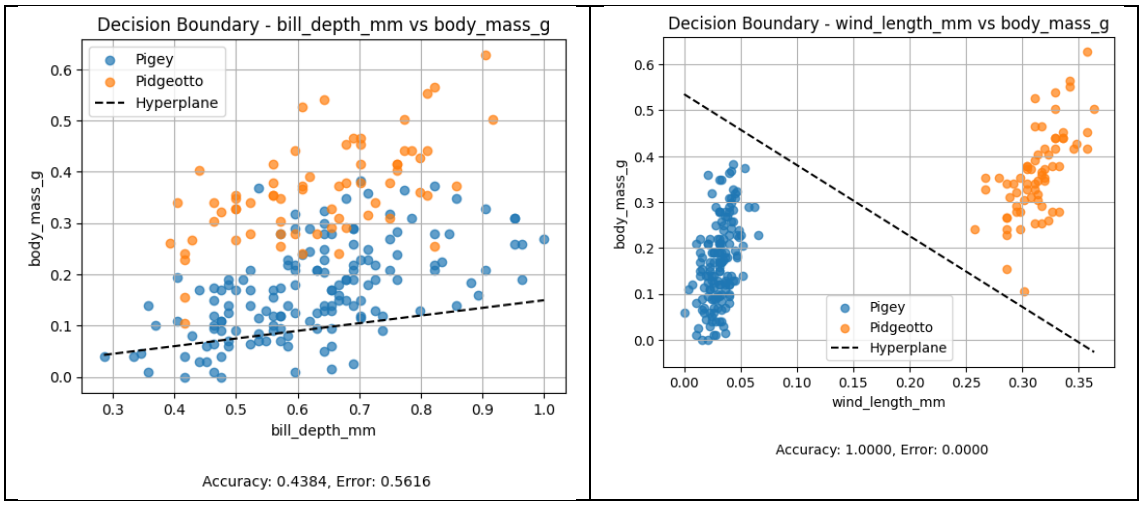
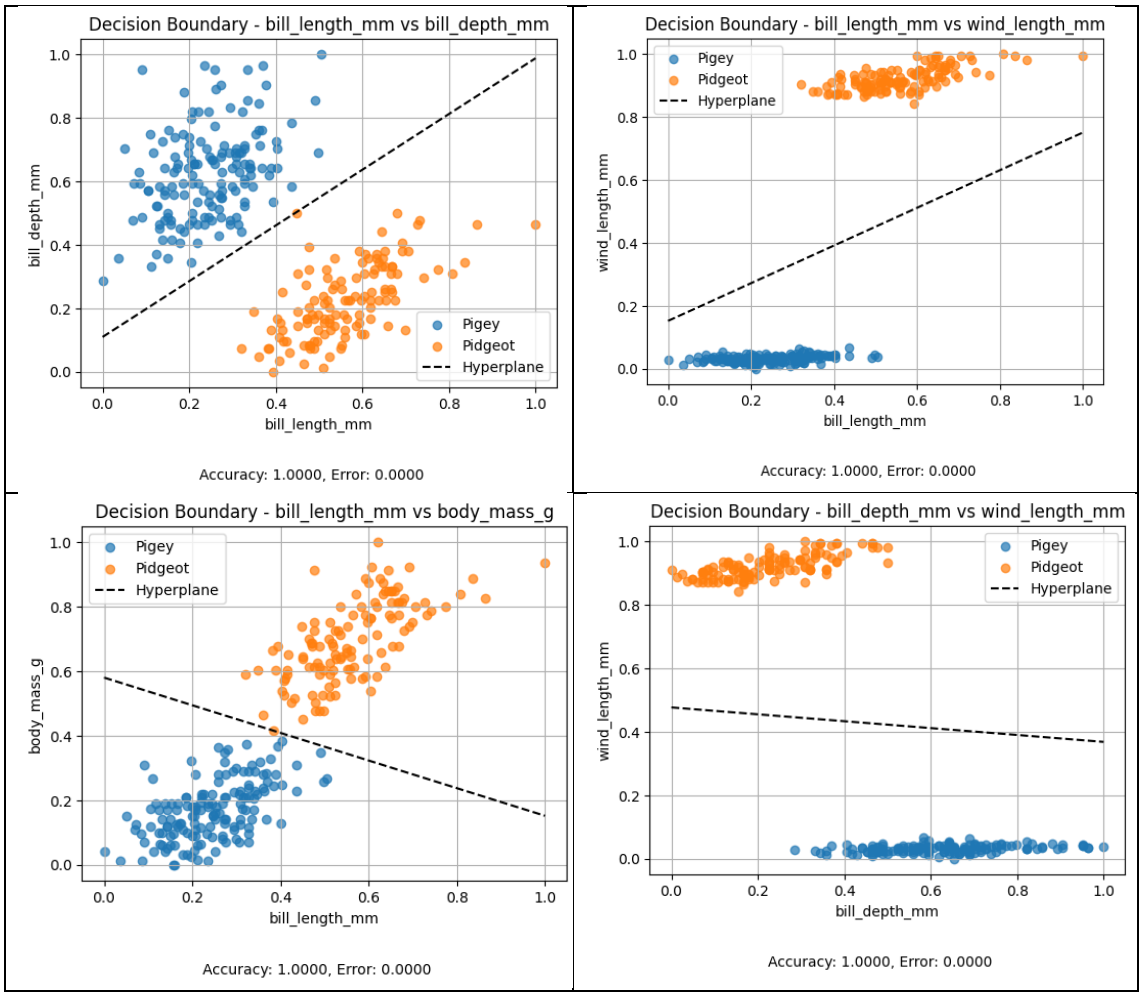


Table 3. Of Diagrams of Decision Boundary for Pigey vs Pidgeotto with 2 features - Selfmade

A.1.2 Pigey vs Pidgeot



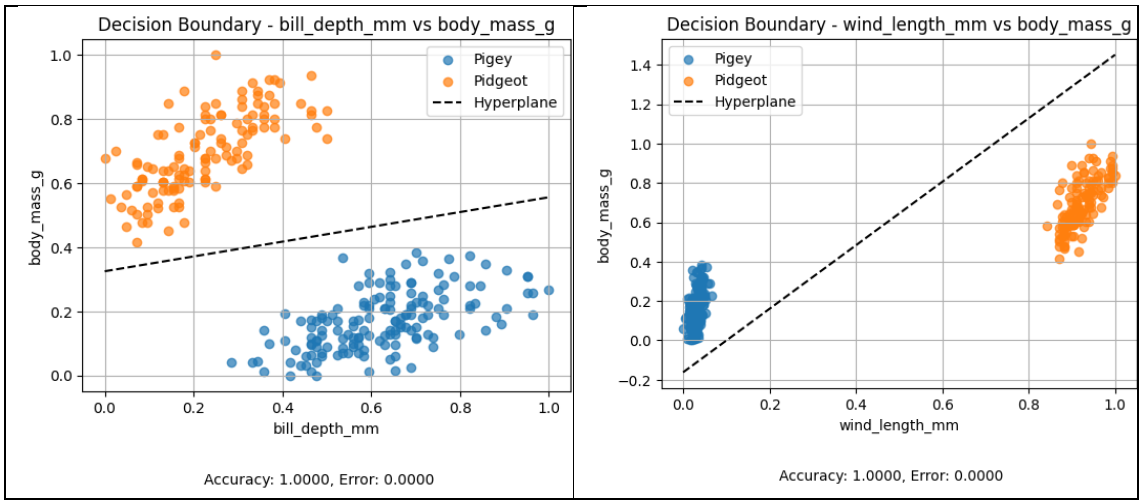
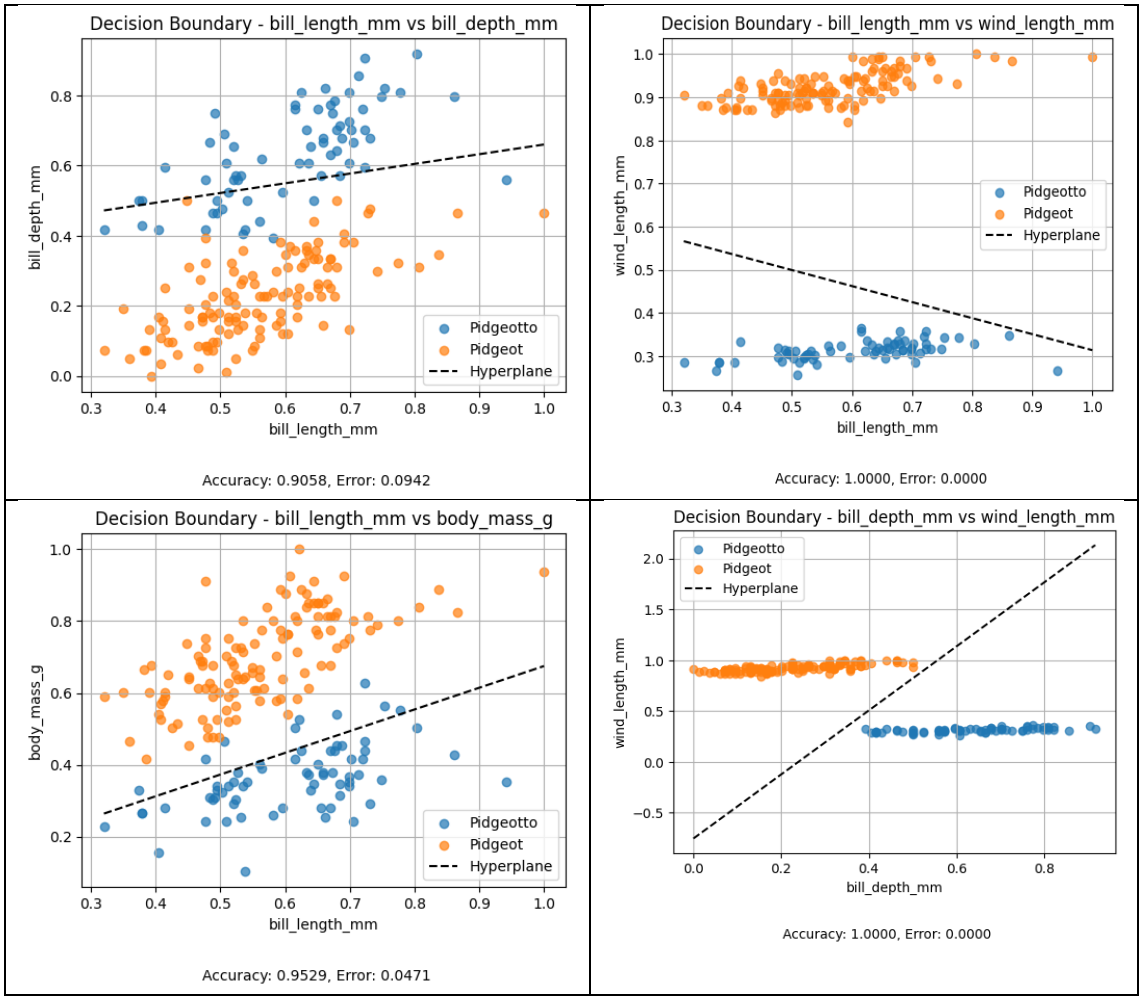


Table 4 Of Diagrams of Decision Boundary for Pigey vs Pidgeot with 2 features - Selfmade

A.1.3 Pigeotto vs Pidgeot



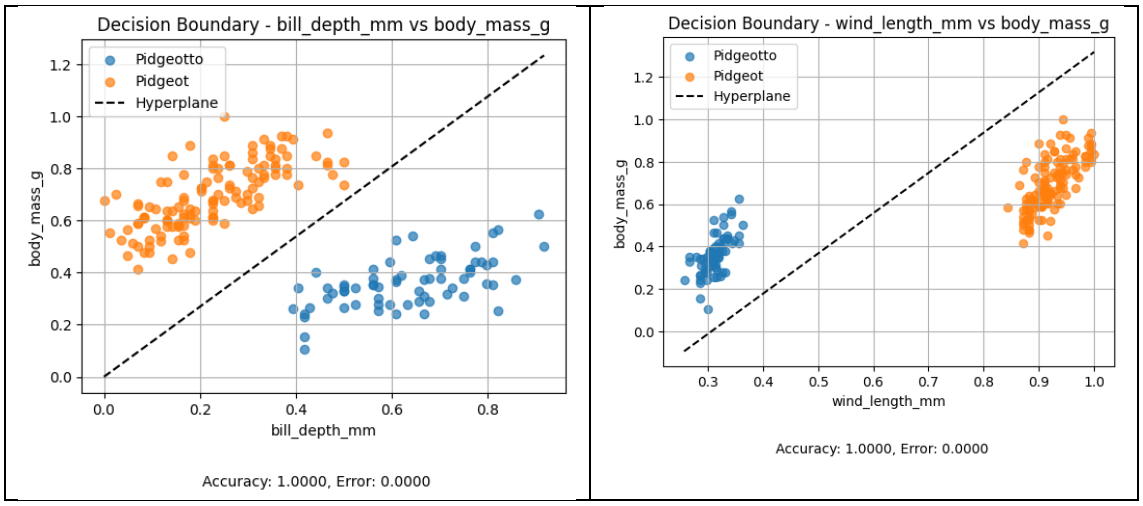


Table 5. Of Diagrams of Decision Boundary for Pigeotto vs Pidgeot with 2 features - Selfmade

A.2 Graphs with 3 Features

Description: These 3D graphs show how combinations of three features separate the two Pokemon types.

A.2.1 Pigey vs Pidgeotto

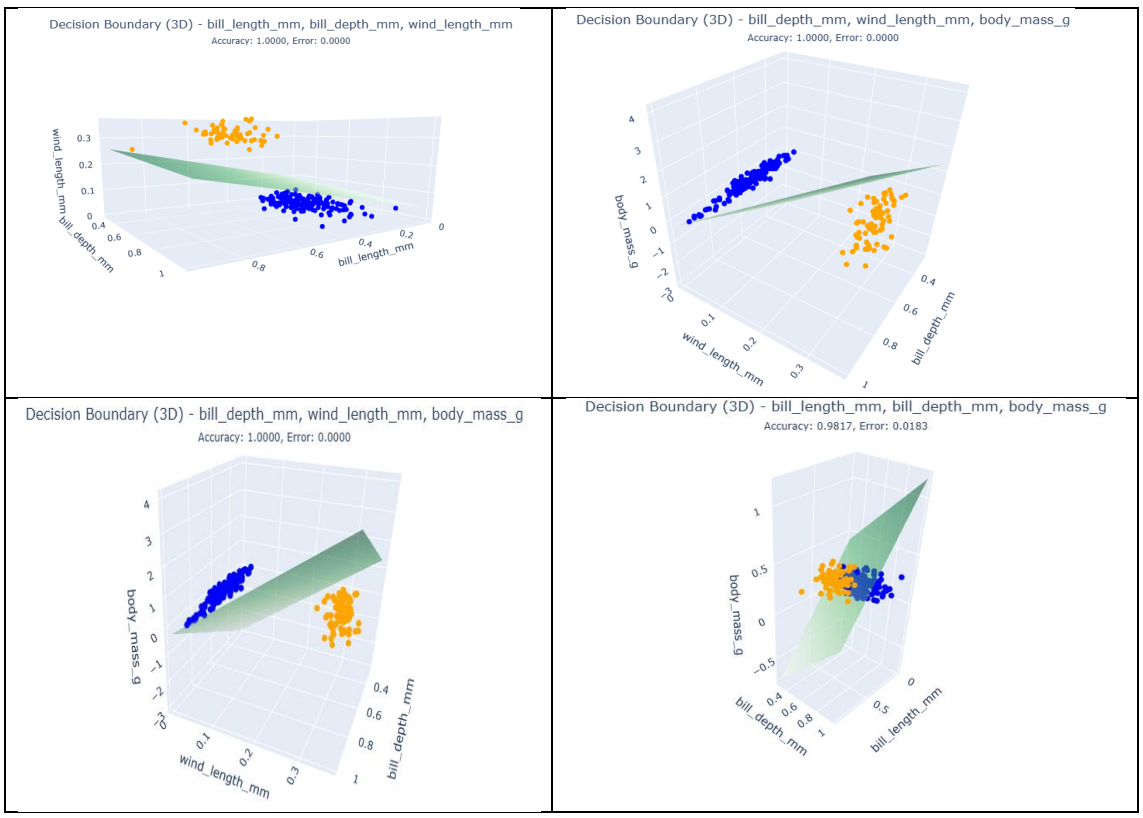


Table 6. Of Diagrams of Decision Boundary for Pigey vs Pidgeotto with 3 features - Selfmade

A.2.2 Pigey vs Pidgeot

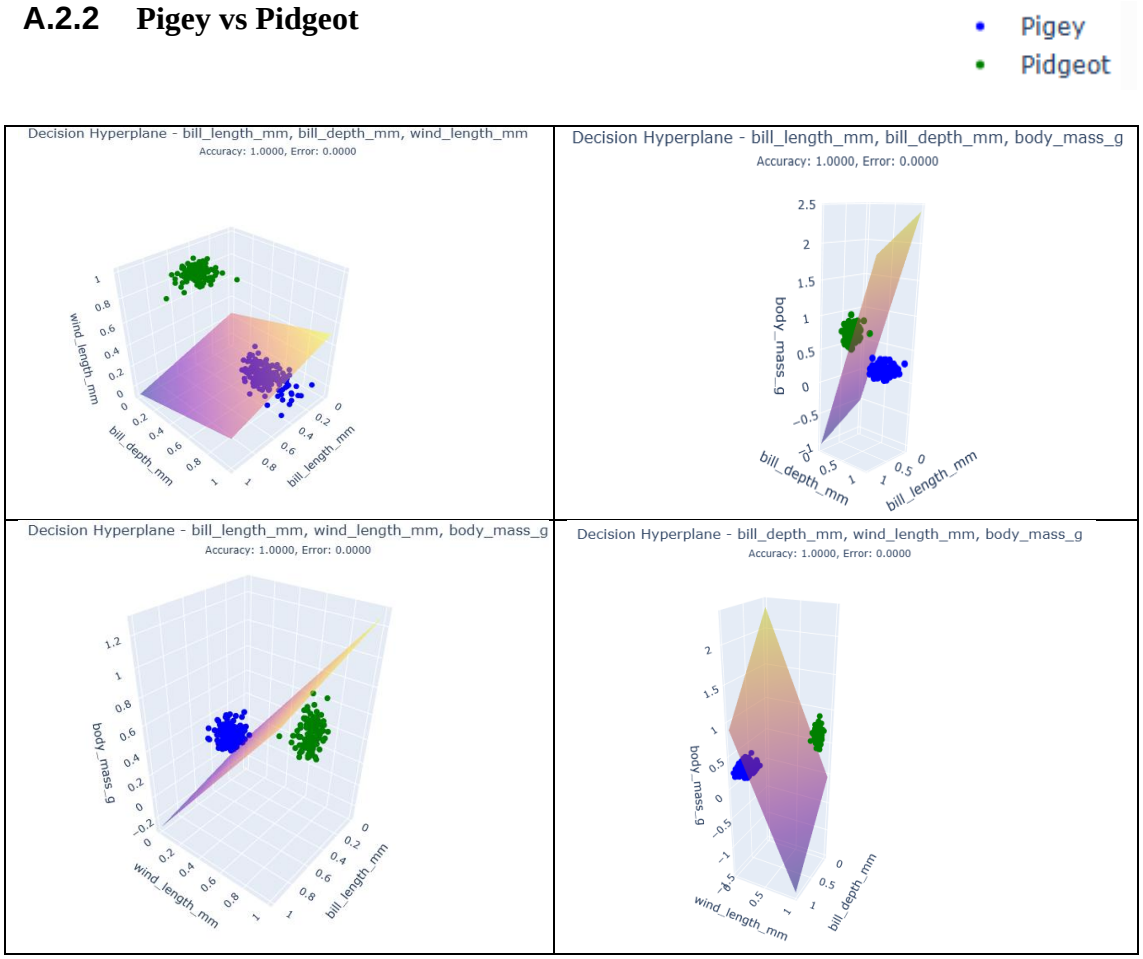
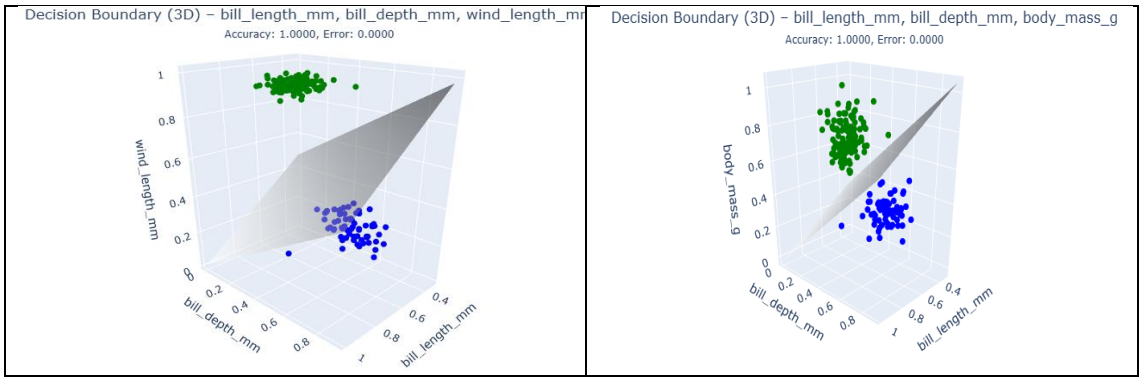


Table 7. Of Diagrams of Decision Boundary for Pigey vs Pidgeot with 3 features - Selfmade

A.2.3 Pigeotto vs Pidgeot



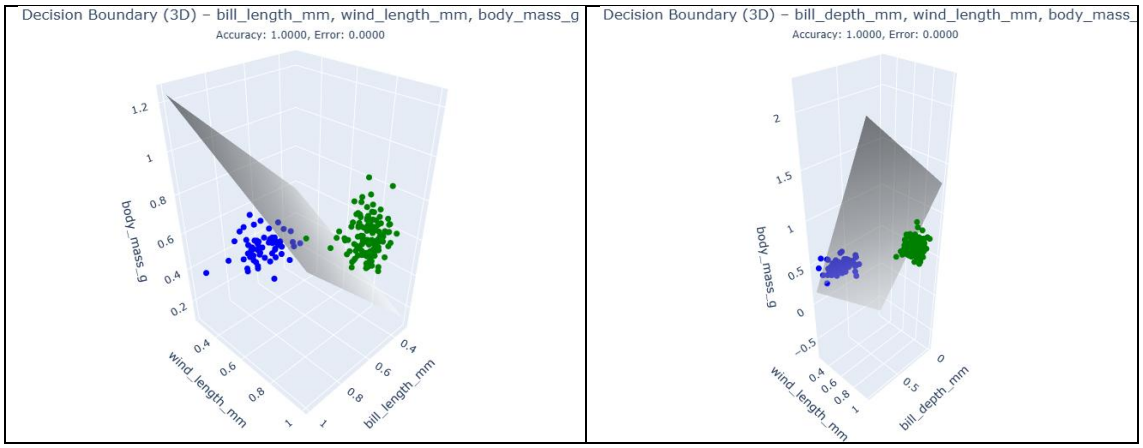


Table 8. Of Diagrams of Decision Boundary for Pigeotto vs Pidgeot with 3 features- Selfmade

A.3 Graphs with 4 Features

Description: These graphs depict the relationships involving four features.

A.3.1 Pigey vs Pidgeotto

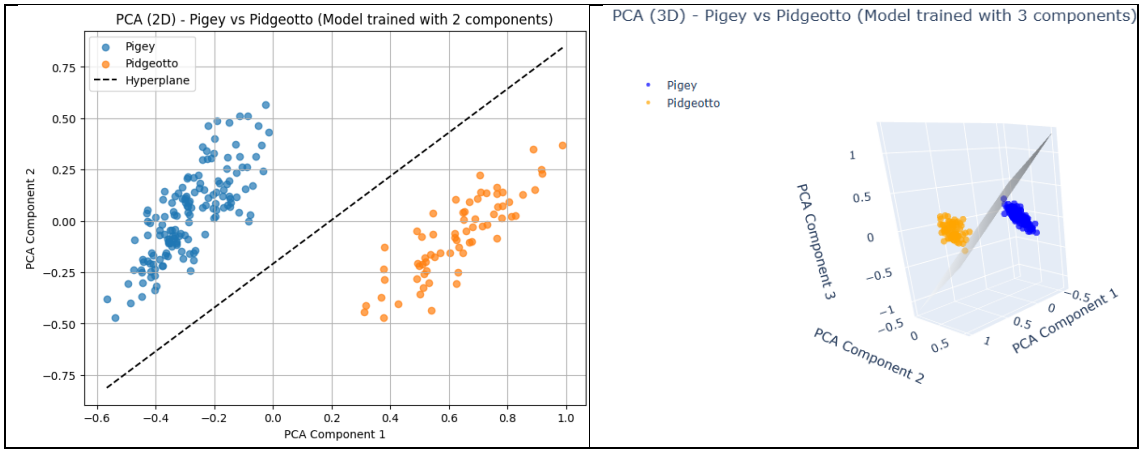


Table 9. Of Diagrams of Decision Boundary for Pigey vs Pidgeotto after PCA with 4 features- Selfmade

A.3.2 Pigey vs Pidgeot

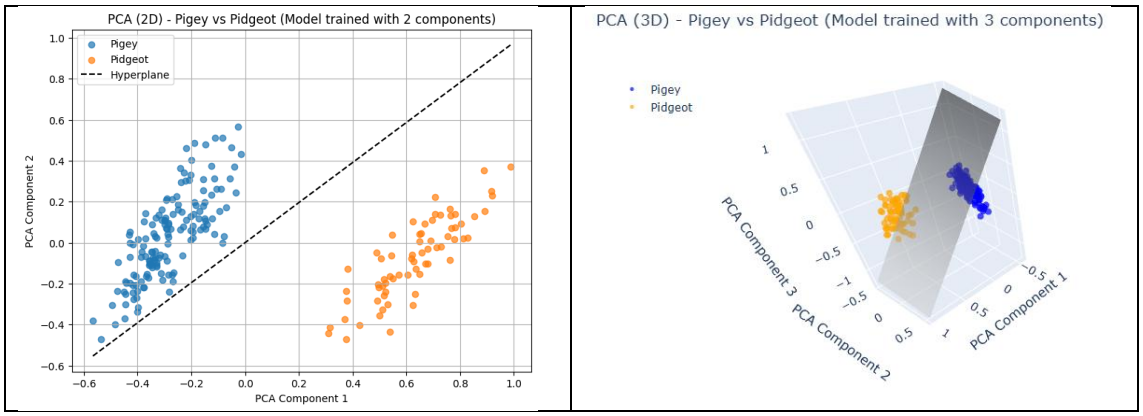


Table 10. Of Diagrams of Decision Boundary for Pigey vs Pidgeot after PCA with 4 features- Selfmade

A.3.3 Pigeotto vs Pidgeot

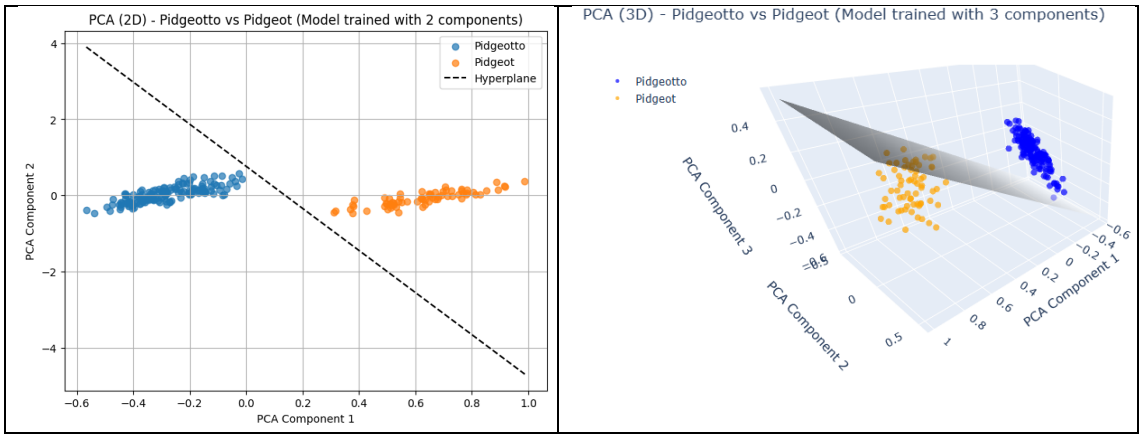


Table 11. Of Diagrams of Decision Boundary for Pigeot vs Pidgeotto after PCA with 4 features- Selfmade

Annex B - Accuracy and Hyperplane Summary

This annex presents a detailed summary of the performance metrics obtained during the training phase for each feature combination. For every tested pair combining the features, we include the accuracy, error rate, and the equation of the decision boundary (hyperplane) generated by the perceptron model.

A.4 2 Features

A.4.1 Pigey vs Pidgeotto

Features	Accuracy	Error	Hyperplane equation
bill_length_mm & bill_depth_mm	0.9361	0.0639	1.7187*x1 + -0.8507*x2 + -0.1000 = 0
bill_length_mm & wind_length_mm	1.0000	0.0000	-0.1361*x1 + 1.0472*x2 + -0.1000 = 0
bill_length_mm & body_mass_g	0.8447	0.1553	1.3695*x1 + 0.3175*x2 + -0.5000 = 0
bill_depth_mm & wind_length_mm	1.0000	0.0000	-0.5357*x1 + 1.2354*x2 + 0.1000 = 0
bill_depth_mm & body_mass_g	0.4384	0.5616	-0.1729*x1 + 1.1575*x2 + -0.0000 = 0
wind_length_mm & body_mass_g	1.0000	0.0000	0.2888*x1 + 0.1869*x2 + -0.1000 = 0

Table 12-Pigey vs Pidgeotto 2 Features -Selfmade

A.4.2 Pigey vs Pidgeot

Features	Accuracy	Error	Hyperplane equation
bill_length_mm & bill_depth_mm	1.0000	0.0000	0.8064*x1 + -0.9169*x2 + 0.1000 = 0
bill_length_mm & wind_length_mm	1.0000	0.0000	-0.3954*x1 + 0.6596*x2 + -0.1000 = 0
bill_length_mm & body_mass_g	1.0000	0.0000	0.5174*x1 + 1.2060*x2 + -0.7000 = 0
bill_depth_mm & wind_length_mm	1.0000	0.0000	0.0228*x1 + 0.2097*x2 + -0.1000 = 0

bill_depth_mm & body_mass_g	1.0000	0.0000	-0.2133*x1 + 0.9230*x2 + -0.3000 = 0
wind_length_mm & body_mass_g	1.0000	0.0000	0.9940*x1 + -0.6160*x2 + -0.1000 = 0

Table 13-Pigey vs Pidgeot 2 Features -Selfmade

A.4.3 Pidgeot vs Pidgeotto

Features	Accuracy	Error	Hyperplane equation
bill_length_mm & bill_depth_mm	0.9058	0.0942	0.2888*x1 + -1.0443*x2 + 0.4000 = 0
bill_length_mm & wind_length_mm	1.0000	0.0000	0.1627*x1 + 0.4379*x2 + -0.3000 = 0
bill_length_mm & body_mass_g	0.9529	0.0471	-0.8584*x1 + 1.4199*x2 + -0.1000 = 0
bill_depth_mm & wind_length_mm	1.0000	0.0000	-0.8357*x1 + 0.2654*x2 + 0.2000 = 0
bill_depth_mm & body_mass_g	1.0000	0.0000	-0.3728*x1 + 0.2767*x2 + 0.0000 = 0
wind_length_mm & body_mass_g	1.0000	0.0000	0.6515*x1 + -0.3425*x2 + -0.2000 = 0

Table 14-Pigeot vs Pidgeotto 2 Features -Selfmade

A.5 3 Features

A.5.1 Pigey vs Pidgeotto

Features	Accuracy	Error	Hyperplane equation
bill_length_mm & bill_depth_mm & wind_length_mm	1.0000	0.0000	-0.1608*x1 + 0.0574*x2 + 0.9135*x3 + -0.1000 = 0
bill_length_mm & bill_depth_mm & body_mass_g	0.9817	0.0183	0.9650*x1 + -0.7907*x2 + 0.7694*x3 + -0.1000 = 0
bill_length_mm & wind_length_mm & body_mass_g	1.0000	0.0000	0.5519*x1 + 0.4918*x2 + -0.0585*x3 + -0.3000 = 0
bill_depth_mm & wind_length_mm & body_mass_g	1.0000	0.0000	0..1892*x1 + 0.5821*x2 + -0.0497*x3 + -0.2000 = 0

Table 15-Pigey vs Pidgeotto 3 Features -Selfmade

A.5.2 Pigey vs Pidgeot

Features	Accuracy	Error	Hyperplane equation
bill_length_mm & bill_depth_mm & wind_length_mm	1.0000	0.0000	0.1319*x1 + -0.1027*x2 + 0.4000*x3 + -0.1000 = 0
bill_length_mm & bill_depth_mm & body_mass_g	1.0000	0.0000	0.0979*x1 + -0.5740*x2 + 0.1968*x3 + 0.1000 = 0
bill_length_mm & wind_length_mm & body_mass_g	1.0000	0.0000	-0.1278*x1 + 0.5806*x2 + -0.4589*x3 + 0.0000 = 0
bill_depth_mm & wind_length_mm & body_mass_g	1.0000	0.0000	0.3047*x1 + 0.5103*x2 + 0.2191*x3 + -0.5000 = 0

Table 16-Pigey vs Pidgeot 3 Features -Selfmade

A.5.3 Pidgeot vs Pidgeotto

Features	Accuracy	Error	Hyperplane equation
bill_length_mm & bill_depth_mm & wind_length_mm	1.0000	0.0000	0.2951*x1 + -0.2372*x2 + 0.4325*x3 + -0.3000 = 0
bill_length_mm & bill_depth_mm & body_mass_g	1.0000	0.0000	0.3149*x1 + -0.4933*x2 + 0.6241*x3 + -0.3000 = 0
bill_length_mm & wind_length_mm & body_mass_g	1.0000	0.0000	-0.4845*x1 + 0.2871*x2 + 0.4958*x3 + -0.2000 = 0
bill_depth_mm & wind_length_mm & body_mass_g	1.0000	0.0000	0.2677*x1 + 0.7740*x2 + -0.2851*x3 + -0.4000 = 0

Table 17-Pigeot vs Pidgeotto 3 Features -Selfmade

A.6 4 Features

A.6.1 Pigey vs Pidgeotto

Features	Accuracy	Error	Hyperplane equation
bill_length_mm & bill_depth_mm & wind_length_mm & body_mass_g	1.0000	0.0000	2D: 0.5100*x1 + -0.4786*x2 + -0.1000 = 0 3D: 0.8192*x1 + -0.6550*x2 + 0.7989*x3 + -0.2000 = 0

Table 18-Pigey vs Pidgeotto 4 Features -Selfmade

A.6.2 Pigey vs Pidgeot

Features	Accuracy	Error	Hyperplane equation
bill_length_mm & bill_depth_mm & wind_length_mm & body_mass_g	1.0000	0.0000	2D: 0.2924*x1 + -0.2992*x2 + -0.0000 = 0 3D: 1.0367*x1 + 0.0348*x2 + 0.6522*x3 + -0.2000 = 0

Table 19-Pigey vs Pidgeot 4 Features -Selfmade

A.6.3 Pidgeot vs Pidgeotto

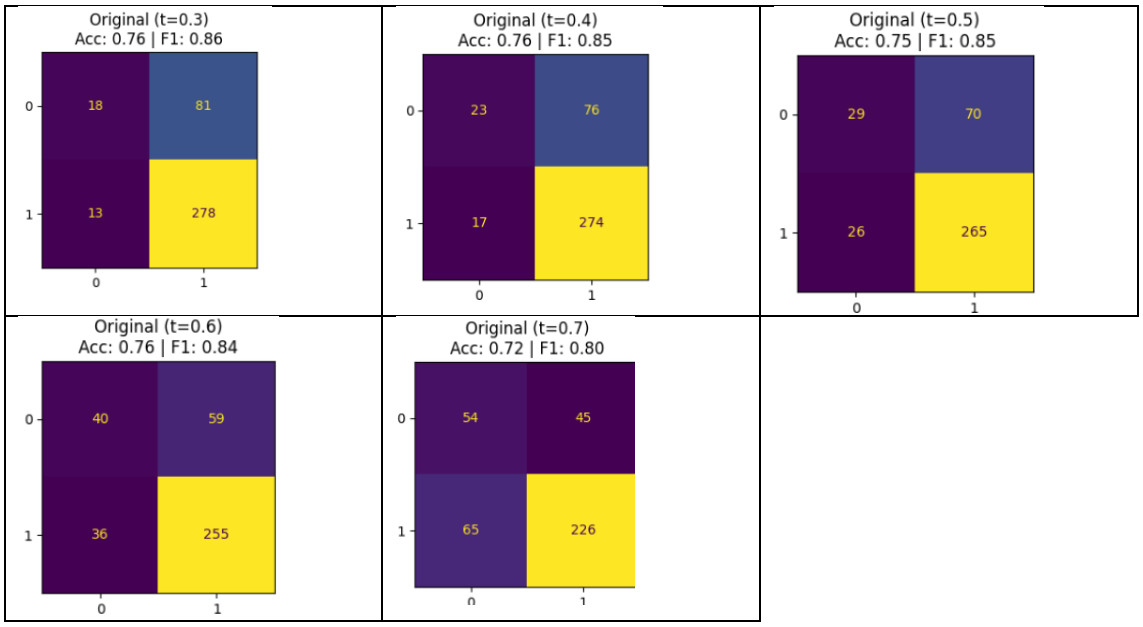
Features	Accuracy	Error	Hyperplane equation
bill_length_mm & bill_depth_mm & wind_length_mm & body_mass_g	1.0000	0.0000	2D: 0.7240*x1 + 0.1309*x2 + -0.1000 = 0 3D: 0.4258*x1 + 0.0150*x2 + -0.5905*x3 + -0.1000 = 0

Table 20-Pidgeot vs Pidgeotto 4 Features -Selfmade

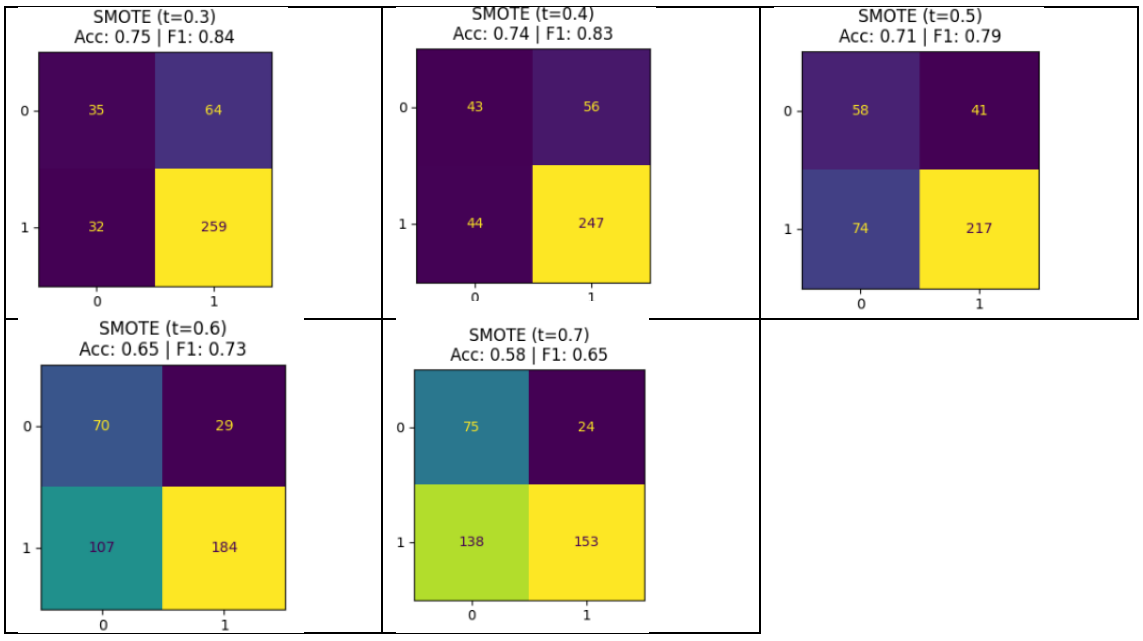
Annex C – Confusion Matrices

In this annex, you will find all the confusion matrices for the three techniques used for data balancing. The first one corresponds to the original dataset, where no balancing technique was applied. The second one presents the results using SMOTE, which generates synthetic examples for the minority class. The third one displays the outcomes obtained with RUS, where a portion of the majority class is removed to balance the class distribution. Each confusion matrix is made for a specific threshold.

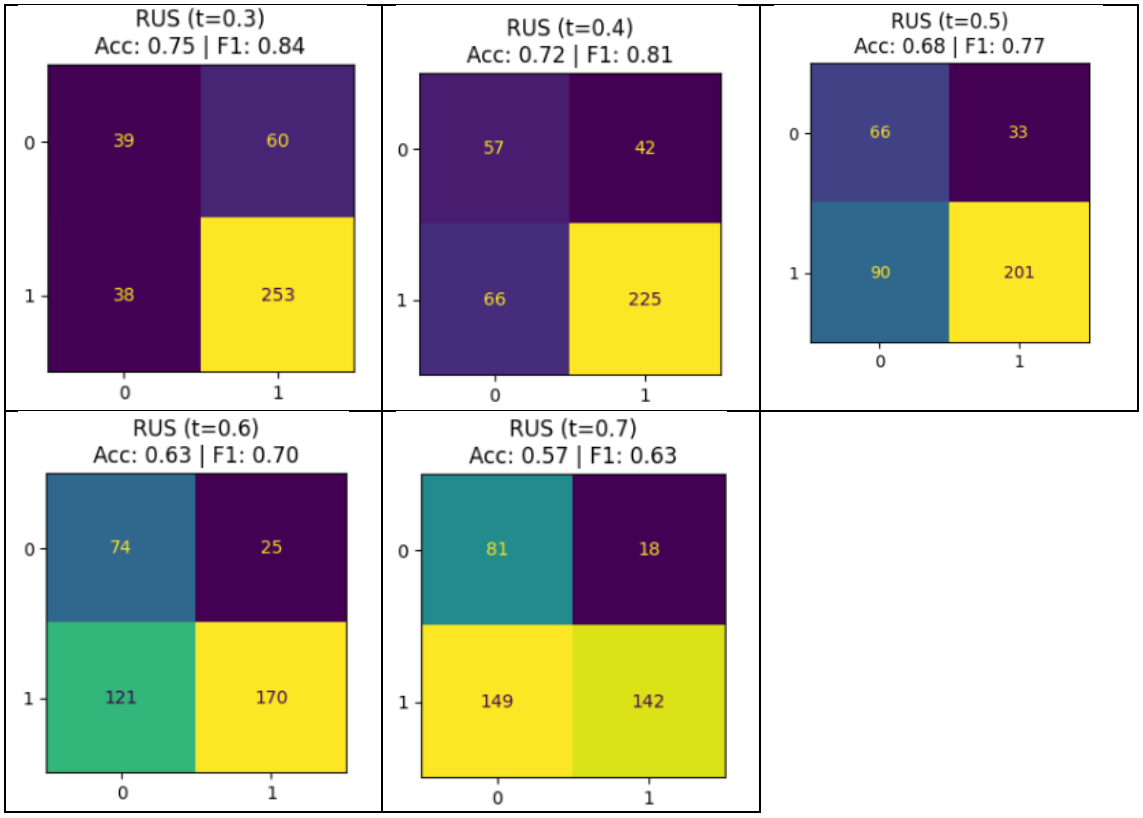
A.7 Original



A.8 SMOTE



A.9 RUS


RUS (t=0.3)
Acc: 0.75 | F1: 0.84
39
60
38
253
RUS (t=0.4)
Acc: 0.72 | F1: 0.81
57
42
66
225
RUS (t=0.5)
Acc: 0.68 | F1: 0.77
66
33
90
201
RUS (t=0.6)
Acc: 0.63 | F1: 0.70
74
25
121
170
RUS (t=0.7)
Acc: 0.57 | F1: 0.63
81
18
149
142