Machine Learning

Carolina Tilley: 99899 João Lopes: 99973

October 2024

1 Project Introduction

For our Machine Learning assignment, we were given a project to understand the various applications and methodologies associated with the field. This project was divided into two parts: regression and classification. From this project, we were able to develop a methodology and understand various machine learning models, concepts, and the pipeline for training, testing, and implementing different models.

The regression component of the project consisted of two distinct problems. In the first one, we estimated the parameters β in a linear regression model to describe the relationship between independent variables and a dependent variable, incorporating outlier removal and generating predictions for the test set. In the second problem, we used an ARX model to predict U_{test} based on U_{train} and Y_{train}. The ARX approach combines both input and output values to construct the matrix X and is sensitive to the time dependency inherent in the data.

For the classification part, we addressed two image-based problems. In the first problem, we developed a binary image classifier to distinguish between images with and without craters. Each image is a 48x48 matrix, and the labels are 0 (without crater) and 1 (with crater). The second problem involved image segmentation, classifying each pixel in the images to create masks identifying crater areas. We were given two data formats: `x_train_a` and `y_train_a` (based on patches) and `x_train_b` and `y_train_b` (based on full images).

Part 1 - Regression with Synthetic Data

1.1 First Problem - Multiple Linear Regression with Outliers

To address this first problem, we began by testing three different outlier removal methods: the Quartile method, the Local Outlier Factor (LOF) method, and the RANSAC method.

1.1.1 Outlier removal

To address the outlier removal problem, we began by investigating the Quartil method as described in the book recommended in the lab guide. This method relies on statistical thresholds, specifically using the interquartile range (IQR) to identify outliers. Therefore, data points that are below $Q_1 - 1.5 \times \text{IQR}$ or above $Q_3 + 1.5 \times \text{IQR}$ are considered outliers, where Q_1 and Q_3 represent the first and third quartiles, respectively. However, this method wasn't the most successful due to its statistical approach, whereas the outliers in our dataset are caused by human error.

Next, we tried to use the LOF method. However, this approach did not perform well, as the outliers in our dataset are once again caused by human error. LOF is more useful when talking about outliers that are situated in sparse regions of the feature space, which was not the case.

Finally, we applied the RANSAC method, which was the most successful of the three approaches. RANSAC is a robust method that works well in cases where outliers are not evenly distributed or follow a specific pattern, as was the case in our study. It works by randomly selecting a subset of points and using them to fit the model regressor chosen as a parameter. Once the model is fitted, RANSAC tests the fitted model against all other data points. The data points that fit the model well are classified as inliers, while the rest are considered outliers. This process is repeated multiple times, each time with a different random subset. We used the Linear Regression model as the estimator within RANSAC to fit the data.

After the process removed what it considers to be outliers, we are left with a mask for our original array that only contains the inliers. This is the array that will be further processed by the different regression models.

Line ID	Ransak	LOF	Quartil
Outliers removed	48	50	45
SSE for linear regression(1.1.2)	1.1383	3126.2271	137.2250

Tabela 1: Outlies Removel

1.1.2 Regression methods

After removing the outliers, we are left with approximately 150 data points for training. We then applied several regression methods on the remaining inlier data, including Linear Regression, Ridge and Lasso.

1. **Linear Regression:** firstly, we tested with the Linear Regression approach. It aims to find the coefficients β that satisfy: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$ or $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$.

   The performance of our model varies depending on the outlier removal method used. The results are presented in Table 1.

2. **Lasso Regression:**Lasso regression is a type of linear regression that uses regularization to prevent overfitting. It achieves this by adding an L1 penalty to the cost function, which helps in reducing the impact of less significant features. This feature selection capability makes Lasso particularly effective when dealing with a high number of features, as it can automatically eliminate some of them.

   The cost function for Lasso regression is given by: Cost Function $= \frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$

   In this equation, λ is a hyperparameter that controls the strength of the regularization. A larger λ forces more coefficients to shrink to zero, effectively performing feature selection, while a smaller λ allows the model to behave more like standard linear regression. The choice of λ is crucial for the Lasso model's performance and is typically determined through cross-validation.

3. **Ridge Regression:**Ridge regression is similar to Lasso, but instead of using an L1 penalty, it uses an L2 penalty. Ridge regression is used to prevent overfitting by shrinking the coefficients toward zero. Unlike Lasso, Ridge does not perform feature selection, as it does not shrink coefficients to zero. It is most effective on datasets with lower dimensionality and also includes the hyperparameter λ, which is discussed in the cross-validation section. The cost function for Ridge regression is given by: Cost Function $= \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2$

1.1.3 Cross Validation

In order to determine the optimal values for Lasso and Ridge, we used the cross-validated versions of these models: LassoCV and RidgeCV. Both models were implemented using the available code from the 'sklearn' library. We specified the parameters 'folds=5' and a range of $\lambda \in [0.001, 0.01, 0.1, 1, 10, 100]$.

From this setup, we obtained the best parameters for both Ridge and Lasso, as well as the corresponding best R^2 and SSE values, which allowed us to compare the models effectively.

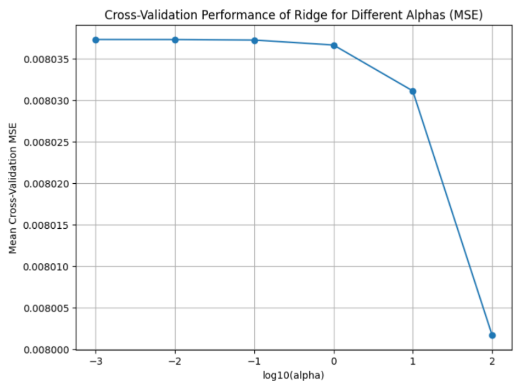

Figura 1: λ variation for Ridge CV

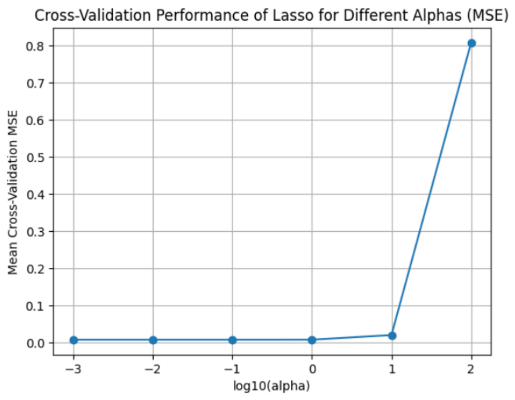

Figura 2: λvariation for Lasso CV

For linear regression, we implemented cross-validation based on k-fold, to evaluate our model's performance

Although we should have evaluate all models using k-fold cross-validation after finding the optimal values for λ in Ridge and Lasso, we ultimately applied this validation only to the linear regression model, as previously mentioned.

In k-fold cross-validation, the dataset is split into several "folds." The model is trained on all but one fold and tested on the remaining one. This process is repeated so that each fold is used as a test set once. This allows us to assess if our model is working properly and ensures that it is not overfitting.

CV	Linear SSE	Ridge SSE	Lasso SSE
3	0.435711	0.427853	0.421671
5	0.251897	0.250434	0.249988
15	0.082657	0.082354	0.082537

Tabela 2: Comparison of SSE for different models across CV folds

1.2 80 20 Validation

In hindsight, we recognize that splitting the dataset into training and testing sets should have been our initial approach for all models. Unfortunately, we only implemented this division in the second part of the project. Revisiting this first part with this understanding, we have now divided the dataset accordingly into training and testing sets. This adjustment enables us to properly validate our models and select the optimal one by combining insights from both the test set evaluation and cross-validation metrics.

1.2.1 Final Results

From the previous sections, we can infer the following: for this part of the project, the best outlier removal algorithm was, by far, RANSAC 1.1.3, which aligns with our research and findings. In the model analysis, we observe that the performances are very similar 1.1.1. Since we are using **SSE**, it's expected that as CV increases, the SSE decreases. Because we didn't include this approach in our initial assessment of the project, we chose Linear Regression as our model, as it performed best on the full dataset, as we shown in table 3. However, with more knowledge and a further evaluation, we can see that LASSO was indeed slightly better at predicting the data.

		Quartil	LOF	RANSAC
Linear Regression	SSE	137.22507	3126.22713	1.13831
	R^2	0.63017	0.06385	0.99570
Lasso Regression	SSE	137.30126	3339.44752	1.15372
	R^2	0.62997	0.0	0.99564
Ridge Regression	SSE	137.22671	3126.32800	1.13905
	R^2	0.63017	0.06382	0.99570

Tabela 3: Comparison of regression methods across different outlier removal techniques.

1.3 Second Problem - The ARX model

To start tackling this problem, we began by examining u_{train} and y_{train}, by plotting these values to observe their behavior.

The **Autoregressive with Exogenous Variables (ARX) model** is defined by specific parameters, which determine how the model makes predictions and which data points from past observations it can use.

$$y(k) + a_1 y(k-1) + \cdots + a_n y(k-n) = b_0 u(k-d) + \cdots + b_m u(k-d-m) + e(k) \tag{1}$$

$$y(k) = \varphi(k)^T \theta + e(k) \tag{2}$$

$$\varphi(k) = [y(k-1), \ldots, y(k-n), u(k-d), \ldots, u(k-d-m)]^T \tag{3}$$

$$\theta = [-a_1, \ldots, -a_n, b_0, \ldots, b_m]^T \tag{4}$$

1. m **- Autoregressive Order**: Defines how many past observations of the target variable y are used to predict the current value y_t.

2. n **- Exogenous Input Order**: Indicates how many past values of external variables x influence y_t.

3. d **- Differencing Order**: Relevant for extending ARX to handle non-stationary data (ARMAX model).

To apply this model, we used three for loops to test the optimal values for *n, m* and *d*, which define the structure of the ARX model. For each combination of *n, m* and *d* values, we generated an X_{train} matrix, using $\varphi(k)$, then used it within the models discussed in Section 1.1.2 to predict a validation set. Once the loops concluded, we were left with the best-performing X_{train} matrix. Using this matrix we can infer the best model.

1.3.1 Models

As previously mentioned, the models we used were those discussed in Section (1.1.2). These models include:

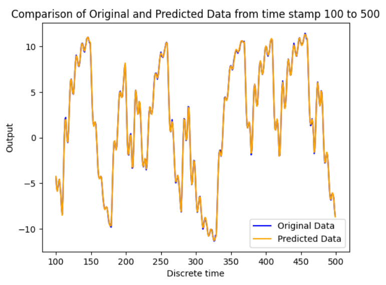

Figura 3: Ridge Prediction

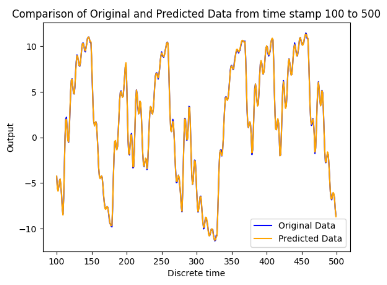

Figura 4: Linear Prediction

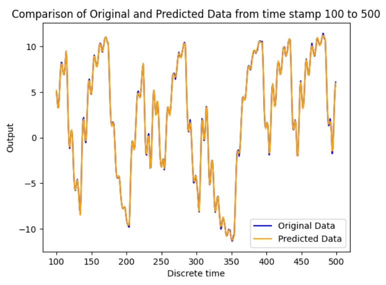

Figura 5: Lasso Prediction

To predict y_{test} using the trained ARX model, we iteratively generate predictions for each time step based on the model's best parameters, n, m, and d. Starting with the last n values of y_{train} as initial lags, we construct a feature vector for each time step in u_{test} using these previous y values and the last m values of u_{test}, delayed by d. This feature vector is then passed to the trained model to predict $y(k)$, which we store and use in subsequent predictions, allowing the model to autoregressively generate y_{test} based on both past predictions and current inputs from u_{test}.

In the assignment, it was specified that *n, m* and *d* should be kept below 10, although we had the option to select higher values if needed. In our initial trials, the best values were *n=9*, *m=9* and *d=6*. Later, we found that increasing$n$ and m yielded improved results, which be discussed further in the results section.

1.3.2 Results

After observing what parameters are the best we computed the SSE for the first 500 points of a validation set

	lin	**lasso**	**ridge**
SSE	18.9733	31.5026	18.9952

Tabela 4: Sum of Squared Errors (SSE) for Different Models

Looking at the values, we see that Ridge and Linear Regression are the models that present the least error. We chose to deliver Ridge because it is very similar to Linear Regression but has added complexity, which we thought would help improve the final results.

For the parameters, we chose $n = 49$, $m = 49$, and $d = 6$ because this combination yielded the best results. While higher values for n and m (such as $n = 49$) may fit the training data better and give lower MSE/SSE on the training set, it can lead to overfitting, where the model captures noise or irrelevant patterns in the training data but performs poorly on new data.

When we delivered the project, we were unsure if the array we provided was optimal because we had two implementations for creating the final y_{test} array. After receiving the scores, we got an SSE of 2000 for the first implementation, and, after discussing with the Professor, we obtained 800 using the second approach. As these values were quite high, we restarted the project and discovered that we were constructing the X matrix incorrectly. For predicting the model we din´t filter the models that weren't stable and we also were using the wrong lag values for y. Fixing this resulted in a much better y_{test}, with an SSE close to 1.

Part 2 - Image Analysis

1.4 First Problem - Image classification

To explore different approaches for developing the best model, we experimented with various combinations of training data and data augmentation, with and without balanced datasets.

Firstly, as a strategy to train our model and be able to test it with new data, we divided our training data into 80% training data and 20% testing data.

1.4.1 Data augmentation

Then, to increase the number of training images, we applied data augmentation techniques. These included horizontal and vertical flipping, along with other transformations such as rotations, zooming, translations and brightness levels.

However, the dataset is imbalanced with a big difference in the number of images with and without craters, as noted in the lab guide. Therefore, in order to avoid a biased model that overly predicts the majority class, we implemented balanced data augmentation to increase the number of training examples for the minority class, which is images without craters. To ensure a similar number of images in both classes, we calculated the ratio between the two and applied data augmentation proportionally to the minority class. The data augmentation algorithm used involved applying a series of transformations to the original dataset, including horizontal and vertical flips, rotation by up to 25%, zoom adjustments of 10%, translations shifting images vertically and horizontally by 10%, and brightness adjustments by 10%. To balance the dataset, we augmented the classes by adding a specific ratio so that the classes would be augmented accordingly, and become balanced.

	Label 1	**Label 2**	**Total**
Balanced	6978	7042	14020
Imbalanced	12439	7042	19481

Tabela 5: Comparison of number if images for Balance and inbalanced dataset.

As we can see the Balanced approach yields a dataset a lot more balanced at the cost of having less images. Next, we combined the original dataset with the augmented images, resulting in the final dataset, which was then used to train the model. This was done with both the balanced and unbalanced datasets for comparison. It is also useful to mention that later in 1.4.2 we combined an extra dataset, which was also submitted to data augmentation.

1.4.2 Models

The first model we implemented was the **K-Nearest Neighbors (KNN) algorithm**. This algorithm operates by selecting a specific number of nearest neighbors, k, and classifying a data point based on the majority label of its k closest neighbors. If k is too small, the model may be sensitive to noise, leading to overfitting; if k is too large, it may over smooth the data, capturing less local detail and leading to underfitting.

We then experimented with the **Convolutional Neural Network (CNN) method**. We tested with 3 convolution layers, each followed by a max-pooling layer. This allowed the model to learn features like edges and shapes, crucial for crater detection. Then, in order to avoid overfitting, we applied dropout layers at a rate of 50%. Finally, after flattening the feature maps, a dense layer with 64 units was used, followed by a final dense layer with one unit and a sigmoid activation function for binary classification, since our output is expected to be 1 or 0. This method performed well due to its ability to capture hierarchical features in the data.

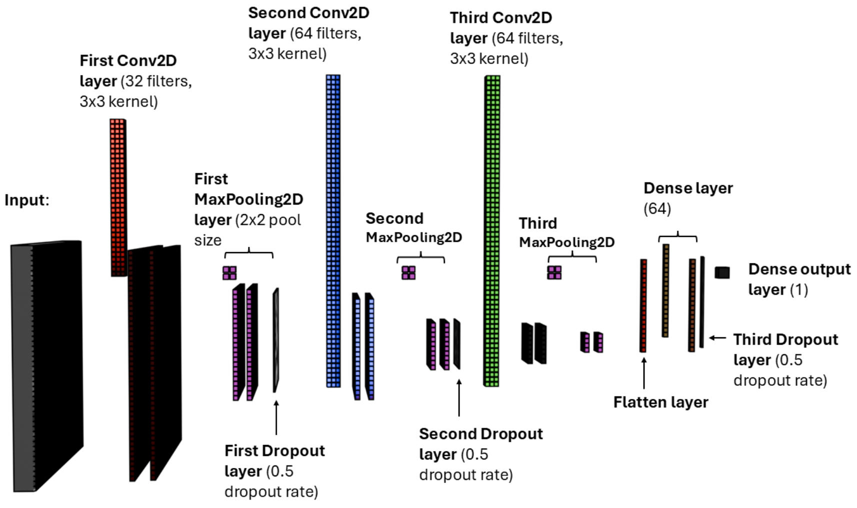
Input:
First Conv2D layer (32 filters, 3x3 kernel)
First MaxPooling2D layer (2x2 pool size
First Dropout layer (0.5 dropout rate)
Second Conv2D layer (64 filters, 3x3 kernel)
Second MaxPooling2D
Second Dropout layer (0.5 dropout rate)
Third Conv2D layer (64 filters, 3x3 kernel)
Third MaxPooling2D
Dense layer (64)
Flatten layer
Dense output layer (1)
Third Dropout layer (0.5 dropout rate)

Figura 6: CNN layers for our implementation

In order to evaluate our model and to prevent overfitting we did crossvalidation. To make sure our CNN was optimal for this data classification we performed k-fold with 5 folds and 80% 20% data split.

	Balanced CV=5	Unbalanced CV=5
F1 Score	89.38	87.55

Tabela 6: Comparison of CV Scores for Balanced and unbalannced datasets.

To evaluate the performance of each method and each combination of training data and data augmentation, with and without balanced datasets, we applied the F1 score (macro). This method treats all classes equally and provides a balanced measure of precision and recall, making it well-suited for imbalanced datasets. This is also the metric that will be used for evaluation. F1 Score $= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Finally, we were provided with an **extra dataset** that did not contain labeled results. Therefore, we used this extra dataset to increase the number of images available for training and testing. However, because the labels were missing, we used the model's predictions to select the images with more than 90% probability of having a crater and those with less than 10% probability of not containing a crater.

1.4.3 Results

After testing both models using an 80% training set and a 20% validation set, we quickly noticed that CNN was significantly better at predicting image labels, as expected. This is clearly illustrated in the following table.

	KNN	CNN
F1	0.2603	0.9004

Tabela 7: F1 score for CNN and KNN

For the CNN testing, we experimented with several combinations, including with/without dropout, with/without balanced data, with/without extra dataset, and others. The best results were achieved using the approach with balanced data augmentation, with dropout, and with the full dataset plus additional data. In the confusion matrices below, we can observe that the CNN model is effective at correctly identifying both classes, with relatively few false positives and false negatives. In contrast, the KNN model performs poorly.

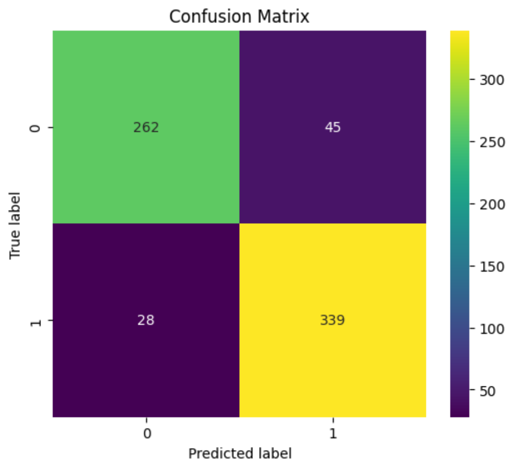

Figura 7: Confusion matrix CNN with 80 20 validation and extra dataset

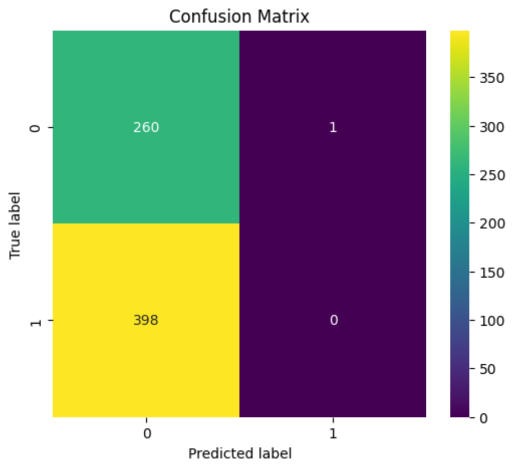

Figura 8: Confusion matrix KNN with 80 20 validation and extra dataset

To estimate the optimal number of epochs for training and assess the impact of the dropout layers, we plotted the training and validation losses across epochs. This allowed us to monitor when the model began to overfit. Around epoch 10, we observed signs of overfitting: while the training loss continued to decrease, the validation loss began to increase. This divergence indicated that the model was absorbing the training data rather than generalizing well to new data.

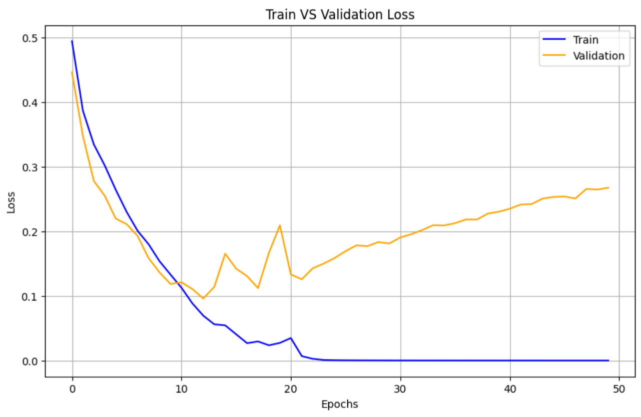

Figura 9: Train VS Validation Loss

1.5 Second Problem - Image segmentation

For this section, as mentioned before, we were given two options for data formats. We chose to use `x_train_b` and `y_train_b` (based on full images).

1.5.1 Data augementation and Class weights

Similar to the classification problem, we performed data augmentation in order to increase the number of training images, as explained before. However, in this problem, we also introduced Gaussion noise transformations as this helps the model generalize better by making it resilient to minor, random pixel fluctuations. Training the model with images with noise, prepares it become less sensitive to similar noise in test data. Additionally, we excluded the vertical flip transformations in order to preserve the spatial orientation typical in crater images.

	N of images before DA	**N of images after DA**	**N Pixel**	
			Label 1	**Label 0**
Dataset	547	3282	2762772	6059244

Tabela 8: Comparison of number of images after and post Data augmentation, with associated label unbalance.

After combining the original dataset with the augmented images, providing the model with a larger set of varied examples, we calculated class weights. This was crucial to address the class imbalance between crater pixels (white) and background pixels (black), with the last one being the majority class. This helped adjust the model's sensitivity to crater pixels during training.

1.5.2 Models

The first model we implemented was the **Random Forest Classifier (RFC)**. This algorithm operates by constructing multiple decision trees. Each tree is trained on a random subset of the data, additionally, at each split within a tree, a random subset of features is considered, reducing correlation among the trees and helping to avoid overfitting. For predicting a majority vote is held in with all the trees. The Random Forest Classifier produces a robust final model, although this approach is effective for preventing overfitting, it can become computationally intensive for larger datasets or when a large number of trees is used.

We then experimented with the **U-Net** architecture, which follows the CNN approach discussed earlier, but utilizes an encoder (contracting path)-decoder (extracting path) structure.These two paths are symmetrical and connected by skip connections to retain spatial information throughout the network. The encoder is responsible for extracting features from the input images through a series of convolutional layers and pooling operations, where each convolutional block has 2 convolutional layers and a max-pooling layer that that reduces the spatial dimensions, effectively down sampling the image. On the other hand, the decoder is responsible for up sampling encoded features back to their original resolution that were lost during the encoding phase and then producing the final output. Between the encoder and decoder lies the bottleneck, a bridge where the encoder transitions into the decoder.

1.5.3 Results

After testing both the RFC and U-Net models using an 80% training set and a 20% validation set, we observed that U-Net performed better compared to RFC, as shown in the table below.

	RFC	U-Net
Balanced Accuracy	0.5502	0.8278

Tabela 9: Balanced Accuracy for RFC and U-Net

We experimented with various Gaussian noise levels during data augmentation and tested different dropout rates within the U-Net model. The optimal values that lead to the best performance were Gaussian noise level of 0.05 and a dropout rate of 0.5. Additionally, we experimented with different threshold values to adjust the classification bias, obtaining the best results by setting predictions greater than 0.5 to 1 and others to 0.

Conclusion

In this project, we explored a range of machine learning techniques across both regression and image analysis tasks.

For the regression problem, we expanded our knowledge in outlier removal methods, which enabled us to improve model accuracy. Additionally, we experimented with regression models such as Linear, Ridge and Lasso and applied cross-validation techniques to optimize model parameters.

For the image analysis task, we first addressed image classification problems in which we experimented with models like K-Nearest Neighbors (KNN) and Convolutional Neural Networks (CNN). Data augmentation techniques, particularly balancing classes, significantly improved model performance. Then , we addressed image segmentation problems where we compared the Random Forest Classifier (RFC) with the U-Net model. Further optimizations with data augmentation and class weights allowed us to achieve optimal segmentation accuracy.

Overall, this project provided valuable insights into selecting and optimizing machine learning models for different tasks, in which we identified robust techniques suited to each problem.