# Machine Learning - Project report

*Group 41: Mariana Mourão, 98473; Hugo Ramos, 96394*

November 8, 2022

## 1 Regression - First problem

Considering a training set with $n$ = 100 samples $\mathcal{T}_r = \left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(100)}, y^{(100)}\right)\right\}$, with $x^{(i)} \in \mathbb{R}^{10}$, $y^{(i)} \in \mathbb{R}$, for $i$ = 1, ..., $n$, it is aimed to train linear regression predictors $\hat{y} = f(x) = [1x^T]\beta$. For the evaluation of the chosen predictor, a test set will be considered, being an independent set of data that wasn't used to train or select the model.

### 1.1 Methodology

The implementation of the various models used throughout this project was made with the help of various public packages, being Python the coding language.

For this task, it was implemented 7 models: simple linear regression, Ridge regression, Lasso regression, linear regression with Lasso feature selection, stepwise feature selection and Polynomial models (using linear regression and Lasso regularization).

Evaluation was made through *Repeated k-fold* cross validation with 10 folds and 3 repeats, meaning that the training set was divided in 10 folds, each of which will be considered the validation set whilst all other folds collectively are used as a training set, being this repeated 3 times. Hence, 30 models will be trained and evaluated, being the overall performance the average of the sum of squared errors (SSE) obtained in each validation fold at each run. In fact, a single run procedure of the k-fold cross validation may result in a noisy estimate of the model performance.

Regression models consists of a set of coefficients $\beta = [a_0, a_1, \ldots, a_{10}]$ fitted to the training data by minimizing a loss function, in this case the SSE, being solved by a linear system of equations(1):

$$SSE = \sum_{n=1}^{n} (y^{(i)} - x^{(i)}\beta)^2 \implies \beta = (X^T X)^{-1} X^T y \tag{1}$$

Ridge regression is an adaptation of linear regression, introducing a regularization term $\lambda$ which penalizes the use of large coefficients, allowing to simplify the model (2):

$$\text{SSE} = \sum_{i=1}^{n} \left(y^{(i)} - x^{(i)}\beta\right)^2 + \lambda \sum_{k=1}^{n} \beta_k^2 \implies \beta = \left(X^T X + \lambda I\right)^{-1} X^T y \tag{2}$$

It was also tried the Lasso regression(3), similar to Ridge regression, however the regularization term $\lambda$ drives the coefficients with less to no importance to zero. This leads to a sparse model which cannot be solved by a linear system of equations.

$$\text{SSE} = \sum_{i=1}^{n} \left(y^{(i)} - x^{(i)}\beta\right)^2 + \lambda \sum_{k=1}^{n} |\beta_k| \tag{3}$$

For both Ridge and Lasso Regression, different $\lambda$ values were picked manually and then iterated until identifying one that achieved highest performance (lowest $SSE$).

Since the lasso regression identifies features with low importance, an algorithm was created to take advantage of this characteristic, performing linear regression only on the features selected by lasso, for a given lambda. A similar approach was taken into consideration, where a model performs stepwise-elimination of the features as long as it improves linear regression model performance. This method uses brute force, so it is not computacional efficient, however it was feasible for the volume of data given and so it was applied (4):

$$\beta = \left(X^T X\right)^{-1} X^T y = \left[\begin{array}{ccccc} a_0 & a_1 & \dots & a_{19} & a_{10} \end{array}\right], \quad \text{where } a_i = 0 \quad \forall i \in \{1, 8, 9\} \tag{4}$$

The features $a_1$, $a_8$ and $a_9$ were eliminated.

Polynomial regression can be considered a linear regression whose features are the power of $x$ (5), allowing to capture non-linear relations and adding complexity which helps to overcome under-fitting. Polynomial regression was tested with a function included in the scikit-learn package which, given the matrix $X$ with the training data, generates a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. Various orders of the polynomial were tested, with $2^{nd}$ order polynomial being the one that achieved the best results. Indeed, more features may result in more overfitting, besides the higher computational burden.

$$f(x) = \beta_0 + \beta_1 x + \cdots + \beta_p x^p \tag{5}$$

For each of these linear models, the data was centered during training and evaluation. Therefore, centering was also performed on the test set when making the predictions on the new data.

## 1.2 Model performance evaluation

The performance of each model is summarized in table 1, being only presented the models whose hyperparameters allowed to achieve the best performances.

**Table 1:** Performances of the linear regressors tested, being the hyperparameters specified.

| Model | Linear Regression | Ridge Regression ($\lambda$ = 0.001) | Lasso Regression ($\lambda$ = 0.002) | Lasso Feature Selection ($\lambda$ = 0.005) | Stepwise Feature Selection | 2nd order Polynomial model | 2nd order polynomial with lasso regression ($\lambda$ = 0.01) |
|---|---|---|---|---|---|---|---|
| Mean SSE | 0.128700 | 0.128702 | 0.12831 | 0.12792 | 0.12433 | 0.70674 | 0.14513 |

## 1.3 Conclusions

Stepwise feature selection was the best-performing model with an $SSE$ of 0.12433 and, thus, was chosen to predict $\hat{y}$ for the test dataset, achieving a $SSE$ of 11.8904664. This substantial difference between the performance in the cross-validation and the test set could be an indication that some mistakes might have occurred during the implementation of the models, either in their training or evaluation. However, this $SSE$ value is in line with the rest of the student groups, indicating that the test set might have a significantly different distribution comparing to that of the training set. Following this line of reasoning, in fact the model chosen isn't as robust if the training and test sets have different distributions, since eliminating the identified unimportant features might no longer be reasonable as they could start to contribute to the predictions. Thus, instead of eliminating features, it would be more suitable to downweighing them, for example with Lasso Regression. Futhermore, the search for the optimal $\lambda$ should

be more refined, with further investigating between the two hyperparameters that achieved the best results.

## 2 Regression - Second problem

In this section, the same objectives as in Section 1 are aimed to be achieved, existing although the additional problem of outliers in the training data, consisting of about 20% of the data.

### 2.1 Methodology

The models adopted in subsection 1.1 were implemented for this problem, except for the polynomials, due to their poor performance and higher computational effort, so we focused on regression and regularization methods. In addition, robust methods were implemented in order to deal with the outliers. .

Huber regression assigns less weight to observations identified as outliers, by using the Huber loss $H_\epsilon(z)$ in the optimization process. The goal is to minimize the cost function 6, where $\sigma$ denotes the standard deviation and $\alpha$ is a regularization parameter. It identifies outliers by considering the residuals, denoted by $z$. If $|z|$ is smaller than a threshold $\epsilon$, the samples are identified as inliers and a squared loss function is applied. Otherwise, it is identified as an outlier and a linear loss is applied, instead of disregarding them completely.

$$\sum_{i=1}^{n}\left(\sigma + H_\epsilon\left(\frac{x^{(i)}\beta - y^{(i)}}{\sigma}\right)\sigma\right) + \alpha\|\beta\|_2^2, \quad H_\epsilon(z) = \begin{cases} z^2, & \text{if } |z| < \epsilon, \\ 2\epsilon|z| - \epsilon^2, & \text{otherwise} \end{cases} \tag{6}$$

RANSAC is a resampling technique which estimates the model's coefficients by using the minimum number of samples, being those randomly selected and iteratively substituted if not fitted with a predefined tolerance. The model parameters are only fitted to the identified inliers.

Theil-Sen regressor involves fitting multiple regression models on all possible combinations of subsets of size $p$ and then aggregating the coefficients. An intercept is calculated if $p \geq n_{features} + 1$. The final slope (and possibly the intercept) is defined as the spatial median of all the least square solutions.

Besides from obtaining an accurate model in the presence of outliers, Huber regressor and RANSAC were used as a detection method to delete outliers for the subsequent regression over the cleaned training set with the regressors referenced in 1.1.

For outlier removal, since it is known that it exists about 20% of outliers, hyperparameters were set in order to achieve this percentage, namely $\epsilon$ and $\alpha$ for the Huber Regressor. For the regressors, hyperparameters, namely $\lambda$, were selected manually and iterated until identifying the one that achieves the best performance, further investigating between the two $\lambda$ that achieved the best results.

Other outlier detection methods algorithms were implemented: Isolation forest, Inter Quantile Range, Local outlier factor and Cook's distance. All of these methods achieved significantly worse performances compared to the reaming models used for the same purpose. For this reason, these algorithms and how they work will not be further discussed.

## 2.2 Model performance evaluation

The performances of the models tested without removing outliers is summarized in table 2. Since the Hubber Regression achieved better results, and in fact the Ransac only identified 16% of data as being outliers, instead of 20%, it is only presented the performances for elimination with Huber regression (table 3)

**Table 2:** Performance of the models without outlier removal, being the hyperparameters specified. SwFS stands for Stepwise Feature Selection.

| **Model** | **Linear Regression** | **Ridge Regression ($\lambda$ = 63)** | **Lasso Regression ($\lambda$ = 0.1)** | **Linear Regression with Lasso Selection ($\lambda$ = 0.1)** | **SwFS** | **Hubber Regression ($\alpha$=30 and $\epsilon$=1.8)** | **Hubber Regression with SwFS ($\alpha$ = 21 and $\epsilon$ = 1.6)** | **Theil-Sen Regression** | **Ransac** |
|---|---|---|---|---|---|---|---|---|---|
| Mean SSE | 72.69241 | 69.02327 | 69.26119 | 71.43357 | 65.21434 | 66.71845 | 61.70788 | 70.45633 | 76.42052 |

**Table 3:** Performances of linear regression models with outlier removal via Huber regression.

| **Model** | **Linear Regression ($\alpha$ = 0.001 and $\epsilon$ = 1.8)** | **Ridge Regression ($\lambda$ = 0.001, $\alpha$ = 0.01 and $\epsilon$ = 1.8)** | **Lasso Regression ($\lambda$ = 0.001, $\alpha$ = 0.1 and $\epsilon$ = 1.8)** |
|---|---|---|---|
| Mean SSE | 0.0919529 | 0.0919534 | 0.09051 |

## 2.3 Conclusions

The presence of outliers in the training set greatly worsens the performance of the algorithms, with the models skewed towards the outlier values, which are far away from the central mass of observations, leading to the linear regressors finding a worse and more biased fit with inferior predictive performance. By removing the outliers, the achieved performances are tremendously better. Indeed, by computing the Residual Plots for Linear regression without and with outlier removal, the goodness-of-fit measure has increased from 0.238 to 0.998, which is to be expected as there is less variance in the predictions after outlier removal. This Residual Plots can be inspected in the notebook submitted. Nevertheless, it is important to mention that as more points are considered outliers, the performance is expected to improve, potentially overfitting the model since it is eliminating samples that it cannot explain. As mentioned, the hyperparameter tuning was made with the intention to reach 20% of outliers, but without this information the decision would have been more difficult. Hubber Regression with $\alpha$ = 0.1 and $\epsilon$ = 1.8 for eliminating 20 detected outliers on the training set, followed by Lasso Regression with $\lambda$ = 0.001, was the model that achieved the best performance of 0.09051. The outlier samples identified were: 15, 18, 24, 29, 30, 33, 34, 36, 47, 48, 62, 63, 65, 70, 71, 72, 83, 88, 93 and 95, which were confirmed later to be the actual outliers. In the test set, the performance was 13.07491486. Once again, the disparity between the cross-validation score and the one obtained in the test set indicates that the distribution of the data is potentially different. It is concluded that although outliers impose a more challenging training process, robust regression methods can be effective in detecting them.

# 3 Classification - First problem

In this section, it is aimed to train a classifier that distinguishes two types of patterns in butterflies' wings, called spots and eyespots. The training set contains 8273 images, divided into 5142 pictures of spots (class 0) and 3131 pictures of eyespots (class 1). Each image is a RGB image with 30x30 pixels. The classifier is evaluated with a testing set containing 1367 patterns.

## 3.1 Methodology

In order to build the classifier, different algorithms were tested, using Sklearn, TensorFlow and Keras: different configurations of Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs), k-means clustering and Support Vector Machines (SVMs). Instead of doing cross-validation as until now, it was opted to split the data set as 90% training set and 10% validation set. This was done since MLPs and CNNs are very time-consuming. The performance of the algorithms was measured as the f1-score of the validation set, being a trade-off between precision and recall.

Given the stochastic nature of the MLPs and CNNs, namely their initialization with random weights, these models do not always find the same solution. For this reason, each configuration was run 5 times, and the median f1-score was considered. All neural networks were trained in mini batch, having divided the training set into batches of size 64, unless otherwise stated. Although in binary classification one-hot encoding of the labels isn't necessary, it was the strategy adopted. MLPs and CNNs were trained by optimizing the categorical cross-entropy loss function.

For each of the MLP's and CNN's configurations, learning curves were monitored, regarding the loss and f1-score in the training and validation sets throughout the epochs. The learning curves with respect to the validation loss have an inflection point, denoting the epoch in which the network starts to overfit and, consequently, will not perform well in the test set. For this reason, an early stopping criterion was considered, in order for the training process to be stopped when the validation loss registers no improvement. Due to the stochasticity of neural networks, it was defined a specific number of consecutive epochs (patience) on which no improvement must be registered in order for the training to be stopped. Combined with the early stopping callback, a ModelCheckpoint callback was also defined, since we are interested in saving the model with the best f1-score on the validation set obtained during training.

Normalization of the input data was done, since neural networks are not linear and small inputs reduce the saturation of the activating functions, helping in the convergence during training.

### 3.1.1 Multilayer Perceptrons

MLPs are fully densely connected neural networks consisting of an input layer, hidden layers and an output layer, where every node on each layer is connected to all other nodes on the next layer. There is no connection between nodes within a single layer and, with the exception of the input layers, all nodes use an activation function. The output layer consisted of two nodes activated by a softmax function, since it was adopted the one-hot encoding format for the labels. Different configurations were tested with respect to the hidden layers, namely changing the number of layers, the type of activation functions, number of nodes and dropout layers. All models were evaluated with a patience of 10. The mini-batch gradient descent method was considered as optimizer for updating the weights, considering a batch size of 64.

Once we obtained the model with the highest f1-score, strategies for dealing with the class imbalance were applied: penalize learning algorithms, which increase the cost of classification mistakes on the minority class by an amount proportional to how under-represented it is, being this tuned by the class_weight argument; resampling algorithms, being generally categorized in three types:

i) Undersampling – removing samples from the majority class, thus discarding potentially useful information that could determine the decision boundary between the classes. For this reason, under-sampling techniques weren't applied alone.

ii) Oversampling – generating samples in the minority class, being the most naïve strategy the generation of new samples by randomly sampling with replacement the currently available

samples (Random Oversampling). In the sklearn package there is available a wide range of oversampling methods, including techniques that generate synthetic data for the minority class, for instance the Synthetic Minority Oversampling Technique (SMOTE) that works by randomly picking a point from the minority class and computing the k-nearest neighbors for this point, being the synthetic points added between the chosen point and its neighbors.

iii) Oversampling and Undersampling combined – A combination of over-sampling the minority class and under-sampling the majority class has been reported to achieve best results in some problems. One technique is the SMOTE-Tomek Links, combining the SMOTE ability to generate synthetic data for minority class, and searching for samples in the majority class that have the lowest Euclidean distance with the minority class and remove those.

### 3.1.2 Convolutional Neural Networks

The full connectivity of the MLPs make them prone to overfit. CNNs are regularized versions of MLPs, assebling patterns of increasing complexity by using filters that capture smaller and simpler patterns in images. The hidden layers are convolutional layers, applying a convolution kernel of specified size to the layer's input matrix, producing a feature/activation map. Different CNN configurations were tested, by changing the number of hidden layers, number of nodes, and adding other possible layers, such as: pooling layers, added after convolutional layers, downsampling the feature maps to introduce local translation invariance, which reduces overfitting; dropout layers, which reduce reliance on any single feature by ensuring that it is not always available, forcing the model to look for different connection, which also reduces overfitting. All convolutional and dense layers had as activation function the Leaky ReLU, since it adds nonlinearity to the network, allowing it to learn non-linear decision boundaries. All CNNs were tested with 20 epochs, a patience of 5 and optimized with the Adam optimizer. The oversampling method that allowed to achieve best results on MLPs was the one applied to the CNNs.

## 3.2 Model performance evaluation

The f1-scores obtained for all the tested MLP configurations are presented in Tables 4 and 5. The f1-scores for the CNNs with SMOTE-Tomek Oversampling are presented in Table 6. The k-means clustering method and SVM were not successful, only achieving f1-scores of 0.41 and 0.56, respectively.

**Table 4:** Performances of the MLP configurations tested. Note that a final Dense 2 Softmax layer is present in all configurations.

| Model id | MLP configuration | Median Validation F1-score |
|---|---|---|
| MLP 1 | Dense 128 ReLU | 0.79481 |
| MLP 2 | Dense 128 ReLU, Dropout 0.5 | 0.79876 |
| MLP 3 | Dense 128 ReLU, Dense 50 ReLU | 0.80117 |
| MLP 4 | Dense 128 ReLU, Dense 50 ReLU, Dense 50 ReLU | 0.81284 |
| MLP 5 | Dense 50 ReLU, Dense 50 ReLU, Dense 25 ReLU | 0.79945 |
| MLP 6 | Dense 128 selu | 0.77129 |
| MLP 7 | Dense 128 ReLU, Dense 50 ReLU, Dense 50 ReLU, Dropout 0.2 | 0.79791 |

**Table 5:** Performances of the MLP 4 tested with strategies for dealing with the class imbalance.

| Model id | MLP configuration | Median Validation F1-score |
|---|---|---|
| MLP 8 | Penalize learning (class 0: 0.8045, class 1: 1.3211) | 0.79481 |
| MLP 9 | Random Oversampling | 0.82670 |
| MLP 10 | SMOTE (k-neighbours = 5) | 0.83295 |
| MLP 11 | SMOTE-Tomek links | 0.83469 |

**Table 6:** Performances of different CNN configurations, with SMOTE-Tomek oversampling. Conv2D and Dense layers have Leaky ReLU(0.1) as activation function, and a final Dense 2 Softmax layer is present in all configurations.

| Model id | CNN configuration | Median Validation F1-score | |
|---|---|---|---|
| CNN 1 | Conv2D 32 (3x3), Maxpooling (2x2), Conv2D 64 (3x3), Maxpooling (2x2), Conv2D 128 (3x3), Maxpooling (2x2), Flatten, Dense 128 | 0.87864 | |
| CNN 2 | Conv2D 32 (3x3), Maxpooling (2x2), Conv2D 64 (3x3), Maxpooling (2x2), Flatten, Dense 128 | Batch size 32 | 0.87029 |
| CNN 3 | | Batch size 64 | 0.88441 |
| CNN 4 | | Batch size 128 | 0.88440 |
| CNN 5 | Conv2D 64 (3x3), Maxpooling (2x2), Conv2D 32 (3x3), Maxpooling (2x2), Flatten, Dense 128 | 0.87997 | |
| CNN 6 | Conv2D 32 (3x3), Averagepooling (2x2), Conv2D 64 (3x3), Averagepooling (2x2), Flatten, Dense 128 | 0.88388 | |
| CNN 7 | Conv2D 32 (3x3), Maxpooling (2x2), Dropout (0.3), Conv2D 64 (3x3), Maxpooling (2x2), Dropout (0.3), Flatten, Dense 128, Dropout (0.3) | 0.88974 | |
| CNN 8 | Conv2D 32 (3x3), Maxpooling (2x2), Conv2D 64 (3x3), Maxpooling (2x2), Flatten, Dense 128, Dropout (0.3) | 0.89125 | |

### 3.3 Conclusions

Penalize learning with class weights did not improve the MLP performance. Comparing the best MLP and CNN models (MLP 11 and CNN 8), the latter reaches better performance. Nevertheless, it was seen that the validation f1 score varied significantly in each trial. It would be better to do Repeated k-fold Stratified Cross-Validation for the evaluation, but that would take quite some time.

As for the analysis regarding the resampling methods, the change in the distribution of classes should have only been applied to the training set, as the intent is to influence the fit of the model and reduce the bias towards the majority class. Resampling should not have been applied to the validation set, whose class distribution should be representative of the test set. This was a mistake made in the notebook submitted for this exercise and made the evaluation very optimistic. The models should be trained and evaluated properly again. Indeed, considering the approach that motivated our selection of the best model (CNN 8) when submitting the predictions on the test set, we reached a f1-score of 0.89125 on the validation set, contrasting with 0.86614 obtained in the test set. Furthermore, the CNN 8 relies on SMOTE-Tomek resampling technique for balancing the dataset, which could introduce non-realistic synthetic sample patterns. This should be further investigated.

## 4 Classification - second problem

This section consists of a multiclass task, where it is aimed to train a classifier that segments a particular type of eyespot into 3 distinct areas, by classifying each pixel in the 30x30x3 eyespot images based on the RGB intensities in a 5x5 neighbourhood surrounding (patches). In fact, given these 5x5 patches, only 26x26 pixels will be classified in each image. The training set contains 50700 of these 5x5 patches, that is, a total of 75 eyespot images. The class distribution over the training set is 1568 white center pixels (class 0), 40438 ring pixels (class 1), 8694

background pixels (class 2). Each patch image is a RGB image with 5x5 pixels. The classifier is evaluated with a testing set containing 33800 patches.

## 4.1 Methodology

The models described in 3.1 were adapted to three classes, by considering 3 nodes in the output layer activated by a softmax function, being the CNNs adapted to the input of 5x5x3 patch images. Once again, the data set is highly imbalanced, with a large number of class 1 samples and a very small number of class 0 samples. For this reason, several other models were tested: Decision Trees, Random Forests, SVM, K-nearest neighbours (K-NNs), Bagging KNN and Logistic Rgression.

The performance of the algorithms was measured as the balanced accuracy of the validation set, being a more accurate metric of the performance of a model when classes are imbalanced, while accuracy simply measures the fraction of correct predictions, being misleading when using a model that is not robust to class imbalance. When the dataset is imbalanced, the model could simply just be predicting only the majority class and haven't really learnt how to classify the data into its classes, being this translated in a misleading high accuracy and, in contrast, a lower balanced accuracy. In fact, if classification reports are analysed, recall and f1 scores are higher for the majority class. The scores need to be almost uniform for all classes, to ensure we can trust the model.

Except for the MLPs and CNNs, hyperparameter tuning using grid search and cross validation was done for the remaining classifiers, allowing to analyse parameter combinations and, thus, inform the choice of best parameters, based on our scoring function (balanced accuracy). For cross validation, the data set was split into 90% training set and 10% validation set, and the training was repeated 5 times, being the split stratified with respect to the labels, to ensure that the percentages of samples for each class in each split is preserved. The training was repeated 5 times and the mean balanced accuracy was considered. When possible, penalize learning with class weights was set and tested with grid search if it improved the results.

Once we obtained the model of each type of classifier with the highest balanced accuracy, modifications to the training set were applied in order to deal with the class imbalance. This time, Random Oversampling, SMOTE and data augmentation were the techniques tested. For data augmentation, it was created new images from the minority classes (class 0 and class 2) through image transformations, specifically horizontal flipping from left to right and rotations of 90º and its multiples (180º and 270º), since other rotations could significantly distort the images. If after these transformations the dataset wasn't perfectly balanced, Random Oversampling of the original images was applied.

### 4.1.1 Decision Trees and Random Forests

A decision tree is a flowchart-like tree structure where an internal node represents a feature, each splitting node represents a decision rule and each leaf corresponds to a label. It learns to partition on the basis of the feature value space such that the samples with the same labels or similar target values are grouped together, being this done recursively. The quality of a candidate split of a node is computed using an impurity function, such as the gini impurity and entropy impurity. Gini impurity is calculated by subtracting the sum of the squared probabilities of each class from 1, while a decrease in entropy impurity can be understood as the increase in the purity of the node. For tree growing, the impurity drop of a node is computed, and the feature that achieves the greatest drop is selected for splitting the node. Decision-tree classifiers can create over-complex trees with greater depth and number of splits, increasing the variance

of the model, not generalizing the data well, that is, they overfit the training data. Pruning is one of the techniques to avoid overfitting, existing two types:

i) pre-pruning, done while growing the tree, stopping non-significant branches from generating. This is done by setting constraints, for example to the maximum depth of the tree (max_depth parameter), the minimum impurity drop (min_impurity_decrease parameter) and the minimum number of samples required to split an internal node (min_samples_split parameter). All these parameters were tested for different values.

ii) post-pruning, done by removing non-significant branches after the complete growing of the decision tree. Minimal cost complexity pruning was tested, which recursively finds the node with the "weakest link", being this characterized by an effective alpha, where the nodes with the smallest effective alpha are pruned first. As alpha increases, more of the tree is pruned (number of nodes and depth decreases), which increases the total impurity of its leaves. Decision Trees were trained using the effective alphas (cpp_alpha parameter), and selected the one that maximized the validation balanced accuracy.

Another approach for decision tree classifiers being less susceptible to overfitting is to aggregate the prediction outcome of multiple decision trees on various sub-samples of the dataset and create a final outcome based on an averaging mechanism (majority voting). This ensemble of randomized decision trees is known as a Random Forests, improving the accuracy of the predictions. The best Decision Tree model, achieved by following the methodology previously described, was also used for setting the parameters for the Random Forest. Additionally, grid search was also performed regarding the min_samples_split and max_depth parameters.

### 4.1.2 K-NNs and Bagging K-NNs

k-NN algorithm classifies unknown data points by finding the most common class among the k-closest neighbours. For this, a distance metric needs to be defined, being the most common the Euclidean distance. Neighbours that are further aways can less strongly influence the prediction, being this tuned by the parameter weights = 'distance'. Bagging was also combined with K-NNs, using initially 100 estimators, reducing afterwards because of increased computational burden with higher number of estimators. Defining 20 estimators didn't significantly worsen the final mean accuracy score in the validation, so this was number of estimators considered.

## 4.2 Model performance evaluation

The performances of the Decision Tree classifiers tested with GridSearchCV are presented in Table 7. The performances of Random Forest Classifiers based on the DT 6 parameters, as well as by performing GridSearchCV for the max_depth and min_samples_split are presented in table 8. For this last model, the performances obtained after applying the strategies to balance the training set are also presented in table 8. Since SMOTE lowered significantly the performance, this strategy wasn't applied for the remaining classifiers.

Regarding the K-NN classifiers, n_neighbours = 3 and weights = 'uniform' yielded the best performance of 0.87183, while if weights = 'distance' the performance was 0.87081. These parameters were set for the K-NN with Bagging, and in fact n_neighbours = 3 and weights = 'distance' yielded the best performance of 0.87649.

MLP 1 and MLP 2 presented in table 3.1 were the only configurations tested, achieving performances of 0.68955 and 0.66000, respectively. The CNN 3 presented in table 3.2 was the one that achieved the better performance of 0.83431 in the imbalanced training set. For K-NN with Bagging , MLP 1 and CNN 3 the results obtained regarding the strategies for dealing with the class imbalance are presented in table 4.2.

**Table 7:** Performances of the different decision trees tested, specifying the parameters.

| Model id | Parameters of the Decision Tree Classifier | Mean validation BAcc |
|---|---|---|
| DT 1 | Criterion = 'gini' with penalize learning | 0.74057 |
| DT 2 | Criterion = 'entropy' with penalize learning | 0.76052 |
| DT 3 | Criterion = 'entropy' with penalize learning and min_impurity = 0.01 | 0.75973 |
| DT 4 | Criterion = 'entropy' with penalize learning and min_samples_split = 253 | 0.82612 |
| DT 5 | Criterion = 'entropy' with penalize learning and max_depth = 15 | 0.80360 |
| DT 6 | Criterion = 'entropy' with penalize learning, max_depth = 15 and min_samples_split = 254 | 0.82934 |
| DT 7 | Criterion = 'entropy' with penalize learning and cpp_alpha = 0.001579 | 0.81773 |

**Table 8:** Performances of Random forests, K-NN with Bagging, MLP 1 and CNN 3, combined with strategies for dealing with class imbalance.

| Stategy \Model | Random Forest DT 6 | Random Forest GridSearchCV | KNN with Bagging | MLP 1 | CNN 3 |
|---|---|---|---|---|---|
| Penalize learning | 0.86258 | 0.87892 | - | 0.85450 | - |
| Random Oversampling | - | 0.87610 | 0.91193 | 0.85834 | 0.90063 |
| SMOTE | - | 0.77763 | - | - | - |
| Data augmentation | - | 0.88721 | 0.91525 | 0.86718 | 0.88979 |

SVM and Logistic regression were not successful, only achieving performances of 0.57585 and 0.59667, respectively. These models could have been further analysed with hyperparameter tuning, but instead we focused on the remaining classifiers.

## 4.3 Conclusions

The model we used to classify the testing set was K-NN with Bagging (n_neighbours = 3 and weights = 'distance') after Random Oversampling of the training set. Although employing our data augmentation strategy resulted in a slightly better performance, by analysing the classification reports over the 5 splits during cross validation it was seen that the macro average of the precision metric deteriorated significantly from roughly 0.81 in average across the splits when doing random oversampling to 0.7 for data augmentation, with the macro average of the recall being similar in both cases. In fact, this behaviour was observed for all the classifiers that were tested with random oversampling and data augmentation, so besides this last could achieve better performances for the balanced accuracy, for the ranking of the models we disregarded data augmentation.

The model selected although having a performance of 0.91193 in cross validation, it was obtained a balanced accuracy of 0.79873 in the test set. The concept of overfitting is difficult to apply to k-NN due to its characteristics. It is a distance-based classifier and therefore can give very optimistic results if the features of the validation set do not vary much from the training ones, meaning that the distances are small. One the other hand, it can also be sensitive to small variations in test features, which can have a big impact on the calculated distances and thus affecting the classification process. This should have been foreseen when selecting the best model for the test set predictions. CNN 3 with Random Oversampling (0.90063), as well as Random Forest with parameters defined by GridSearchCV combined with Penalize learning (0.87892), can both potentially be more robust and, consequently, achieving better balanced accuracy in the test set. In fact, segmentation of a given set of test images was performed by these three models, and visually both the CNN 3 and the Random Forest seemed to perform better. This can be inspected in the notebook submitted. Nevertheless, even for this models, to gain insight as to whether they are overfitting or not, it should be computed the mean balanced accuracy on both train and validation datasets. If these performances are comparable, it constitutes evidence of the model's certainty, being the model able to generalize well to new data.