TÉCNICO LISBOA

Instituto Superior Técnico

Aprendizagem Automática / Machine Learning

MEEC

Machine Learning Project, 2022/2023

Students:
Afonso Brito Caiado Alemão | 96135
Rui Pedro Canário Daniel | 96317

Group: **46**

Teacher:
Ana Catarina Fidalgo Barata

September 12, 2023

Contents

0 Introduction

Throughout this project we use the Python programming language because it has powerful libraries for building Machine Learning applications and it is a widespread language in industry. The project is split into 2 parts (regression and image analysis).

For each problem, we have access to a training set (feature vectors and real outputs) and a test set (just the feature vectors). This data is stored in numpy (.npy) format.

We implement and train our machine learning approaches using the training set. For each question we research and try some models and pick only one to apply to the test set and perform the submission.

1 Part 1 - Regression with Synthetic Data

1.1 First Problem - Basic Linear Regression

The first problem is a basic linear regression problem. We have a training set with $n = 100$ examples. Each example comprises a feature vector, $x^{(i)} \in \mathbb{R}^{10}$, with 10 features, and an outcome $y^{(i)} \in \mathbb{R}$, for $i = 1, ..., n$.

We trained a linear predictor (1) using the sum of squared errors (SSE) criterion, computed in the training set.

$$\hat{y} = f(x) = \begin{bmatrix} 1 & x^T \end{bmatrix} \beta, \tag{1}$$

For that purpose we research and try some different models:

- In the first one (Ordinary Least Squares, that we denominate **Simple Linear Regression**) is the minimization of the SSE cost functional that leads to a system of equations that are denoted normal equations (2). In this case, X and y are given by 3.

$$(X^T X)\hat{\beta} = X^T y, \tag{2}$$

$$X = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_p^{(1)} \\ 1 & x_1^{(2)} & \dots & x_p^{(2)} \\ 1 & \vdots & \dots & \vdots \\ 1 & x_1^{(n)} & \dots & x_p^{(2)} \end{bmatrix} \quad , \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix} \tag{3}$$

The normal equations have a unique solution if $det(X^T X) \neq 0$. If that does not happen, the inverse of matrix $X^T X$ may not exist due to small amount of data like the number of data points smaller than the number of features or due to the existence of redundant features (linearly dependent).

In the case that the inverse of matrix $X^T X$ exists, we can estimate the $\hat{\beta}$ parameter: equation 4.

$$\hat{\beta} = (X^T X)^{-1} X^T y, \tag{4}$$

- The second criterion used was **Ridge Regression**. This model addresses some of the problems of ordinary least squares by imposing a penalty on the size of the coefficients, which prevents the occurrence of overfitting. The Ridge coefficients minimize a penalized residual sum of squares:

$$\hat{\beta}_{ridge} = arg\min_{\beta} ||y - X\beta||^2 + \alpha||\beta||^2, \tag{5}$$

In equation 5, $\|.\|$ denotes the Euclidean norm. The term $||\beta||^2$ penalizes the use of large coefficients and it is denoted a regularization term. This criterion aims to represent the data, keeping the coefficients small. α represents the trade-off between both objectives.

In fact, the complexity parameter $\alpha \geq 0$ controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.

Furthermore, we assume that training data X and y have zero mean. The coefficient β_0 is not usually included in the regularization term.

The Ridge Regression can be solved by computing the gradient vector and making it equal to zero, leading to equation 6, where matrix $(X^T + \alpha I)$ is always non-singular, even if $(X^T X)$ is singular.

$$\hat{\beta}_{ridge} = (X^T + \alpha I)^{-1} X^T y, \tag{6}$$

- The third criterion used was **Lasso Regression**. The Lasso is a linear model that estimates sparse coefficients, aiming to minimize the sum of squared errors (with $\beta_0 = 0$) but with a different constraint on the coefficients that penalizes large errors less.

The Lagrangian formulation is given by equation 7, where $||\beta||_1$ is the l_1 norm of the coefficient vector.

$$\hat{\beta}_{lasso} = arg\min_{\beta} ||y - X\beta||_2^2 + \alpha||\beta||_1, \tag{7}$$

This is useful in some contexts when we wish to find sparse solutions for β (with some zero coefficients)

which corresponds to selecting only a subset of features (feature selection).

In fact, the Lasso estimate is often a sparse vector of coefficients where less important features receive a zero coefficient.

In both second and third model used, X and y are given by expression 8.

$$X = \begin{bmatrix} x_1^{(1)} & ... & x_p^{(1)} \\ x_1^{(2)} & ... & x_p^{(2)} \\ \vdots & ... & \vdots \\ x_1^{(n)} & ... & x_p^{(2)} \end{bmatrix} \quad , \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix} \tag{8}$$

- The fourth criterion used was **Polinomial Model** given by equation 9 (x scalar).

$$f(x) = \beta_0 + \beta_1 x + ... + \beta_p x^p \tag{9}$$

The model is non-linear in x but linear in the parameters β_i. Therefore, the β coefficients can be obtained by the least squares method, leading to a linear set of equations. This model becomes numerically unstable when we increase the order of the polynomial.

Evaluation

To evaluate and compare the performances of the models, we used **cross validation** because we do not have a large amount of data. The data set is split into k folds T_k (subsets with the same number of examples). One fold is used for testing and the others for training. After, the test fold rotates K times. We shuffle the data before spliting it with some different random states, and used the average of the score (SSE of validation data) as metric.

For Ridge and Lasso Regression we used cross validation to check the optimal *alpha* values that minimize the average SSE of the validation data.

We wrote code in file *GR46.py* for the cross validation process described above and we report the results in figure 1, including the optimal *alpha* values.

Nº folds (k)	Simple Linear Regression	Ridge Regression	Lasso Regression	Polynomial Model	
				degree = 2	degree = 3
2	0,6704	0,6898 for alpha = 0,299	0,73389 for alpha = 0,0069	108,759	68,086
5	0,2653	0,2641 for alpha = 0,2357	0,2571 for alpha = 0,0031	8,596	27,358
10	0,1256	0,1267 for alpha = 0,1075	0,1238 for alpha = 0,0029	0,574	15,882
20	0,0626	0,0623 for alpha = 0,1050	0,0602 for alpha = 0,0017	0,268	8,282

Figure 1: Table with the average SSE of validation data obtained with cross validation.

When analyzing the obtained data, we choose to use a Ridge Regression. We choose $alpha = 0.2357$ because the cross validation test with 5 folds seems to be what gives the most appropriate treatment to the set of 100 data points that we have. Simple Linear Regression and Lasso Regression with the *alpha* value in the table also seem reasonable. We did not choose Lasso Regression, despite being the one that gets the best results, because in this exercise the data do not have outliers, and Lasso corresponds to selecting only a subset of features (feature selection).

For the Polynomial Model is clear that the model is overfitted. It fits too closely to the training dataset. The consequence of this is that the model may perform well on a training dataset but not as well on a testing dataset. For the 100 data training we obtain $SSE_{train} \approx 0.38142$ for $degree = 2$, and $SSE_{train} \approx 4.40998 \cdot 10^{-27}$ for $degree = 3$. However, when we used cross validation in these model, the results were disappointing: this model is not suitable. There is a large difference between the evaluation of the model in the training set and in an independent set. That means that the model is too specialized in

representing the training data and performs much worse with new data.

Statistical evaluation

To evaluate the performance, we applied the estimated predictor to an independent set of data with $n^{'} = 1000$ examples: x_{test}.

$$T_{test} = \{(x^{(1)}, y^{(1)}), ..., (x^{(n)}, y^{(n)})\}, \tag{10}$$

We predict the outcome for x_{test} and save it in *Ytest_pred_Regression1.npy* file.

The teachers used the metric of SSE to compare the predictions $\hat{y}^{(i)}$ and the true values $y^{(i)}$: $SSE = 11.86412561$.

We reached the 38th position in the leaderboard and the first place obtained $SSE = 11.1951974$. This demonstrates that our strategy for solving this task was good. To improve our solution we should have chosen Lasso Regression because, despite there are no outliers in the data, that model provides the best results in figure 1.

1.2 Second Problem - Linear Regression with Outliers

The second problem is similar to the previous one but some of the training examples (about 20%) are outliers. Our goal is to devise a method to estimate a predictor in the presence of outliers in the training data. When it comes to evaluating the model, we assume that the test set, T_{test}, does not have any outliers.

In a first phase, we identify and remove the outliers by creating a method called *RemoveOutliers*. In this, we create a Simple Linear Regression with the initial data that allows to identify and remove the outliers: data points whose squared error is above some threshold (that we consider initially equal to 64). Then we repeated the same process, using half of the previous threshold. This process is repeated until there is an iteration in which no data points are removed (there are no more outliers). This way we have the x_{train} and y_{train} datasets without outliers: 20 data points were removed.

Evaluation

We apply to the training set without outliers the same process as in the last question. We report in figure 2 a table with the average SSE of validation data obtained with cross validation for $k = \{4, 5, 8, 10\}$ folds.

Nº folds (k)	Simple Linear Regression	Ridge Regression	Lasso Regression	Polynomial Model	
				degree = 2	degree = 3
4	0,2259	0,2442 for alpha = 0,0727	0,2361 for alpha = 0,0071	14,871	18,253
5	0,1838	0,1915 for alpha = 0,0144	0,1831 for alpha = 0,0065	9,557	16,496
8	0,1134	0,1171 for alpha = 0,0082	0,1149 for alpha = 0,0065	4,198	9,605
10	0,0882	0,0928 for alpha = 0,0127	0,0910 for alpha = 0,0063	5,401	8,138

Figure 2: Table with the average SSE of validation data obtained with cross validation for $k = \{4, 5, 8, 10\}$.

Through the analysis of the obtained data, we chose the model that has the best average of SSE for 5 folds: Lasso Regression. We choose $alpha = 0.0065$ because it minimizes SSE for cross validation with 5 and with 8 folds which seem to give the most adequate treatment to the set of 80 data points that we have. Simple Linear Regression and Ridge Regression (with the optimal *alpha* in the figure) also seem reasonable. For the Polynomial Model is clear that the model is overfitted, as in the last question.

Statistical evaluation

To evaluate the performance, we applied the estimated predictor to an independent set of data with $n^{'} = 1000$ examples: x_{test}.

We predict the outcome for x_{test} and save it in *Ytest_pred_Regression2.npy* file.

The teachers used the metric of SSE to compare the predictions $\hat{y}^{(i)}$ and the true values $y^{(i)}$: $SSE = 13.27814961$.

We reached the 52th position in the leaderboard and the first place obtained $SSE = 12.26182582$. This demonstrates that our strategy for solving this task was reasonably good. To improve our solution we could have tested other different models like Elastic Net Regression or Stochastic Gradient Descent Regression to solve this problem.

2 Part 2 - Image Analysis

In the second part, our goal is to classify and segment two types of patterns that develop in the wings of butterflies: the spots and the eyespots. Spots are patterns that develop a single spot of color. Eyespots are patterns that develop spots and rings of color. Both types are illustrated in Figure 3. The detection of the two types of patterns was done automatically for a large number of butterflies of different species using a object detection method known as YOLO. After detection, the patterns bounding boxes have been resized to a common size of 30x30 pixels. Since the images are in color, the patterns have 3 color channels (RGB), thus each input is 30x30x3.

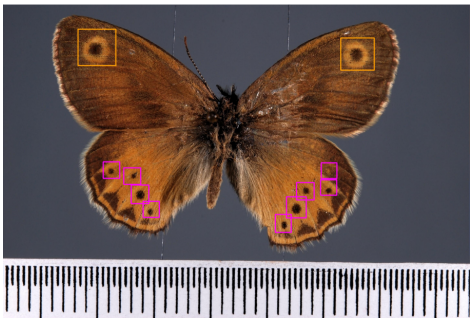

Figure 3: Identification of the two types of patterns that develop in the wings of butterflies.

2.1 First Task

The first classification task is a binary one, where we want to create a model that predicts the type of pattern that has been detected. The label is either 0 (spot) or 1 (eyespot). The dataset is imbalanced, since there is a big difference in the number of spots and eyespots in our dataset.

The metric used by the teaching team to evaluate the submissions for this task is the F1 score. We investigate which are the most suitable classifiers for image tasks and try some different models (described below).

-K-Nearest Neighbor

We wish to predict variable y knowing an input vector $x \in \mathbb{R}^p$ and a training set, given by equation 11.

$$T_{train} = \{(x^{(1)}, y^{(1)}), ..., (x^{(n)}, y^{(n)})\}, \tag{11}$$

It's possible to predict y for new values of x by finding the training pattern $x^{(i)}$ nearest to x and approximating y by $y^{(i)}$. If we reorder the training set according to equation 12, the nearest neighbor (NN) method assigns x to the outcome of the nearest neighbor: $f(x) = y_{(1)}$.

$$\|x_{(1)} - x\| \leq \|x_{(2)} - x\| \leq ... \leq \|x_{(n)} - x\| \tag{12}$$

This method can be extended to take into account k nearest neighbors of x. In classification problems, the predicted class is chosen $f(x)$ as the most voted class in the sequence $(y_{(1)}, \ldots, y_{(k)})$.

Using cross-validation, we estimate the value of k nearest neighbors that maximizes F1 score for our training data.

-Naïve Bayes Algorithm

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes Theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes Theorem states the relationship in 13, given class variable y and dependent feature vector x_1 through x_n:

$$P(y|x_1,...,x_n) = \frac{P(y)P(x_1,...,x_n|y)}{P(x_1,...,x_n)} \tag{13}$$

Using the naive conditional independence assumption that $P(x_i|y,x_1,...,x_n) = P(x_i|y)$ for all i, this relationship in 13 is simplified to the following equation 14:

$$P(y|x_1,...,x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1,...,x_n)} \tag{14}$$

Since $P(x_1,...,x_n)$ is constant given the input, we have that:

$$\hat{y} = arg\max_y P(y)\prod_{i=1}^{n} P(x_i|y) \tag{15}$$

For this task, we tested four Bayes classifiers (Gaussian, Multinomial, Complement and Bernoulli), in which the differences are mainly the assumptions they make regarding the distribution of $P(x_i|y)$.

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by equation 16.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}}\exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right) \tag{16}$$

The Naive Bayes algorithm for multinomially distributed data is one of the classic Naive Bayes variants used in text classification (where the data are typically represented as word vector counts). For this case, feature vectors represent the frequencies with which certain events have been generated by a multinomial distribution.

In the multivariate Bernoulli event model, features are independent booleans (binary variables) describing inputs. Is similar to the Multinomial Naive Bayes but the predictors are boolean variables. The parameters that we use to predict the class variable take up only values true or false, for example if a word occurs in the text or not.

The Complement Naive Bayes (CNB) algorithm is an adaptation of the standard Multinomial Naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the complement of each class to compute the model's weights.

-MLP

A multilayer perceptron (MLP) is a fully connected class of feedforward artificial neural network (ANN). It can be used for regression and for classification tasks. In classification tasks the output units typically have logistic or Softmax activation functions and the network is often trained with negative log-likelihood (cross-entropy) loss. It utilizes backpropagation for training.

An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function.

Initially, we convert train labels to one-hot encoding and split train data in 70% train data, 15% validation data to use in the model fit and 15% validation data for metric evaluation. Then we create a sequential model, start by adding a flatten layer to convert the 3D images to 1D.

Next, we add hidden layers to MLP. The choice of the number of layers is made by cross validation. We discuss the number of neurons per layer further ahead. We use 'relu' as activation function for all neurons in the hidden layers. We end the network with a softmax layer: a dense layer that will return an array of size equal to the number of classes.

This model uses the input image directly in the form of a pixel array. For instance, if input images have the same pattern in different positions, its features will be represented in different regions of the input array. MLP need to recognize patterns independently from where they may appear on the input, what makes the learning process more complex, leading to the need for a larger train data set. It also uses more elements in

the input than necessary because some image zones are irrelevant. In this way, overfitting can occur.

We also create an Early stopping monitor that will stop training when the validation loss is not improving. This criterion was used to avoid overfitting, given that the training of the network stopped when the loss of the validation data didn't improve, over the best configuration, in 10 consecutive epochs (*patience* = 10).

Finally, we fit the MLP to training and validation data using 'categorical_crossentropy' as the loss function, on mini-batch mode (batch size of 32) and Adam as the optimizer. The metrics used to optimization were tf.keras.metrics.Precision(*class_id* = 1) and tf.keras.metrics.Recall(*class_id* = 1) in order to optimize the F1 score for *class_id* = 1. As stopping criterion, we choose the number of epochs reaching 200.

-CNN

Convolutional Neural Network (CNN) is a type of deep neural network primarily used to recognize patterns in data. Like MLP, it is a Neural Network composed of a collection of neurons that are organized in layers, each with their own learnable weights and biases. However, it tries to solve the problems of the MLP with the use of convolutional and max-pooling layers before the conversion to an array.

The convolutional layer tries to find features in the input image by using of filters and separating the result in different images (important information). Then, the max-pooling layer downsamples the feature map extracted by the convolution layer, i.e, it reduces the number of pixels of the different images by running a filter across the features map. It only takes specific information from that filter, either the average of the values coming under the filter or the maximum depending on the pooling method selected. This reduces the feature map's spatial size and translates the features exact spatial information to rough information. Only after this process is repeated a second and a third time, the smaller images are turned into an input array to serve as input to a fully connected Feed-Forward Neural Network (FFNN).

Using convolutional layers plus max-pooling layers has many advantages over using just the MLP/FFNN. In first place, the FFNN receives as input an array with fewer elements, where only the most important features are represented, which, in turn, reduces the number of weights in the training. Another advantage of CNNs is that they can recognize pixel patterns (features) independently from where they are located in the image array, so the same feature is represented in the same region of the input array. Consequently, there is no longer the need for the neural network to identify where the feature is, taking down one step of complexity from the whole problem.

In fact, CNNs try to learn the features of the images, instead of literally learning each individual pixel of an image, as it happens in MLP. This property makes CNN models less prone to overfitting (when compared with MLPs) and allow to make accurate predictions of images of the same object but whose pixels are in different locations or when the object in the image is rotated.

Now we are going to describe our implementation of a CNN for this task. Initially, we convert train labels to one-hot encoding and split train data in 70% train data, 15% validation data to use in the model fit and 15% validation data for metric evaluation.

Overfitting generally occurs when there are a small number of training examples. In the first layer we apply data augmentation that takes the approach of generating additional training data from existing examples by augmenting them using random transformations that yield believable-looking images. This helps expose the model to more aspects of the data and generalize better. For that we used the following Keras preprocessing layers: tf.keras.layers.RandomFlip, tf.keras.layers.RandomRotation, tf.keras.layers.RandomZoom, tf.keras.layers.RandomBrightness and tf.keras.layers.RandomContrast. Then, it will standardize input values to be in [0, 1] because we should seek to make your input values small.

Our model consists of three convolution blocks with a max pooling layer in each of them. Then, we apply another technique to reduce overfitting: dropout regularization to the network. It randomly drops out 20% of output units randomly from the layer during the training process (by setting the activation to zero). Finally, there's a flatten layer followed by a fully-connected dense layer with 128 units on top of it, that is activated by a ReLU activation function. The end of the network has a softmax layer: a dense layer that will return an array of size equal to the number of classes.

We also create an Early stopping monitor that will stop training when the validation loss is not improving. This criterion was used to avoid overfitting, given that the training of the network stopped when the loss of the validation data didn't improve, over the best configuration, in 10 consecutive epochs (*patience* = 10).

Then, we fit the CNN to training and validation data using 'categorical_crossentropy' as the loss function, on mini-batch mode (batch size of 32) and Adam as the optimizer. The metrics used to optimization were tf.keras.metrics.Precision($class\_id = 1$) and tf.keras.metrics.Recall($class\_id = 1$) in order to optimize the F1 score for $class\_id = 1$. As stopping criterion, we choose the number of epochs reaching 200.

-Random Forest

Random Forest is a supervised learning algorithm that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. The more trees it has, the more robust a forest is. In fact, it creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. The sub-sample size is controlled with the *max_samples* parameter if $bootstrap = True$ (default), otherwise the whole data-set is used to build each tree.

However, Random Forest is slow in generating predictions because it has multiple decision trees. Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming.

-Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods that can be used for classification. The main idea is to separate the cloud of data in two regions, using a carefully chosen hyperplane.

As advantages, they are effective in high dimensional spaces and still effective if the number of dimensions is greater than the number of samples. It's memory efficient because uses a subset of training points in the decision function and is versatile: different kernel functions can be specified for the decision function.

As disadvantages if the number of features is much greater than the number of samples, avoid overfitting in choosing kernel functions and regularization term is crucial and also they do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

We test the most common choices for kernel: Linear, Polynomial and RBF. Some of this kernels depend on hyperparameters that have to be specified or learned during the training phase. We used GridSearchCV for cross validation.

Evaluation

To evaluate and compare the performances of the models tested, we used **cross validation**. The data set is split into k folds T_k (subsets with the same number of examples). One fold is used for testing and the others for training. After, the test fold rotates K times. We used the F1 score as metric.

In terms of data processing, we try some different modes: unchanged data, oversampling using SMOTE, oversampling with RandomOverSampler, oversampling by duplicating images of minority classes through rotations and tonal changes, undersampling and applying class weights.

For K-Nearest Neighbor we used cross validation to check the optimal value of number of neighbors, k, that maximize the average F1 score of the validation data. As, in the case of ties, the answer will be the class that happens to appear first in the set of neighbors, we only tested odd values for k.

For MLP we also used cross validation to estimate the optimal value for the number of hidden layers and for the neuron distribution in the network. We tried three different distributions: all layers with 32 neurons; all layers with 64 neurons; the first layer with 32 neurons, the second with 64 neurons, the third with 128 neurons and so on (number of neurons in ith layer $= 32 \cdot 2^{i-1}$).

For that we wrote code in file *GR046.py* and obtain the results in figure 4.

When analyzing the obtained data, we found that the best option is to use CNN. Inside CNN, we get the best results for oversampling by duplicating images of minority classes through rotations and tonal changes.

Statistical evaluation

To evaluate the performance, we applied the estimated predictor to an independent set of data with $n' = 1367$ examples: x_{test}. We predict the outcome for x_{test} and save it in *ytest_Classification1.npy* file.

The teachers used the metric of F1 score to compare the predictions $\hat{y}^{(i)}$ and the true values $y^{(i)}$: $F1$ score $= 0.933717579$. We reached the 4th position in the leaderboard and the first place obtained $F1$ score $= 0.945205479$. This demonstrates that our strategy for solving this task was successful.

	Training Data					
	Given Data	Oversampling with SMOTE	Oversampling with RandomOverSampler	Oversampling with our implementation	Undersampling	Class weights
CNN with 3 folds	0,7761	0,7679	0,7777	0,7914	0,7729	0,7806
MLP (layer increasing, starting with 32 neurons)	0,7141 for nº of layers = 2	0,7184 for nº of layers = 5	0,7217 for nº of layers = 5	0,7199 for nº of layers = 4	0,7187 for nº layers = 3	0,7228 for nº layers = 2
MLP (constant layer, starting with 32 neurons)	0,7103 for nº layers = 2	0,7175 for nº layers = 19	0,7215 for nº layers = 19	0,7257 for nº of layers = 15	0,7168 for nº layers = 20	0,7222 for nº layers = 22
MLP (constant layer, starting with 64 neurons)	0,7245 for nº layers = 2	0,7229 for nº layers = 3	0,7308 for nº layers = 14	0,7261 for nº layers = 6	0,7197 for nº layers = 11	0,7248 nº layers = 15
K-Nearest Neighbor	0,6677 for nº neighbors = 1	0,7080 for nº neighbors = 7	0,7008 for nº neighbors = 47	0,6899 for nº neighbors = 19	0,7085 for nº neighbors = 10	-
Naive Bayes (Gaussian)	0,5608	0,5751	0,5589	0,6197	0,5600	-
Naive Bayes (Multinomial)	0,6258	0,6296	0,6284	0,6309	0,6270	-
Naive Bayes (Complement)	0,6289	0,6317	0,6275	0,6279	0,6267	-
Naive Bayes (Bernoulli)	0,4041	0,4259	0,4047	0,4043	0,4073	-
Random Forest	0,5348	0,6404	0,6500	0,6530	0,6494	-
SVM (kernel = Linear)	0,6200	0,6462	0,6317	0,5015	0,6272	-
SVM (kernel = RBF)	0,6982	0,6793	0,6764	0,6594	0,7022	-
SVM (kernel = Polynomial)	0,6268	0,6751	0,6358	0,5285	0,6566	-

Figure 4: Table with the F1 score of validation data obtained with cross validation using 5 folds except for CNN (3 folds).

2.2 Second Task

The second task consists in the segmentation of a particular type of eyespot that develops in butterflies of the bicyclus anynana species. These eyespots have a white center, surrounded by a black ring and then a golden ring. Our goal is to segment three distinct areas: the white ring, the black and gold rings combined, and the background. The classification is a multiclass problem for which the label is an integer from 0 to 2, denoting background, rings, and white center, respectively.

In this pixel classification task, we want to classify each pixel in a 30x30 RGB eyespot image using as features the set of pixels in a 5x5 neighborhood surrounding it.

This dataset is also imbalanced, since there is a big difference in the number of pixels in the three areas of the eyespots. The metric used by the teaching team to evaluate the submissions for this task was the Balanced Accuracy.

We follow the same strategy that we used for the previous task. However, it was necessary to make changes to some of the used methods, in addition to the fact that we have to generalize the classification problem to three possible output labels.

We need to create a new CNN due to the decrease in the size of the input images used on the network. In the first layer the model apply data augmentation as in the previous task but now we used the following Keras preprocessing layers: tf.keras.layers.RandomFlip and tf.keras.layers.RandomRotation. Then, it will standardize input values to be in [0, 1]. Our model consists of one convolution block with a max pooling layer (we chose this model by testing and verifying that Balanced Accuracy is maximized with this organization for the layers). These have the padding argument adapted to the 5x5 input shape. Then, we apply the same procedure as in the prior task.

Evaluation

To evaluate and compare the performances of the models tested, we used **cross validation**. The data set is split into k folds T_k (subsets with the same number of examples). One fold is used for testing and the others for training. After, the test fold rotates K times. We used the Balanced Accuracy score as metric.

In terms of data processing, we try some different modes: unchanged data, oversampling using SMOTE, oversampling with RandomOverSampler, oversampling by duplicating images of minority classes through rotations and tonal changes, undersampling and applying class weights.

For K-Nearest Neighbor we used cross validation to check the optimal value of number of neighbors, k, that maximize the Balanced Accuracy of the validation data. As, in the case of ties, the answer will be the class that happens to appear first in the set of neighbors, we only tested odd values for k (in range [1; 300]).

For MLP we also used cross validation to estimate the optimal value for the number of hidden layers and for the neuron distribution in the network. We tried the same different neuron distributions as in 2.1.

For that we wrote code in file *GR046.py* and obtain the results in figure 5.

	Training Data					
	Given Data	Oversampling with SMOTE	Oversampling with RandomOverSampler	Oversampling with our implementation	Undersampling	Class weights
CNN	0,7427	0,7747	0,8115	0,7988	0,7962	0,6767
MLP (layer increasing, starting with 32 neurons)	0,8066 for nº of layers = 4	0,8448 for nº of layers = 6	0,8470 for nº of layers = 4	0,8476 for nº of layers = 4	0,8377 for nº layers = 2	0,7430 for nº of layers = 3
MLP (constant layer, starting with 32 neurons)	0,7973 for nº of layers = 3	0,8224 for nº of layers = 15	0,8672 for nº of layers = 9	0,8500 for nº of layers = 9	0,85042 for nº layers = 12	0,7897 for nº layers = 6
MLP (constant layer, starting with 64 neurons)	0,7901 for nº layers = 3	0,8304 for nº of layers = 7	0,8783 for nº layers = 2	0,8538 for nº layers = 11	0,8469 for nº layers = 14	0,7591 for layers = 7
K-Nearest Neighbor	0,8789 for nº neighbors = 3	0,9267 for nº neighbors = 3	0,9159 for nº neighbors = 3	0,9038 for nº neighbors = 3	0,8380 for nº neighbors = 1	-
Naive Bayes (Gaussian)	0,7152	0,7164	0,6364	0,5748	0,7144	-
Naive Bayes (Multinomial)	0,3338	0,6375	0,5642	0,6352	0,3436	-
Naive Bayes (Complement)	0,5330	0,5571	0,5625	0,5212	0,5585	-
Naive Bayes (Bernoulli)	0,3333	0,3333	0,3333	0,3333	0,3333	-
Random Forest	0,3679	0,7726	0,7757	0,7218	0,6564	-
SVM (kernel = Linear)	0,7650	0,7853	0,7761	0,7353	0,7673	-
SVM (kernel = RBF)	0,8905	0,9008	0,8882	0,8889	0,8643	-
SVM (kernel = Polynomial)	0,3333	0,3333	0,3333	0,3333	0,3333	-

Figure 5: Table with the Balanced Accuracy of validation data obtained with cross validation using generally 5 folds, except for CNN and MLP (3 folds), and for K-Nearest Neighbor (50 folds).

When analyzing the obtained data, we found that the best option is to use K-Nearest Neighbor with $k = 3$ neighbors, for oversampling with SMOTE.

Statistical evaluation

To evaluate the performance, we applied the estimated predictor to an independent set of data with $n^{'} = 33800$ examples: x_{test}. We predict the outcome for x_{test} and save it in *ytest_Classification2.npy* file.

The teachers used the metric of Balanced Accuracy to compare the predictions $\hat{y}^{(i)}$ and the true values $y^{(i)}$: Balanced Accuracy = 0.832829355. We reached the 51th position in the leaderboard and the first place obtained Balanced Accuracy = 0.891721161. This demonstrates that our strategy for solving this task was reasonably good. To improve our solution we could have increased our training data by using data augmentation after the use of oversampling, by applying random image operations like flip, rotation, zoom, brightness and contrast.

3 Conclusion

Throughout this project we implement and train our machine learning approaches for Regression and Classification tasks.

For each problem, we implement and train our machine learning approaches using the training set that we split in training data and validation data. To estimate hyperparameters we used cross validation. Then, we pick only the model that maximizes the evaluation metric and apply it to the test set in order to perform the submission.

4 Bibliography

[1] Machine Learning Slides - Slides of Theoretical Classes 2022/2023, 1st Semester (MEEC), by Professor Jorge S. Marques;

[2] 1st Semester 2022/2023, Machine Learning, Project Statement Prepared by Prof. Margarida Silveira, Prof. Catarina Barata and Prof. Jorge Marques;

[3] Python 3.8.10;

[4] https://scikit-learn.org/stable/;

[5] https://keras.io/;

[6] https://www.tensorflow.org/tutorials/images/classification;