Machine Learning Project

Ana Matoso, 89787 | Inês Arana, 89805

Part 1 - Regression

First Problem

The training set we were given had 100 samples, $\mathcal{T} = ((x^{(1)}, y^{(1)}), ..., (x^{(100)}, y^{(100)}))$, with $x^{(i)} \in \mathbb{R}^{20}$, $y^{(i)} \in \mathbb{R}$. With this in mind, several predictors were tested, namely the linear model, in order to predict the outcome vector, $\hat{y}$.

The linear model is based on the fact that the outcomes depend linearly on each feature, as it can be seen by equation 1 where X is the design matrix and Y the column vector of outcomes.

$$y = \begin{bmatrix}1 & x^T\end{bmatrix} \beta \Leftrightarrow Y = X\beta \tag{1}$$

The coefficients (β) are calculated by minimizing the cost function (the sum of squared errors), in equation 2 where R is the regularization term (explained further ahead).

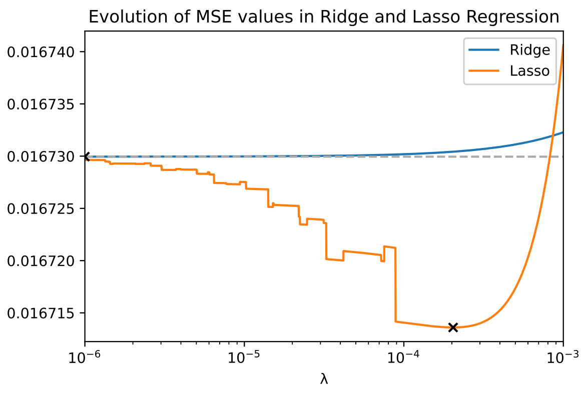
$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \|y - \hat{y}\| + R = \underset{\beta}{\operatorname{argmin}} \|y - X\beta\| + R \tag{2}$$

In the case of linear regression, $R = 0$ and thus $\hat{\beta}_{lr} = (X^T X)^{-1} X^T Y$.

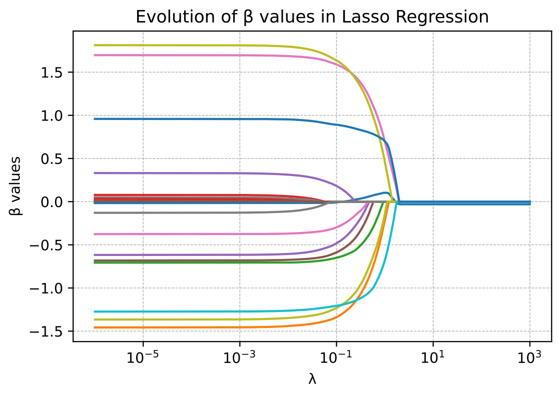
For the case of Ridge and Lasso Regression, $R \neq 0$ and therefore yields a penalization for large coefficients so it represents the data more. In the case of Ridge Regression, $R = \lambda \|\beta\|^2$ thus $\hat{\beta}_{ridge} = (X^T X + \lambda I)^{-1} X^T Y$ [1].

On the other hand, for the case of Lasso Regression, $R = \lambda \|\beta\|_1^2$, where $\| \cdot \|_1$ represents the l_1 norm which is the sum of the magnitudes of the coefficients [2]. This optimization problem cannot be solved by a linear system of equations unlike the Ridge Regression. Therefore it needs to be solved numerically. In order to do that, we used the scikit-learn Python library which uses the coordinate descent as the algorithm to fit the coefficients.

To choose λ, we compared the mean squared errors by using 5-fold cross validation for Ridge and Lasso regression, having tested values between 10^{-6} and 10^3. The graph of the mean squared error as a function of λ is present in figure 1a. In that same figure, the crosses represent the minimum values of the error of each method. As it can be seen the minimum value is achieved using the Lasso Regression when $\lambda \approx 0.0002$. Moreover, the dashed line represents the mean squared error when using the Linear Regression (without regularization) which, as expected, is where both lasso and ridge regression tend to when $\lambda \to 0$. As a further analysis we plotted the evolution of the coefficients with the value of λ (figure 1b), and we detected that for $\lambda \approx 0.0002$, in the Lasso Regression, there are three features whose β value is smaller than 0.01 which means that they don't have much influence on the result in the case where the error is minimum.



(a) Mean Squared Error as a function of lambda in ridge and lasso regression



(b) β values as a function of the regularization parameter in Lasso Regression

As it was said previously, we used the k-fold cross validation method in order to assess and compare each of the predictor's performance as well as to tune hyperparameters. This was done by calculating the mean squared error (MSE) of the k times we tested with the unseen fold of the training data. It is important to note that we divided the error of each fold by the size of fold (s) in order to allow comparison between cross validations with different k values thus obtaining an average point-wise value of the error. The equation we used to compare model performance can be seen in equation 3.

$$MSE = \frac{1}{k}\sum_{i=1}^{k}\frac{SSE_i}{s} \tag{3}$$

Additionally, some other models were tested: Linear Support Vector Regression (SVR) with an epsilon of 0.07 and 100000 as maximum iterations; Stochastic Gradient Descent (SGD) with 'epsilon_insensitive' as loss function and an epsilon of 0.05; Gaussian Processes (GP) in which the kernel used was the dot product. These three models with the respective parameters had comparable results with the linear regression ones.

Furthermore, other models were tested such as K-Nearest Neighbours, Support Vector Machines and Decision Tree however they yielded errors that were over 100 times greater than the ones that were already tested (10.2, 9.9, 18.7, respectively) so their results will not be presented amongst the other results.

Results

Using the above mentioned predictors, we performed a k-fold cross validation using $k = 5$ and $k = 10$ for each predictor. The results for 5-fold and 10-fold cross validation are displayed in table 1 (the minimum values are in bold) and in figure 2 it is present the results in a bar plot. It is important to note that for Lasso Regression, $\lambda \approx 0.0002$ and for Ridge Regression $\lambda \approx 10^{-6}$.

Table 1: Mean squared errors for each predictor using 5- and 10-fold cross validation

Predictor	**5-fold**	**10-fold**
Linear Reg.	0.0167299	0.0162493
Ridge Reg.	0.0167299	0.0162493
Lasso Reg.	**0.0167136**	**0.0162237**
Linear SVR	0.0176589	0.0175636
SGD	0.0201422	0.0176588
GP	0.0167297	0.0162493

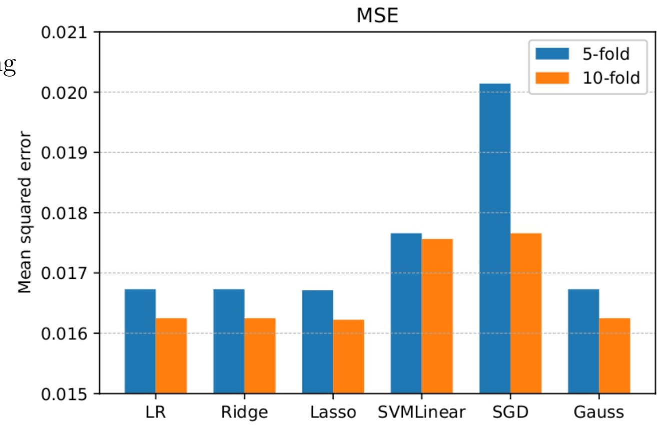


Figure 2: Mean Square Error of each predictor

As it can be seen by table 1, the mean squared error is the lowest when the Lasso Regression is used given the lambda parameter found previously. Additionally, when comparing the error between the linear regression and Ridge Regression they are very similar, which is what we expect due to the reason we explained in the previous section. It can also be seen that the Gauss Processes also yields similar values to the linear regressor ones which might be due to the fact that the data might have a somewhat Gaussian distribution.

Given all these results, the predictor we chose is the Lasso regressor with the value of λ of equation 4 since it was the one who performed best.

$$\lambda = 0.00020371870326560236 \tag{4}$$

Second Problem

In the second part of the problem, we were told there are at most 10% of outliers within the data set and as such they needed to be removed. In order to do that, we compared four outlier detection methods: Local Outlier Factor (LOF), Isolation Forest, Minimum Covariance Determinant (MCD) and One Class SVM. We also used one clustering algorithm, DBSCAN. Then, for each outlier detection method we tried several pedictors: Linear Regressor, Ridge Regression, Lasso Regression, Linear SVR, Stochastic Gradient Descend, Gauss Processes, Elastic Net, Least angle Regression, Least Angle Regression with Lasso, Orthogonal Matching

Pursuit, Bayesian Ridge Regression, Linear SVM and RANSAC. Note that for every outlier detector used, a verification of whether it removed 10% or less of outliers was performed.

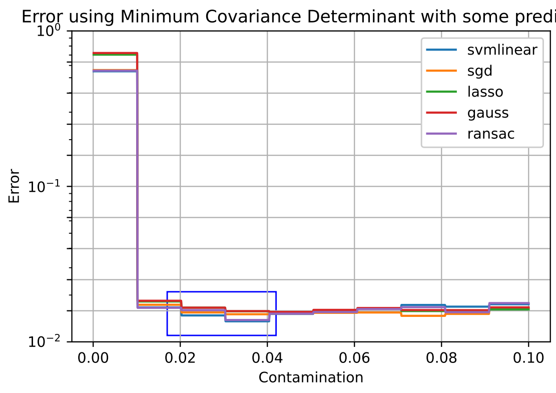
Given that most of these methods have hyperparameters associated to them we also did a fine tuning of them for each combination of predictor and outlier detection method with each value of contamination. In table 2 it is present the values of parameters used, as well as the function in which they are used.

Table 2: Parameter's range of values

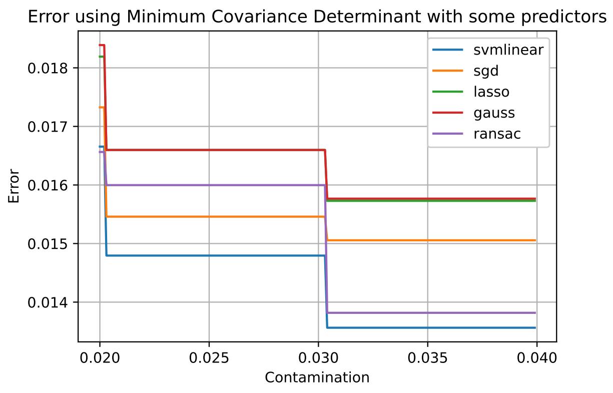
Hyperparameter	**Values**	**Function**
contamination	`linspace(0.0001,0.1,1000)`	LOF, MCD, Isolation Forest
eps	`linspace(3,5,1001)`	DBSCAN
nu	`linspace(0.01,1,1000)`	One-Class SVM
lambda	`logspace(-6, 0, 100)`	Lasso Regression

Results

When analysing the results of the outlier detection methods, minimum covariance determinant and local outlier factor, shown in figures 3 and 4, both show a similar behaviour with the change in contamination value: the error decreases greatly after a certain contamination value, steadying after that. This shows that, in these two outlier detection methods, true outliers were found and removed, even if in different quantities (due to the different contamination value the error drop occurs at).

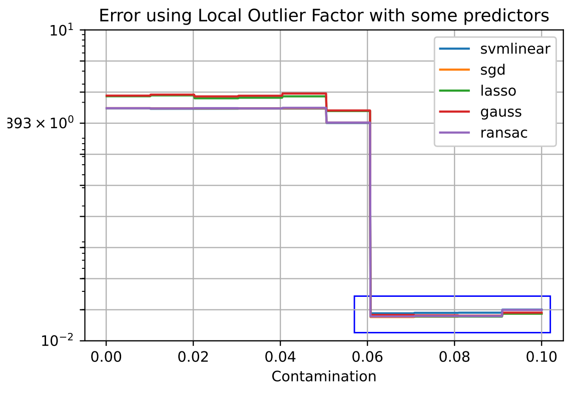


(a) Error as a function of contamination

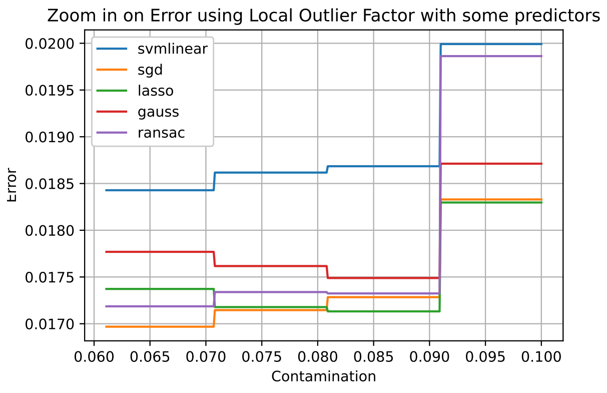


(b) Zoom in on the region inside the blue box

Figure 3: Error as a function of contamination when using minimum covariance determinant method to detect outliers



(a) Error as a function of contamination



(b) Zoom in on the region inside the blue box

Figure 4: Error as a function of contamination when using local outlier factor method to detect outliers

Moreover, when using isolation forest or one class SVM to detect outliers, the error for the different predictors was much higher than the other two methods already mentioned.

In fact, when using isolation forest (figure 5) the error increases greatly with the contamination which means that it is possible that this algorithm might be removing valid data points. When looking at figure 6, it can be seen that the value of the error is very unstable which makes one class SVM an untrustworthy algorithm to chose (though it has a huge drop in error for a very specific nu which reaches similar values of error as the first two outlier detection methods).

Nevertheless, both of these methods are known to perform better on larger data sets therefore since ours only had 100 samples, perhaps that was the reason why it resulted in poorer outlier detection and consequent higher error values when compared to the other methods. Additionally, in the case of isolation forest, although considered a good method for outlier removal, it uses an anomaly score to determine whether or not a data point is an outlier, which can compromise accuracy in some cases [3, 4].

What's more, concerning DBSCAN, it showed no difference when changing the eps parameter (within the range mentioned), having stayed always with an error of 1.66 which is far from the other methods stated.

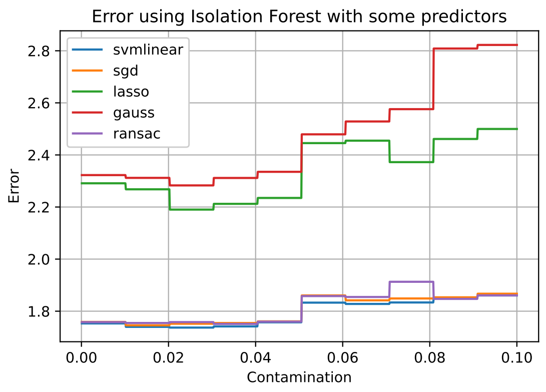


Figure 5: Error as a function of contamination when using isolation forest method to detect outliers

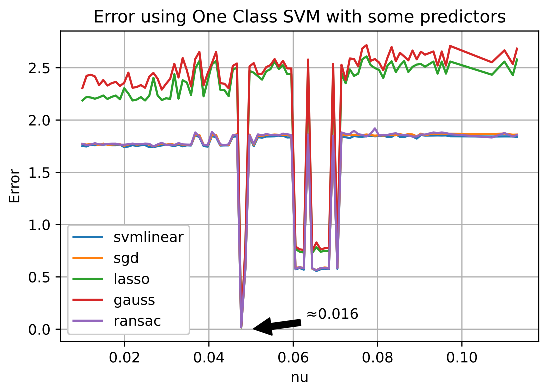


Figure 6: Error as a function of nu when using one class SVM method to detect outliers

After comparing the different outlier detection methods and predictors with the different hyperparameters, the model with the lower error when performing a 5-fold cross validation was the SVM Linear predictor, using Elliptical Envelope as the Minimum Covariance Determinant method to remove outliers and considering a contamination of around 0.035 (see figure 3b).

The chosen outlier detection method, Elliptical Envelope, has a simple principal: creating an ellipse around the data set, and exclude the points outside this region. This is done by fitting a Gaussian into the data and excluding the data points not well fit by these functions. This is done by finding a robust version of the covariance matrix with the smallest determinant, hence the name Minimum Covariance Determinant, that still contains a large portion of the data, defined by the contamination value. Given that this was the model that better fitted our data set, it suggests that the data has a Gaussian distribution [5, 6].

As for the predictor model used, the SVM Linear provided the best results, minimizing the error. This predictor consists of the support vector regression model using a linear kernel. Unlike the linear regression presented previously, this model uses a threshold value, ϵ, to define the margins in which the error is acceptable (equation 5). Additionally, the objective is to minimize the l_2-norm of the coefficient vector: min $\frac{1}{2}||w||^2$. The parameter ϵ was tuned to obtain the best model, that is to minimize the norm [7].

$$|y_i - w_i x_i| < \epsilon \tag{5}$$

However, it is worth commenting that, when using minimum covariance determinant method to detect and remove outliers, the predictors used show very similar behaviours for the range of contamination values used.

Part 2 - Classification

First Classification Task

In this problem, we were given a set of images of faces in order for us to develop a machine learning algorithm that can differentiate between male and female faces. In this data set, the two classes are fairly balanced (57% to 43%). Provided this problem, we decided to compare three types of algorithms: Convolutional Neural Networks (CNN), Multilayer Perceptron (MLP) and Support Vector Machine (SVM).

Regarding the first two methods mentioned, the images were reshaped to a 50x50 matrix and the y vector was put in one-hot encoding format. Plus, to compile the model, the adam optimizer was used since it was the method that showed to be more efficient in finding an optimum point [8]. The loss function used was the categorical crossentropy and an early stopping criteria was used in order to avoid overfitting. More specifically, the validation accuracy was monitored with a patience of 10 epochs, saving the weights which yielded the best validation accuracy. It is also important to note that we divided the data set in training set and validation set, corresponding to 80% and 20% of the original data set, respectively. What's more, a limit of 100 epochs was (empirically) chosen since it was seen to be enough in order to have sufficient number of epochs to learn.

Convolutional Neural Networks

In order to design a Convolution Neural Network, there needs to be an input layer, at least one convolutional layer and one output layer. The convolutional layer is nothing more than the application of a 2D filter (or kernel) to the input matrix.

In order to determine the number of hidden layers and what each one does, we tried many combinations of convolutional layers, max pooling layers and dense layers. After several attempts we found that the architecture of figure 7 was the one which yielded the highest balanced accuracy in the validation set. This includes a first layer of rescaling the greyscale to values between 0 and 1, two convolutional layers followed by maxpooling layers, a flattening layer that transforms the tensor (3D matrix) into a column vector and a fully connected layer of 32 neurons and finally the classifier. Moreover, the hidden layers all have ReLU as the activation function and the output layer has the softmax as activation function. Furthermore, we used the mini-batch mode with a batch size of 50.

Multilayer Perceptron

In order to design a multilayer perceptron we tried to add several fully connected (Dense) layers with a varying amount of neurons. The model which yielded the best result was the one with four dense layers where the first layer has 100 neurons, the second 50, the third 30 and the last one 2 (since we have two classes). It is important to note that the hidden layers all have ReLU as the activation function and the output layer has the sigmoid as activation function. Additionally, a batch size of 100 was used.

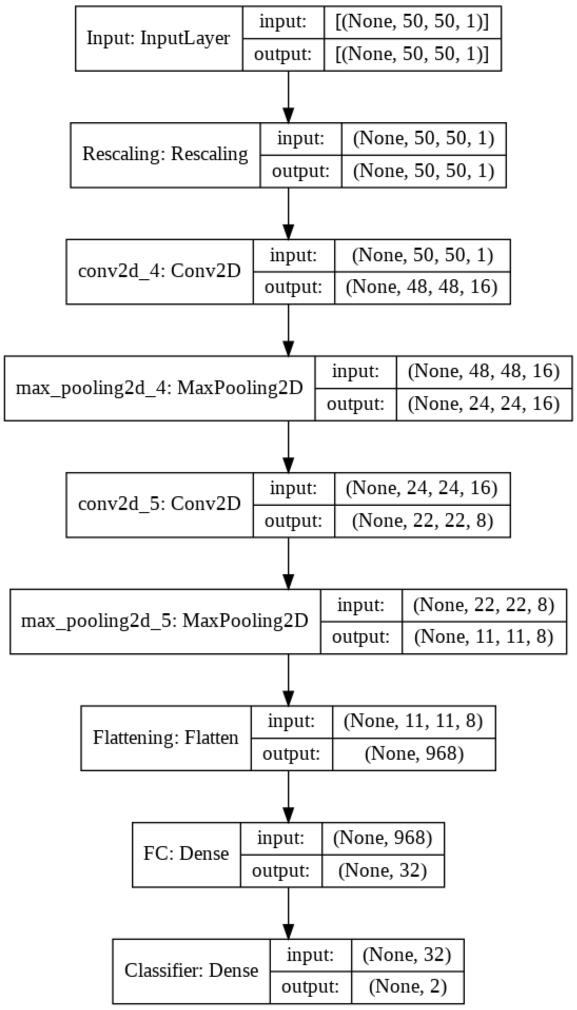


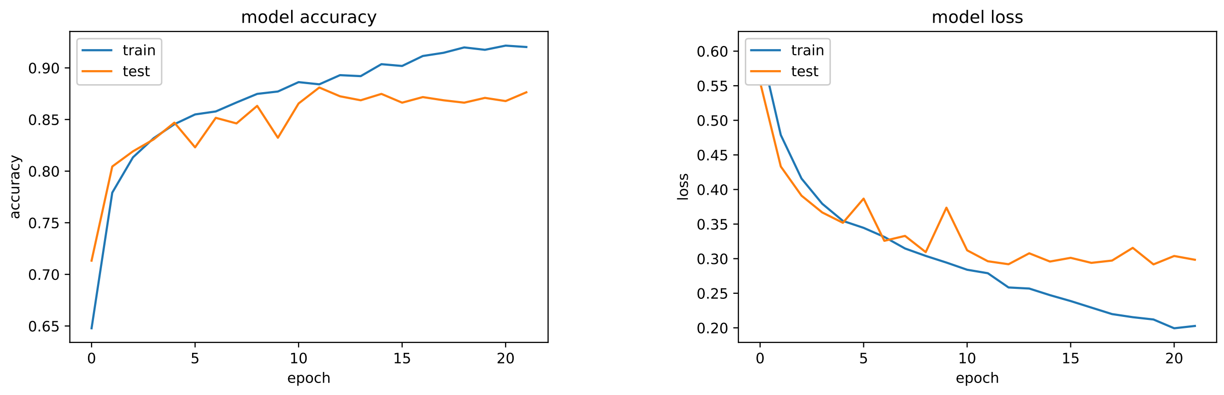
Figure 7: Layers of the CNN

Support Vector Machines

To use SVM, we used the respective scikit-learn library. We tested different kernels, namely radial basis function (RBF), polynomial and linear, and different values of C (1,2 and 10), and two different gammas (scale and auto).

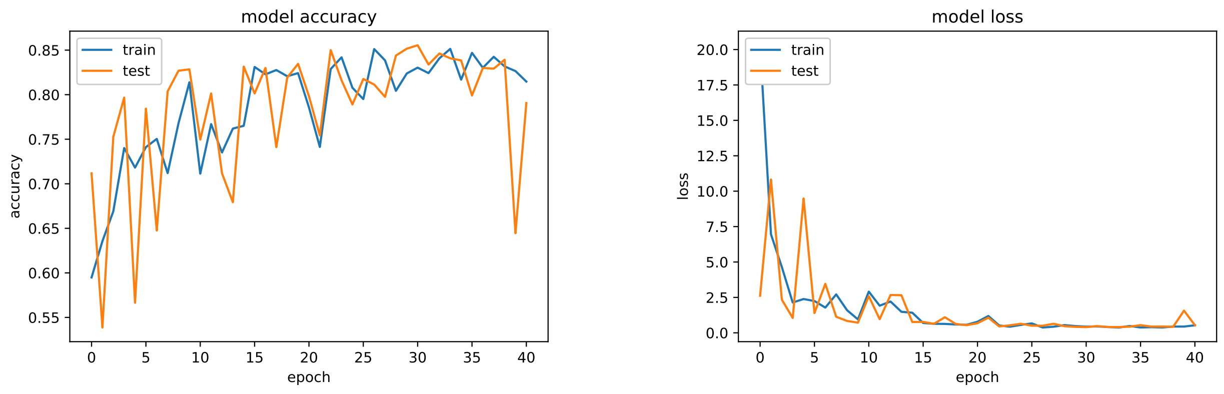
Results

Regarding the results, for the first two algorithms it was plotted the accuracy and the loss along the epochs in figures 8 and 9. It is important to note that the minimum values of the loss in the validation (test) set go down to about the same value in both algorithms which is around 0.3.



(a) Evolution of Accuracy across epochs (b) Evolution of Loss across epochs

Figure 8: Results when using a Convolution Neural Network



(a) Evolution of Accuracy across epochs (b) Evolution of Loss across epochs

Figure 9: Results when using a Multilayer Perceptron

As it can be seen in figure 8, when using the CNN model, the accuracy and loss graphs (in the test set) tend to stabilize as the epochs increase which is good since that means that there is no overfitting happening. In addition, when using this model, the balanced accuracy reached was 88.7%.

Concerning the MLP model in figure 9, it can be seen that even though the loss stabilizes, the accuracy in the test set is very unstable. In this algorithm, the best model (best weights) only yielded 67% of balanced accuracy which is quite low, especially when comparing to the CNN model.

Moreover, in both models, it can be seen that the limit of 100 epochs is never reached which means that the algorithms stop due to the early stopping criteria mentioned above which prevents overfitting.

Regarding the last algorithm, Support Vector Machine, table 3 presents the balanced accuracy when changing the kernel and gamma parameters. The results when changing the C parameter are not shown since C=1 yielded always better results than when C=2 or C=10 so the results in the table are for C=1. Note that the best result is in bold.

Table 3: Balanced Accuracy for Support Vector Machines

Gamma / Kernel	Auto	Scale
Linear	0.7884038491506935	0.7884038491506935
RBF	0.6391741071428572	0.852221308103661
Polynomial	0.8667291189375994	**0.8786427204187663**

As it can be seen, given the best hyperparameters of support vector machine, interestingly enough, this algorithm reaches similar values of the CNN model described above even though it ”sees” the images as flat vectors whilst the CNN ”sees” them like 50x50 pixel images.

In conclusion, given the balanced accuracy results presented, the algorithm we chose was the CNN model described above due to the fact that it had the highest balanced accuracy.

Second Classification Task

In this task, unlike in the previous section, we have a multi-class problem with 4 classes which correspond to the ethnicity of the people in the database. For this, we built on the models that were described in the last task and did some tweaks in their hyperparameters to see if it would yield better accuracy. Additionally, Random Forest and k-Nearest Neighbours (k-NN) models were also evaluated. In the end, an ensemble of classifiers was used to improve accuracy of the final model. [9]

The CNN and MLP models had as input 50x50 images unlike the SVM, Random Forest and k-NN models which received a column vector with size 2500. For the first two methods, the y vector was put in one-hot encoding format.

Additionally, regarding the neural network models (CNN and MLP) to compile these models, the adam optimizer was again used due to the reason stated in the previous task. The loss function used was also the categorical crossentropy and an early stopping criteria was used in order to avoid overfitting. More specifically, the validation accuracy was monitored with a patience of 7 epochs and still saving the weights which yielded the best validation accuracy as before.

In this task, we also divided the data set in training set and validation set using the same ratio as in the previous section. To ensure an accurate representation of each class, we verified the class distribution in the complete data set and compared it with the chosen training and validation sets, and concluded it was similar. Furthermore, a limit of 100 epochs was (empirically) chosen.

Convolutional Neural Network

In this task, we tested several convolutional neural networks, using as a base the one used in the last task, which can be seen in figure 7. After several attempts, the model in which the highest validation accuracy value was obtained was number 6 (according to the numbering in the code), which is represented layer by layer in figure 10.

This model has a first layer of rescaling the greyscale to values between 0 and 1, followed by a 2D convolutional layer with kernel size of 5x5 and 32 output filters and a 2D maxpooling layer, using a 2x2 kernel. Then, another 2D convolutional layer, this time with a 3x3 kernel and 16 output filters, was used, again followed by a 2D maxpooling layer, with the same 2x2 kernel. Plus, the convolutional layers mentioned all have zero padding around the image (padding=’same’). After this, a dropout layer with a rate of 0.3 is used to prevent overfitting. Then there is a flattening layer that transforms the tensor (3D matrix) into a column vector and fully connected layer of 100 neurons and finally the classifier. Moreover, the hidden layers all have ReLU as the activation function and the output layer has the softmax as activation function. Furthermore, a batch size of 32 was used.

In table 4, most of the models tested as well as the validation accuracy obtained for each one, are presented. Note that, since each model in the table was built based on model number 6, changes are shown compared to this one (again the numbering of each model is done based on the submitted code). The

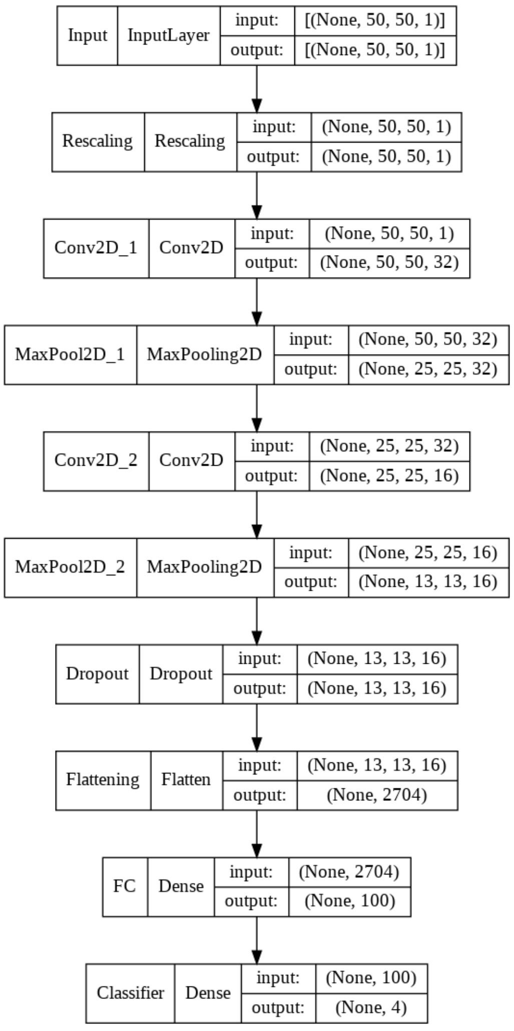

Input | InputLayer | input: [(None, 50, 50, 1)] | output: [(None, 50, 50, 1)]
Rescaling | Rescaling | input: (None, 50, 50, 1) | output: (None, 50, 50, 1)
Conv2D_1 | Conv2D | input: (None, 50, 50, 1) | output: (None, 50, 50, 32)
MaxPool2D_1 | MaxPooling2D | input: (None, 50, 50, 32) | output: (None, 25, 25, 32)
Conv2D_2 | Conv2D | input: (None, 25, 25, 32) | output: (None, 25, 25, 16)
MaxPool2D_2 | MaxPooling2D | input: (None, 25, 25, 16) | output: (None, 13, 13, 16)
Dropout | Dropout | input: (None, 13, 13, 16) | output: (None, 13, 13, 16)
Flattening | Flatten | input: (None, 13, 13, 16) | output: (None, 2704)
FC | Dense | input: (None, 2704) | output: (None, 100)
Classifier | Dense | input: (None, 100) | output: (None, 4)

Figure 10: Layers of CNN model

changes made consisted of replacing one layer with another, adding layers or simply modifying an existing layer. For example, in models 9 and 10, the Dropout layer used in the previous models was removed, being replaced by SpatialDropout and GaussianDropout layers, respectively, in order to analyse its the effect on the model. However, since the results obtained weren't better than the original model, these layers weren't used in the final model.

Table 4: Changes made to each used CNN model compared with the best one

Model	**Changes**	**Balanced Accuracy**
6	-	**83.1%**
6.1	20% of Dropout instead of 30%	81.8%
6.3	40% of Dropout instead of 30%	82.4%
1	different number of output filters and kernel size for Conv2D layers	81.0%
5	different number of output filters for Conv2D and MaxPool2D layers	81.0%
7	add Conv2D and MaxPool2D layers, with different number of output filters	79.8%
12	add Conv2D and MaxPool2D layers, with different number of output filters and kernel size	79.7%
8	use LocallyConnected2D instead of first Conv2D layer	79.8%
9	use SpatialDropout2D instead of Dropout layer	78.1%
10	use GaussianDropout2D instead of Dropout layer	78.1%

Other Models

The first model tested after CNN was a Multilayer Perceptron. To test this model, several combinations of layers with different parameters were tested. The model with the best result had five dense layers with 1300 neurons, then 500, 200, 50 and finally 4 (number of classes). Like in the last section, all hidden layers have ReLU as the activation function and the output layer has the softmax as activation function. Additionally, a batch size of 50 was used.

The following method was a Support Vector Machine (SVM). As in the previous section, we tested different kernels, namely radial basis function (RBF), polynomial and linear, and different values of C (1, 2, 3, 4 and 5), and two different gammas (scale and auto).

Moreover, a random forest model was also tested. This model uses ensemble learning by constructing a set of random decision trees and then classifying the input data based on the class that is selected in most of these trees (mode). Nevertheless, we had to select a combination of hyperparameters in order to get reasonable results. The combination we found that got the best result was when the criterion was changed to entropy, the maximum depth was set to 10 and the number of estimators was set to 1000.

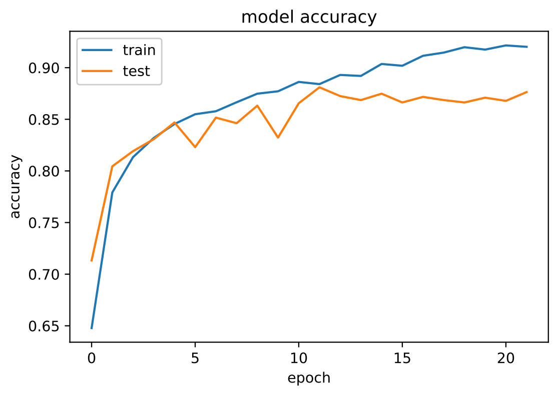
Another model we tested was k-Nearest Neighbours. This model for each input data point checks what are the labels of the k nearest points (in feature space), and assigns to that data point the most frequent label found within those k neighbours (majority vote). In our case, due to the size of the training set, we increased the number of neighbours from 5 (default) to 500 (having tested also 100, 200, and 1000).

Ensemble Learning

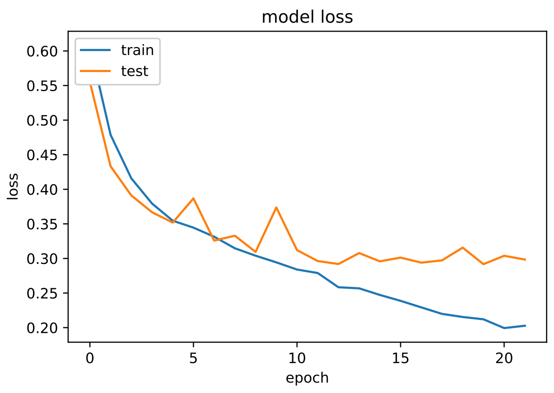
Given all these results, in order to increase the balanced accuracy of the result, we decided to use an ensemble of classifiers in order to make a prediction, more specifically a stacking method. What this means is that for each type of classifier we calculated a prediction which was then used in a voting system so that for each image, the most voted class among all classifiers was the chosen one. With this in mind, we tested some combinations of classifiers seeking the combination that yielded the best balanced accuracy.

Results

To analyse the chosen CNN model (model 6), the balanced accuracy was calculated (seen in table 4) as well as the evolution of accuracy and loss across epochs which can be seen in figure 11. When looking at this figure, the accuracy and loss graphs (in the test set) tend to stabilize as the epochs increase which is good since that means that there is no overfitting happening, showing the same behaviour described in the previous section and shown in figure 8.



(a) Evolution of Accuracy across epochs



(b) Evolution of Loss across epochs

Figure 11: Results when using a Convolutional Neural Network

Furthermore, the confusion matrix for the validation set is presented in figure 12. This matrix shows the distribution of predicted labels compared to the true value of the label. The highest value occurs for class 0, which is expected since this is the class with highest representation in the dataset. In addition, it can be seen that there is no case where it missclassifies a class more than it rightfully labels it.

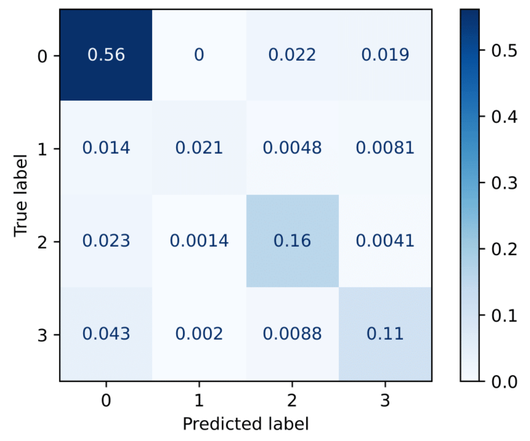
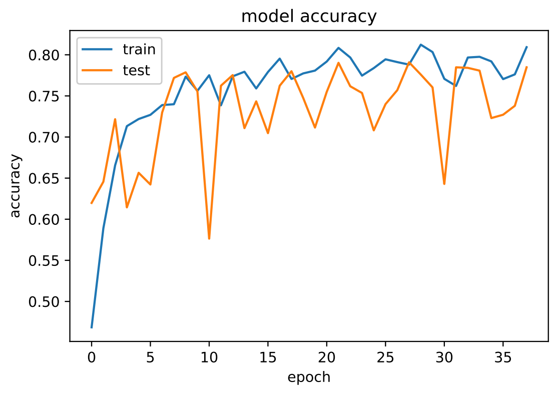
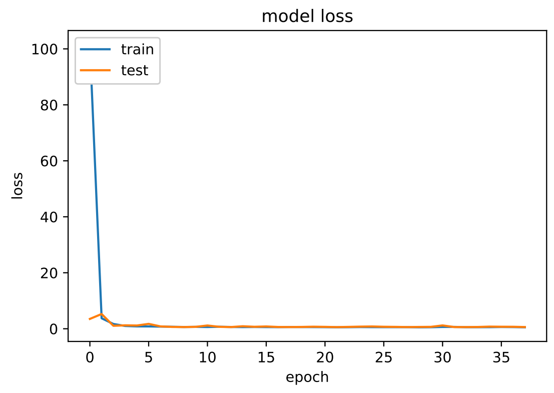


Figure 12: Confusion Matrix of CNN model 6

Regarding the MLP model, the evolution of accuracy and loss across epochs can be seen in figure 13, in which it can be seen that the behaviour of this model is similar to the MLP used in the previous section. The accuracy of the test set is very unstable with the loss stabilizing quickly. Additionally, the highest balanced accuracy value obtained was 73.3%, which is very low compared to the CNN model presented previously.



(a) Evolution of Accuracy across epochs



(b) Evolution of Loss across epochs

Figure 13: Results when using a Multilayer Perceptron

The results obtained when using SVM are in table 5. As it happened in the last section, C=1 always yielded better results than the other values of C so the results in the table are for C=1. The best result is in bold. In this task, the best SVM model was using an RBF kernel with gamma, the kernel coefficient, set to 'scale', which is the default value.

Table 5: Balanced Accuracy for Support Vector Machines

Kernel \ Gamma	Auto	Scale
Linear	0.6513559558419911	0.6513559558419911
RBF	0.5343303874915023	**0.8052884659308603**
Polynomial	0.7024362093015344	0.7385982807482376

In table 6, the results obtained for the best model of each type of classifier are presented. As it can be seen, the MLP model had the worst results, with a 73.3% balanced accuracy. This is in accordance with the instability of the model mentioned previously. Furthermore, the Random Forest model yielded a balanced accuracy of 83.0%, which is comparable to the best value obtained with the CNN. This could be a result of the ensemble learning used to compute the model, resulting in a more reliable classifier. Although the SVM and k-NN models didn't yield the best results, they still yielded rather good results which are comparable to the CNN and Random Forest ones.

Table 6: Balanced Accuracy obtained for the best model in each type of classifier

Model	**Balanced Accuracy**
CNN	**83.1%**
MLP	73.3%
SVM	80.5%
Random Forest	83.0%
k-NN	80.2%

After this, we did several combinations of all the models in the ensemble algorithm reaching the conclusion that the best result was the one which combined the following models: CNN6, CNN6.3, random forest, k-nearest neighbours and support vector machine. Using this ensemble method we achieved a balanced accuracy of 89.42526807110139% in the validation set. This result is better than all the results stated above which is expected since one advantage of using ensemble methods is that its accuracy is greater than each of the methods that are in it alone [10]. Therefore, this ensemble method with the five models mentioned above was the method chosen to make the prediction.

References

[1] Donald E. Hilt and Donald W. Seegrist. *Ridge, a computer program for Calculating Ridge Regression estimates.* Dept. of Agriculture, Forest Service, Northeastern Forest Experiment Station, 1977.

[2] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288. DOI: 10.1111/j.2517-6161.1996.tb02080.x.

[3] Eryk Lewinson. *Outlier Detection with Isolation Forest.* 2018. URL: https://towardsdatascience.com/isolation-forest-a-tree-based-algorithm-for-anomaly-detection-4a1669f9b782. (accessed: 28.10.2021).

[4] Jason Brownlee. *One-Class Classification Algorithms for Imbalanced Datasets, Imbalanced Classification.* 2020. URL: https://machinelearningmastery.com/one-class-classification-algorithms/. (accessed: 28.10.2021).

[5] Andreas C. Müller. *Outlier Detection.* 2020. URL: https://amueller.github.io/aml/03-unsupervised-learning/03-outlier-detection.html#id1. (accessed: 06.11.2021).

[6] scikit-learn developers. *Elliptic Envelope.* 2021. URL: https://scikit-learn.org/stable/modules/generated/sklearn.covariance.EllipticEnvelope.html. (accessed: 06.11.2021).

[7] Tom Sharp. *An Introduction to Support Vector Regression (SVR).* 2020. URL: https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2. (accessed: 28.10.2021).

[8] Sanket Doshi. *Various optimization algorithms for training neural network.* Aug. 2020. URL: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6.

[9] Colleen Dunlap. *Non-Binary Image Classifying CNN Tutorial (5 categories) with Keras.* 2019. URL: https://medium.com/analytics-vidhya/non-binary-image-classifying-cnn-tutorial-5-categories-df431449c7d5. (accessed: 10.10.2021).

[10] Ludmila Kuncheva and Chris Whitaker. "Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy". In: *Machine Learning* 51 (May 2003), pp. 181–207. DOI: 10.1023/A:1022859003006.