Aprendizagem Automática Final Report

Bernardo Santos - 93434 and Vasco Pearson - 97015

Instituto Superior Técnico

Professors: Jorge dos Santos Salvador Marques and Chrysoula Zerva

---

1 Introduction

This project was made in the scope of the Machine Learning course. With it, we not only learned fundamental knowledge about machine learning in general, but also applied techniques on concrete applications. This project was divided in two main parts, regression and classification.

Concerning regression, we applied several concepts like the cross-validation method to several models, namely, linear regression, ridge regression and lasso regression. Furthermore, we used the z-score and the interquartile range concepts to interpret boxplots and identify outliers.

Finally, in the classification we used convolutional neural networks that are most commonly applied to computer vision applications, where we train several models and select the best one, depending on the classification task.

2 Regression Part 1

In this problem we where given a training set with 100 examples. $T_r = \{(x^{(1)}, y^{(1)}), ..., (x^{(100)}, y^{(100)})\}$, with $x^{(i)} \in \mathbb{R}^{20}$, $y^{(i)} \in \mathbb{R}$ and we wish to train predictors $\hat{y} = f(x)$. We trained a Linear Regression model, a Ridge Regression model and a Lasso Regression Model and in every model we use K-Fold Cross Validation to assess the results of our model on a new independent data set.

2.1 Linear Regression

A Multiple Linear Regression model is the simplest model that we will fit to this data. A Linear Regression model is a model that explains the scalar response of a dependent variable to one or more independent variables (in our case we have more than one independent variable). In the OLS (ordinary least squares) aproach, the model is fit by minimizing the total loss in the training set. The loss function is the following:

$$L_{OLS}(\hat{\beta}) = \sum_{i=1}^{n} (y_i - x_i\hat{\beta})^2$$

2.2 Ridge Regression

Ridge Regression is a model tuning method that is used to estimate the coefficients of multiple regression models when the independent variables are highly correlated, in other words, the data suffers from multicollinearity. Ridge Regression performs L2 regularization by adding an L2 penalty, which equals the square of the magnitude of coefficients. All the coefficients are shrunk by the same factor so none are eliminated. In this case, the OLS loss function is augmented in such a way that we not only minimize the sum of squared residuals but also penalize the size of the parameter estimates, in order to shrink them towards zero. The loss function is the following:

$$L_{Ridge}(\hat{\beta}) = \sum_{i=1}^{n} (y_i - x_i\hat{\beta})^2 + \lambda \sum_{j=1}^{m} \hat{\beta}_j{}^2$$

The λ parameter is the regularization penalty. Setting λ to 0 is the same as using the OLS, while the larger its value is, the stronger is the coefficients' size penalized. As λ becomes larger, the variance decreases, and the bias increases.

2.3 Lasso Regression

Lasso Regression is similar to Ridge Regression in the way that it is well-suited for models showing high levels of muticollinearity and also adds a penalty for non-zero coefficients, but unlike ridge regression which penalizes the sum of squared coefficients (L2 penalty), lasso penalizes the sum of their absolute values (L1 penalty).

$$L_{Lasso}(\hat{\beta}) = \sum_{i=1}^{n}(y_i - x_i\hat{\beta})^2 + \lambda \sum_{j=1}^{m}|\hat{\beta}_j|$$

This type of regularization can result in sparse models with few coefficients because some coefficients can become zero and be eliminated from the model. Larger penalties result in more coefficient values closer to zero, which is the ideal for producing simpler models.

2.4 Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. Assuming that some data is Independent and Identically Distributed, we assume that all samples come from the same generative process and that the generative process has no memory of past generated samples. It is mostly used in machine learning in order to estimate the accuracy of a machine learning model on unseen data. In other words, to use a limited sample in order to estimate how the model is expected to react in general when used to make predictions on data not used during the training of the model.

In our models we use K-Fold Cross Validation, which is a type of Cross Validation that consists of splitting our data into k subsamples of equal size. Of those k subsamples, one is used as a validation set where we are going to evaluate our model, while the other $k-1$ are used to train the model. The method has the following steps:

- Shuffle the dataset randomly;
- Divide the dataset into k groups;
- For each group:

  Take the group as a hold out or a validation data set;

  Take the remaining groups as a training data set;

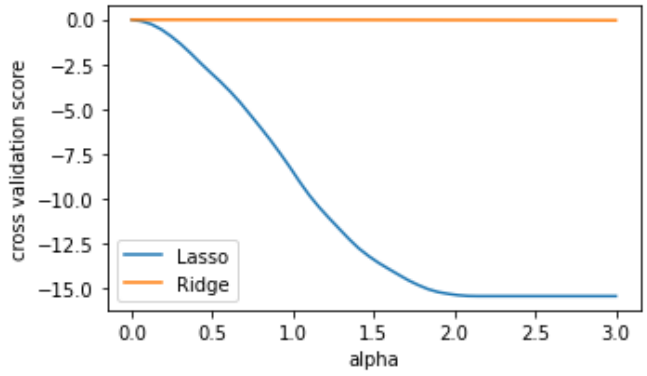
  Fit a model on the training set and evaluate it on the validation set;

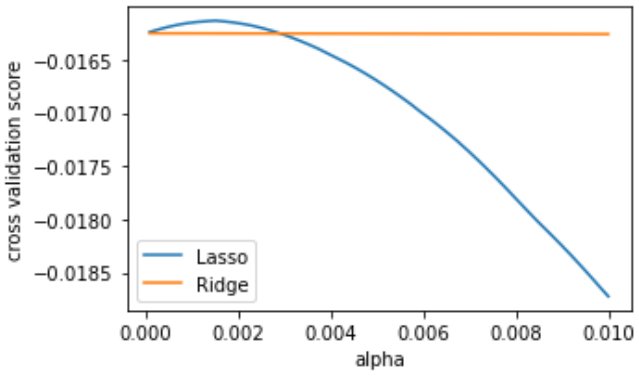
  Save the evaluation score and discard the model.
- Summarize the quality of the model using the sample of model evaluation scores.

Importantly, each observation in the data sample is attributed to an individual group and stays in that group throughout the process. So, each sample is given the opportunity to be used in the validation set 1 time and used to train the model $k-1$ times.

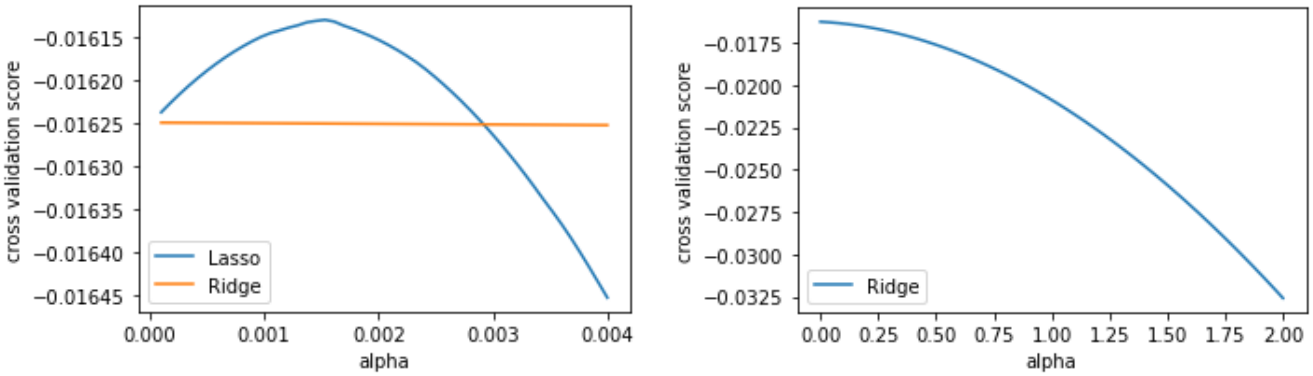
2.5 Results and interpretation

To understand what model was the best, we plotted (with *pyplot* from *matplotlib*) the cross validation score (using 10-fold cross validation) as a function of alpha (or lambda as we called it before), the regularization penalty. This was useful to get answers on what was the best model between the ridge and the lasso regression, and what was the value of alpha that optimized the regression. The results are shown bellow.





From these two plots we can conclude that lasso regression has a higher cross validation score if the right alpha is chosen (somewhere between 0 and 0.002). Due to this, we made a third plot to get a more accurate idea of the range in which that alpha value is located. We also plotted the values of alpha for the ridge regression, since we couldn't understand if the values where increasing or decreasing in the previous plots.



Here we can see that the optimal alpha for lasso regression is between 0.001 and 0.002, and that the best alpha for ridge regression is 0 (which makes it equal to linear regression). By using the python functions *RidgeCV* and *LassoCV* (from *sklearn.linear_model*) we where able to apply ridge and lasso regression to the data and evaluate the results using 10-fold cross validation. The advantage of using these functions is that they give us the optimal alpha value for each regression. We obtained

$$\alpha_{Lasso} = 0.0015771543086172347$$

$$\alpha_{Ridge} = 0$$

as we expected.

To confirm our results, we used $KFold$ from *sklearn.model_selection* to evaluate each of the three models. To make this task easier we created three functions: *Get_score* that determines the score of the model given as parameter; *Predict_val* that predicts the dependent variable of the validation set given as parameter; And *Average* that returns the average of a list. Using a for loop, $KFold$ from *sklearn.model_selection* and *mean_squared_error* from *sklearn.metrics*, we were able to get two lists for each model: one with the scores and one with the mean square error. We then computed the average, which is shown bellow:

$$Scores_{LinearRegression} : 0.9986814491686788$$

$$Scores_{RidgeRegression} : 0.9986813346451633$$

$$Scores_{LassoRegression} : 0.9986852583775196$$

$$MSE_{LinearRegression} : 0.016249346626408735$$

$$MSE_{RidgeRegression} : 0.016250079136195104$$

$$MSE_{LassoRegression} : 0.01613064793147894$$

Here we computed the ridge regression with $\alpha = 0.001$, since it is the same as linear regression if $\alpha = 0$, and for lasso regression we assigned $\alpha = 0.0015771543086172347$, as seen before to be the optimal value. The results are what we expected, so we can conclude that the lasso regression with $\alpha = 0.0015771543086172347$ is the best model to apply to this dataset.

We trained the model again using Lasso Regression with $\alpha = 0.0015771543086172347$ but this time using the entire dataset. We then submitted the predictions for the test set obtained with this model.

3 Regression Part 2

The regression part 2 problem is identical to the first one but some of the training examples (less then 10%) are not generated by the model used to generate the other data, so they can be considered as outliers.

3.1 Z-score

Z-score tells us how many standard deviations away a given observation or data point is from the mean. In order to use Z-score we need to know the mean and the standard deviation of what is being observed or measured. It is calculated by subtracting the population mean from an individual raw score and then dividing the difference by the population standard deviation. The formula is shown bellow.

$$Z = \frac{x - \mu}{\sigma}$$

However, when the population mean and the population standard deviation are unknown, we calculate the Z-score using the sample mean and the sample standard deviation. In this case we have:

$$Z = \frac{x - \bar{x}}{S}$$

The further away an observation's Z-score is from zero, the more unusual it is. A standard cut-off value for finding outliers are Z-scores of $+/-3$ or further from zero.

3.2 Box-plots and Interquartile range

Interquartile range is a measure of statistical dispersion, in other words it measures the spread of data. It is defined as

$$IQR = Q3 - Q1$$

where $Q3$ is the upper quartile (or $75th$ percentile of the data) and $Q1$ is the lower quartile (or $25th$ percentile of the data).

A box-plot gives us information about the distribution of the data, through its quartiles. Moreover, a box-plot gives a sense how much the data is actually spread about, what is its range, and informs us about its skewness. It displays the distribution of data based on a five number summary ("minimum", first quartile ($Q1$), median, third quartile ($Q3$), and "maximum").

- "Minimum": The lowest data point, excluding outliers;
- "Maximum": The highest data point, excluding outliers;
- $Q1$: The first quartile (or $25th$ percentile of the data);
- $Q3$: The third quartile (or $75th$ percentile of the data)
- Median: The middle point of the dataset;

Outliers are usually defined as data points that are outside the following range:

$$[Q1 - 1.5IQR, Q3 + 1.5IQR]$$

3.3 Results and interpretation

First we attempted to find outliers using Z-score, however, using the usual cutoff of $+/-3$, we where left with only two data points. Since we where expecting to find a bit less than 10% of the training examples to be outliers (around 100 data points), we then decided to lower this cutoff to 2 which led to us having many more data points to analyse.

We also plotted the box-plots for each column in order to find new outliers. The plot with outliers defined as points not in $[Q1 - 1.5IQR, Q3 + 1.5IQR]$ is the following:

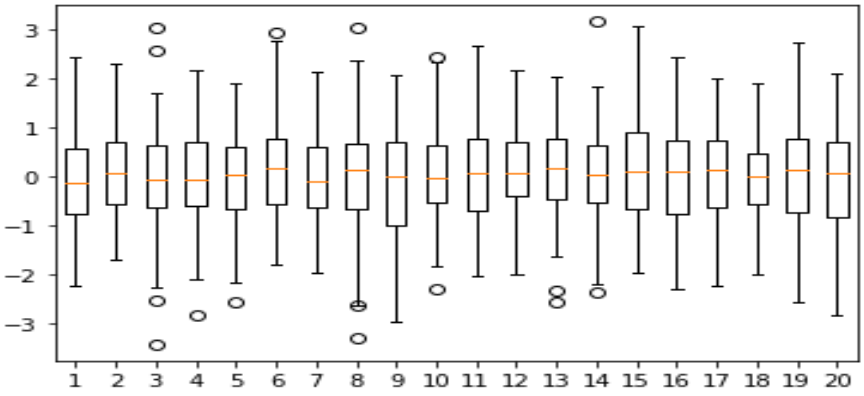


Figure 1: Box plot 1

However, for the same reason as before, we decided to consider an outlier any data point not in $[Q1 - 0.9IQR, Q3 + 0.9IQR]$, bellow is the corresponding plot.

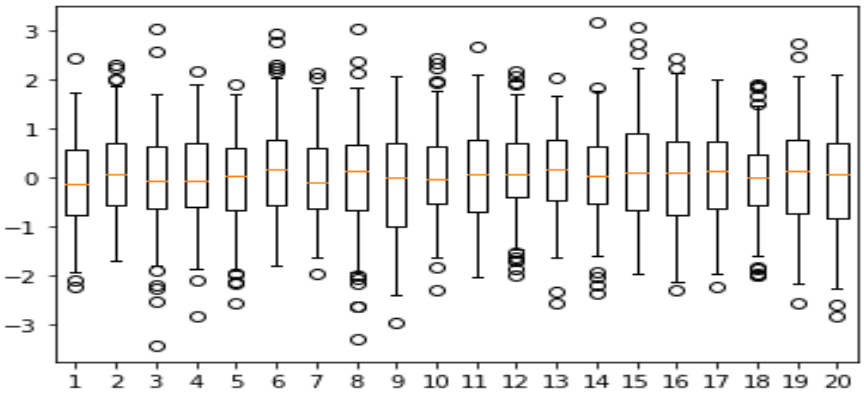
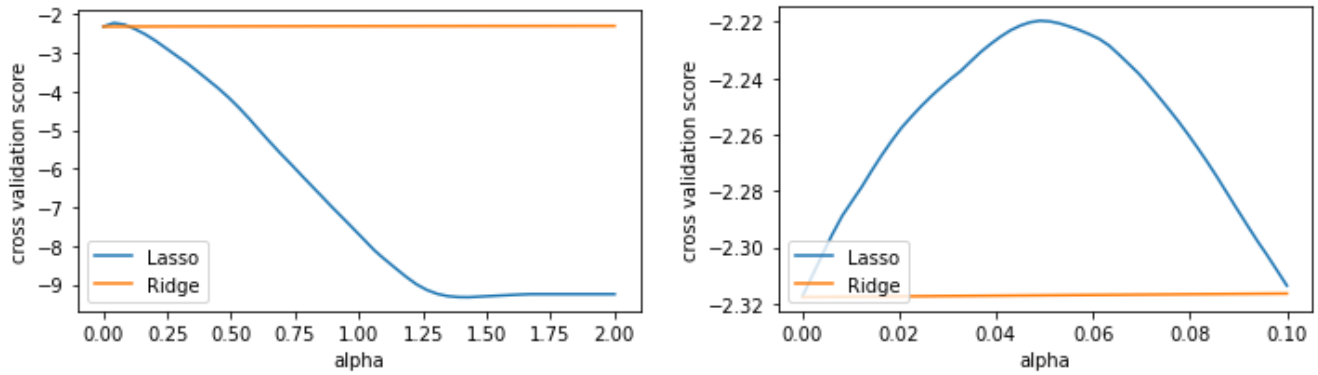


Figure 2: Box plot 2

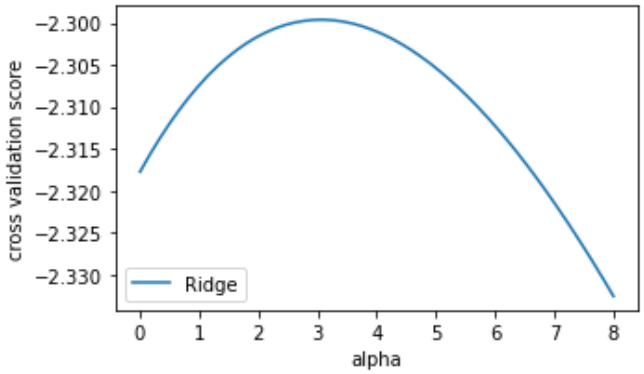
To find out to what training example these data points corresponded, we made three functions, $find\_iqr$, $find\_q1$ and $find\_q3$, that compute the value of IQR, $Q1$ and $Q3$ respectively. In a for loop, we applied these functions to every column and found out from which observation the outliers were.

In the end, we decided that we would consider an outlier any training example that had more that 3 data points with a z-score higher than 3 and also more than 3 data points labeled as outliers in the box-plot. This left us with 12, 15, 31, 59, 61, 71, 84, 97 as the indices of the outliers.

After removing these points, we followed the procedure from the previous exercise. The comparisons between lasso and ridge regression are shown in the plots bellow:



We can see that lasso regression appears to be better than ridge, however when we plot the alpha values for the ridge regression we get the following plot:



To see which model is best, we get the alpha values from the *RidgeCV* and *LassoCV* functions just as we did in the previous section and plot the average of the score and the average of the mean square error obtained when evaluating the model with a 10-fold cross validation. The values obtained using $\alpha_{Lasso} = 0.0494188376753507$ $\alpha_{Ridge} = 3.4$ were:

$$Scores_{LinearRegression} : 0.7369185345023236$$

$$Scores_{RidgeRegression} : 0.7396474529642756$$

$$Scores_{LassoRegression} : 0.7489475364631348$$

$$MSE_{LinearRegression} : 2.317743888859833$$

$$MSE_{RidgeRegression} : 2.2997597183299754$$

$$MSE_{LassoRegression} : 2.2198475624209655$$

So, again, the lasso regression had the highest score values and lowest mean square error values, meaning it was the best model to fit on this data and the one we chose.

So again we trained the model again using Lasso Regression with $\alpha_{Lasso} = 0.0494188376753507$ but using the entire dataset. We then submitted the predictions for the test set obtained with this model.

4 Classification Part 1

In this task we want to train a classifier that will classify a dataset of grayscale face images (50x50 pixels). This first task is a binary classification task where our model is supposed to predict the gender of each subject. The label is either 0 (male) or 1 (female).

4.1 Convolutional neural networks

A convolutional neural network (CNN) is a class of artificial neural network. CNNs are most commonly applied to computer vision applications. They can take in an input image, assign importance to various aspects in the image and be able to differentiate one from the other. The architecture of a CNN was inspired by the organization of the Visual Cortex, in a way that neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. A big advantage of CNNs is that they use little pre-processing compared to other image classification algorithms. They have the ability to learn how to optimize the necessary filters while in other methods filters are hand engineered.

The architecture of a convolutional neural network consists of inputs, hidden layers and an output layer. In a CNN, the hidden layers include convolutional layers, pooling layers and fully connected layers.

- **Convolutional layers** repeatedly apply a kernel to the input (the input is tensor). This is done by performing a dot product of the convolutional kernel with the layers input matrix which results in a map of activations called the feature map. The most commonly used activation function (and the one we will use) is *ReLU*. The convolution kernel slides along the input matrix of the current layer and generates a feature map, which will be the input of the next layer. This map indicates the location and strength of detected features in the input. A feature map has two spatial axes (height and width) and a depth axis (called the channels axis). For an RGB image, the dimension of the depth axis is 3 (for red, green and blue), however in our case (black and white images) the dimension of the depth

axis is 1. The convolution operation extracts patches from its input feature map and applies the same transformation to all of these patches, producing an output feature map. This output feature map is still a 3D tensor, but its depth is a parameter of the layer. Now, the different channels in the depth axis stand for filters and not for colors.

Convolutional layers, contrary to dense layers, learn local patters (in our case patterns found in small 2D 3x3 windows). This characteristic gives CNNs two interesting properties: The patterns they learn are translation invariant, meaning that after they learn the pattern they can recognize it anywhere; They can learn spatial hierarchies of patterns, meaning that a first convolutional layer will learn small patterns such as lines and edges while the next convolutional layers can learn larger patterns made out of those smaller patterns.

The two key parameters that define convolutions are: The size of the patches extracted from the inputs, typically 3x3 or 5x5; The depth of the output feature map, in other words the number of filters.

- **Pooling layers** reduce the dimensionality of the data, decreasing the computational power required to process the data. There are two types of pooling, max pooling returns the maximum value from the portion of the image covered by the kernel, while average pooling returns the average of that portion. Max pooling tends to work better than average pooling because features tend to encode the spatial presence of some pattern over the different tiles of the feature map and it is more informative to look at the maximal presence of different features than at their average presence.

- **Fully connected layers** connect every neuron in one layer to every neuron in the next layer. The flattened matrix goes through a fully connected layer to classify the images. This happens by feeding the flattened matrix to a feed-forward neural network and applying backpropagation to every iteration of training.

There are also a number of hyperparameters that can be used to tune our model. A few of the most important ones that we used in our search for the best model are the kernel size, padding, stride, number of filters, pooling type and size, learning rate, number of epochs, batch size, activation function, number of hidden layers and units and dropout for regularization. These are all described bellow.

- **Kernel** size is expressed as the dimensions of the convolutional kernel. We will use a common kernel size of 3x3, which corresponds to the number of pixels that will be processed together.

- **Padding** is the addition of pixels on the borders of an image. We use padding because the image shrinks every time a convolutional operation is performed, and also because the pixels on the corners and edges are used much less than the ones in the middle. Padding helps solve these issues. The pixels we add usually have the value of 0. Adding padding to an image processed by a CNN allows for more accurate analysis of images.

- **Stride** is the number of pixels by which the kernel moves along the feature map. When the stride is 1 we move the kernel 1 pixel at a time. When the stride is 2 we move the kernel 2 pixels at a time and so on.

- **Number of filters** corresponds to the the number of channels in the depth axis of the feature map. The size of the feature map decreases with each convolution and pooling layer, so it is common to have less filters in the lower layers and more in the higher layers. This equalizes computation at each layer.

- **Pooling type and size** corresponds to choosing the type of pooling (we have seen before we can have max pooling or average pooling) and choosing the size of the pooling operation. Usually 2x2 pooling is applied (which is what we will do), however for large input volumes it may be necessary to use greater pooling, like 4x4.

- **Learning rate** controls how much to update the weights in the optimization algorithm. There are several options we can use (depending on the choice of optimizer such as SGD, Adam or RMSProp), such as fixed learning rate, gradually decreasing learning rate, momentum based methods or adaptive learning rates.

- **Number of epochs** is the the number of times the entire training set passes through the CNN. One epoch means that each sample in the training dataset has had an opportunity to update the model parameters.

- **Batch size** is the number of training examples utilised in one iteration. CNN models don't process an entire dataset at once, they break the data into small batches. There three options for the batch size: batch mode where the batch size is equal to the total dataset; mini-batch mode where the batch size is greater than one but less than the total dataset size (usually a number that can be divided into the total dataset size); stochastic mode where the batch size is equal to one. The one that is usually preferred for CNNs is mini batch mode.

- **Activation function** The activation function is the non linear transformation that we do over the input signal. There are different types of activation functions (Sigmoid, Tanh, ReLU), but the most commonly used for CNNs is ReLU (it is also the one we will use).

- **Number of hidden layers and units** are the number of layers and units we choose for the model. It is usually good to add more layers until the validation error no longer improves. However, more layers means it will be more computationally expensive to train the network. Having a small amount of units may lead to underfitting while having more units is usually not harmful if the appropriate regularization is applied.

- **Dropout** is a regularization technique that prevents the model from overfitting. It randomly sets the outgoing edges of hidden units to zero at each update in the training phase. This is done according to a desired probability. The default value for this probability is 0.5.

4.2 Training our model

The first step was to format our data into appropriately preprocessed floating-point tensors before being fed into the network. The datasets we where given where 2D matrices, so we had to reshape them into the appropriate size which is a 4D array where each dimension corresponds to $(batchsize, height, width, depth)$. In our case the batch size was the number of observations, the height and width where both 50 because our images are 50x50 and the depth is 1 because we are not dealing with color.

In general, it isn't safe to feed into a neural network data that takes relatively large values, so to make learning easier for our network we rescaled all our pixel values to take values between 0 and 1 by dividing them by 255. We also used one hot encoding on our Y_{train} data.

We also used data augmentation to prevent overfitting. Overfitting is caused by having too few samples to learn from, which then makes it impossible to train a model that can generalize to new data. Data augmentation generates more training data from existing training samples, by augmenting the samples via a number of random transformations, such as rotating and shifting, that yield believable-looking images. Our results improved when we tried data augmentation so we applied it.

We implemented *EarlyStopping* which in our case monitors the accuracy of the validation set with a patience of 3 (meaning that after 3 epochs with no improvement the training will be stopped). The parameter *restore_best_weights* was set to true so the weights from the epoch with the best value of accuracy for the validation set will be restored.

For this task we trained innumerous models before choosing the best one. We tried models with different numbers of layers and units, we changed the kernel size, tried using padding and changing the stride, used different numbers of filters and sizes of pooling, saw how the results changed when using different activation functions, optimizers, epochs and when changing the batch size and added droppout layers in different places with different probabilities.

The final model we obtained was a model with 3 convolutional layers, the first one had 32 filters and the other two had 128. They all had a kernel size of 3x3 and *ReLU* as the activation function. Every convolutional layer was followed by a pooling layer that performed max pooling with a 2x2 pooling size. At the end of the convolutional and pooling layers we added a dropout layer with a rate of 0.25. We used the *sigmoid* function as an activation function in the last layer, since we are dealing with a binary problem. For the same reason, we used *binary_crossentropy* as the loss function. As an optimizer, we used the *Adam* optimization algorithm with the default learning rate because it was yielding the best results.

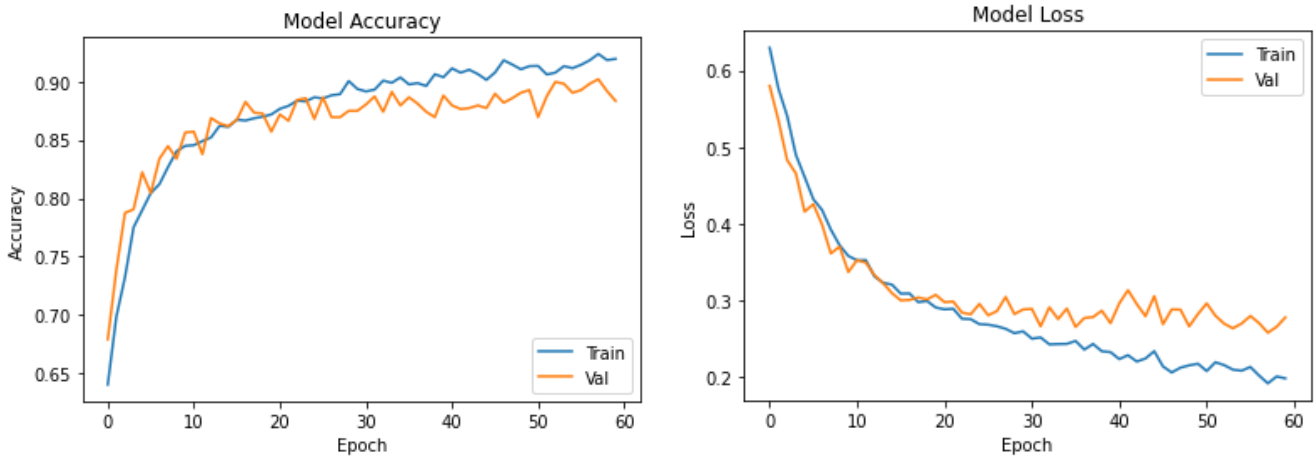
We then fed the last output tensor (of shape (4, 4, 128)) into a densely connected classifier network. Here we started with a dense layer (or fully connected layer) with 128 units and also with *ReLU* as its activation function. This layer if followed by another dropout layer with a rate of 0.5 and then a final dense layer with only two units, since this is a binary classification problem. Bellow we show our final model summary.

Layer (type)	Output Shape	Param #
conv2d_277 (Conv2D)	(None, 48, 48, 32)	320
max_pooling2d_258 (MaxPoolin	(None, 24, 24, 32)	0
conv2d_278 (Conv2D)	(None, 22, 22, 128)	36992
max_pooling2d_259 (MaxPoolin	(None, 11, 11, 128)	0
conv2d_279 (Conv2D)	(None, 9, 9, 128)	147584
max_pooling2d_260 (MaxPoolin	(None, 4, 4, 128)	0
dropout_196 (Dropout)	(None, 4, 4, 128)	0
flatten_98 (Flatten)	(None, 2048)	0
dense_197 (Dense)	(None, 128)	262272
dropout_197 (Dropout)	(None, 128)	0
dense_198 (Dense)	(None, 2)	258

Total params: 447,426
Trainable params: 447,426
Non-trainable params: 0

Figure 3: Model Summary

We ran this model for 60 epochs and got a top validation accuracy of 0.9019 with a loss of 0.2537. Bellow are the plots for the validation accuracy and loss.



To finalize the task, we transformed our predictions into a list of zeros (for men) and ones (for women), depending on which probability was higher.

5 Classification Part 2

In this task we want to train a classifier that will classify a dataset of grayscale face images (50x50 pixels). This task is a multiclass classification task where our model is supposed to predict the ethnicity of each subject. The labels can be 0 (caucasian), 1 (african), 2 (asian) or 3 (indian).

5.1 Training our model

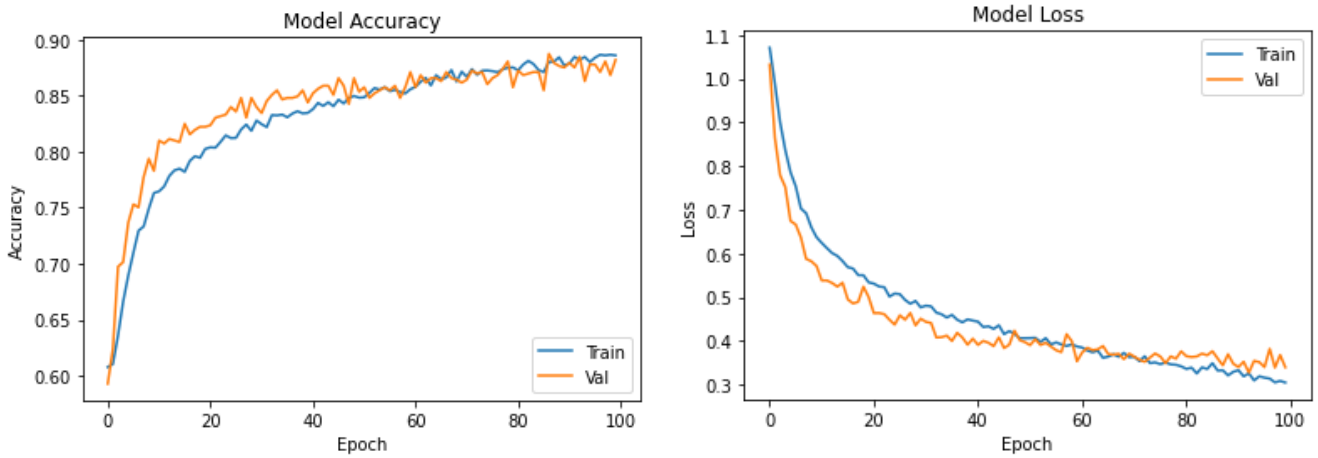
In this task we applied the same preprocessing steps as in the previous task, such as formating and rescaling the values. We also applied one hot encoding to the Y_{train} data and split the data into training

data and validation data. We decided 10% of the data would be enough for the validation set since the dataset is large enough and we would have more data to train our model.

We started with the same model we used in the previous task and attempted to make changes in order to improve the performance. The first obvious change that had to be made was changing the number of units in the final *Dense* layer to 4, since we now have 4 classes. We also had to change the activation function in the final layer to $softmax$, and the loss function to $categorical\_crossentropy$. Both of these changes are because we are no longer dealing with a binary classification problem. The results we obtained after these changes were good but not great, so we tried changing other parameters.

Changing the architecture of the model, such as the number of convolutional layers and dropout layers, didn't help us much, so we added data augmentation and lowered the learning rate of the optimization function and the results were significantly better. The data augmentation lead to less overfitting and the learning rate got us better loss and accuracy values. The data augmentation had quite small changes, we set $rotation\_range$, $width\_range$, $height\_range$, $shear\_range$ and $zoom\_range$ to 0.1 and we also set $horizontal\_flip$ to $True$. When increasing the values to something higher than 0.1, the data would overfit less (the validation accuracy and loss would be similar to the one of the training data) but the model would learn less (meaning the accuracy would be lower and the loss higher), so we kept them at 0.1. The optimal learning rate we ended up with was 0.0005 (for the Adam optimization algorithm).

In this problem, since the classes are imbalanced, we evaluate the model based on the balanced accuracy. However, when using the early stopping, we cannot monitor the balanced accuracy, so we monitor the loss of the validation set and then compute the balanced accuracy on the validation set. Bellow are the plots for the accuracy and the loss of our final model. The lowest loss of the validation set was 0.3281 and the highest accuracy of the validation set was 0.8874. The corresponding plots are bellow.



The balanced accuracy of this model on the validation set was 0.83, which is the best out of the models we tried.

After obtaining the best model, we used the predict method to get the class of each observation on the test set and transformed the probabilities obtained into integer values from 0 to 3, corresponding to the class with higher probability.

6 Conclusion

In conclusion, we would like to mention some of the mistakes we made throughout the project.

In the second part of the project in the regression framework, we only considered the outliers in the X_{train} dataset, leaving out the Y_{train}. This made our results significantly worse.

Also, for the first task of the classification section we submitted a file with the values of a previous model that wasn't our final one. The submission prior to that would have been more correct than the one we ended up submitting.

7 Bibliography

Regression Modeling Strategies by Frank E. Harrell;
Deep Learning with Python by François Chollet.