Instituto Superior Técnico

M.Sc. in Electrical and Computer Engineering

Machine Learning

2021/2022 - First Semester

Regression and Classification

Group: 40

1. **N.°:** 101692 **Name:** Fred Intwaza Kwizera
2. **N.°:** 101449 **Name:** Gunnar Skaalheim

Instructor: Chrysoula Zerva

Contents

1 Regression **1**

1.1 First Problem 1

1.1.1 Data Analysis 1

1.1.2 Implementation and results 1

1.1.3 Discussion 2

1.2 Second Problem 2

1.2.1 Data Preparation 2

1.2.2 Implementation and results 3

1.2.3 Discussion 3

1.3 Conclusion 4

2 Classification **5**

2.1 First Problem 5

2.1.1 Data Preparation 5

2.1.2 Implementation and Result 5

2.1.3 Discussion 6

2.2 Second Problem 6

2.2.1 Data Preparation 6

2.2.2 Implementation and Result 7

2.2.3 Discussion 8

2.3 Conclusion 8

1 Regression

Regression is a statistical method that attempts to determine the relationship between one dependent variable $y^{(i)}$ and a series of other variables $x^{(i)}$. As there exist several different regression methods, a main objective is to find an appropriate method that suits the training set well or well enough.

The following section consider two regression problems. In both problems, a training set $\mathcal{T}_r = \{(x^{(i)}, y^{(i)}), i = 1, \cdots, 100\}$, $x^{(i)} \in \mathbb{R}^{20}$ and $y^{(i)} \in \mathbb{R}$ was given, but in the second problem, a small part of the examples can be considered as outliers (less then 10%). The goal for both problems is to build a predictor that best predicts unseen samples.

1.1 First Problem

1.1.1 Data Analysis

A simple linear regressor assumes that all features are linear dependent of y. Considering the correlation between the features in x and the output y, one can see that the data set has linear tendency in some of the features. This is evident in the bottom column. Clearly, there are not many correlated features, which indicates that a simple linear regression may not be sufficient.

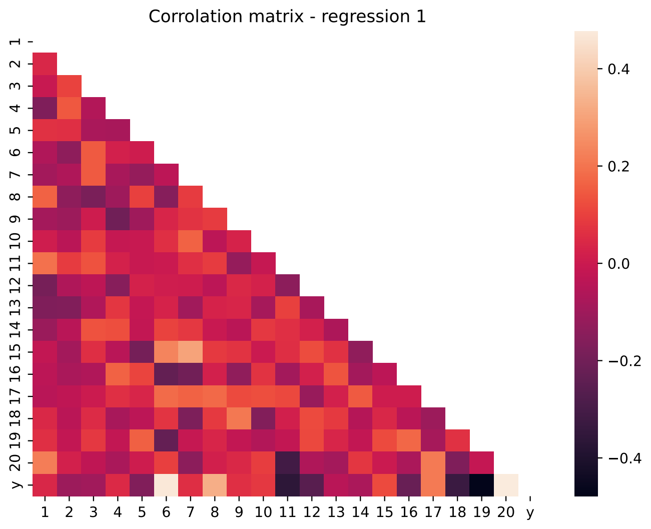


Figure 1: Correlation matrix of $\mathcal{T}_r$

1.1.2 Implementation and results

Having several correlated variables may indicate that the training set is sufficiently linear, such that a regularized linear regression would be able to solve the regression problem with an acceptable performance. In such a situation the lasso regressor may be a good solution, as it excludes the impact from the most uncorrelated features. Nevertheless, excluding features may cause a loss of small, but crucial linear dependencies. The ridge regression method resolves this issue by preventing the coefficients $\hat{\beta}$ from being set to 0, however allowing small values. Considering the correlation observed in figure 1, one could expect that ElasticNet - a linear combination of lasso and ridge, could be a sufficient regression method.

ElasticNet have two parameters, weighting how much penalization the l_2-norm and l_1-norm of the coefficients $\hat{\beta}$ should be. Since the solution of the minimization problem, finding $\hat{\beta}$, is highly depends on how the data is separated into training and test set, repeated cross validation was used. Cross validation uses different portions of the data to test and train a model on different iterations.

A hyperparameter search on a cross validated ElasticNet was runned and resulted in the parameters `l1_ratio = 1`, `alpha = 0.001`. This is equivalent to a lasso regression with a weighting of $\lambda = \texttt{alpha} = 0.001$ on the l_1-norm of $\hat{\beta}$. Further, this means that ElasticNet have through hyperparamer search concluded that extracting a feature is better than just reducing its impact towards 0. The final MSE score of the model on the test data was `0.014866414`, with a ranking of `47`.

1.1.3 Discussion

Looking at the estimated coefficients in $\hat{\beta}$, it is observed that $\hat{\beta}_2 = 0$, meaning that the regressor have excluded feature 2 from the training set. This should be expected due to the weak correlation between x_2 and y in figure 1. The most colored features in figure 1, e.g. feature 6 and 19 had the respective beta values $\hat{\beta}_6 = 1.696$ and $\hat{\beta}_{19} = -1.273$. Both the magnitude and sign of the beta values match their entries in the correlation matrix.

1.2 Second Problem

In a real worlds situation, the data may not always represent the desired measurement. The data may contain extreme values that are outside the range of what is expected and that is unlike the other data. This can be due to noise or incorrect measurement[1], among other things. It is therefore important to clean the data sample to ensure that the observations best represent the problem.

1.2.1 Data Preparation

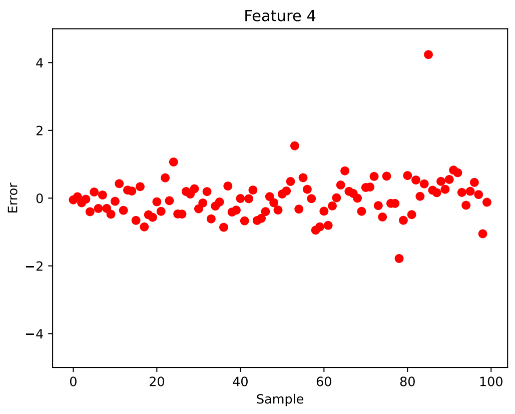
The approach used to detect and remove outliers from the training set is described in the algorithm below. This approach utilises the linear correlation between features to detect outliers. Each target y_i, is added to it's respective sample vector, resulting in a *number of samples* $\times 21$ matrix denoted as *data*. For each feature, a linear model is fitted, where the feature is used as reference, and the remaining features as input. The model then makes predictions on the same feature $\hat{y}$. The true value of the feature y, and its prediction $\hat{y}$, is used as basis for detecting outliers. In the GET-OUTLIER-INDICES algorithm, the error between y and $\hat{y}$ is computed. An error larger then 3 standard deviations is unlikely[2], and samples corresponding to these errors are regarded as outliers. An example of one iteration of the first for loop of the algorithm is shown in the residual plot in figure 2. The figure depicts the error between the true and predicted value of feature 4 for each sample, for both the data set from the first and second task. It is apparent that the outliers of the second data set increases the uncertainty of the predictions. The outermost points of the figure are regarded as outlier and are removed from the training set.

[1]Jason Brownlee, 'How to Remove Outliers for Machine Learning', April 25 2018, https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/

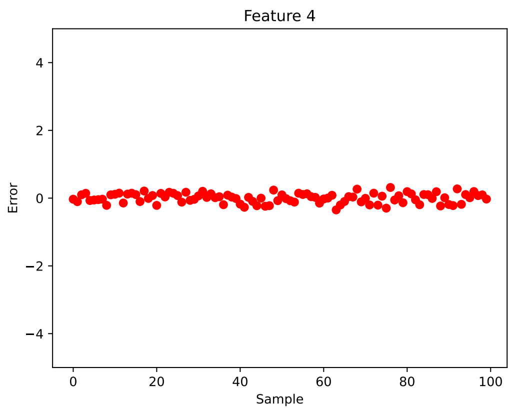
[2]Jason Brownlee, '8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset', August 19 2015, https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

Algorithm 1 Remove-outliers

```
Input $x_{raw}$, $y_{raw}$
Output $x$, $y$
  $data \leftarrow x_{raw}, y_{raw}$
  $outliersIndices \leftarrow \emptyset$
  for $feature$ in $data$ do
      $y \leftarrow feature$
      $x \leftarrow data/feature$
      Fit $linearmodel$ to $x$ and $y$
      $\hat{y} \leftarrow linearmodel(x)$
      $indices \leftarrow GET - OUTLIER - INDICES(y, \hat{y})$
      $outliersIndices \leftarrow outliersIndices \cup indices$
  end for
  $x \leftarrow REMOVE(x_{raw}, outliersIndices)$
  $y \leftarrow REMOVE(y_{raw}, outliersIndices)$
```



(a) Dataset with outliers



(b) Dataset without outliers

Figure 2: Prediction error of feature 4

Compering the correlation matrix of the filtered data versus the correlation matrix prior to the removal of outliers, an increase in correlation of `5.26%` between y and the features in x was observed.

1.2.2 Implementation and results

When considering that the examples of the second problem are the same as for the first problem, disregarding the outliers, it is reasonable to use the same approach for selecting an regressor as used in the first problem. After processing the data as described in the section above, the ElasticNet regressor was tuned in the same manner as for the first problem. The resulting hyper parameters was `alpha=0.001` and `l1_ratio=1`. Effectively resulting in a Lasso regressor. Cross validation was used to validate good performance. The final MSE score of the model on the test data was `0,013454185`, with a ranking of `42`.

1.2.3 Discussion

Based on the final ranking on the resulting regressor, the argument can be made that the algorithm for "cleaning" the data was effective. Further, in this problem most of the work was related to outlier filtering and not on tuning the regressor. There is no guarantee that the REMOVE-OUTLIERS algorithm detected all outliers. In hindsight, better performance could have been obtained by selecting a regressor more robust against outliers.

1.3 Conclusion

This section has considered two regression problems. One without, and one with outliers. The challenges associated with a training set containing outliers was addressed by attempting to remove the outliers. Surprisingly enough, the regressor from the second problem had a better performance on the test set then the regressor from the first problem had, despite the same model being used in both task. This may be due to better parameters being chosen during the tuning. Good performance supports the argument that the outlier removal algorithm was efficient.

2 Classification

The following section considers two classification problems, where an adaptation of the UTK-Face data set was provided. The training set contains gray scale images of approximately 7300 subjects, with 50x50 pixels. The first task describes a binary classification problem where the gender of a subject was to be predicted. The labels of the subject was either 0 for male, or 1 for female. The second classification task is a multi-class problem in which the ethnicity of the subject was to be predicted. The label of the subject was an integer from 0 to 3, denoting Caucasian, African, Asian or Indian.

2.1 First Problem

From the provided data there were 43% were male, and 57% were female. Based on this distribution, the training set can arguably be denoted as balanced, but with a minor under representation of men. There are various models for solving this problem, in the following the Convolutional Neural Networks will be implemented. A CNN classifier can predict the label of an image based on internal representations that extract useful information. This classifier is the most popular neural network model used for image classification.

2.1.1 Data Preparation

To implement the CNN classifier, the python library Keras was utilized. To be able to use the interface, the training data which was a linear array for each sample needed to be reshaped to a 50x50 array for each sample. In addition, the class labels had to be represented as a set of of indicator variables through one hot encoding. The training data was further split to a training and a validation set to be able to validate the performance of the final model.

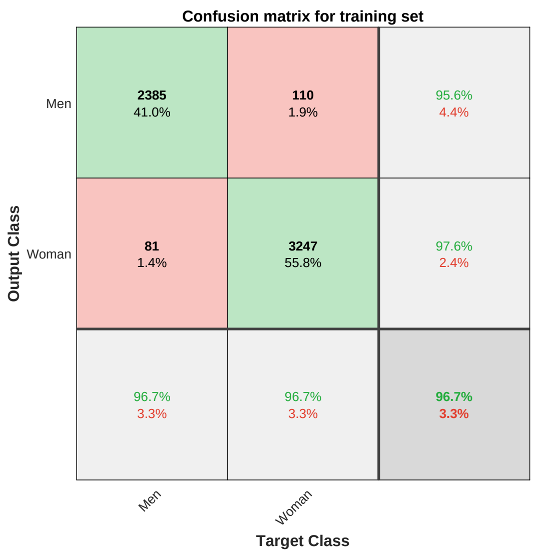
2.1.2 Implementation and Result

The choice of architecture of a CNN, i.e. layer types, number of layers and number of neurons in each layer, is not trivial. In addition, the final architecture will result in numerous tuning parameters. To tackle this complexity, the Keras tuner was used for tuning architecture and its parameters. The tuner was to test different numbers of convolution layers, and the size of the respective kernal. A validation split was utilized in the tuner to prevent an overfitted model. In addition, the performance metric of the tuning was set to balanced accuracy and the loss to binary cross entropy. For each convolution and dense layer, the activation function ReLU was utilized. The softmax function was used in the final dense layer. The resulting architecture from the tuning is shown in table 1.

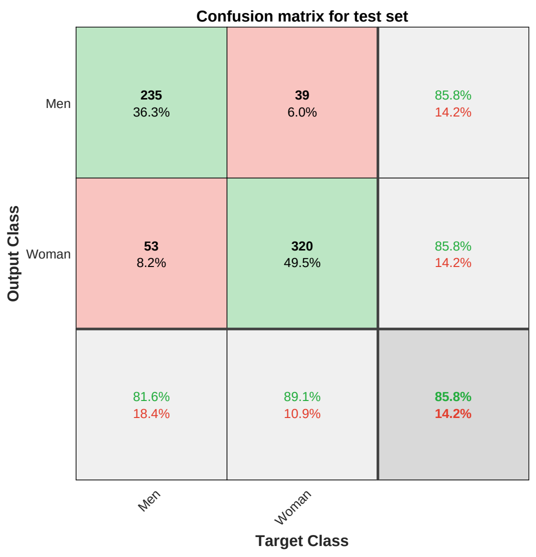
Layer 1 - Convolution	kernal: 64
Layer 2 - Convolution	kernal: 16
Layer 3 - Pooling	size : 3x3
Layer 4 - Flatten	
Layer 5 - Dense	nurons: 512
Layer 6 - Dense	nurons: 2

Table 1: Architecture of the CNN classifier

It was experimented with adding pooling 2D layers between all or some of the convolutions as this could amplify facial characteristics. However, this did not result in better performance on the validation set. To mitigate any overfitting, different dropouts was attempted before the flatten layer. This did not result in better performance either. Evaluating the confusion matrices of the training set and the test set in figure 3, we observe some miss classifications in both the training and test set.



(a) Training set



(b) Test set

Figure 3: Confusion matrices of training and test data, genders

Based on the confusion matrix Figure 3 the final performance of the classifier on the test set was a balanced accuracy of `0.8785757`, with a ranking of `87`.

2.1.3 Discussion

Due to the relative performance of the model on the test set, it is clear that a number of measures could have been implemented to improve the model. Prior to tuning the CNN model, the assumption was made that the training data was sufficiently balanced. A way to increase the performance of the model may have been to reject this assumption and implement stratified sampling, which would ensure a consistent balance between each men and woman in the training and validation set.

2.2 Second Problem

The following multi-class problem is based on the same training set as the previous task and will consequently result in a smaller pool of examples for each class. This will have to be considered when developing a classifier.

2.2.1 Data Preparation

When evaluating the data, the distribution of the number of images per class for Caucasian, African, Asian or Indian is `60.79%`, `4.63%`, `18.16%` and `16.41%` respectively. Consequently, the training set is clearly imbalanced. Using the same approach as in the first task would result in a biased classifier. An attempt to balance the data was made before developing a model.

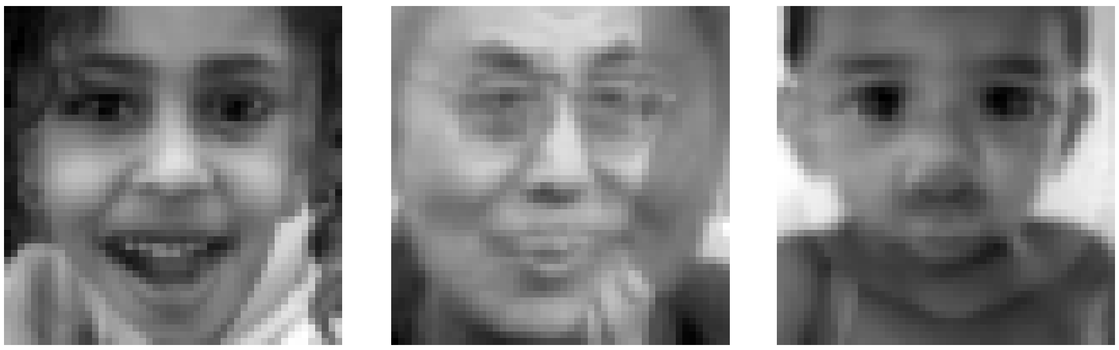
There are numerous methods for tackling imbalance between classes. This includes collecting more data, choosing balanced accuracy as performance metric, resampling data among others. When resampling the training set, either copies of the under-represented classes can

Layer 1 - Convolution	kernal: 16
Layer 2 - Convolution	kernal: 64
Layer 3 - Convolution	kernal: 64
Layer 3 - Convolution	kernal: 16
Layer 4 - Pooling	size : 3x3
Layer 5 - Flatten	
Layer 6 - Dense	nurons: 512
Layer 7 - Dense	nurons: 4

Table 2: Architecture of the CNN classifier

be added or instances of the over-sampled classes can be deleted. An alternative method to adding copies of under-sampled classes is to generate synthetic samples. The process of adding more samples to the training set is denoted over-sampling, while deleting samples is denoted under-sampling.

One method for generating synthetic samples is called the Synthetic Minority Oversampling Technique, or SMOTE. The method considers each sample as a linear array, draws lines between adjacent sample and synthetic minority instances somewhere on these lines. The python library imbalanced-learn provides the framework for utilizing SMOTE. Using SMOTE resulted in a training set with `4030` samples each. Examples of synthetic samples are shown in figure 4.

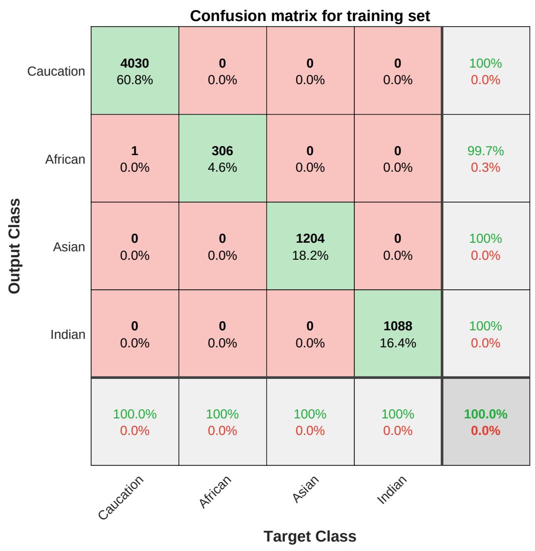


(a) African (b) Asian (c) Indian

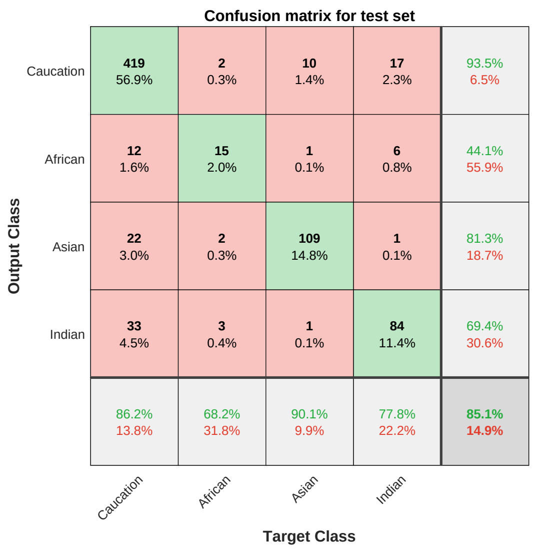
Figure 4: Synthetic images

2.2.2 Implementation and Result

A CNN classifier was implemented using the training set prepared in the previous section. Recalling that we have four classes, in contrast to two as in the previous problem, the labels of images were one hot encoded, resulting in four indicator variables.
The approach for selecting the architecture of CNN classifier was the same as for the previous task. For good measure, stratified sampling was implemented to ensure no imbalance when training the model. The resulting architecture yields is summarized in figure 2.
The same trial of adding pooling layers and trying different dropouts were experimented with, as in the previous task, without resulting in better performance. The respective confusion matrix of the model is presented in figure 5.
As expected, there is a significantly higher performance on the training set versus the test set. Only `44.12` % of the Africans in the test set was classified as African. A low percentage



(a) Training set



(b) Test set

Figure 5: Confusion matrices of training and test data, ethnicity

for correct classified Indians is also present. Despite the over-sampling, samples from the minority classes are still classified as the over-represented class. For example, observe that 12 of 34 out of Africans is classified as a Caucasian. The same yields for the 33 out of 121 Indians that are classified as Caucasians.
The final performance of the classifier on the test set was `0.719443159` with the ranking of `96`.

2.2.3 Discussion

Also for this model, There is room for improvement. Because the task is a multi-class problem, and the distinction between ethnicity often appear in smaller traits, a deeper network that picks up on these traits, may have resulted in better performance. This is however generally a difficult assessment to make in machine learning. The data set is arguably too small for under sampling[3]. However, under-sampling the over-represented classes in combination with SMOTE could have improved the model.

2.3 Conclusion

The two classification problems were solved using a CNN classifier and arguably produced a sufficient classifier. The challenges assisted with an imbalanced training set was addressed, and its effects were mitigated by preparing the training data prior to selecting a model. Numerous methods were discussed for improving the performance of the classifier, and preventing over-fitting. Experience has shown that an impotent factor when training a model is the limitation in computation power, which limits the search space during tuning in order to be able to obtain a feasible model within a reasonable time. Due to the huge amount of possible configurations of a classifier, luck is also an impotent factor when considering performance.

[3]Jason Brownlee, '8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset', August 19 2015, https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/