Machine Learning

LEEC

Project Report

Autores:

André Santos (96152) andresantos1.alf@gmail.com
Nuno Santos (96296) nuno.j.santos@tecnico.ulisboa.pt

Grupo 3

2022/2023 – 1º Semestre, P1

1 Part 1 - Regression with Synthetic Data

On this first problem, we were tasked with analysing regressions in order to determine which method better adapts to a given set of data. The criteria used for this was the sum of squared errors (SSE).

1.1 First Problem - Linear Regression

On this first problem, we were given a training set with $n = 15$ examples and 10 features. We were tasked with training a linear predictor for a test set with $n' = 1000$ examples, using the given training set. In order to accomplish this, it was decided to test a linear regression, as well as Ridge regression and Lasso regression. In order to cross-validate our models, we decided to use `kfold` with `k=5`, which uses 12 elements for training and 3 for testing at a time. It was also decided to test several values of `alpha` for both Ridge and Lasso, in order to then utilize the one that minimized the sum of squared errors. In order to test the values of alpha, we added a for cycle which calculated the sum of the SSE from each fold in each iteration, in order to decide what was the best value. The `sklearn` functions `RidgeCV` and `LassoCV` should've been used, as these do this process automatically and store the best alpha. This process resulted in the following alpha and SSE values:

Table 1: Mean of SSE values for initial approach

	Linear	**Ridge**	**Lasso**
Mean of SSE	51.568	9.909	10.375
Alpha	–	1.78	0.07

As it can be seen, the best model to use by this metric is Ridge regression. This led to a poor result, due to the fact that 12 examples were not enough to train the model. A better approach would have been the use of `leave one out`, which corresponds to a `kfold` with `k=15`. This way, we could've used 14 of the 15 examples for training. For this approach, and replacing the method used to obtain the values of alpha described above with `RidgeCV` and `LassoCV`, the following SSE values were obtained.

Table 2: Mean of SSE values for correct approach

	Linear	**Ridge**	**Lasso**
Mean of SSE	7.864	5.298	3.672

As it can be seen, the best approach is now the Lasso regression, and all the values obtained are much lower (since we are usng more samples to train the model). This is due to the fact that the number of features available is too high for the amount of samples. As such, Lasso regression, which allows for feature selection and represses those with less importance, results in a better model.

1.2 Second Problem - Linear Regression with Two Models

In this problem, we were given a data set generated by two diferent models, and were tasked to split them correctly and train each one as well as possible. In order to split the model, we analysed some options, namely `Kmeans`, `Gaussian Mixture` and residual based separation. Both `Kmeans` and `Gaussian Mixture` were implemented using methods available in sci-kit learn, while the residual based separation was implemented using a linear regression. The clusters were then tested with the same regression models used in the previous problem (linear, ridge and lasso regressions) and evaluated using their SSE. The clusters produced by `Kmeans` and `Gaussian Mixture` are shown below:

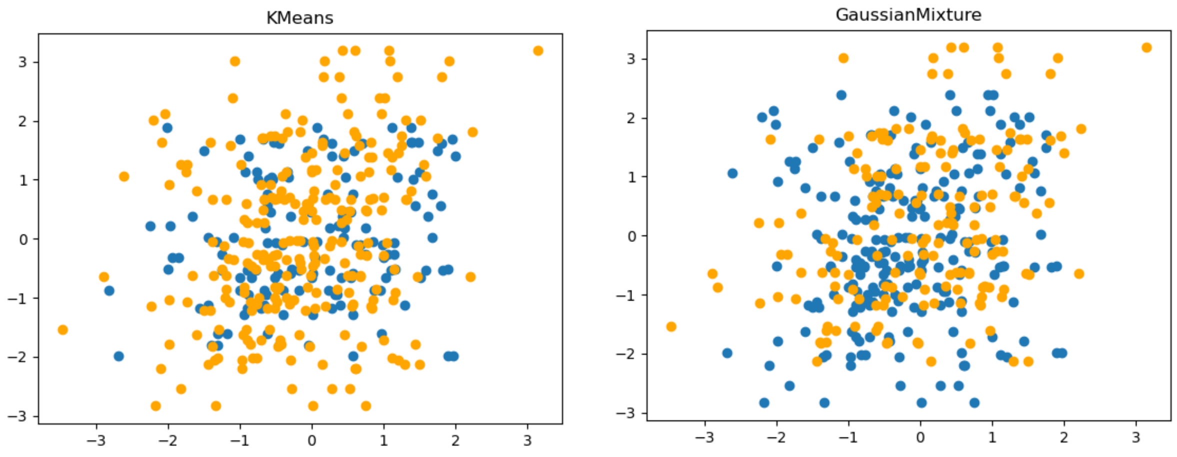


Figure 1: Clusters for KMeans and Gaussian Mixture

As it can be seen, the clusters do not appear too organized; there appears to be a clear separation in the data that neither method achieves completely. Therefore, it was decided to test a residual based separation. In order to do this, we performed a linear regression over all the data. Afterwards, we calculated the residuals, by subtracting the predicted values from the ones used in training. We then used the mean and the standard deviation of the calculated residuals in order to define a threshold, using the formula $threshold = mean\_residuals + k \times std\_residuals$, with k being a constant. This resulted in the following clusters:

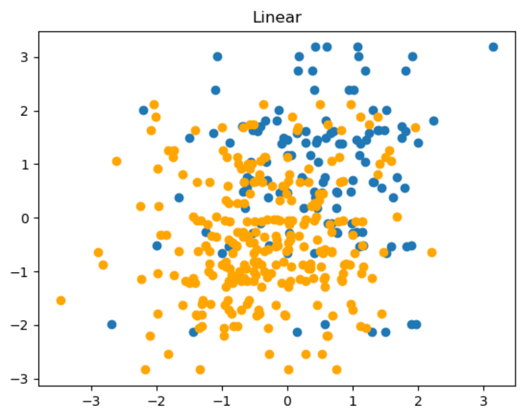


Figure 2: Clusters for KMeans and Gaussian Mixture

This was the method chosen, which resulted in a very poor result. In order to improve the SSE value, it would've been necessary to continue iterating this algorithm, lowering the threshold each time it ran. In order to do this, we would've used R2 Squared to compute the error. This would've resulted in less and less values being added to one of the clusters on each iteration, until the clusters were split.

On the second part of this problem we had to take the clusters obtained, and decide what type of regression to use with each one. In order to do this, the logic used on the first problem was reapplied here: we used `Kmeans` with `k=15`, equivalent to using `leave one out`, then tested Linear regression, Ridge regression, and Lasso regression, using the best alpha possible for both Ridge and Lasso. We obtained the following SSE values:

Table 3: Mean of SSE values for initial approach

	Linear	**Ridge**	**Lasso**
Mean of SSE (Cluster 0)	1.540	1.544	1.501
Mean of SSE (Cluster 1)	1.265	1.247	1.271

As can be seen, the lowest values are obtained for Lasso and Ridge, respectively for clusters 0 and 1, which were the regressions used for the submission. If the data split was done correctly by doing more iterations of the implemented algorithm, it is likely that different results would've been produced.

2 Part 2 - Image Analysis

On the second part of the project, we were tasked with two image classification tasks, in order to apply our knowledge of CNNs and other methods used for classification. The criteria used for this part was balanced accuracy.

2.1 First Task

2.1.1 Overview

For the first task in image analysis, the goal was to create a model that could predict whether an image is a nevu (label 0) or a melanoma (label 1). Since there are two classes, this consists of a binary classification task. In order to achieve this, we decided to test a logistic regression model and a CNN (convolutional neural network) and decided to use the one that produced best results. Since the data set was imbalanced, it was also necessary to test which data balancing technique produced the best results for our data: class weights, data augmentation or over/under sampling techniques like SMOTE (Synthetic Minority Oversampling Technique). The first one attributes a higher weight to the minority class, the second one creates artificial images by means of rotation, flipping or small shifts in image position and the third one also produces artificial images, but by different means: images are created by interpolating the already existing ones.

2.1.2 Used Models

For the CNN, we decided to test the following architecture:

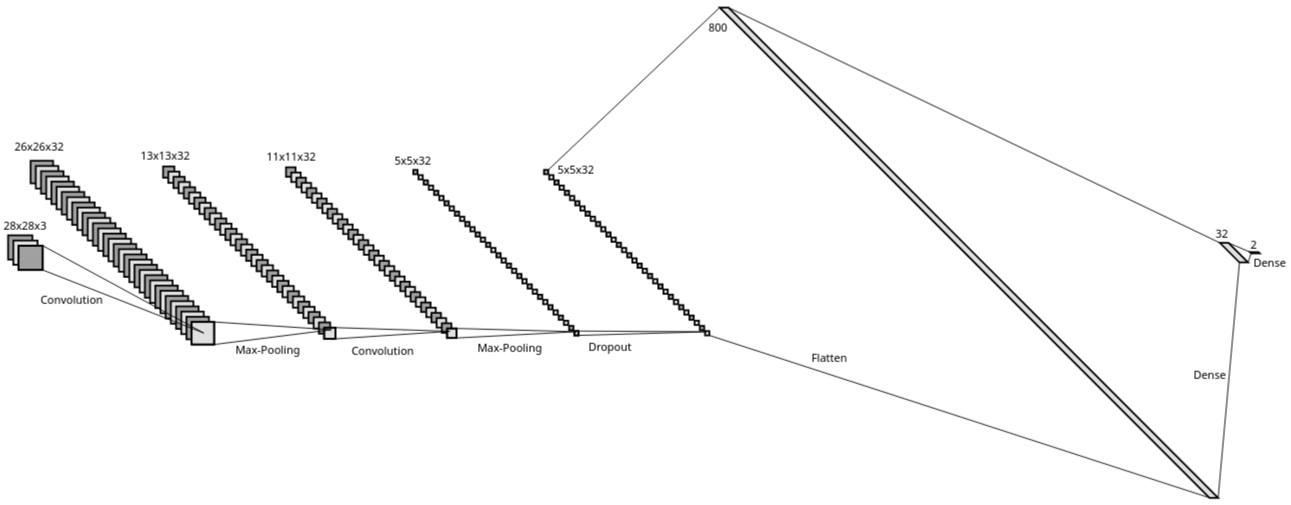


Figure 3: CNN for first task

The CNN receives as input a 28x28x3 matrix, which corresponds to a 28x28 pixel image and it's three color filters, RGB (red, green and blue). The first and third layers are convolutions layers with 32 filters of size 3x3 and activation function *ReLU*. The second and fourth layers are max pooling layers with filter size 2x2. The fifth and sixth layers are dropout and flatten, respectively, and the final two are dense layers, with activation functions *ReLU* and *softmax* for the final one. In order to evaluate the model, we utilized *balanced accuracy*, which is calculated by dividing the sum of the *sensitivity* and *specitivity* by two. The *sensitivity* corresponds to the "true positive" rate, and the *specitivity* corresponds to the "true negative" rate. Both of these can be obtained through the confusion matrix, shown below.

		Predicted		
		Negative	Positive	Total
Actual	Negative	TN	FP	$TN + FP$
	Positive	FN	TP	$FN + TP$
	Total	$TN + FN$	$FP + TP$	N

The *sensitivy* is calculated by dividing the number of true positives by the sum of true positives and false negatives and the *specitivity* is calculated by dividing the number of true negatives by the sum of true negatives and false positives. We also chose the optimizer *Adam* and, since we have two exit (Dense) layers, we used *categorical crossentropy* as our loss function. In order to reduce overfitting, along with the Dropout layer we also utilized *early stopping* as a callback function, with *patience* set to 20, as well as the *model checkpoint* callback in order to store the model in the instant in which the balaced accuracy is at it's maximum. Our model was trained with 200 epochs.

For the Logistic Regression, we used a model with the "lbfgs" solver, and 3000 iterations to predict the classes.

2.1.3 Data Balancing

Before trying the data balancing methods, a base test was done to obtain a reference, so that the improvement of the methods tested could be measured. The CNN managed a better score than the Logistic Regression.

For balancing, firstly class weighting was tried by using compute_class_weight (from *skelarn*), which computes weights of each class by dividing the number of samples by the product of the number of classes and the number of samples from said class; this improved the Logistic Regression model considerably, but not the CNN, which showed better results without it, so we moved on to other methods.

Next, over and under sampling was then tried, using *SMOTE*, *ADASYN* (Adaptive Synthetic Algorithm, similar to SMOTE, but the number of samples generated depends on the local distribution of each class), *ENN* (Edited Nearest Neighbours, , which removes samples closer to the decision boundary) and *SMOTEENN* (*SMOTE* + *ENN*).

For the Logistic Regression, the improvement was not as great as class weighting (*ADASYN* was the technique that showed the best results, value shown on table 5).

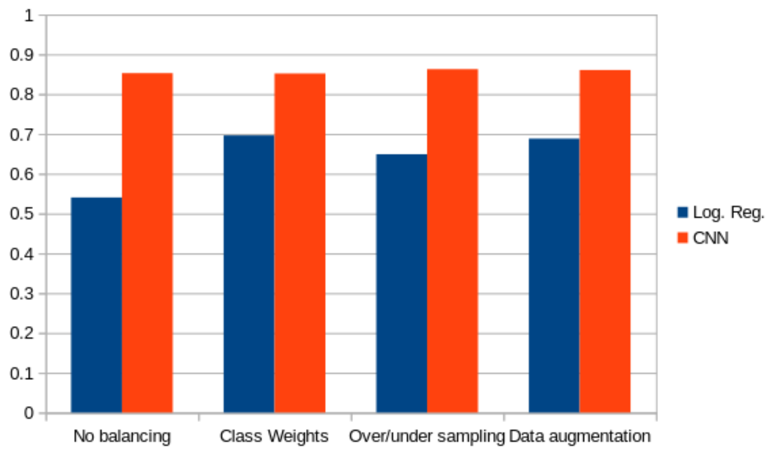
For the CNN, it improved over the unbalanced set, which was a step in the right direction. *SMOTEENN* was the technique with the best results. The values on Table 5 were obtained using this technique.

Finally, data augmentation using image flipping, rotating, and shifting was used to try and balance the dataset. The results for the Log. Regression model were an improvement over the reference value, but not as much as the class weighting technique. For the CNN, the results improved over the reference as well, getting a balanced accuracy just below the over and under sampling combination (SMOTEENN).

In the end we decided to use the data augmentation, as with it we got smaller losses when compared to over/under sampling.

Table 4: Balanced Accuracy of Val. set for both models with various data balancing techniques

	Log. Regression	**Convolutional NN**
No data balancing	0.5404	0.8526
Class Weighting	0.6962	0.8518
Over/Under Sampling	0.6489	0.8626
Data Augmentation	0.6882	0.8604



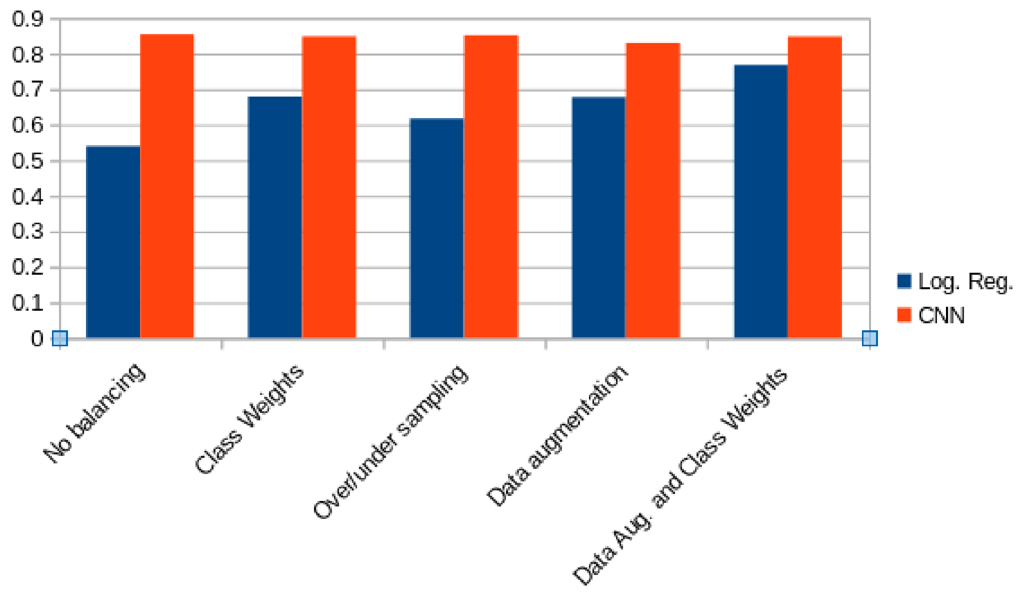
2.1.4 Post Submission Review

The code submitted resulted in a balanced accuracy of 0,6686 for the BAcc_In_Domain set, and 0,7125 for the BAcc_Out_Domain set, which showed a poor result, so the code was reviewed in search of possible causes of this poor performance.

After realizing that the model had been trained to target the validation data, and that it would not the best metric to evaluate the model, the tests were reevaluated with a separate test set, with data that was not involved in any part of the training process. The values for the accuracies are shown below.

Table 5: Balanced Accuracy of Test set for both models with various data balancing techniques

	Log. Regression	**Convolutional NN**
No data balancing	0.5401	0.8552
Class Weighting	0.6789	0.8490
Over/Under Sampling	0.6172	0.8517
Data Augmentation	0.6776	0.8307
Data Aug. + Class Weighting	0.7687	0.8484



The results from the Test set now show that, for the CNN model, none of the techniques offer any improvement. To see if the model layers and data augmentation were perhaps poorly set, some modifications were tried to them, but regrettably none resulted in better results. Additionally, a combination of data augmentation and class weighting was tested, and it improves the results of the Log. Regression model considerably, but like the other techniques, it does not improve the CNN. We also produced the confusion matrix for the new data augmentation:

		Predicted		
		Negative	Positive	Total
Actual	Negative	996	100	1096
	Positive	67	88	155
	Total	1063	188	1251

As it can be seen, the number of false positives is very high, higher than the amount of true positives. This means our model has a lot of trouble identifying melanomas. On the other hand, the number of false negatives is very low when compared to the amout of true negatives. This suggests that the augmentation was not implemented properly, as the melanomas were the class with less elements. The identification of nevu is relatively successful, which can not be said for the melanomas.

2.2 Second Task

For the second classification task, we were required to adapt the previous program to do multi-class classification. There were now 6 possible classes, coming from two distinct data-sets: dermoscopy and blood cell microscopy. The first three classes belong to the first data-set, while the latter belong to the second one.

Using the information acquired from the post submission review above, we made some changes to the program. The training data was now split into three parts: *train*, *test* and *validation*. The CNN was also changed to the architecture shown below:

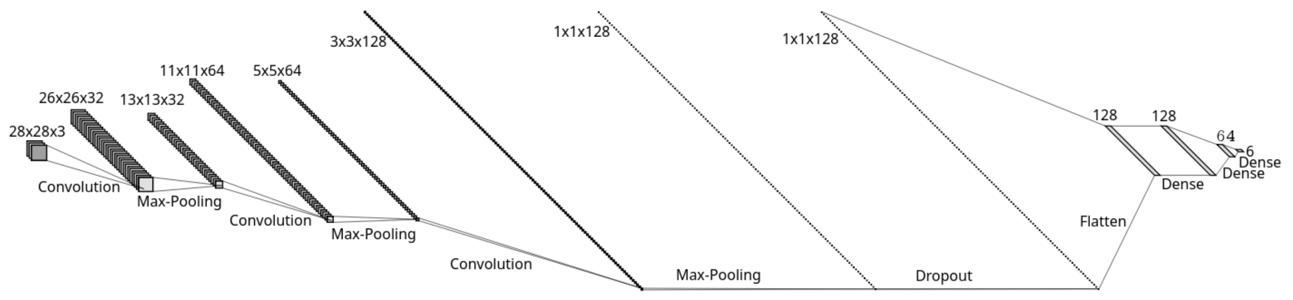


Figure 4: CNN for second task

As it can be seen, several layers were added: one convolution, one max-pooling and one dense. The convolution layer filters have also changed, increasing in size from 32, to 64, to 128. An extra dense layer was added, to account for this change and the increase in classes.

The data augmentation algorithm was also changed and class weights were again tried. It now iterates over every class except the one with most samples, and creates all variations of each image (horizontal flip, horizontal flip and 90 degree rotation, vertical flip, 90 degree rotation, shift left, shift right), adding them to the train set. It iterates until it goes trough all images of the same class, or until the ratio between the amount of images from that class and the amount from the biggest class is bigger than 0.9.

As for the class weights, we utilized the function `compute_class_weight`, using *'balanced'* for the class weight. We then represent the weights as a dictionary in order to be able to use them for 6 classes. Below are shown the results obtained from the two approaches described above:

Table 6: Balanced Accuracy of Val. set for both models with various data balancing techniques

	Balanced Accuracy
Class Weighting	0.91379
Data Augmentation	0.92341

As it can be seen, data augmentation again produced a better result, so it was the method chosen for the submission. Again, the results obtained were not very good. Some changes could be done to the approach, in order to take advantage of the fact that there are two different data sets and their respective characteristics. Firstly, using two different CNNs, one for each data set, would've likely yielded better results, since as it is, there may be too much data for only one neural network, and with diferent characteristics. Ideally, we would've identified to which data set a certain image belonged in order to know what CNN to use.

The data augmentation process could've also been improved: the approach taken can still leave certain classes with far less elements than others, and even those that have more elements can still not have enough to match the bigger class. Implementing class weights along with data augmentation would've improved this scenario, giving bigger imporance to images from class with less elements, resulting in a more balanced training set.

3 Conclusions

Throughout the development of this project, we were able to deepen the knowledge of machine learning methods acquired in the theoretical and practical classes. Trying several methods for each problem allowed us to see what limits certain methods and what works in what contexts. The deliverable results also gave us the opportunity to further identify problems in the approaches we ended up taking, with discussion in laboratory classes further clarifying doubts. Overall, although the deliverable results were subpar, we still fell like we learned and were able to apply a lot of what was taught.