TÉCNICO LISBOA

Machine Learning

MEEC

Final Report

Authors:

André Godinho (99892) andre.v.godinho@tecnico.ulisboa.pt
Ricardo Silva (100071) ricardo.querido@tecnico.ulisboa.pt

Group 116

2023/2024 – 1º Semester, P1

1 Part 1 - Regression with Synthetic Data

The Part 1 of this assignment focuses on regression analysis using synthetic data. The training dataset for both tasks related to this part comprises multi-feature vectors for each of the samples, along with corresponding outcomes, which we will use to build an effective linear predictor to estimate accurate outcomes.

1.1 First Task - Linear Regression

1.1.1 Problem Statement

In this problem, we set out to create a linear predictor for a given dataset with 15 samples, each comprising of 10 distinct features, along with corresponding outcomes. Our task is to develop a linear model that can accurately predict outcomes based on these feature vectors. The metric used to evaluate each model was the Sum of Squared Errors (SSE) which is defined by

$$\text{SSE} = \sum_{i=1}^{n} \left[y^{(i)} - \hat{y}^{(i)} \right]^2$$

1.1.2 First approach

To address this problem, we undertook a comprehensive exploration of the fundamental techniques of linear regression. Firstly, we started by crafting a direct linear model to understand how this model would fit the data given. The liner regression model is given by the following equation:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

where $p = 10$, as there are 10 features. We then computed the SSE and the coefficient of determination, R^2, which values were 5.32136 and 0.9246, respectively.

1.1.3 Cross-validation

These results already shows us that this model seems to have a pretty accurate prediction based on the training data. Since we were training the model with all the data available and no data is being used to validate predictions, our predictor could be suffering **overfitting**. In this case, our model fits really well to training data but when new data is fed to the model it may not perform that well. Therefore, in order to get a more reliable and unbiased picture of our model's performance, we started using **cross-validation** in our next models. The cross-validation method that we decided to use was leave-one-out, since it isn't computationally expensive to perform in small datasets and doesn't require to specify any configuration. The overall idea behind LOO, is to generate multiple rounds of cross-validation using different subsets of training and validation partitions, in this case 14 samples to train and 1 to validate, where the validation results are combined over the rounds to give an estimate of the model's predictive performance. This is the output of the linear model using cross-validation:

1.1.4 Regularization

Because we have a small number of samples represented with a high number of features, there is a high risk of the model over-fit the training data.

Lasso and Ridged are two regularization techniques that we used to add a penalty term to the loss function during training and thus making the model less prone to overfitting.

1.1.5 Ridge Regularization

Ridge modifies the over-fitted or under fitted models by adding the penalty equivalent to the sum of the squares of the magnitude of coefficients. It's cost function is defined below.

$$E_{\text{Ridge}} = \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|^2$$

In the cost function, the penalty term is represented by Lambda λ. By changing the values of the penalty term function, we are controlling the penalty term. The higher the penalty, the more it reduces the magnitude of coefficients. Therefore, it is used to prevent multicollinearity between features. and reduce the model complexity.

1.1.6 Lasso Regularization

On the other hand, Lasso modifies models by adding the penalty equivalent to the sum of the absolute values of coefficients.

Lasso regression also performs coefficient minimization, but instead of squaring the magnitudes of the coefficients, it takes the true values of coefficients. It's cost function is defined below.

$$E_{\text{Ridge}} = \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|$$

This small change on the cost function leads Lasso to be particularly useful for feature selection because it tends to drive some coefficients to zero, effectively eliminating those features from the model. Conversely, Ridge shrinks the coefficients but does not force any of them to be exactly zero. It typically retains all features in the model.

As we intend to exploit the most of feature selection with this regularization techniques in order to simplify the model and make it more interpretable, we emphasize Lasso's more in this report.

1.1.7 Lasso and Feature selection

In order to use the Lasso method properly, we first ran LassoCV using leave-one-out cross-validation to test and run an array of alpha parameters (penalty term) in order to understand which one would minimize the SSE. This was its output:

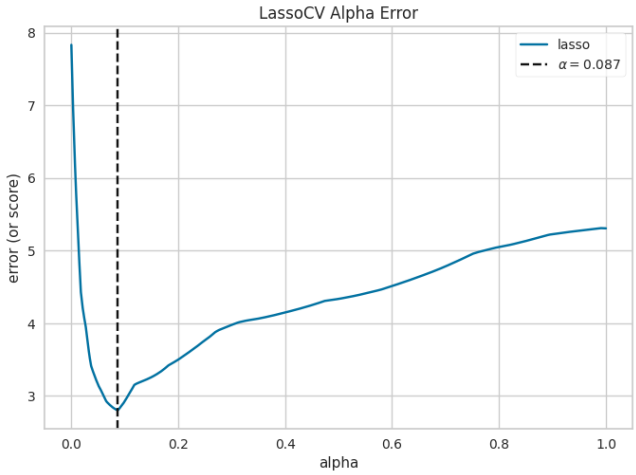

Figure 1: Cross-validated alpha-selection

We then proceeded to conduct experiments with the Lasso algorithm using $\alpha = 0.087$. We observed that not all features held equal importance in predicting the output. In fact, some of them appeared to be irrelevant or redundant. Lasso assigned the following weights to the different features:

Feature	1	2	3	4	5	6	7	8	9	10
Weight	**0**	-0.183	0.348	-0.067	**0.004**	0.57	-0.145	**0.018**	**0.006**	0.191

Table 1: Normalized weighted feature vector

To note that we standardized the feature vector so that t makes features comparable. That was made to center the data around a mean of 0 and a standard deviation of 1.

After some experiments, we decided to consider irrelevant all features with weight which absolute value is < 0.05. So from now on we discarded features 1, 5, 8 and 9.

1.1.8 Evaluation and Results

We then employed a bunch of different methods and got the results presented on Table 6.

Analysing the results, we concluded that removing the features increased the performance of the models that don't intrinsically remove these features, such as Ridge and Linear regression.

The model which end up getting the best scores, and thus, the one we chose to deliver, was the **Linear regression** with CV, excluding the features mentioned before. The choosing process was pretty straight forward since this model represented both the best score and the lowest SSE. We ended up achieving **10th place**, affirming that the linear model provides a pretty accurate solution for this problem.

Model	SSE	R^2
Linear regression	5.32136	0.92459
Linear regression with CV	7.86417	0.88856
Polynomial regression	2.43815	0.96545
Polynomial regression with CV	21.97484	0.96545
Lasso with CV	2.80591	0.96023
Ridge with CV	3.15431	0.95530
Lasso with CV excluding features	1.91738	0.96563
Ridge with CV excluding features	1.91441	0.97287
Linear regression with CV excluding features	**1.83866**	**0.97394**

Table 2: Results for the different tested models

1.2 Second Task - Linear Regression with Two Models

1.2.1 Problem Statement

The second problem closely resembles the previous one; however, the available data was generated by two distinct linear models. The challenge in this case was to initially find a method for dividing the dataset into two clusters, each generated by one of the two different models. Subsequently, the same methods employed in the first problem were to be applied.

The training dataset for this task comprised 100 samples, each represented by four features along with corresponding outcomes.

1.2.2 Ransac

For this data grouping, we experimented with two distinct approaches. The first approach utilized the RANSAC estimator, a method designed to estimate parameters of a mathematical model from a dataset that may contain outliers.

RANSAC operates by randomly selecting a minimal subset of data points, fitting a model to this subset, and then verifying the model against the remaining data. Outliers that do not fit the model well are disregarded. To implement this, we utilized the RANSACRegressor() method from the sklearn library. To assess its performance, we created linear models for both inliers (cluster 1) and outliers (cluster 2). The results are displayed in Table 3.

1.2.3 Gaussian Mixture

After the Ransac approach we decided to take a closer on how the data was distributed. An histogram regarding the features values distribution is presented on Image 2. The conclusion is that the the distribution of the feature values resembles to a mixture of multiple Gaussian (normal) distributions. Thus, and to explore data grouping from a different perspective, we employed a Gaussian Mixture Model (GMM).

Gaussian Mixture Models are a type of probabilistic model that assumes data points are generated from a mixture of several Gaussian distributions. By utilizing GMM, our aim was to uncover the underlying structure within the dataset and identify distinct clusters generated by the two different linear models.

So we used the GaussianMixture function from the sklearn library, setting the number of components to 2 to represent the two distinct clusters in the data. The results are displayed in Table 4.

	Size	Scores	
		SSE	R^2
Cluster 1	73	1.35089	0.94224
Cluster 2	27	2.37391	0.98416

Table 3: Ransac performance

	Size	Scores	
		SSE	R^2
Cluster 1	57	1.56197	0.96278
Cluster 2	43	1.57660	0.97593

Table 4: Gaussian Mixture performance

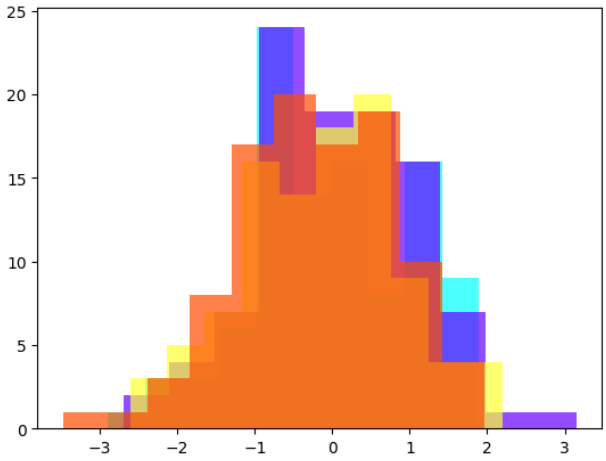

Figure 2: Distribution of feature values

1.2.4 Discussion of Results

Given the nature of the problem at hand, involving distinct clusters generated by different linear models, we opted to use the Gaussian Mixture solution over RANSAC for the fact that Gaussian Mixture extracted two balanced clusters with two pretty strong linear regressions. While Ransac extracted two highly imbalanced clusters, leading to a strong linear regression for one cluster and a poor one for the other.

This decision ended up securing us **28th place**. Although the result was positive, it shows that probably Gaussian Mixture was not the optimal solution for the data clustering.

2 Part 2 - Image Analysis

The second part of this project delves into the realm of image analysis. In this section, we are presented with two classification challenges involving dermoscopy and blood cell microscopy images.

2.1 First Task - Binary Image Classification

Our objective in this task is to develop a robust machine learning model that can accurately distinguish between two key categories of dermoscopy images: melanoma and nevu.

The dataset for this task includes 6254 samples of 28x28 pixel dermoscopy images, each represented in RGB color space, resulting in a feature vector with 2352 elements (28x28x3). The labels for this task are binary, with 0 representing nevu images and 1 representing melanoma images.

One important note is that the dataset is imbalanced, with a significant difference in the number of melanoma (14.3%) and nevu (85.7%) images in the training dataset. Therefore, we

must address the challenge of imbalanced data during model training and evaluation. There are a lot of ways to tackle this issue, we chose to increase the number of samples of the minority class but could also have under-sampled the majority class. We didn't choose the latter because we could lose valuable information in doing so.

2.1.1 Data Augmentation

In order to help the model generalize better and reduce the risk of overfitting to the majority class, we applied data augmentation to the original data. This was achieved by doing a two step process: balancing the training dataset and augmenting the validation dataset and the already balanced dataset, where we splitted the original dataset in 70% and 30% partitions, being the biggest for training. In the first step and in the process of augmenting the validation dataset we chose to apply random flip, brightness, translation and zooming transformations. In order to avoid *data leakage*, we used different transformations such as rotations and contrast. Even though applying data augmentation is a great method to feed data that the model will never see twice, the inputs it sees are still heavily intercorrelated because they come from a small number of original images, and since we can't produce new information, we can only remix existing information. As such, this may not be enough to completely get rid of *overfitting*. As to why we didn't balance the data, we knew that the test data is taken from an unbalanced dataset, therefore the chances that it would be unbalanced were high, and that's why we kept the validation test unbalanced, in order to recreate a similar profile to the test one. For some reason that we can't explain, when data augmentation was applied to the already balanced training dataset the model's performance seemed to improve, but when comparing to the performance of our model when we didn't the performance was way worse. After all the transformations we got a training and validation dataset sizes of 31844 and 9385 images, respectively.

2.1.2 Convolutional Neural Network

The model that we chose for this task was the CNN, a class of deep learning models that have proven to be incredibly effective in various computer vision tasks. They are specifically designed to process and analyze visual data, making them ideal for tasks like ours. To approach the architecture problem we firstly thought about doing transfer learning but since the majority of available models were trained in images of higher resolution, like the VGG16 that was trained with 224x224 RGB images, this wasn't a viable option. Therefore, with inspiration in the 'Deep Learning with Python' book, written by François Chollet, we came up with the following architecture:

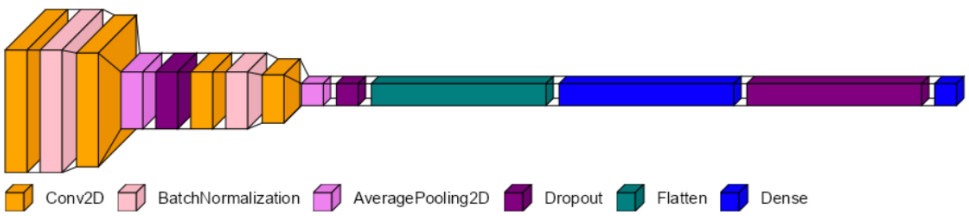

Figure 3: CNN architecture

One of the key elements that we kept thinking while building our CNN was the characteristics of the input. Since we will feed low resolution images to our model we opted to use padding and Average Pooling instead of Max Pooling in order to utilize the most amount of information possible. As we stated in the data augmentation subsection, since we are generating new data that is heavily intercorrelated to the original images, we may still be vulnerable to overfitting. That's why we also added Dropout layers as a regularization method which proved a good solution to fight off overfitting. The batch normalization layers served the purpose to accelerate the convergence process and reduce sensitivity to the initial weights of the network.

2.1.3 Evaluation and Results

As we can see in the table below, the model that had the best performance was without a doubt the CNN. Even though the validation balanced accuracy computed was approximately 87%, this performance wasn't reflected when fed with test data. Even though it didn't perform as expected, we secured the **54th place** on the leaderboard, with a 4.5% relative error from the best 'In Domain' model and 14.5% from the best 'Out Domain' model. It is trivial that our model would perform best with the 'In Domain' test data since they share similar features. Something that we could've done to improve our model was mixing multiple models, instead of really only on the CNN. In this cenario, different models would be looking to different features, resulting in a more reliable classifier for our task.

Model	Validation Balanced Accuracy
Convolutional Neural Network	0.8745
Random Forest Classifier	0.7897
Support Vector Machine	0.7654
Multilayer Perceptron	0.7061

Table 5: Results for the different tested models

2.2 Second Task - Multiple Class Image Classification

In contrast to the last task, this one presents six classes of images and come from two distinct datasets: dermoscopy and blood cell microscopy. The class labels are the following, 0 (nevu), 1 (melanoma), 2 (vascular lesions), 3 (granulocytes), 4 (basophils), and 5 (lymphocytes), where the first three classes come from the dermoscopy dataset and the latter from the blood cell one. As displayed in the Image 4 the problem of splitting the datasets won't be complex since the difference in the images is clear as day. This task, similarly to the previous one, presents us another problem of an unbalanced dataset, since the portion for each class is 50.4%, 8.4%, 1.1%, 21.7%, 9.3%, 9.1%, respectively. The major problem in this task will be our choice of model to approach this problem. Should we stick to the solution of task 3 or should we explore different models?

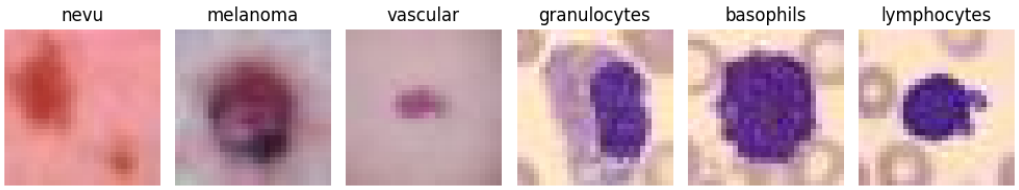

Figure 4: Different classes of images

2.2.1 Choice of Model

In order to answer our questions we started by applying the same strategy as we did on the previous task, we balanced the training data and fed it to a CNN with six outputs. After experimenting with the task 3's solution we explored another path where we create a model built upon 3 different models as we can see in the Image 5. The Model 1 had to have a validation balanced accuracy close to 100% in order to avoid accumulating error, which would be detrimental to our model's performance. As we previously said, this task wouldn't be too complex, with a simple CNN the model quickly reached a 99.99% validation balanced accuracy. Now for both children models we tried mixing more than one model by doing the weighted average of the probabilities of each model's guess since it proved to be a good solution for task 3. These were the results built upon the model 2:

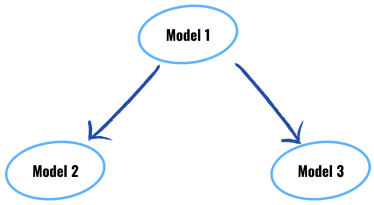

Figure 5: Our model's structure

Model	Validation Balanced Accuracy
CNN	0.8695
CNN + HGC	0.8826
CNN + RFC	0.8836
CNN + SVC	0.8844
CNN + SVC + RFC	0.8732
CNN + SVC + HGC	0.8638
HGC + SVC + RFC	0.8057

Table 6: Results for the mixed models tested for Model 2

2.2.2 Discussion of Results

As we can see by looking at the confusion matrixes, the biggest problem was having a good solution to separate nevu images from melanoma, like we did in task 3. At the end we got two final best models, the CNN with 6 outputs with a VBA of 87.58% and the branched model built upon 3 CNNs which had a VBA of 87.69%. Even though both models present similar performances, in the end, we chose the latter model which granted us the **4th** place in the leaderboard.

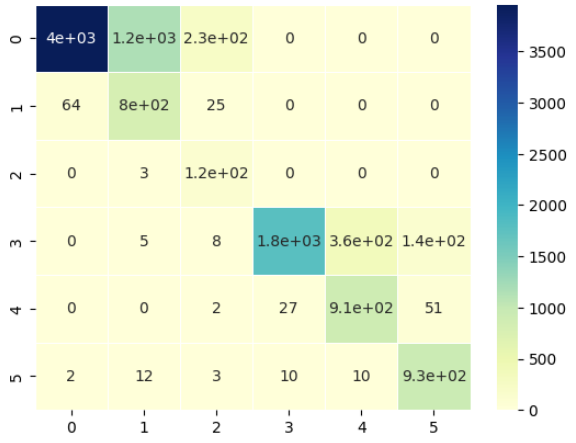

Figure 6: Confusion Matrix of the CNN with 6 outputs

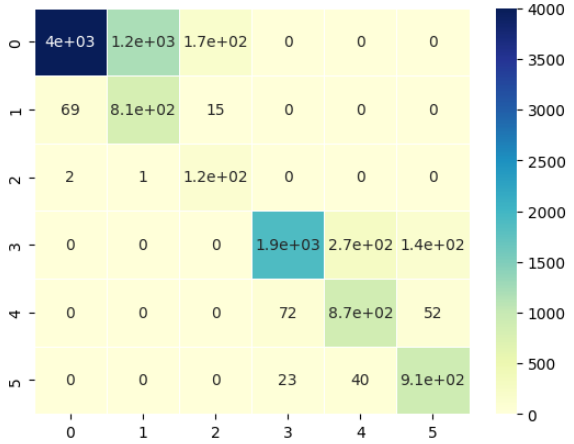

Figure 7: Confusion matrix of the branched model