Programming Assignment Report

Gianluca Covini, # 526045

9 gennaio 2026

The aim of this assignment is to build a classifier able to identify the seven species of beans Barbunya, Bombay, Cali, Dermason, Horoz, Seker, and Sira, that we will indicate with numbers from 0 to 6.

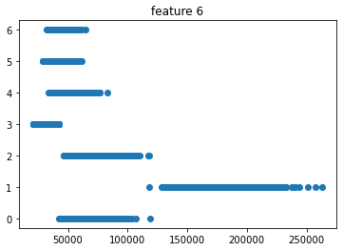
In this report, we present the results generated by the code found in the accompanying file, `MLExam.py`. The scripts within this Python file are organized in the same sequence as the results discussed herein, facilitating easy cross-reference.

Analysis and comment of the data

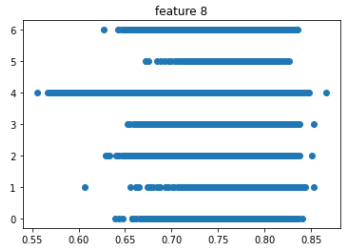
The data we have available for our project is made of 13611 samples which consist in the description of beans through 16 features that we indicate with numbers from 0 to 15. 11611 samples (85%) compose the training set, while validation and test set are composed of 1000 samples each.

To visualize the data we implemented two functions: the `plot_features1d`, which shows for each class a scatter plot of a single feature, and the `plot_features2d` which shows two different features in a 2d scatter plot and uses the colours to highlight the classes. These functions help identify key features.

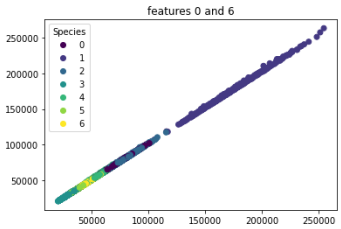
The following figures illustrate some feature distributions within the training set:



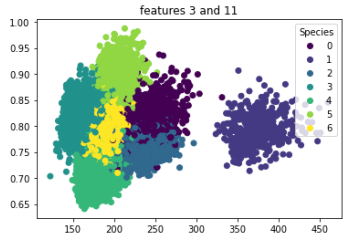
(a) Feature 6



(b) Feature 8



(c) Features 0 and 6



(d) Features 3 and 11

As we notice from the plots some features (for example the feature 6) seem to be more discriminative than others (such as the feature 8) as they assume different values for different classes. So, looking at the plots, we can do a first estimate of which features are more discriminative. We can then verify it by calculating the variance of the features between the classes using the function `between_class_variance`; we notice that the features with high variance (that will probably be more discriminative) are features 0, 1, 2, 3, 6 and 7, while features 8, 9, 12, 13 and 15 have a low variance; the other features assume medium values.

We can then do other exploratory analysis looking at the 2d plots, in order to find correlations between feautres. In fact, we do not only want reliable and discriminative features themselves, but also features which are not strongly correlated, as it would result in redundant data. From the plots we can observe that some features (such as the 0 and the 6) are strongly correlated, while other (such as the 3 and the 11) are not. As before, we can verify our observations mathematically; for this aim, we implemented the function `feature_corr` which calculates the correlation coefficient between for each couple of features. In fact, for example, the correlation coefficient between feature 0 and feature 6 is almost 1, while between feature 3 and feature 11 is -0.01. These results are easily explainable by looking at the interpretation of the features: feature 0 represents the area, while feature 6 the convex area, so it is easy to understand why the two values are so strongly correlated. Other correlations happen in the same way: for example, between features 0 (area) and 1 (perimeter) we have a correlation of 0.97.

To conclude our exploratory analysis we can compute some statistics of the data in the training set; these could suggest us how to proceed in the next sections; to do so, we implemented the function `extract_stats`. From the computation we notice that the range of values assumed by different features changes a lot. The feature 0, in fact, assumes values between 20 420 and 254 616, while the feature 13 between 0.0006 and 0.0037. Also the variances are very different, going from 3.5×10^{-7} of feature 13 to 853383622 of feature 0.

Data pre-processing

The observations made in the section above suggest us how to proceed in designing and implement a suitable data pre-processing procedure.

While extracting the statistics from the training data, in fact, we already highlighted the strong differences that occur between ranges and variances of the different features. This makes necessary to apply a normalization to the features in order to obtain good results from the models we will train. We can choose between several feature normalization techniques: good choices seem to be min-max scaling and mean-var scaling, since the minimum, maximum, mean and variance of the different features have very distant values. A good choice could also be whitening, because it also solves the problem of correlation between features. On the other hand, istance normalization and max-abs normalization seem to be not so useful: the first because the features represent different measures, and the second because we don't have strong preferences for sparse solutions.

We look for the best choice by applying different normalizations to our data and calculating the accuracy on the validation set. For this process we use a multinomial logistic regression model with L2 regularization. The learning rate is set to 10^{-3} and the regularization parameter λ to 0.5 (we discuss the implementation in the next section).

The results obtained speak for themselves:

Normalization technique	**Validation accuracy**
Min-Max normalization	25.5%
Mean-var normalization	69.3%
Whitening	90.8%

It should be noted that the algorithm fails to find a solution when the data is not normalized.

As we expected the model trained after a whitening transformation outperforms the other normalization techniques. So, from now on, we will assume that the data we will use only data already normalized with whitening.

Another important pre-processing step regards the feature selection. Too many features, in fact, can lead the model to overfitting, in particular in this case, as we observed in the previous section where we have features strongly correlated or not so discriminative and reliable. A way to reduce the number of feature is to apply a recursive feature elimination. We implemented this technique in the function `recursive_feature_elimination`, using once again the logistic regression model we introduced above.

The result of the RFE is the elimination of the ninth feature: without this feature, in fact, we obtain an accuracy over the validation set of 90.11%, greater than 9.08% obtained without

elimination. Even if the validation accuracy is similar, we decided to eliminate the ninth feature for the sake of simplicity. For now on, if not explicitly said, we will consider the data without the ninth feature.

We tried also to apply the Principal Component Analysis but we noticed that also in this case the validation accuracy do not increase substantially (we obtain again an accuracy of 9.08%). In this case we decided to not apply PCA to our data in order to mantain more information.

Lastly, we want to select two features in order to build a classifier that can be drawn on a plane. We tried using PCA but the validation accuracy is too low (25.5%), so we proceeded considering each couple of features and computing the accuracy over the validation set. We used using the same logistic regression model. A decent result was obtained using the features 12 and 13 (of the original data, without elimination) as the validation accuracy reached 55.6% (the highest obtained).

The training data of the two features can be visualized in the following plot:

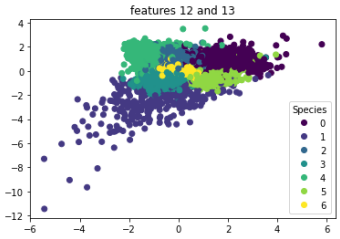


Figura 1: Two features data

Classification models and analysis

In this section we train different classification models to achieve our initial task. Since it is a multinomial classification problem we have different valuable options: we will see in detail the implementation and the results of multinomial logistic regression and multi layer perceptron.

For each model we will use data normalized with whitening and with the ninth feature removed, as we said in the previous section.

Logistic regression

The first model we implemented is a multinomial logistic regression. While choosing a linear model, we preferred it to SVM because of its greater ability to adapt to multi classes problems. We implemented a `multinomial_logreg_train` function that trains the models and returns the weights and the biases, a `multinomial_logreg_inference` which caluclates the predictions, and a `accuracy_multinomial_logreg`. The model was implemented with L2 regularization to avoid overfitting. We adopted this choice since we do not have preerences for sparse solutions that would justify the use of L1 regularization.

Due to computational limitations, we opted for exploratory analysis over grid search for hyper-parameter tuning. We found that a learning rate of 10^{-3} and a parameter λ equal to 0.1 are values good enough, since they result in a validation accuracy of 90.9%.

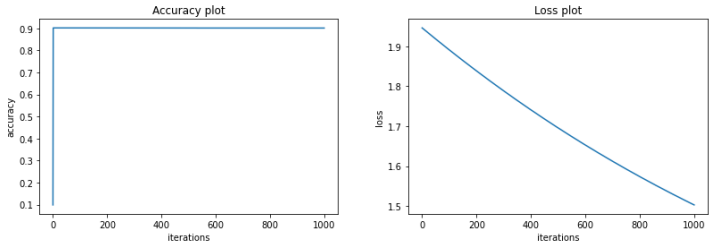
We then let the model work and here are the results obtained:

Train accuracy	**Validation accuracy**	**Test accuracy**
90.2%	90.9%	90.5%

The results are good: the model tends not to overfit and obtains quite good results on the training set.

We can make other interesting observations: looking at the accuracy-iteration plot we observe that the algorithm converges very rapidly and the accuracy then tends to slightly reduce in the following iterations.

An idea to overcome this phenomenon is to use the early stopping technique, reducing the number of iterations based on the training accuracy or validation accuracy results.



(a) Accuracy plot (b) Loss plot

Furthermore we can study the weights obtaineds: generally they are well balanced, since they assume similar small values (the means of the weights over the different classes are of similar orders of magnitude). This means that the model is well trained, thanks also to the L2 regularization, since it relies on different features, so it is more prone to generalization.

Finally, the model applied to the two-features data yielded these results:

Train accuracy	**Validation accuracy**	**Test accuracy**
56.7%	55.6%	53.3%

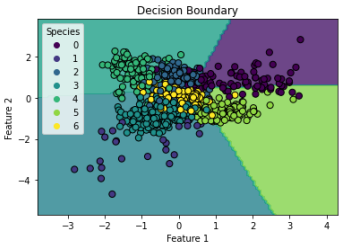


Figura 2: Logistic regression on the two-features data

The model is underfitting, so a possible idea to improve the results is to increase the complexity of the model moving beyond the linearity of the logistic regression.

Multi Layer Perceptron

Given the abundance of data available for our project, Artificial Neural Networks (ANNs) emerge as a highly viable approach for classification tasks. Specifically, we will employ a Multi-Layer Perceptron (MLP) model for our analysis.

The training of the Multi-Layer Perceptron offers several possibilities, given the variety of network structures and parameters involved. Our network will have 15 input neurons (the number of features used) and 7 output layers (the number of classes), but we will try different combinations for the hidden neurons.

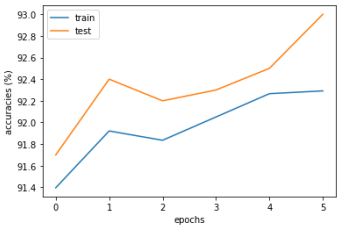
We will train the network using stochastic gradient descent with mini batches, so the first consideratinos involve the dimension of the mini bathces.

Here are the results varying the dimension of batches using a network without hidden layers; we are training the mlp for 5 epochs:

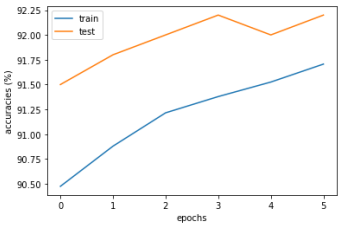
Batch dimension	**Train accuracy**	**Validation accuracy**	**Test accuracy**
10	91.5%	92.3%	92.3%
5	91.7%	92.4%	92.2%
1	92.3%	93%	93%

To decide the parameter we have to rely on the validation accuracy. The result is that the model with mini-batches of dimension 1 achieves the best results. We also notice, as expected, that

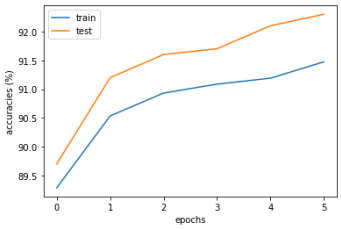
the use of small minibatches makes the convergence more noisy (even if it has several advantages). In fact, it is interesting to observe that the accuracy plot we have monotone convergence only for bigger mini-batches, even if they all converge to the solution.



(a) Mini batches dimension of 1



(b) Mini batches dimension of 5



(c) Mini batches dimension of 10

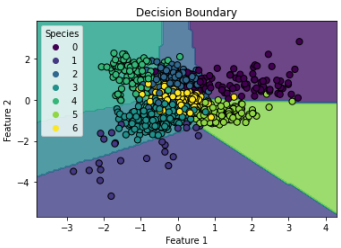
The second question arises from network architecture. We tried different possibilities varying depth and width of the hidden layers and calculating the accuracy results. We fixed the mini batches dimension to 1 and the number of eopchs to 5. We expect that the deeper the network the highest the accuracy. Here are the results:

Architecture	**Training accuracy**	**Validation accuracy**	**Test accuracy**
0 hidden layers	92.3%	93%	93%
1 hidden layer; width 100	93.3%	93.7%	93.8%
2 hidden layers; width 100, 100	93.4%	93.4%	92.7%
3 hidden layers; width 150, 150, 150	93.6%	93.5%	93.7%

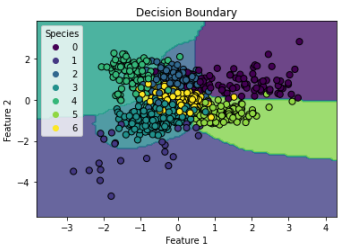
As we expected, the deeper network obtains the best results.

Lastly, we implemented the model for the two-features problem and we obtained the following numerical and visual results:

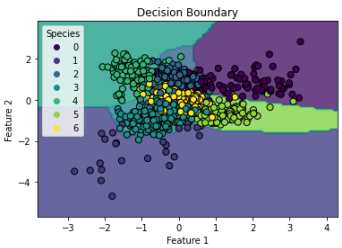
Architecture	**Train accuracy**	**Validation accuracy**	**Test accuracy**
0 hidden layers	82%	82.5%	82.9%
1 hidden layer; width 100	82.1%	80.4%	82%
3 hidden layers; width 150, 150, 150	82%	81.3%	82%



(a) No hidden layers



(b) 1 hidden layer; width 100



(c) 3 hidden layers; width 150, 150, 150

As we expected, augmenting the complexity of the model leads us to far better results. In particular a non linear decision boundary as the one computed by MLP is more suitable for our data.

Statement

I affirm that this report is the result of my own work and that I did not share any part of it with anyone else except the teacher.