Homework 2: Linear Classification

James Baldwin

1 Introduction

I perform a linear classification using the pocket algorithm, the pseudo-code for which is described in *Learning from Data*[1]. The algorithm itself, as well as all helper functions, were constructed from scratch. The exception to this is the function `init_weights`, which utilizes the built-in pseudo-inverse matrix function `pinv` from the `numpy.linalg` class in order to initialize the weights using linear regression. Vectorization was implemented where possible. Exploratory data analysis (EDA) and data transformation (DT) was carried out in order to attempt to reduce the data and improve the model's generalization capabilities. The error associated with the training data (Ein) and the error associated with a separate, unseen test set (Eout) are recorded for the untouched data and then following each stage of DT. Plots of the performance of the model are provided for each of these stages. Using various method of DT, I was able to reduce the error on the unseen test set (measured as the percentage of missed classifications) to **1.45%**.

In Section 2 I briefly discuss the source of the data and how it was accessed. In Section 3 I discuss my solution to this problem, including the EDA and DT techniques that I used to attempt to improve the performance of the model.

2 Data

The data for this exercise comes from the UC Irvine Machine Learning repository and is accessed using the `scikitlearn` databases [2, 3]. It consists of 569 data points over 30 features which are "computed from a digitized image of a fine needle aspirate (FNA) of a breast mass" [2].

3 Solution

The first step in my EDA was to take a look at the data documentation to check the data format, whether any data was missing, and to see how the targets (diagnoses) were classified. From [2] I could see that the data had 569 data points containing 30 features, which matched the shape of my dataframe after importing the dataset. I could also see that there was no missing data, so nothing needed to be done in terms of handling NaN values, for example.

Next I could see that the diagnoses were stored as a [0,1] pair corresponding to 'malignant' and 'benign', respectively.

From this I had a clear plan for my first DT, which was to adjust the targets to map to $[-1, +1]$ rather than $[0, 1]$. The reasons for this have to do with the way the pocket algorithm - a variant of the perceptron learning algorithm (PLA) - works. This algorithm attempts to map the effects of multiple features to a simple binary classification. This process is made much simpler by reducing it to a question of whether the output is positive or negative, rather than whether the output is above or below some offset (eg 0.5).

Next, I used the `describe()` function from the `pandas` library in order to take a look at the raw values and the features themselves. From this I could see an obvious next step would be to normalize the data, as the values associated with the different features ranged widely from around 1000 to less than 0.5. I utilized the `MinMaxScaler()` function from the `sklearn.preprocessing` library in order to transform all the data to be in the range $[0, 1]$.

Once the data was normalized and the target mappings updated, I decided to take a look at the correlation between the features. This idea came from a discussion with Zeehan while we were discussing the homework. To do this, I made use of the `corr()` function from `pandas`, which creates a correlation matrix of the data. I decided to only look at features that were highly correlated or anti-correlated, so I set a threshold of 0.9, and set all elements in the correlation matrix with an absolute value lower than this to zero. I used the `heatmap()` function from the `seaborn` library to create the heat map, which allowed me to easily visualize any strong correlations. The resulting heat map can be seen in Figure 7. After inspecting the map I decided to remove five features, namely: mean perimeter, mean area, worst radius, worst perimeter, and worst area. These features showed a strong correlation with the mean radius feature, and my thought was that the information from these features may all be contained in a single feature.

The next strategy that I employed in my EDA was to visualize the linear separability of each of the 30 features. As this was a binary classification problem, I felt that it might be useful to remove any features which did not show any linear separability around the diagnosis. To do this I simply plotted the values of each feature on a scatter plot, and colored the data point according to its target classification. A sampling of six of the resulting plots can be seen in Figure 8. The plots showed three different levels of linear separability: clear, ambiguous, and absent. For my model, I removed only those features which showed an absence of linear separability, and retained all clear and ambiguous features. This choice was based on the inherent error of performing this evaluation by eye. With more time, I would perform an analytical linear separation each of the features, and made a decision based on a quantitative measure of their linear separability. For these purposes, though, I felt the by-inspection method would suffice.

The last DT I performed was a combination of the previous two. Using the sparse data with the non-linearly separable features removed, I created another heat map to visualize the correlation between the remaining features. This time

I did not set any threshold as there were less features, so I wanted to retain all of the details contained in the correlation matrix. Not surprisingly, the same features from the first heat map showed strong correlations. I again removed these features from the data and ran the model again. The heat map from this step can be seen in Figure 9.

4 Training and Validation

4.1 Experiment

This experiment was done in a Jupyter notebook. The data was first loaded in, in full, using `scikitlearn`, and stored using a `pandas` data frame. The original dataset contained 569 rows and 30 columns, corresponding to the 30 features, each with 569 data points in the set. The targets were stored separately. The pipeline essentially follows the order of the notebook, so I will discuss the progression here as it pertains to how the experiments were performed.

First, the weights were initialized using linear regression inside the function `init_weights`. Inside of this function, the zeroth order weight, the bias term with a value of one, is also added to each column of the data so that it now contains 569 rows and 31 columns. Next a test set is separated out from the data on which Eout will eventually be evaluated. This is done inside the function `separate_test`. In brief, 69 random data points and their features are chosen and extracted from the data set, leaving a reduced set of 500 rows and 31 columns from which the training sets will be constructed. The training sets of variable size N are constructed using the function `create_train_val`. The different sizes for the data sets that I chose were: 10, 50, 100, 200, 250, 300, 350, 400, 450, and 500. Using the `train_test_split` function from the `sklearnmodel_selection` library, I stored the ten sets each for the inputs and targets.

With the data separated into training sets, I can now run the model. The model is an implementation of the pocket algorithm, which is itself a version of a PLA. The pocket algorithm works by making predictions on the data set using a set of weights, and then updating the weights using a misclassified point. The error for Ein is calculated using the new weights, and the procedure continues until there are either no misclassified points, or the maximum iterations is reached. The algorithm returns the best set of weights which minimized Ein over the given data set, along with the Ein associated with those weights.

In order to run this algorithm on the variable sized sets and compare them to Eout I use two more helper functions: `calculate_Ein_vals()` and `calculate_Eouts`. The former takes in the training sets and their targets and runs the pocket algorithm on each in turn, storing the weights and the Ein values associated with each into an array. The latter takes in these best weights and runs predictions on the test set with them, storing the Eout associated with the different data sizes into an array. The Eins and Eouts associated with each different value of N is then plotted, with the size of the data set on the x-axis and the error (in

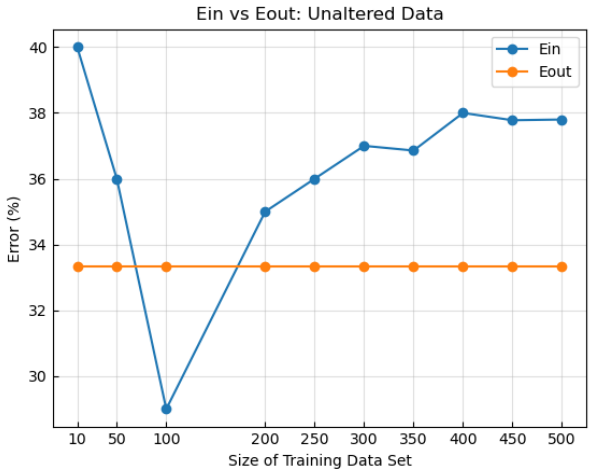

Figure 1: Ein vs Eout with no data transformation.

%) on the y-axis.

4.2 Result

The minimum error for Eout reached using the various methods, measured as the percentage of misclassified examples in the test set, is **1.45%**. The results of running the pocket algorithm on the full data set and then after each DT are presented here. In Figure 1 you can see the initial comparison between Ein and Eout. While there is an arbitrary jump at a data size of 100, in general the unaltered data produces an Eout settling around 38%, while the Eout stays constant at 33.33%. In Figure 2 we can see the immediate improvement of giving the algorithm a range that it is more attuned to evaluating. Our Ein settles to between 6 and 8 percent, while the Eout actually drops to below 5%. Normalizing sees another noticeable improvement, as seen in Figure 3. The Ein error never gets above 1%, while the Eout error generally settles around 4%. Removing highly correlated features comes with no real improvement, and in fact performs slightly worse, as seen in Figure 4. The best overall performance comes from removing features which show no linear separability, as seen in Figure 5. For data sizes of 10 and 50 the algorithm actually converges quite quickly for Ein, though at such small samples this is more due to chance than accuracy. Overall, though the Ein values remain at or below 4%, while we see a pretty strong Eout convergence around 1.45%, with a small spike at $N = 350$, though my feeling is that this would smooth out if run multiple times. Finally, one again we see that removing highly correlated features - this time from the linearly separable reduced data - also worsens performance, seen in Figure 6.

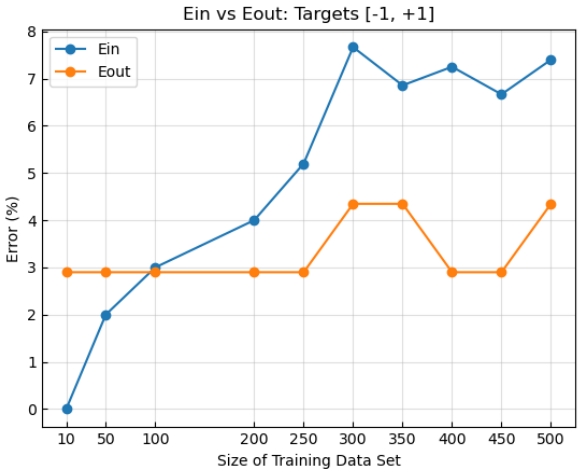

Figure 2: Ein vs Eout with targets adjusted to [-1, +1].

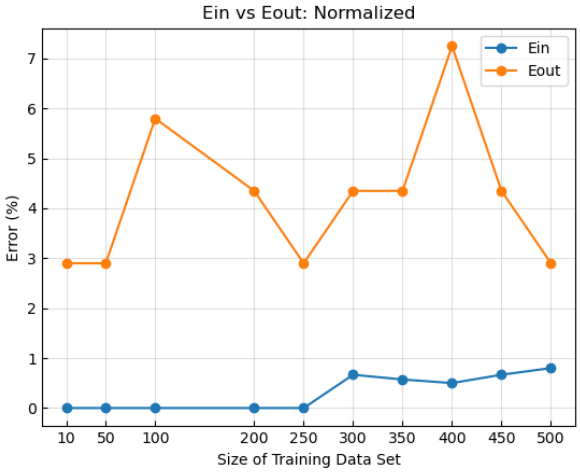

Figure 3: Ein vs Eout with data normalized.

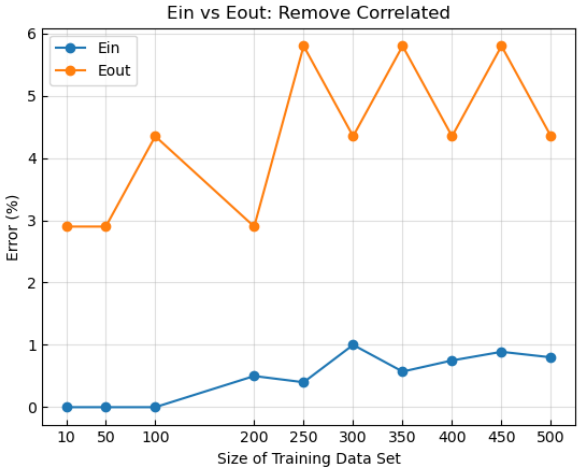

Figure 4: Ein vs Eout with highly correlated features removed.

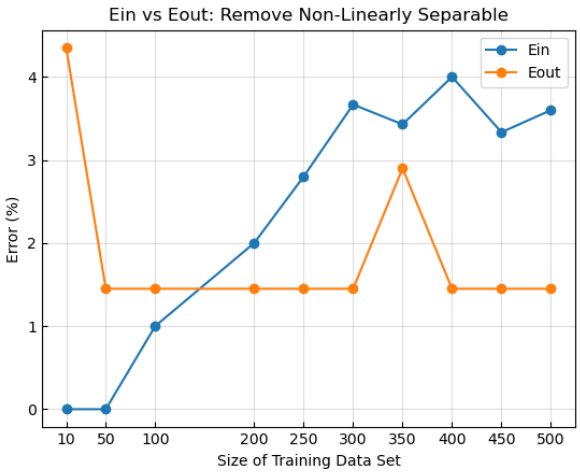

Figure 5: Ein vs Eout with non-linearly separable features removed.

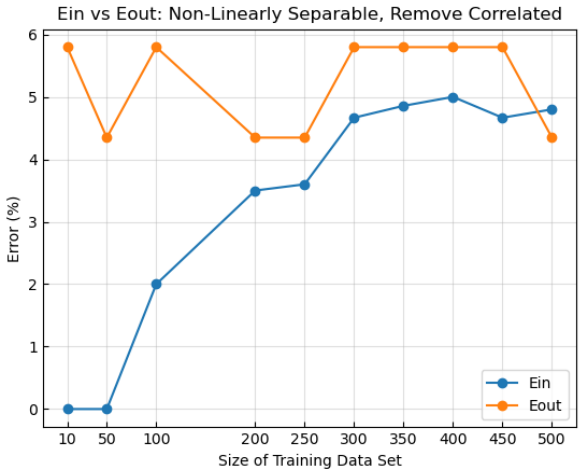

Figure 6: Ein vs Eout with non-linearly separable features, and those highly correlated among them, removed.

4.3 Discussion

The plots suggest that certain data transformations can make a lot of difference with very minimal effort, and also that some seemingly reasonable transformations can actually harm performance. These results contained both surprises and expected results. Perhaps the biggest surprise was the level to which adjusting the target range allowed the algorithm to work much more effectively. This is not surprising in hind sight, as I discussed above, but it was interesting to witness in real time how one could go from a more than 40% error to around 3% error with a single line of code difference. I think the main lesson that I took from this experiment was the idea that when training a model on a set of data, it is often best to spend the vast majority of your time inspecting and reducing your data in order to ensure you are allowing the model to work at its most effective level. The methods utilized here were mainly done using subjective measures, sometimes by eye. If given more time one would presumably like to make these data transformations in a more systematic and quantitative manner.

5 Appendix

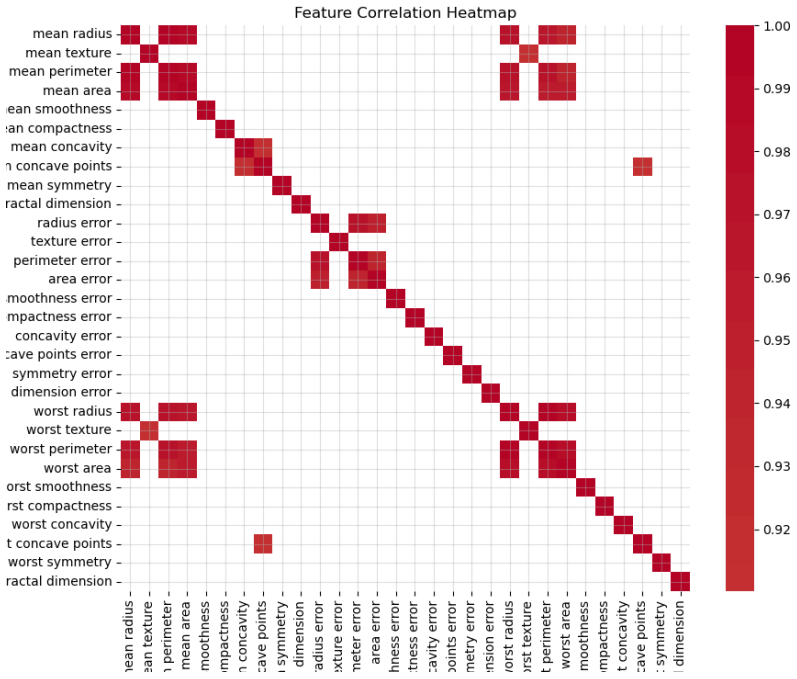

Figure 7: The correlation matrix for the entire data set (30 features). Note the strong correlation between the mean radius, the mean perimeter, the mean area, the worst radius, worst perimeter, and worst area. The hypothesis was that highly correlated features may contain redundant information. This turned out not to be the case.

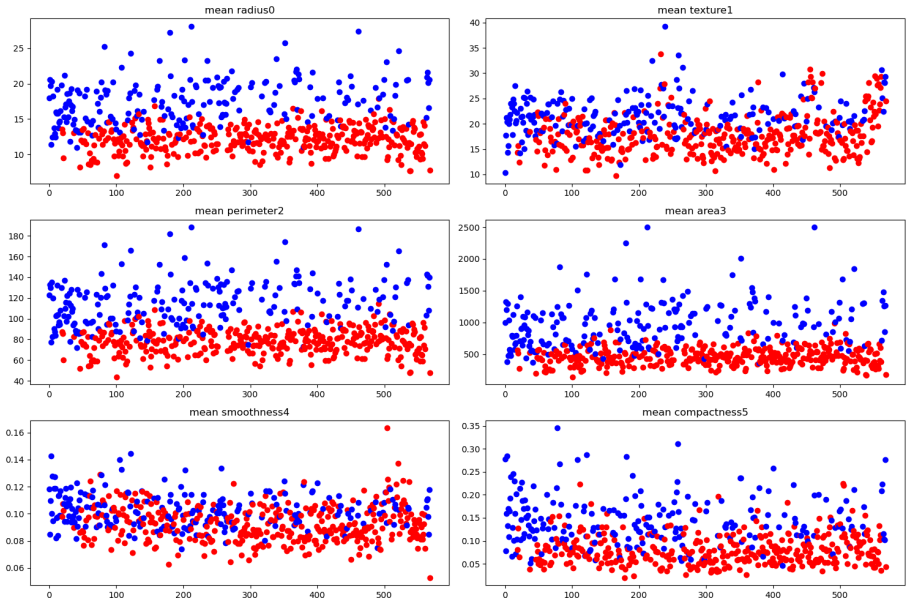

Figure 8: A sampling of the plots made in testing the linear separability of each of the 30 features in the data set. Note the clear linear separability of mean radius, mean perimeter, and mean area, the more ambiguous linear separability of the mean texture and the mean compactness, and the total lack of linear separability of the mean smoothness. In my DT I removed only those features which showed no linear separability and left those that displayed clear or ambiguous separability in the dataset.

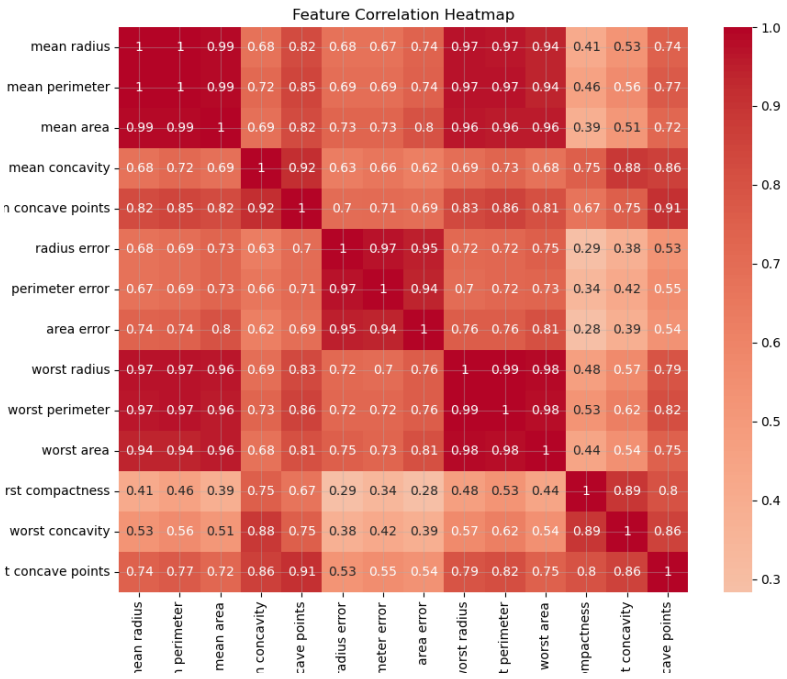

Figure 9: Correlation matrix for the reduced data set. The same features from the first correlation matrix show here, though in the reduced dataset are more clearly visible.

References

[1] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from Data*. AMLBook, 2012.

[2] Dheeru Dua and Casey Graff. Uci machine learning repository: Breast cancer wisconsin (diagnostic) data set, 2019. Accessed: 2024-10-09.

[3] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Mathieu Brucher, Matthieu Perrot, and Édouard Duchesnay. scikit-learn: Machine learning in python – ‘load$_b$$reast_c$$ancer$‘, 2024. *Accessed* : $2024 - 10 - 09$.