# MLP vs Random Forest

Lorenz Wolf

October 20, 2021

## Preliminaries

Firstly note that before starting the analysis, the dataset is split into two equally large parts, one train set and one test set. For the split sklearns StratifiedShuffleSplit is used with the seed set to zero. The following analysis will be done on the train set only and the test set will only be used for the final model evaluation.

## Data

The train set contains data on 516 individuals. The column, denoted z, is a class indicator corresponding to different chromosome types. The variables, V2, V3,. . . V12 relate to the size and shape of the chromosome and the remainder relate to quantitative descriptions of the banding pattern. The feature vectors for each individual have been normalised, so that they can be compared across different individuals.

Firstly note that all features have some missing values. Each feature is only missing very few values, thus performing listwise deletion would lead to a greater loss of data than necessary. To deal with the missing values mean imputation is performed. We note that both categories are roughly equally represented in the dataset. An inspection of the summary statistics uncovers a great variety in the sample standard deviations of the feature columns, taking values between 61 (V3) and 247 (V28), which could lead to problems in the model training. Furthermore, some of the features are strongly correlated. In particular, V2 and V3 have a correlation of 0.9949, but also the pairs 7 and 11 as well as 9 and 12 show a strong correlation.

Plotting the histograms for each feature and splitting by category does reveal some underlying structure in the distributions of some of the features, while for others nothing specific can be observed (see Figure 1). Specifically we note that for variables V2, V3, V5, V9, V12, V16, and V19 larger values tend to be more commonly associated with type 1 and rarely with type 0, while for V17 this pattern is reversed.

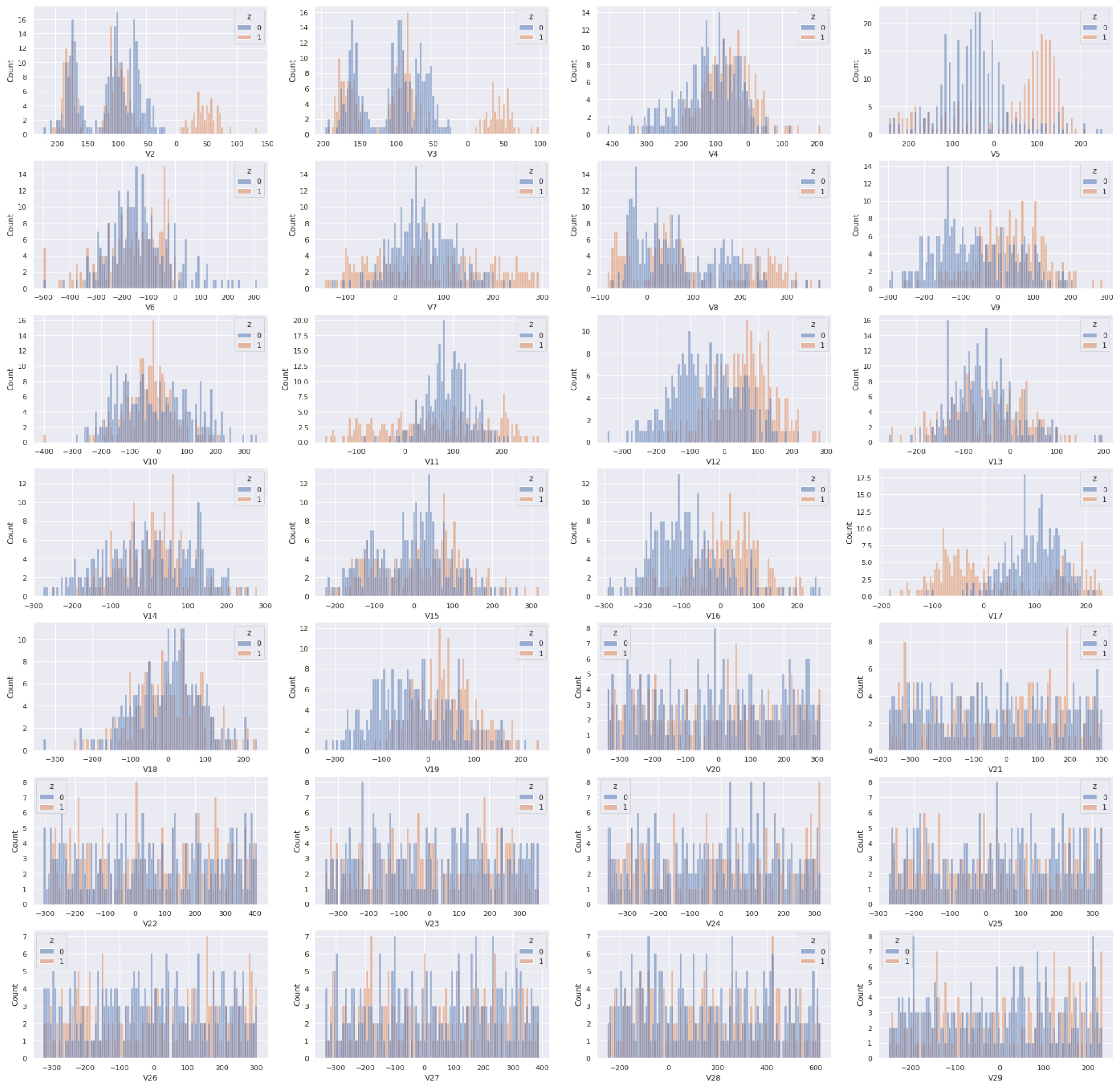

Figure 1: Histograms of the data

## Model training and tuning

Note that by definition minimising the error rate is equivalent to maximising the accuracy, which is simpler to implement due to existing metrics in the Sklearn and Tensorflow packages.

Training the random forest (RF) as well as the multi layer perceptron (MLP) requires tuning of several hyper-parameters. A common approach to tuning hyper-parameters is grid search combined with a cross-validation procedure to assess the model performance for each combination of hyper-parameters where the hyper-parameters are given by the grid and each possible combination is tested. The problem with this approach is that for a computationally expensive training process such as for the MLP classifier the grid search is slow. Furthermore, it does not scale with the number of parameters as the number of possible combinations blows up. A second approach is randomised search which simply chooses the parameter values to try out at random. While this solves the issue of computational cost it does not guarantee a good set of parameters and we have no control on the choice of parameter values to try out. A solution is offered by Bayesian optimisation[5]. To perform the Bayesian optimisation Python's scikit-optimize package is used (great tutorial). To train the MLP we utilise Bayesian optimisation, while for the Random Forest randomised search is used. We will describe the procedures in more detail below.

We aim to choose the hyper-parameters in order to minimise the validation error rate which is equivalent to minimising the negative validation accuracy, hence this is the target function f(.).

## Training the MLP

First, consider training the MLP. The target function takes as inputs a combination of hyper-parameters, compiles and trains a model with those hyper-parameters for 10 training epochs, and with a batch size of 89. The output of f(.) is the negative accuracy on the validation set (negative since the optimisation function performs minimisation) For the purpose of this classification task the MLPs considered consist of $n_L$ stacked blocks containing a dense layer with $n_N$ neurons, a dropout layer with specified rate and a given activation function. The output layer is another dense layer with 1 neuron and sigmoid activation function for the binary classification. The optimizer used for learning is Adam with a learning rate to be tuned. For details on the optimisation algorithm refer to [3]). The hyper-parameters and corresponding ranges considered are

- $n_L$: number of hidden layers, between 1 and 8
- $n_N$: number of nodes per layer, between 5 and 512
- Dropout rate, between $10^{-2}$ and 0.5
- Activation function for all layers (except output layer): Sigmoid or Relu
- Learning rate, between $10^{-6}$ and $10^{-2}$

The following is the algorithm used for optimisation. For the initialisation the default parameters chosen by hand tuning, 1 hidden layer, 300 neurons, relu activation, learning rate are used and the first 10 points to evaluate are chosen at random until the Gaussian process (GP) surrogate is fit. Gaussian procceses are common due to their good performance on small data sets. For the GP a Matern kernel is used.

1. Initialize surrogate by training on small number of initial datapoints $\{x_i, y_i\}_{i=1}^{n_{\text{init}}}$ ($n_{\text{init}} = 0$ sometimes possible)
2. Derive acquisition function $\zeta$ from surrogate
3. (Approximately) optimize acquisition function to find next point to evaluate
$$x_{n+1} = \underset{x \in \mathcal{X}}{\arg\max} \zeta(x)$$
4. Evaluate the target function at $x_{n+1}$, yielding $y_{n+1} = f(x_{n+1}) + \epsilon_{n+1}$, where $\epsilon_{n+1}$ represents any evaluation noise, and add $(x_{n+1}, y_{n+1})$ to the data set
5. Refit/update surrogate to incorporate new data
6. If computational budget remaining go back to Step 2 , else return final surrogate (estimate for optimum usually taken as $x_{i^*}$ where $i^* = \arg\max_{i \in \{1,\ldots,n\}} \mathbb{E}[f_{\text{surrogate}}(x_i)]$)

Let $\mu(x)$ and $\sigma(x)$ be the current corresponding marginal posterior mean and standard deviation of the GP surrogate. Furthermore let $\mu^+ = max_{i \in \{1,\ldots,n\}} \mu(x_i)$. As acquisition function we use the expected improvement given by
$$\zeta_{\text{EI}}(x) := \int_{\mu^+ + \xi}^{\infty} \left(f(x) - \mu^+ - \xi\right) \mathcal{N}\left(f(x); \mu(x), \sigma^2(x)\right) df(x). \quad \text{(see [5])}$$

Performing the Bayesian optimisation to find the best set of hyper-parameters leads to an MLP with learning rate 0.0053, 1 hidden layer with 302 neurons, a relu activation function and a dropout rate of 0.3. Then a model with those hyper-parameters is trained on the entire train set for 200 epochs. Early stopping to monitor the validation accuracy is implemented to prevent the model from overfitting. For further performance improvement l2-norm regularisation is implemented with a regularisation coefficient of $10^{-5}$. Another popular regularisation method is batch normalisation. However, since we have implemented drop out already we do not take this option for reasons explored in [1].

### Training the RF

To train the RF classifier a randomised search combined with cross validation is implemented. We use the RandomizedSearchCV() function for search and cross validation, and RandomForestClassifier() for the random forest model, both contained in Python's Sklearn. Note that alternatively to cross validation the out of bag error could have been used as well.
The grid of hyper parameters considered during the randomised search is as follows:

- Number of trees in the forest: {2,8,14,..., 296}
- Number of features to consider at every split: {2,5,8,..., 26}
- Maximum number of levels in a tree: {1,2,..., 30, None}

The samples each tree is trained on are bootstrap samples to introduce more diversity in the trees and improve generalisability of the model. As node impurity measure we use the Gini index defined by $\sum_{k=1}^{K} \hat{p}_{mk} (1 - \hat{p}_{mk})$ where $\hat{p}_{mk}$ is the proportion of class k observations in node m. A model is trained for 200 randomly selected parameter combinations and 5 fold cross validation on the train set is performed. The average validation accuracy is computed for each parameter combination and the parameters producing the highest average accuracy are selected. The selected parameters are 146 trees, considering 5 features at every split and allowing a maximal depth of 7. During cross validation this set of parameters obtained an average validation accuracy of 0.95. For the final RF classifier, the model with tuned hyper-parameters is fit to the entire train set.

## Comparison of predictions

The fully trained classifiers are then used to predict on the so far untouched test set. Prediction with a random forest is done by majority vote amongst the trees. The random forest obtains an accuracy 0f 0.94 while the MLP obtains 0.91. Thus in terms of accuracy the random forest does seem to perform better on the test set. In Table 1 the accuracy, recall and precision for both models on the test set are presented. Interestingly both models show a lower recall on class 1 than on class 0.
The confusion matrices for RF and MLP classifier are shown in Figures 2 and 3 respectively. The confusion matrices show indeed some small differences with the biggest difference in class 1.

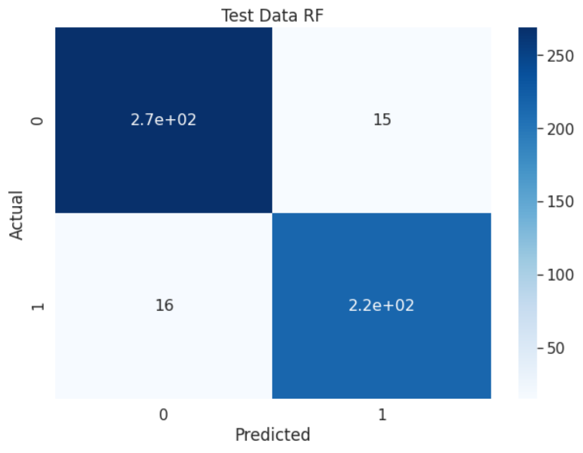

Figure 2: Confusion matrix for final MLP classifier on test data.

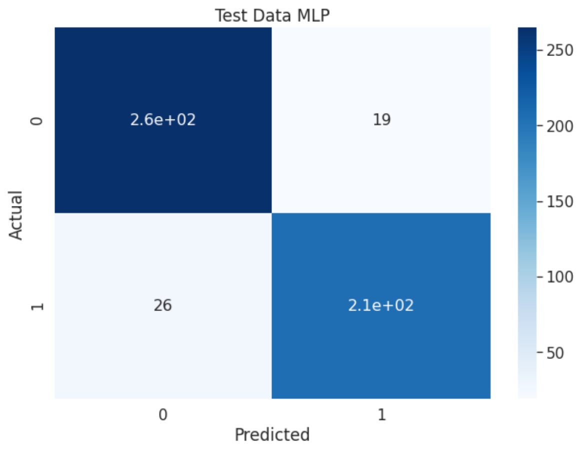

Figure 3: Confusion matrix for final MLP classifier on test data.

| Measure | RF | MLP |
|---|---|---|
| Precision | 0.94 | 0.91 |
| Recall | 0.94 | 0.91 |
| Accuracy | 0.94 | 0.91 |

Table 1: Measures of performance for the classifiers on the test set.

To quantify the difference of the predictions we use McNemar's test. We test $H_0$ : the two model predictions disagree to the same amount vs. $H_1$ : they don't disagree to the same amount. The McNemar's test statistic value is computed to be 2.1667 which yields a p value of 0.0151. Thus the null hypothesis is rejected at the 5% significance level. Due to the better performance in terms of error rate/accuracy the random forest classifier is recommended.

Note the following concerns regarding the recommendation. Firstly we do not have a measure of the variability of the model performance, as they were only evaluated once on the test set. The variability induced by the training set nor the randomness in the training procedure are taken into account (see [2]. Secondly, for a reasonable comparison a representative test set is required. With 27 variables the space is large and 516 observations might not be enough to give a sensible representation. Furthermore, it is argued that a large test set size is needed in order to make a definitive conclusion and detect a small difference in error rate/accuracy using McNemar's test. The test set is not large enough and thus results should be treated with care (see [4]).

Finally, we focus on the use case of such a model. Classifying chromosomes into different types could for example be part of an illness prediction model for which it is important to know why a certain prediction was made, so that the practitioner can explain the diagnose to a patient. In other words, interpretability is key. The random forest as well as the MLP are not easily interpretable and further steps, such as an analysis of the weights of the MLP or the feature importance of the random forest, would need to be taken in order to better understand the model predictions.

## Reject option

We now implement the reject option for the random forest classifier. Points for which we have

$$\left\{\mathbf{x} \mid 1 - \max_i p\left(\omega_i \mid \mathbf{x}\right) > 0.4\right\},$$

where $p\left(\omega_i \mid \mathbf{x}\right)$ is the posterior probability of class membership to $\omega_i$, are rejected, i.e. more information is required. The confusion matrix including not rejected points is shown in Figure 4. The classifier with reject option has precision 0.96, recall 0.95 and accuracy 0.96. In total 45 points are rejected.

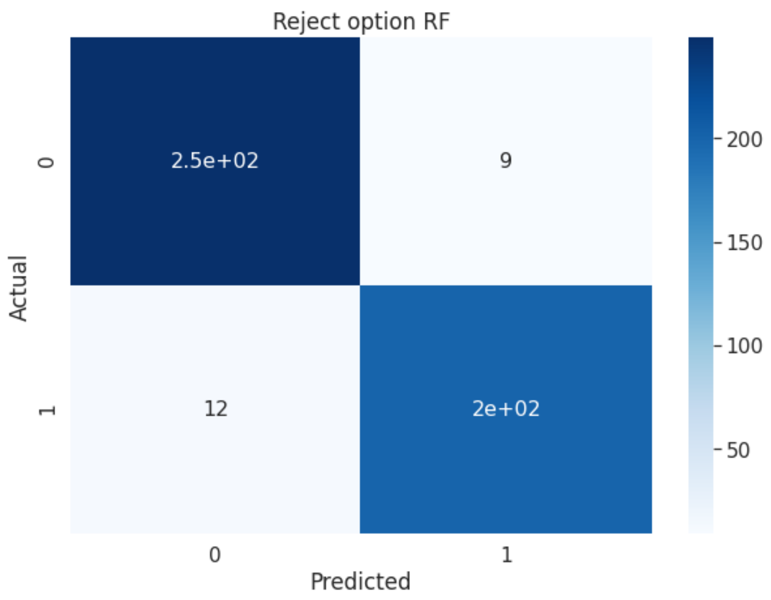

Figure 4: Confusion matrix with reject option implemented.

## References

[1] Xiang Li et al. *Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift.* 2018. arXiv: 1801.05134 [cs.LG].

[2] Thomas G. Dietterich. "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms". In: *Neural Computation* 10.7 (Oct. 1998), pp. 1895–1923.

[3] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. arXiv: 1412.6980 [cs.LG].

[4] Brian D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996. DOI: 10.1017/CBO9780511812651.

[5] Bobak Shahriari et al. “Taking the Human Out of the Loop: A Review of Bayesian Optimization”. In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175. DOI: 10.1109/JPROC.2015.2494218.