FYS-STK3155/4155 Applied Data Analysis and Machine Learning - Project 3

Lotsberg, Bernhard Nornes
Nguyen, Anh-Nguyet Lise

`https://github.com/liseanh/FYS-STK4155-project3`

November - December 2019

Abstract

Whole page: 6.24123in Column: 3.01682in

1 Introduction

In popular culture the neural network is probably the most well known form of machine learning. In recent years many other statistical learning methods have proven themselves as well however. In this project we compare the performance of neural networks and gradient boosting in the case of binary classification. In addition to these, we also use the much simpler k-nearest neighbours method as a baseline for classification performance.

2 Data

The data set we will analyse in this project is the MAGIC Gamma Telescope data set retrieved from the UCI Machine Learning Repository, which was generated by a Monte Carlo (MC) program described by D. Heck et. al. [4] to simulate high energy gamma particle registration in a Cherenkov gamma telescope. The set consists of ten explanatory variables and a binary response variable `class` which specifies whether the measured photons resulted from a gamma particle (`class = g`) or a hadron (`class = h`). The entire data set consists of 19020 instances with no missing values, with outcome distribution as shown in Figure 1. The explanatory and response variables are defined as the following by the UCI Machine Learning Repository [2]:

1. `fLength`: continuous # major axis of ellipse [mm]
2. `fWidth`: continuous # minor axis of ellipse [mm]
3. `fSize`: continuous # 10-log of sum of content of all pixels [in #phot]
4. `fConc`: continuous # ratio of sum of two highest pixels over `fSize` [ratio]
5. `fConc1`: continuous # ratio of highest pixel over `fSize` [ratio]
6. `fAsym`: continuous # distance from highest pixel to center, projected onto major axis [mm]
7. `fM3Long`: continuous # 3rd root of third moment along major axis [mm]
8. `fM3Trans`: continuous # 3rd root of third moment along minor axis [mm]
9. `fAlpha`: continuous # angle of major axis with vector to origin [deg]
10. `fDist`: continuous # distance from origin

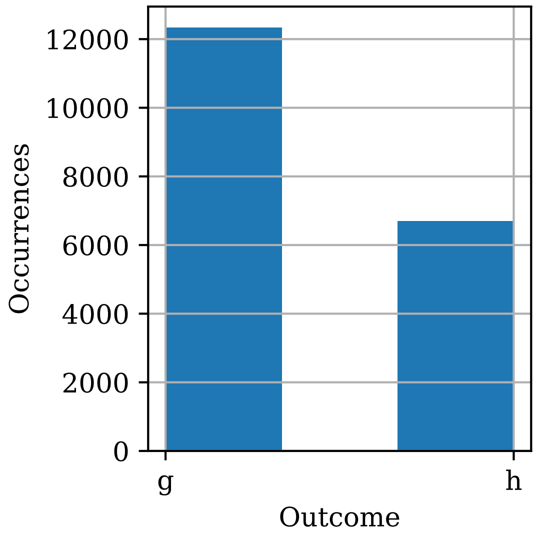

Figure 1: Frequencies of the outcomes g and h in the data set. The numbers of instances for the categories were 12332 and 6688 for g and h respectively.

to center of ellipse [mm]

11. `class:` g, h # gamma (signal), hadron (background)

3 Methods

3.1 *k*-Nearest Neighbour (kNN)

The k-nearest neighbour method is a non-parametric method that consists of using the k observations in the training set in feature space closest to a point x to form a model $\hat{y}$. For regression and binary classification, the method considers a point x and the k closest points x_i in the neighbourhood $N_k(x)$ defined by k, such that

$$\hat{y} = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i, \tag{1}$$

i.e. the model output is the average of the k closest points to x.

As we are using this method for binary classification, we assign the classes such that

$$\hat{Y} = \begin{cases} \text{g if } \hat{y} \leq 0.5 \\ \text{h if } \hat{y} > 0.5 \end{cases} [3]. \tag{2}$$

Alternatively, the observation can be assigned according to the majority class of its neighbours, such that the observation is classified as the most common class in its neighbourhood. This is the method used for multi-class classification.

To implement kNN and the hyperparameter optimisation in this project we use the Scikit-Learn library.

3.1.1 Hyperparameter tuning

The kNN method is a simple approach that only requires tuning of a single hyperparameter k. Using $k = 1$ results in a highly complex model with high variance, while using $k = N$, where N is the total number of samples in the training set, results in a simple, highly biased model. To find the optimal value for k in our case, we use 5-fold cross-validation on the training set, keeping 1/3rd of the total data as test set.

3.2 Multilayer Perceptron (MLP)

A multilayer perceptron (MLP) is a feed-forward neural network. It contains an input layer, one or more hidden layers, with tunable number of nodes and layers, and a final output layer. The information flows only in one direction from the input layer to the output layer. The input and output values of the nodes are determined by an activation function, in addition to the weights and biases of the nodes. For a more extensive explanation of how the different components of the MLP works, please refer to our previous work, *Project 2: Regression and Classification* [5].

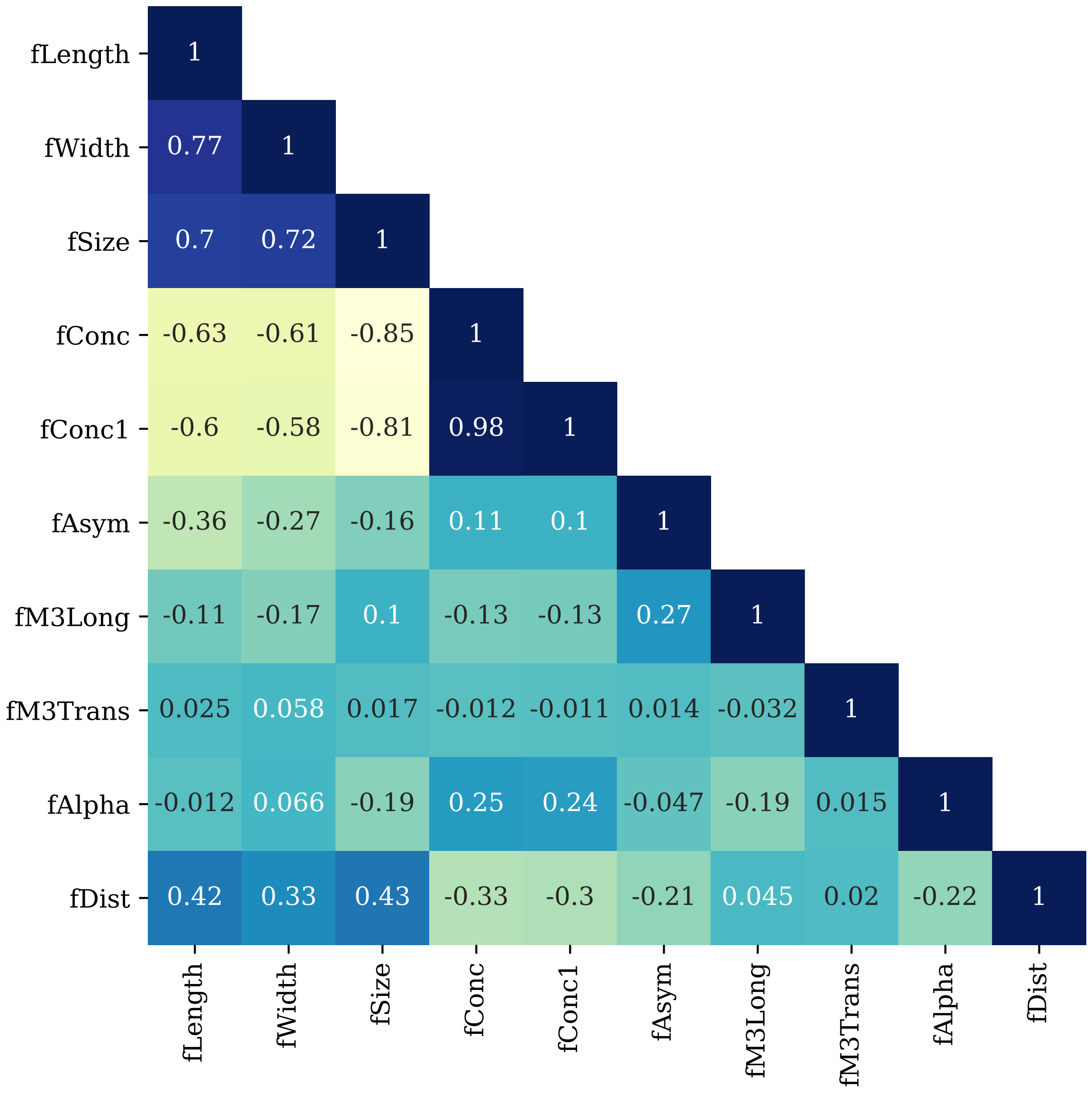

Figure 2: Correlation matrix of the features in the train set. Upper triangle excluded for readability.

We have chosen to use the Rectified Linear Unit (ReLU) function as our activitation function $f(z)$ between the layers, which is given by

$$f(z) = \max(0,\, z), \tag{3}$$

and its gradient by

$$f'(z) = \begin{cases} 0, & z < 0 \\ 1, & z > 0 \end{cases}. \tag{4}$$

For the activation function of the final output layer, we use the softmax function to achieve the categorical outcomes,

$$f(z_i) = \frac{\exp(z_i)}{\sum \exp(z_j)} \tag{5}$$

For this project we will be using the TensorFlow library to implement the MLP with the addition of the Scikit-Optimize library to optimise the hyperparameters.

3.2.1 Hyperparameter tuning

Neural network models normally have high variance and low bias

3.3 Extreme Gradient Boosting (XGB)

Boosting is a learning technique based on the idea of combining several weak learners b_m into a final strong learner $f_{m_{\text{stop}}}$, where each f_m is improved from the previous iteration f_{m-1} through the step h_m,

$$f_{m_{\text{stop}}} = \sum_{m=0}^{m_{\text{stop}}} h_m. \tag{6}$$

Gradient boosting is one such method. We consider a loss function $L(y, f(x))$, which is iteratively minimised by calculating the negative gradient of the loss function to obtain the minimisation direction. To prevent overfitting, the boosting step size $\nu \in (0,1)$ is introduced as a tuning parameter, with smaller values of ν resulting in larger shrinkage,

$$f_m(x) = f_{m-1}(x) + \nu h_m. \tag{7}$$

The gradient boosting algorithm goes as follows: [1]

1. Initialize the estimate $f_0(x)$
2. for $m = 1, \ldots, m_{\text{stop}}$:
   (a) Compute negative gradient vector
   $$u_m = -\frac{\partial L(y,\, f(x))}{\partial f(x)}\bigg|_{f(x)=\hat{f}_{m-1}(x)}$$
   (b) Fit the base learner to the negative gradient vector, $b_m(u_m, x)$
   (c) Update the estimate
   $$f_m(x) = f_{m-1}(x) + \nu b_m(u_m, x)$$
3. Compute final estimate,
$$\hat{f}_{m_{\text{stop}}}(x) = \sum_{m=0}^{m_{\text{stop}}-1} \nu b_m(u_m, x)$$

For this project, we have chosen to boost decision trees using a variant of gradient boosting called extreme gradient boosting (XGB).

3.3.1 Hyperparameter tuning

3.4 Model evaluation

To evaluate the models

confusion matrix and f1 score

4 Results

Batch	Drop_1	Drop_2	Nodes_1	Nodes_2	Epochs
109	0.13	0.095	17	4	55

Table 1: Estimated hyperparameters for the neural network model employed in this project.

η	Depth	α	λ	$\text{Weight}_{\text{min}}$	Epochs
0.048	9	0.0074	0.0035	3	125

Table 2: Estimated hyperparameters for the XGB model employed in this project.

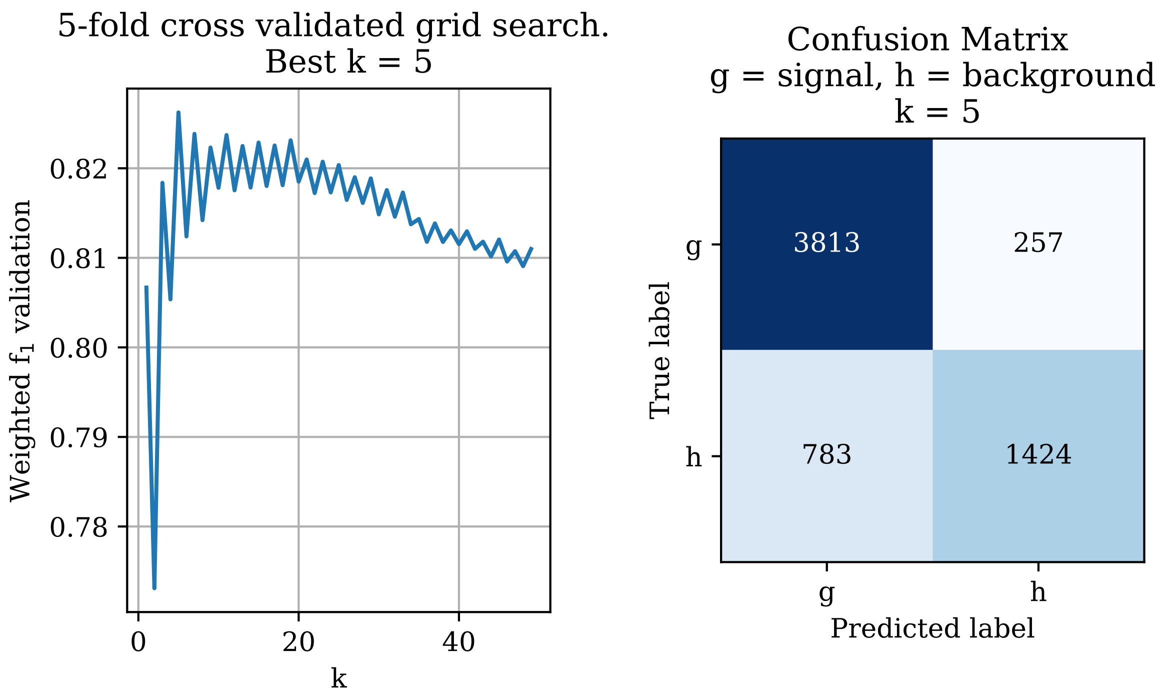

Figure 3: Results from tuned kNN using cross validation. The confusion matrix was found using the test set.

	kNN	MLP	XGB
Train F_1	0.88	0.87	0.95
Test F_1	0.83	0.87	0.88

Table 3: Train and test F_1 scores for k Nearest Neighbours, Multilayer Perceptron and XGBoost.

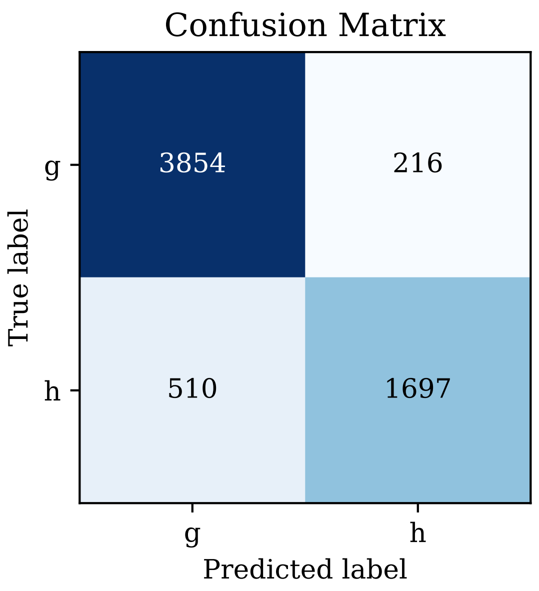

Figure 5: Confusion matrix of the gradient boosted model applied to the test set.

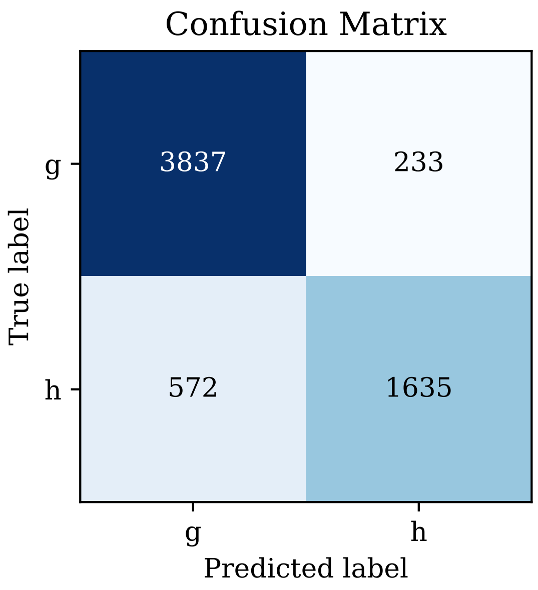

Figure 4: Confusion matrix of the neural network model applied to the test set.

5 Discussion

6 Conclusion

Acknowledgements

We want to thank the University of California, Irving for providing us with the data used in this project.

References

[1] Riccardo De Bin. STK-IN4300 Statistical Learning Methods in Data Science: Lecture 10, 2019. Retrieved: 09-12-2019.

[2] Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017.

[3] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*, volume 27. Springer Series in Statistics, 2009.

[4] Dieter Heck, G Schatz, J Knapp, T Thouw, and JN Capdevielle. CORSIKA: a Monte Carlo code to simulate extensive air showers. Technical report, 1998.

[5] Bernhard Nornes Lotsberg and Anh-Nguyet Lise Nguyen. *Project 2: Classification and Regression*. FYS-STK3155/4155 Applied Data Analysis and Machine Learning. 2019.