2406 – Simple binary classification problem with XGBoost

Lorenzo Cavezza, Giulia Doda, Giacomo Longaroni, and Laura Ravagnani
(Dated: May 1, 2024)

This work explores the powerful tool of XGBoost applying it to a classification task on a 4-dimensional dataset. The hyperparameters of XGBoost are tuned with k-fold cross-validation to maximize validation accuracy. Regularization does not improve accuracy on the test set. XGBoost is then applied to a reduced dataset. The highest accuracy can be achieved by considering only the two most important features. Next, XGBoost is compared with a fully connected feed-forward neural network (NN), both in terms of validation accuracy and computational efficiency. By varying the number of data samples in the training set N', XGBoost outperforms the NN especially at low N'. Finally, a data augmentation procedure improves the test accuracy.

INTRODUCTION

The main topic of this paper is the study of a 4-dimensional dataset using supervised learning algorithms to perform classification. The dataset is shown in Figure 1: it is clear that the first two features combine in a geometric way, while the others show no seemingly meaningful patterns. The dataset is split into 75% training and 25% testing data.
In particular, the aim is to investigate the performance of XGBoost (eXtreme Gradient Boosting) [1]. In the first part of the work, the main goal is to find the XGBoost model that classifies our data better in terms of validation accuracy. First, a grid search is performed over different XGBoost hyperparameters with k-fold cross-validation. Later, the regularization parameter is studied on its own to best understand its role in improving the performances.
Next, the paper investigates the feature importance in the classification task. XGBoost performance is tested on the most important features to see if the test accuracy can improve. The performance is also evaluated on different feature combinations to better understand each feature's role in classification.
This paper then compares the performance of XGBoost with that of a fully connected feed-forward Neural Network (NN), both on the full dataset and on the dataset restricted to the two most important features. Their performance is evaluated both in terms of validation accuracy and computational efficiency, with a particular focus on the learning time.
Finally, since the dataset contains only 4000 samples, the paper describes the implementation of a data augmentation strategy to best train the XGBoost model and improve test accuracy.

METHODS

XGBoost hyperparameters tuning – The first objective is to build the best model of XGBoost to be used for classification. The algorithm is based on decision trees aggregations created by iteratively adding new trees to the ensemble. Parametrizing the prediction of a decision tree j on data point $(y_i, \mathbf{x}_i)$ as $g_j(\mathbf{x}_i)$, the whole ensemble gives the prediction

$$\hat{y}_i = g_A(\mathbf{x}_i) = \sum_{j=1}^{M} g_j(\mathbf{x}_i) \tag{1}$$

where M is the number of trees in the aggregation [2].
In order to select the best XGBoost classifier in terms of accuracy we use the SCIKIT-LEARN tool `GridSearchCV` and perform a grid search over different values of the following hyperparameters [3]:

- `gamma` (γ): minimum loss reduction required to make a further partition on a leaf node of the tree;
- `learning_rate` (η): boosting learning rate. It reduces the step size in order to reach the optimal solution more precisely;
- `max_depth`: maximum tree depth;
- `n_estimators` (M): total number of trees in the ensemble.

For each model, the performance is evaluated using k-folds cross validation: for every combination of parameters the training set is split into k parts, $k-1$ are used for training and the remaining one as validation. The final performance estimate is computed as the validation accuracy mean.

Regularization analysis – One key element in Gradient-Boosted Trees is the cost function

$$C(\mathbf{X}, g_A) = \sum_{i=1}^{N} l(y_i, \hat{y}_i) + \sum_{j=1}^{M} \Omega(g_j)\,. \tag{2}$$

At each step, we compute its gradient with respect to the predicted value and add trees in the direction of the gradient. This function is composed by a term that evaluates the goodness of the predictions and, to avoid overfitting, a regularization term $\Omega(g_j)$ that depends on the γ parameter [2]. In order to understand the role of regularization in the computation of the loss we use the

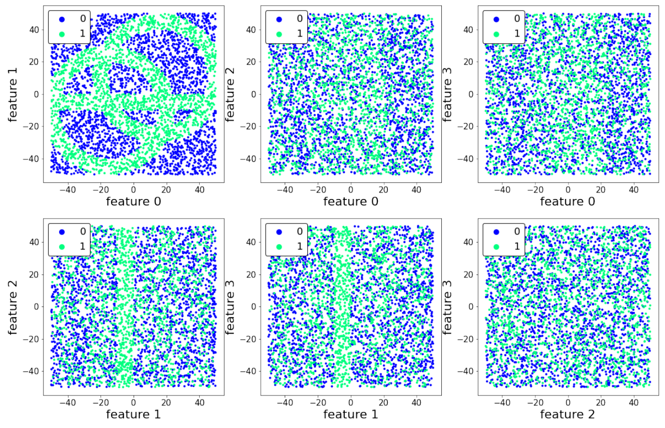

FIG. 1. Projections of the full dataset over different combinations of its dimensions (features): features 0 and 1 creates a clear geometric pattern, while features 2 and 3 are almost uniformly distributed. Data are of different colors depending on the true label.

best model found before and study how the accuracy trend changes with the value of the parameter λ using cross validation.

Dimensionality reduction – First, the best XGBoost model is applied on the test set and we then perform a feature importance analysis using the built-in feature importance method implemented in the XGBoost library. The algorithm performance is subsequently studied on a reduced dataset with $L' < L$ features, where $L = 4$ are the features that compose the whole dataset and L' indicates a subsample. All the possible subsets of features are analyzed. The algorithm is trained on the reduced training set and tested on the reduced test set. The aim is to verify if applying the algorithm to a reduced dataset excluding less important features can improve test accuracy. In addition to test accuracy, every XGBoost performance is evaluated also through the Receiver Operating Characteristic (ROC) curve. The ROC curve shows the evolution of the true positive rate as a function of the false positive rate. It is an extremely insightful tools when evaluating the performance of binary classifiers: both the area (AUC - Area Under the Curve) and the shape can provide additional information about the features role in classification. With ROC curves it is therefore possible to evaluate the training quality and compare the different binary classifiers trained on different portions of the training set.

XGBoost *vs* NN – The analysis continues by comparing the best XGBoost model with a feed-forward NN implemented with KERAS [4], specifically to examine the accuracy of the two models and their respective learning times. XGBoost hyperparameters are the optimal ones previously identified. The NN architecture is defined by an input layer of 300 neurons, two hidden layers of 200 and 100 neurons, and an output layer with only 2. The activation functions used are the `ReLU` and the `softmax` for the output layer. Moreover the NN implements $L2$ regularization and uses `Adam` optimizer. The comparison between the two models is conducted through cross-validation, with the number of samples N' to train the models increasing each time; data samples are randomly selected from the dataset. Finally, this analysis is repeated considering only the 2 most important features for each example in the dataset.

Data Augmentation – Lastly a data augmentation method is developed to create a richer dataset that could better train the XGBoost model and get an higher test accuracy. The implementation of Data Augmentation consists of 3 main steps described as follows:

- A ***k*-Nearest Neighbours Augmentation** (kNN) phase to better fill the geometric structures inside the original dataset. It consists in generating new points between two k-nearest neighbors with the same label, with some constraints that help to better build the structures, such as choosing k as a random integer (between 1 and 12 in our case) and limiting the generation to only distances above the tenth percentile of the distance distribution, in order to fill the gaps in the original dataset.

- A **Local Voting Aided Uniform Augmentation** phase to generate a new dataset using the kNN augmented one as reference. It consists of generating 70k points with a uniform distribution over the domain of our dataset and assigning their labels based on a strict majority vote (over 90%) on a local neighborhood in the reference set. New points that are not assigned are placed in a temporary "border" set.

- A final **2-folds Majority Voting** phase to assign labels to the border set. It consists in dividing the reference set into two random folds and performing a local vote on the border set within each fold and then a global vote between the 2 folds. This process is repeated several times with different random folds until every point of the border set is assigned.

After this process a well defined dataset of 70k points is created. We evaluate then test accuracy after a grid search on XGBoost hyperparameters.

RESULTS

XGBoost Hyperparameter tuning – The best values of the hyperparameters obtained maximizing the accuracy on the training set are the following: $\gamma = 0.01$,

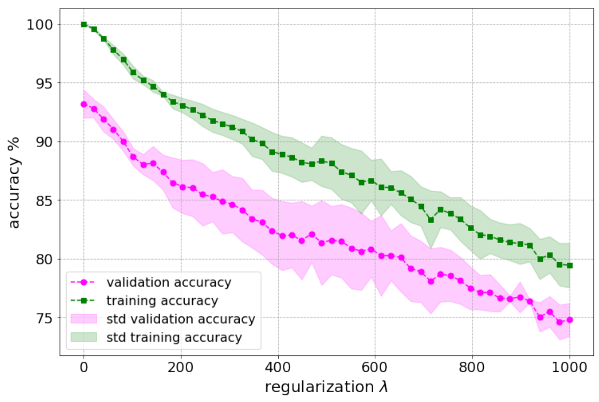

FIG. 2. Training (green) and validation (magenta) accuracy means computed with cross validation for different values of regularization parameter λ.

$\eta = 0.15$, max_depth $= 22$, $M = 90$. This analysis allows to increase the test accuracy up to 95.4% without taking into account regularization.

Regularization analysis – The goal of regularization is to avoid overfitting of the data. The idea is to look for a trade-off between having a complex model that performs very well only on the training set and having a simpler one that works better on the test set. The expected trends are then the increasing of test or validation accuracy at the expense of the performance on the training set. The evolution of the accuracy is shown in Figure 2: the introduction of the regularization in the cost function does not improve validation accuracy, on the contrary it leads the model to underfit the data. Since the best accuracy is obtained with $\lambda = 0$ the rest of the analysis is performed without regularization.

Dimensionality reduction – The most important features are 0 and 1. Table I collects the test accuracies for every feature subset, while Figure 3 shows ROC curves for every feature combination and the corresponding AUCs. Combining these results, it is clear that learning fails without features 0 and 1. In particular, with them, the test accuracy is slightly higher than the one obtained on the full test set, and the corresponding ROC curve has a slightly better shape and is closer to the perfect classifier than the curve corresponding to the full training set. Considering also the AUC values, discarding the less important features does not show any improvement. Without features 0 and 1, the classification performance is close to the random classifier and the training is essentially pointless. Furthermore, it is clear that the presence of feature 1 gives the classifiers a good start, but feature 1 alone without feature 0 cannot solve the task completely. It is also noteworthy that the two least important features show an even worse performance than the random classifier.

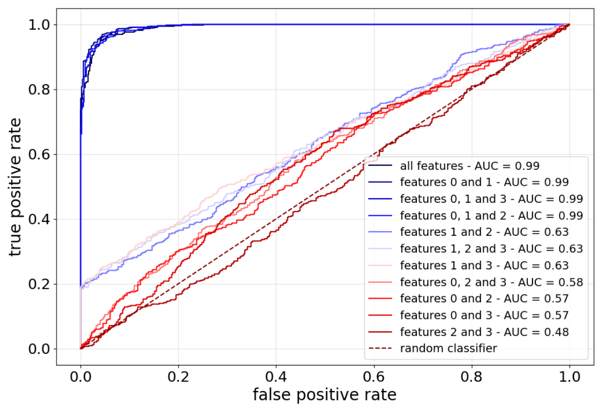

FIG. 3. ROC curves corresponding to different feature subsamples. The best curve shapes with their respective AUC values correspond to the presence of the two most important features, i.e. features 0 and 1. The selection of only features 0 and 1 performs slightly better compared to the full dataset. However, the AUC values in the presence of both features 0 and 1 are the same. The presence of feature 1 alone identifies a group of curves with a good start for the ROC curve, but it is not sufficient to complete the classification correctly, and the corresponding AUC values are not satisfactory. The remaining curves indicate that the learning has failed, since they are comparable to the random classifier curve.

TABLE I. Test accuracy with the corresponding feature selection. The highest test accuracy is obtained with the two most important features.

features	test accuracy	features	test accuracy
(0,1)	96.0%	(1,2)	58.8%
(0,1,3)	95.8%	(0,3)	57.0%
(0,1,2)	95.2%	(0,2,3)	56.5%
(1,3)	61.3%	(0,2)	54.6%
(1,2,3)	59.8%	(2,3)	49.8%

full dataset test accuracy: 95.4%

XGBoost vs NN – The results are illustrated in Figure 4. The plots show the performance of the two models in terms of accuracy and learning times.

Accuracy – Both models perform well on the dataset, with XGB generally outperforming NN for every fraction of the dataset in the cross-validation. The highest accuracies achieved are (96.0 ± 1.5) % (XGB) and (91.2 ± 0.8)% (NN) using all features for the training, while (95.9 ± 0.3)% (XGB) and (93.0 ± 0.3)% (NN) using only the 2 most significant features. This result, considering the previous analysis of feature importance, highlights again how features 2 and 3 do not add information to the problem, but instead introduce noise causing a worsening of the predictive power of the model, especially for the NN. The differences in accuracy between the models are more pronounced when trained on sets with fewer data. In this case, XGB manages to maintain

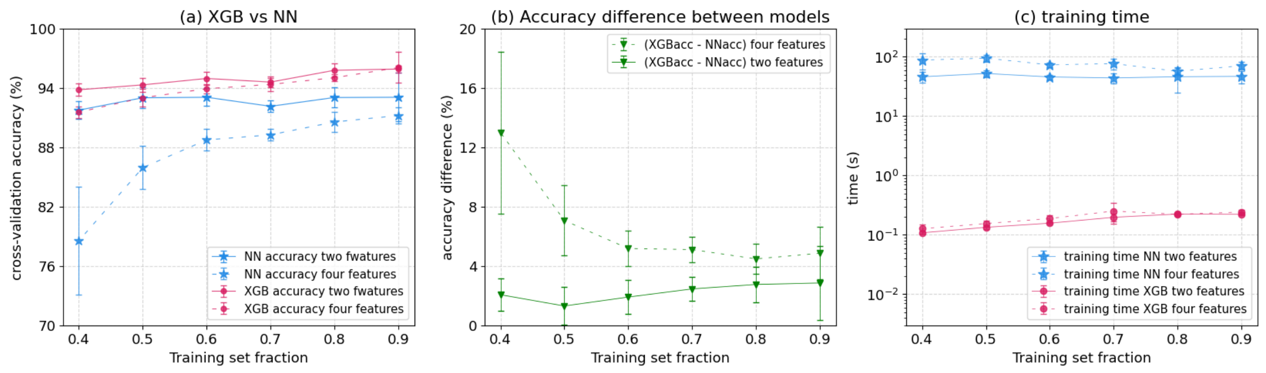

FIG. 4. The plots compare model performance across different training set fraction, highlighting (a) validation accuracy, (b) validation accuracy difference, and (c) training time. Solid lines show models trained with the two key features, and dashed lines for models trained with all four features.

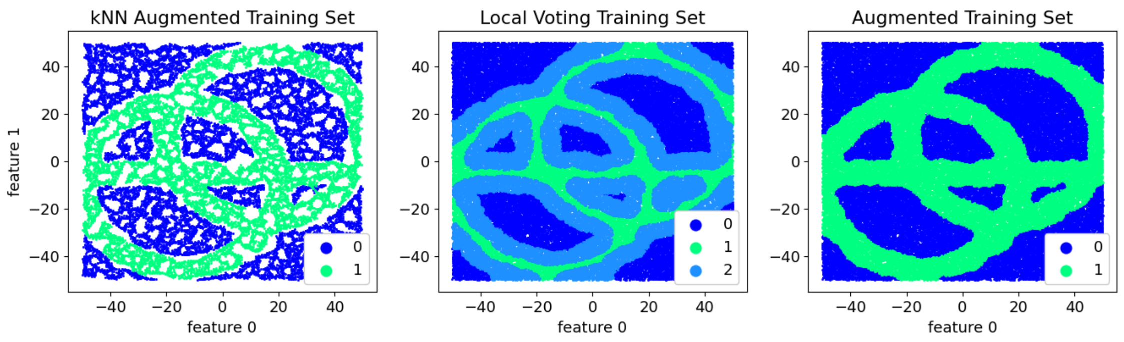

FIG. 5. Data Augmentation phases: the plot on the left panel shows the preliminary kNN Augmentation that is used as reference in the later ones; then the Uniform Augmentation aided with Local Voting is shown in the center panel, with the "border set" in light blue (label 2); lastly the final augmentation after the 2-folds majority voting is shown on the right panel.

a stable accuracy above 90%, on the contrary the NN trained on the four features shows greater accuracy errors, with also an accuracy difference compared to XGB of more than 10%.

Training time – As highlighted in Figure 4c, XGBoost demonstrates remarkable efficiency. Specifically, the training phase for XGB ranges 0.1-0.3 seconds. On the other hand, training a NN requires significantly more time, in this case between 30 and 100 seconds.

Data Augmentation – The results of each step in the augmentation is shown in Figure 5, a slight final improvement in the data structures can be observed. We feed the augmented dataset to the XGBoost model defined with GridSearch and get a test accuracy of about 97%. Every step of the augmentation helps bringing up the accuracy: the first local voting reaches about 96% and the last 1% is achieved thanks to the "border" set. It could appear a very small improvement, but given the already high accuracy it is a clearly satisfactory result. This method can be easily implemented in other datasets of this type and can help training a better model if the available data is too little.

CONCLUSIONS

The best XGBoost classifier trained with the GridSearch paradigm was simple enough to avoid overfitting, making the regularization inconsequential. The dimensionality reduction proved features 2 and 3 irrelevant for the classification, but discarding them does not permit a significant accuracy improvement. The developed NN model could not outperform XGBoost valid performances. Finally, the custom data augmentation method proved useful to slightly improve test accuracy.

Authors contribution statement – This work and the final manuscript are equally contributed by all group members.

[1] XGBoost. XGBoost documentation
[2] P. Mehta et al., *A high-bias, low-variance introduction to Machine Learning for physicists*, **8**, 45-49 (2019).
[3] Scikit-learn. Scikit-learn documentation: GridSearchCV
[4] Keras. Keras documentation