

# XGBoost

## Even-Odd Classification Report

Vrishab Anurag Venkataraghavan

### 1 XGBoost Hyperparameter Tuning

The XGBoost Classifier I implemented was tuned to maximize the accuracy score on the MNIST\_validation dataset. Using 32 different combinations of key hyperparameters, I did a grid search

The hyperparameters and their values being checked for the grid search were:

- `max_depth`: [5, 6]
- `subsample`: [0.8, 1.0]
- `min_child_weight`: [1, 5]
- `gamma`: [0, 0.5]
- `reg_lambda`: [1.0, 2.0]

Other parameters were held constant during this search: `n_estimators=100`, `learning_rate=0.1`, `colsample_bytree=0.5`, and `use_exact_greedy=False`. For every iteration of a different combination of hyperparameters, the val accuracy after every 5 trees was checked, to see the evolution of performance. During separate runs with  $n\_estimators = 200$ , it was noted that performance flatlined after  $\approx 100$  iterations. Also, while testing with both the Exact Greedy Algorithm and Approximate Algorithm, it was noted that there was no significant increase in performance, while training time was at least 1.5x the Approximate implementation in the Exact Greedy implementation

Table 1 presents the complete results of this grid search. The best-performing model (with 0.9672 accuracy) was achieved with two different parameter sets. The one with `max_depth=6`, `subsample=0.8`, `min_child_weight=5`, `gamma=0`, and `reg_lambda=1.0` was selected as the optimal model, as it had a lower training time (150.04s) than the other top-performing run (178.99s).

Table 1: Full XGBoost Hyperparameter Tuning Results

| n_estimators | max_depth | lr         | col_by_tree | subsample  | min_child | gamma      | lambda     | use_exact    | eps        | Accuracy      | Time (s)      |
|--------------|-----------|------------|-------------|------------|-----------|------------|------------|--------------|------------|---------------|---------------|
| 100          | 6         | 0.1        | 0.5         | 0.8        | 1         | 0.0        | 1.0        | False        | 0.3        | 0.9648        | 157.26        |
| 100          | 6         | 0.1        | 0.5         | 0.8        | 1         | 0.0        | 2.0        | False        | 0.3        | 0.9632        | 164.25        |
| 100          | 6         | 0.1        | 0.5         | 0.8        | 1         | 0.5        | 1.0        | False        | 0.3        | 0.9668        | 157.24        |
| 100          | 6         | 0.1        | 0.5         | 0.8        | 1         | 0.5        | 2.0        | False        | 0.3        | 0.9628        | 164.43        |
| <b>100</b>   | <b>6</b>  | <b>0.1</b> | <b>0.5</b>  | <b>0.8</b> | <b>5</b>  | <b>0.0</b> | <b>1.0</b> | <b>False</b> | <b>0.3</b> | <b>0.9672</b> | <b>150.04</b> |
| 100          | 6         | 0.1        | 0.5         | 0.8        | 5         | 0.0        | 2.0        | False        | 0.3        | 0.9640        | 145.91        |
| 100          | 6         | 0.1        | 0.5         | 0.8        | 5         | 0.5        | 1.0        | False        | 0.3        | 0.9612        | 157.36        |
| 100          | 6         | 0.1        | 0.5         | 0.8        | 5         | 0.5        | 2.0        | False        | 0.3        | 0.9656        | 152.16        |
| 100          | 6         | 0.1        | 0.5         | 0.8        | 5         | 0.5        | 2.0        | False        | 0.3        | 0.9620        | 169.32        |
| 100          | 6         | 0.1        | 0.5         | 1.0        | 1         | 0.0        | 1.0        | False        | 0.3        | 0.9664        | 172.84        |
| 100          | 6         | 0.1        | 0.5         | 1.0        | 1         | 0.0        | 2.0        | False        | 0.3        | 0.9648        | 161.23        |
| 100          | 6         | 0.1        | 0.5         | 1.0        | 1         | 0.5        | 1.0        | False        | 0.3        | 0.9672        | 178.99        |
| <b>100</b>   | <b>6</b>  | <b>0.1</b> | <b>0.5</b>  | <b>1.0</b> | <b>1</b>  | <b>0.5</b> | <b>2.0</b> | <b>False</b> | <b>0.3</b> | <b>0.9632</b> | <b>170.10</b> |
| 100          | 6         | 0.1        | 0.5         | 1.0        | 5         | 0.0        | 1.0        | False        | 0.3        | 0.9644        | 173.65        |
| 100          | 6         | 0.1        | 0.5         | 1.0        | 5         | 0.0        | 2.0        | False        | 0.3        | 0.9644        | 165.31        |
| 100          | 6         | 0.1        | 0.5         | 1.0        | 5         | 0.5        | 2.0        | False        | 0.3        | 0.9648        | 152.59        |
| 100          | 5         | 0.1        | 0.5         | 0.8        | 1         | 0.0        | 1.0        | False        | 0.3        | 0.9592        | 114.67        |
| 100          | 5         | 0.1        | 0.5         | 0.8        | 1         | 0.0        | 2.0        | False        | 0.3        | 0.9616        | 115.30        |
| 100          | 5         | 0.1        | 0.5         | 0.8        | 1         | 0.5        | 1.0        | False        | 0.3        | 0.9580        | 108.97        |
| 100          | 5         | 0.1        | 0.5         | 0.8        | 1         | 0.5        | 2.0        | False        | 0.3        | 0.9596        | 113.25        |
| 100          | 5         | 0.1        | 0.5         | 0.8        | 5         | 0.0        | 1.0        | False        | 0.3        | 0.9572        | 106.20        |
| 100          | 5         | 0.1        | 0.5         | 0.8        | 5         | 0.0        | 2.0        | False        | 0.3        | 0.9596        | 106.07        |
| 100          | 5         | 0.1        | 0.5         | 0.8        | 5         | 0.5        | 1.0        | False        | 0.3        | 0.9572        | 105.16        |
| 100          | 5         | 0.1        | 0.5         | 0.8        | 5         | 0.5        | 2.0        | False        | 0.3        | 0.9564        | 119.26        |
| 100          | 5         | 0.1        | 0.5         | 1.0        | 1         | 0.0        | 1.0        | False        | 0.3        | 0.9596        | 118.60        |
| 100          | 5         | 0.1        | 0.5         | 1.0        | 1         | 0.0        | 2.0        | False        | 0.3        | 0.9584        | 120.28        |
| 100          | 5         | 0.1        | 0.5         | 1.0        | 1         | 0.5        | 1.0        | False        | 0.3        | 0.9580        | 118.03        |
| 100          | 5         | 0.1        | 0.5         | 1.0        | 1         | 0.5        | 2.0        | False        | 0.3        | 0.9608        | 115.31        |
| 100          | 5         | 0.1        | 0.5         | 1.0        | 5         | 0.5        | 1.0        | False        | 0.3        | 0.9604        | 111.81        |
| 100          | 5         | 0.1        | 0.5         | 1.0        | 5         | 0.5        | 2.0        | False        | 0.3        | 0.9580        | 111.81        |

## 2 Model Performance Comparison

A comparison was then run between Logistic Regression, Random Forest, and the XGBoost implementation. The Logistic Regression and Random Forest implementations were taken from the notebooks discussed in class, with minimal changes. All models were trained on the `MNIST_train.csv` dataset (10,002 samples) and evaluated on the `MNIST_validation.csv` dataset (2,499 samples).

The XGBoost model in this specific comparison used the best parameters from the comparison file (`n_estimators=100, max_depth=6, colsample_bytree=0.5, subsample=0.8, learning_rate=0.1`). The results, including training time and key classification metrics, are summarized in Table 2.

Table 2: Performance Comparison of Classifiers on the Validation Set

| Model               | Training Time (s) | Accuracy      | Precision     | Recall        | F1 Score      |
|---------------------|-------------------|---------------|---------------|---------------|---------------|
| Logistic Regression | 10.27             | 0.8792        | 0.8801        | 0.8730        | 0.8765        |
| Random Forest       | 108.65            | 0.9088        | 0.9433        | 0.8644        | 0.9032        |
| <b>XGBoost</b>      | <b>158.17</b>     | <b>0.9632</b> | <b>0.9595</b> | <b>0.9658</b> | <b>0.9627</b> |

## 3 Thoughts and Observations

First, I will mention the results of the experiments and testing conducted

- **Performance vs. Cost Trade-off of XGBoost:** The results in Table 2 clearly show the trade-off between performance and computational cost. The XGBoost classifier, while by far the slowest to train (158.17s), achieved a significantly higher accuracy (96.32%) than the other models. Logistic Regression was much faster (10.27s) but both LR and Random Forest performance plateaued at around 88-91% accuracy.
- **Model Complexity:** The implementation complexity also reflected in the performance results. Logistic Regression was the most straightforward. The Random Forest required a functional Decision Tree, along with mechanisms for bagging and feature subsampling. The XGBoost implementation was by far the most complex, implementing custom splitting logic based on gradients and Hessians to calculate gain, and introducing several more hyperparameters to tune. The ensemble methods as a whole clearly outperformed Logistic Regression, showing their power in predicting on structured data (although RF, while using somewhat comparable hyperparameters to XGBoost still got severely outstripped by a margin of around 5.44%)

Overall, throughout several iterations of running, testing, debugging, rewriting, I learnt a lot of valuable steps to include, general practices which might help me in future implementations (such as printing val score for every 5 trees), and how to go about implementing something as complex as this. Through the different files (demarcated in my experimentation folder with a suffix from `_v1` to `_v11`), I learnt what I was missing (starting from just a GBM), I learnt what else I should be implementing and what features from the paper were important to include (also how to effectively implement them for the MNIST dataset - such as sparsity handling). While I was not able to include parallelisation features from Section 4 of the paper, I still managed to introduce the other features detailed. Transitioning from the theory of these algorithms and the word of the paper to the practicalities of their implementation proved to definitely be a challenge, but one I hope I have risen to.