University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Computer Science

A Study on Gradient Boosting Classifiers

A large-scale experimental analysis of hyperparameter effect on binary classification models

Juliano Garcia de Oliveira

Undergraduate Thesis

mac 499 — Capstone Project

Program: Computer Science
Advisor: Prof. Dr. Roberto Hirata Jr.

São Paulo
November 2019

Too much consistency is as bad for the mind as it is for the body. Consistency is contrary to nature, contrary to life. The only completely consistent people are the dead.
— Aldous Huxley

Resumo

Juliano Garcia de Oliveira. **Um Estudo sobre Classificadores de *Gradient Boosting*: *Uma análise experimental em larga escala do efeito de hiperparâmetros em modelos de classificação binária***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2019.

O *Gradient Boosting Machines* (GBMs) é um algoritmo supervisionado de aprendizado de máquina que vem obtendo excelente resultados em uma ampla gama de problemas e vencendo diversas competições de aprendizado de máquina. Ao construir um modelo de aprendizado de máquina, a otimização de hiperparâmetros pode se tornar uma tarefa dispendiosa e demorada, dependendo do número e do espaço de busca do procedimento de otimização. Usuários de aprendizado de máquina que não são pesquisadores experientes ou profissionais de ciência de dados podem ter dificuldade para definir quais hiperparâmetros e valores escolher ao iniciar o ajuste do modelo, especialmente com implementações mais recentes de GBMs como a biblioteca XGBoost e LightGBM. Neste trabalho, um experimento em larga escala com 70 conjuntos de dados é realizado usando a plataforma OpenML, medindo a sensibilidade das métricas de avaliação de classificadores binários a alterações em três hiperparâmetros da biblioteca LightGBM. Um arcabouço estatístico sólido é aplicado aos resultados do estudo, analisando o comportamento através de três pontos de vista diferentes: resultados por hiperparâmetros, resultados por características do conjunto de dados e resultados por métrica de desempenho. Os experimentos realizados indicam relações interessantes dos hiperparâmetros nos classificadores de *gradient boosting*, descobrindo quais combinações de hiperparâmetros resultaram em modelos com a maior alteração nas métricas, quais delas são mais sensíveis e quais características dos conjuntos de dados estudados se destacaram. Estes resultados são apresentados aqui para facilitar a criação de modelos de classificação baseados em GBMs em aprendizado de máquina.

Palavras-chave: Importância de Hiperparâmetros. Gradient Boosting. Aprendizado de Máquina Supervisionado. Análise Experimental. Classificação. Seleção de Modelos. Estudo Empírico.

Abstract

Juliano Garcia de Oliveira. **A Study on Gradient Boosting Classifiers:** ***A large-scale experimental analysis of hyperparameter effect on binary classification models***. Undergraduate Thesis (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2019.

Gradient Boosting Machines (GBMs) is a supervised machine learning algorithm that has been achieving state-of-the-art results in a wide range of different problems and winning machine learning competitions. When building any machine learning model, the hyperparameter optimization can become a costly and time-consuming task depending on the number and the hyperparameter space of the tuning procedure. Machine learning users that are not experienced researchers or data science professionals can struggle to define which hyperparameters and values to choose when starting the model tuning, especially with newer GBMs implementations like the XGBoost and LightGBM library. In this work, a large-scale experiment with 70 datasets is conducted using the OpenML platform, measuring the sensitivity of binary classifiers evaluation metrics to changes in three LightGBM hyperparameters. A solid statistical framework is applied to the study results, analyzing the behavior from three different viewpoints: results by hyperparameters, results by characteristics of the dataset and results by performance metric. The carried out experiments indicate insightful relationships of the hyperparameters in gradient boosting classifiers, uncovering which combinations of hyperparameters resulted in models with the highest change in the metrics from the baseline, what metrics are most sensitive and which characteristics of the studied datasets stood out. These results are hereby here presented to facilitate the model building of gradient boosting classifiers for machine learning users.

Keywords: Hyperparameter Importance. Gradient Boosting. Supervised Machine Learning. Experimental Analysis. Classification. Model Selection. Empirical Study.

Contents

Symbols and Notations ix

List of Figures xi

List of Tables xiii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Objective . . . 2
- 1.3 Structure . . . 2

2 Fundamentals of Supervised Learning 3
- 2.1 Function Approximation and Optimization . . . 3
- 2.2 Validation and Evaluation . . . 4
- 2.3 Classification Task . . . 4
- 2.4 Classification Metrics . . . 5
  - 2.4.1 AUC . . . 6
  - 2.4.2 Brier Score . . . 7
  - 2.4.3 Logloss . . . 7
- 2.5 Model Tuning . . . 8
- 2.6 Decision Trees . . . 9

3 Gradient Boosting Machines 11
- 3.1 Additive Model . . . 11
- 3.2 Gradient Descent . . . 11
- 3.3 Boosting and *AdaBoost* . . . 12
- 3.4 GBMs . . . 13
- 3.5 *XGBoost* and *LightGBM* . . . 14
- 3.6 Hyperparameters . . . 16

4 Study Structure 19
- 4.1 Tools . . . 19
- 4.2 OpenML Datasets . . . 20
  - 4.2.1 Data Cleaning . . . 22
- 4.3 Descriptive Dataset Statistics . . . 22
- 4.4 Hyperparameter Space . . . 23

4.4.1 Hyperparameters Tree . . . 23
4.4.2 Maximum Depth . . . 25
4.4.3 Learning Rate . . . 25
4.4.4 Number of Estimators . . . 27
4.5 Final Experiment Pipeline . . . 29
4.5.1 Label Encoding . . . 29

5 Experimental Analysis **31**
5.1 Single Dataset Experiment . . . 31
5.1.1 Individual Hyperparameter Impact . . . 32
5.1.2 Hyperparameter Pairs Impact . . . 33
5.1.3 Hyperparameter Triples Impact . . . 35
5.2 Dataset Aggregation and Clustering . . . 36
5.2.1 Aggregation . . . 38
5.2.2 Clustering Strategy . . . 39
5.3 Experiment Definitions . . . 42
5.3.1 δ_{metric} Definition . . . 43
5.3.2 Ordering and Visualization . . . 44
5.4 Statistical Framework . . . 45
5.4.1 Experimental Scenarios . . . 46
5.4.2 Analysis of Variance . . . 46
5.4.3 ANOVA . . . 47
5.4.4 Testing of Assumptions . . . 48
5.4.5 Kruskal–Wallis: A Nonparametric Approach . . . 51

6 Results and Discussion **53**
6.1 Statistical Significance . . . 53
6.2 Treatment Effect and Single-Factor Plots . . . 56
6.3 Result by Hyperparameters . . . 58
6.3.1 η_{NE} . . . 58
6.3.2 η_{MD} . . . 59
6.3.3 η_{LR} . . . 60
6.3.4 $\eta_{MD,LR}$ . . . 61
6.3.5 $\eta_{MD,NE}$ . . . 63
6.3.6 $\eta_{LR,NE}$ . . . 64
6.3.7 $\eta_{NE,MD,LR}$ . . . 65
6.4 Results by Hyperparameter Combination Effect . . . 67
6.5 Results by Cluster . . . 68
6.6 Results by Metrics . . . 70

7 Conclusion **73**
7.1 Limitations and Future Work . . . 73

A Cluster Ridgelines Plots **75**

B Statistical Tests Results **81**

Bibliography **85**

Symbols and Notations

Some of the symbols and notations are based on Goodfellow et al. [2016], with slight modifications.

$\boldsymbol{x}^{(i)}$	The i-th example from a dataset, vector or scalar
$y^{(i)}$	The target associated with $\boldsymbol{x}^{(i)}$
$\hat{y}^{(i)}$	The prediction associated with $\boldsymbol{x}^{(i)}$
$(\boldsymbol{x}^{(i)}, y^{(i)})$	A supervised dataset pair
$\boldsymbol{X}$	A $n \times p$ matrix, with input example $\boldsymbol{x}^{(i)}$ in the i-th row
$\boldsymbol{X}_{i,j}$	Element i, j of matrix $\boldsymbol{X}$
$\boldsymbol{X}_{i,:}$	Row i of matrix $\boldsymbol{X}$
$\boldsymbol{X}_{:,i}$	Column i of matrix $\boldsymbol{X}$
y	Vector of the actual targets of $\boldsymbol{X}$
$\hat{\text{y}}$	Vector of the predictions of the targets, where the i-th position is the prediction of $\boldsymbol{x}^{(i)}$
D_i	Number of instances of the ith dataset
LR	Notation for `learning_rate`
NE	Notation for `num_estimators`
MD	Notation for `max_depth`
$Hspace_i$	Hyperparameter space of the ith dataset
δ_{metric}	difference between a new metric and the baseline
$\eta^{(k)}$	Union of all hyperparameters of the kth cluster
$\eta_Q^{(k)}$	A partition Q of $\eta^{(k)}$
$\mathcal{S}(C_k, \eta_Q^{(k)}, m)$	Experimental scenario in the kth cluster, partitioned by hyperparameters Q and metric m
H_0	Null hypothesis

List of Figures

2.1 Example of decision boundary for a binary classifier, from Hastie et al. [2009] 5
2.2 ROC curve of a logistic regression classifier, from Kuhn and Johnson [2013]. Two points of the ROC curve are highlighted, showing different values of Specificity and Sensitivity. 7
2.3 Schema of a single 5-fold Cross-Validation iteration, from Hastie et al. [2009] 8
2.4 Example of a trained decision tree, from Russell and Norvig [2010] 9

3.1 Training of an AdaBoost classifier, image from Marsh [2016] 13
3.2 Example of structure score calculation for a single tree, from Chen and Guestrin [2015] 15
3.3 Leaf-wise tree growth, from LightGBM documentation (Ke et al. [2017b]) 16

4.1 Number of datasets after filtering is close to 400 20
4.2 Example of dataset information from OpenML 21
4.3 Number of data points distribution 21
4.4 Hyperparameter tree 24
4.5 Learning rate distribution, changing D_i and N_{LR}. Bigger dataset sizes generate higher learning rates. 27
4.6 Number of estimators distribution - changing D_i, for a fixed $N_{NE} = 20$. Bigger dataset sizes generate higher *num_estimators* values. 28

5.1 Individual impact of **`max_depth`** on the *BNG(kr-vs-kp)* dataset: one can observe that when $max\_depth > 14$ the model starts to overfit. 32
5.2 Individual impact of **`num_estimators`** on the *BNG(kr-vs-kp)* dataset 33
5.3 Pair impact of $MD_i \times NE_i$ on the *BNG(kr-vs-kp)* dataset 34
5.4 Pair impact of $MD_i \times LR_i$ on the *BNG(kr-vs-kp)* dataset 35
5.5 Triple impact of hyperparameters on the *BNG(kr-vs-kp)* dataset, showing only **test** set AUC 36
5.6 Feature types of the benchmark datasets 37
5.7 Cardinality of categorical features of the benchmark datasets 37
5.8 Skewness distribution of the numeric features in the dataset 38
5.9 t-SNE projection of all $\mathcal{P}_i$ 39
5.10 Dendrogram of the old clustering method 40
5.11 t-SNE projection with the assigned clusters and the centroids 41
5.12 Ridgeline plot of `categorical_ratio`. Clusters are represented by different colors 42
5.13 Ridgeline plot of `mean_skewness`. Clusters are represented by different colors . . . 42

5.14 δ_{AUC} for Cluster 3, changing only `max_depth`. The colors represent different datasets 44
5.15 δ_{Brier} for Cluster 2, changing both `max_depth` and `learning_rate`. The colors represent different datasets . . . 45
5.16 Two Normal Q-Q Plots from Cluster 1 results . . . 50

6.1 δ_{AUC} for scenario $\mathcal{S}(C_3, \eta^{(3)}_{LR,NE}, AUC)$, which failed the Levene Test; Some datasets δ_{AUC} increased the variance in treatment groups with very high learning rate and number of estimators . . . 55
6.2 Example of post hoc plot with the pairwise differences in treatment means . . . 56
6.3 SFM plot of $\mathcal{S}(C_3, \eta^{(3)}_{MD,LR}, AUC)$ . . . 57
6.4 SFM plot for $\mathcal{S}(C_1, \eta^{(1)}_{NE}, Logloss)$ . . . 58
6.5 SFM plot for $\mathcal{S}(C_6, \eta^{(6)}_{NE}, AUC)$ . . . 59
6.6 SFM plot for $\mathcal{S}(C_3, \eta^{(3)}_{MD}, AUC)$ . . . 60
6.7 δ_{AUC} plot for $\mathcal{S}(C_3, \eta^{(3)}_{LR}, AUC)$ . . . 60
6.8 SFM plot for $\mathcal{S}(C_1, \eta^{(1)}_{LR}, Brier)$ . . . 61
6.9 SFM plot for $\mathcal{S}(C_3, \eta^{(3)}_{MD,LR}, Brier)$ . . . 62
6.10 SFM plot for $\mathcal{S}(C_1, \eta^{(1)}_{MD,LR}, Brier)$ . . . 62
6.11 SFM plot for $\mathcal{S}(C_3, \eta^{(3)}_{MD,NE}, AUC)$ . . . 63
6.12 SFM plot for $\mathcal{S}(C_2, \eta^{(2)}_{MD,NE}, Logloss)$ . . . 64
6.13 SFM plot for $\mathcal{S}(C_1, \eta^{(1)}_{LR,NE}, Brier)$ . . . 65
6.14 $\delta_{Logloss}$ for scenario $\mathcal{S}(C_2, \eta^{(2)}_{NE,MD,LR}, Logloss)$ . . . 66
6.15 SFM plot for $\mathcal{S}(C_2, \eta^{(2)}_{NE,MD,LR}, Logloss)$ . . . 66
6.16 SFM plot for $\mathcal{S}(C_1, \eta^{(1)}_{NE,MD,LR}, Logloss)$ . . . 67
6.17 δ_{AUC} plot for $\mathcal{S}(C_3, \eta^{(3)}_{MD,NE}, AUC)$ - only two datasets seem to change the AUC . . . 69
6.18 Two SFM plots of η_{MD} and δ_{Brier} on Cluster 2 and 5, respectively . . . 69
6.19 δ_{AUC} plot for $\mathcal{S}(C_5, \eta^{(5)}_{NE,MD,LR}, AUC)$ . . . 71

A.1 Ridgeline plot of `num_rows` . . . 75
A.2 Ridgeline plot of `num_features` . . . 76
A.3 Ridgeline plot of `mean_variance` . . . 76
A.4 Ridgeline plot of `mean_skewness` . . . 77
A.5 Ridgeline plot of `num_categorical` . . . 77
A.6 Ridgeline plot of `sum_cardinality_over_categorical` . . . 78
A.7 Ridgeline plot of `categorical_ratio` . . . 78
A.8 Ridgeline plot of `numeric_ratio` . . . 79
A.9 Ridgeline plot of `boolean_ratio` . . . 79

List of Tables

2.1 The confusion matrix for binary classification tasks 6

5.1 Datasets from Cluster 2 and Cluster 3 41
5.2 Table exemplifying a general Single-Factor Experiment, based on Montgomery [2017] 46
5.3 SFM model for scenario $\mathcal{S}(C_1, \eta^{(1)}_{max\_depth}, Brier)$ 49
5.4 Statistical p-values of the experimental scenarios in Cluster 2. All the normality assumptions were rejected in this case 52

6.1 All experimental scenarios and their respective Kruskal–Wallis results for equality of treatment means 54
6.2 Percentage of statistically significant results for each comparison and metric 67
6.3 Percentage of statistically significant results in each cluster, by metric 68
6.4 Percentage of statistically significant results of each metric 70

B.1 Statistical p-values of the experimental scenarios in Cluster 1 81
B.2 Statistical p-values of the experimental scenarios in Cluster 2 82
B.3 Statistical p-values of the experimental scenarios in Cluster 3 82
B.4 Statistical p-values of the experimental scenarios in Cluster 4 83
B.5 Statistical p-values of the experimental scenarios in Cluster 5 83
B.6 Statistical p-values of the experimental scenarios in Cluster 6 84

Chapter 1

Introduction

1.1 Motivation

Machine Learning techniques are widely used in the modern industry to leverage useful insights and applications from data. In a range of very different areas, from medical labs to financial institutions, machine learning models are already a fundamental part of the business. There is a rising trend of academic research on the theory and applications of machine learning techniques, which is evidence of the growing importance of robust machine learning models for modern scientific and industrial applications.

Gradient Boosting Machines (GBMs) is one of the techniques which has been achieving state-of-the-art results, especially with problems in structured datasets. *State of Data Science and Machine Learning 2019* from Kaggle [2019], an industry-wide survey that presents a comprehensive view of the state of data science and machine learning, reports that GBMs is the third most used algorithm chosen by the respondents. The XGBoost and LightGBM libraries are the commonly used implementations nowadays, using decision trees as the base learners in the algorithm.

In the applied research and industry a considerable amount of a data scientist's time is reserved to experimentation and improvements to new or existing machine learning models. An important component of this process is the experimentation with parameters that the algorithm does not learn directly from the data, the *hyperparameters*. Despite the increase in the development of automated tuning techniques and the currently available hyperparameter optimization methods, XGBoost and LightGBM libraries have several hyperparameters to be optimized, which depending on the amount of data and the desired number of hyperparameters renders these tuning techniques infeasible.

Recent research has been tackling this problem with different solutions and perspectives. In Probst et al. [2018], the concept of hyperparameter *tunability* is formalized and new default values of hyperparameters are calculated for some algorithms. In another paper, van Rijn and Hutter [2018] attempts to measure the relative importance of each hyperparameter over different datasets, including the gradient boosting algorithm AdaBoost.

To expand on the knowledge of how hyperparameters alter the machine learning model performance, this study proposes a large-scale experiment relating dataset's characteristics, gradient boosting classification models, a set of hyperparameters and selected evaluation metrics for classification tasks.

1.2 Objective

The main objective of this work is to study and obtain insights about the main hyperparameters of gradient boosting binary classification models, using a statistically robust framework in the analysis. The research objective of the studies can be summarized in the following viewpoints:

1. How each type of hyperparameter (or combinations of them) affects the performance of the model, overall. Is there a hyperparameter which does not affect much the performance? What is the combination of hyperparameter that impacts the most? How sensitive is the gradient boosting algorithm to the studied hyperparameters?

2. How different characteristics of a dataset affect the hyperparameters impact, if there is a difference in them. Datasets with high number of features will have a different impact on hyperparameter? How the datasets are sensitive to hyperparameter changes in the algorithm?

3. How the performance metrics of classification models relate, obtain insights about the relative behavior of the performance metrics and the hyperparameters. Is there a pattern in the metrics behavior? Which combination of hyperparameter causes the highest change in a performance metric?

1.3 Structure

The next two chapters of this work, Chapter 2 and Chapter 3 introduce the required theoretical background for supervised machine learning and gradient boosting machines, including an explanation of the studied hyperparameters. Chapter 4 describe the structure of the experiment, which datasets were used, how the hyperparameters values were created and the experiment pipeline. In Chapter 5 the experimental analysis is outlined, with the description of the aggregation method used, definitions of concepts for the final analysis and a complete characterization of the statistical framework used in the results. The main results are presented in 6, with the outcomes observed from the analysis along with discussions of what was observed. Finally, Chapter 7 present the conclusions from the overall study and a discussion about some of its limitations that could be addressed in future work.

There are two complementary materials for this work: the first ones are the appendices A and B at the end of this dissertation, which contain all the ridgeline plots and statistical results of the experiment. The second complementary material is a *Jupyter Notebook* with the code and complete list of plots, results and analysis performed on the data.

Chapter 2

Fundamentals of Supervised Learning

In *Machine Learning* one of the main objectives is to discover patterns from data, obtain insights and solve a multitude of problems in different fields. Tasks that deal with data in the form of $(\boldsymbol{x}^{(i)}, y^{(i)})$, with $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$, where the true outcome $y^{(i)} \in \mathbb{R}$ is available for at least some instances of the dataset, is part of the *supervised learning* area of machine learning.

In Russell and Norvig [2010] the task of *supervised learning* is defined as:

> "Given a *training set* of N example input-output pairs
>
> $$(\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), ..., (\boldsymbol{x}^{(N)}, y^{(N)})$$
>
> where each $y^{(i)}$ was generated by an unknown function $y = f(\boldsymbol{x})$, discover a function h which approximates the true function f.
>
> The $\boldsymbol{x}$ and y can be any value(...). The function h is a **hypothesis**. Learning is a search through the space of possible hypothesis for one that will perform well, even on new examples beyond the training set."

Some classical examples of supervised learning tasks include: Email Spam detection, where $\boldsymbol{x}^{(i)}$ is the text (or a representation of it) of an email, and the label $y^{(i)}$ is a Boolean indicator of whether the i-th email is a spam or not; Classification of handwritten digits, where $\boldsymbol{x}^{(i)}$ is an image or the representation of a handwritten number and $y^{(i)}$ the corresponding number; Predicting house prices, where $\boldsymbol{x}^{(i)}$ is a set of different characteristics of a house and $y^{(i)}$ is its price.

2.1 Function Approximation and Optimization

A supervised learning task can be understood as an optimization problem. As explained in Hastie et al. [2009], this function-fitting paradigm basically

> "attempts to learn f by example through a *teacher*. One observes the system under study, both the inputs and outputs, and assembles a training set $(\boldsymbol{x}^{(i)}, y^{(i)})$. The observed input values to the system $\boldsymbol{x}^{(i)}$ are also fed into an artificial system, known as a learning algorithm (usually a computer program), which also produces outputs $\hat{f}(\boldsymbol{x}^{(i)})$ in response to the inputs. The learning algorithm has the property that it can modify its input/output relationship $\hat{f}$ in response to differences $y^{(i)} - \hat{f}(\boldsymbol{x}^{(i)})$ between the

> original and generated outputs. This process is known as *learning by example*. Upon completion of the learning process the hope is that the artificial and real outputs will be close enough to be useful for all sets of inputs likely to be encountered in practice (generalization)."

The functions can be estimated using a range of different **models**, i.e. specific hypotheses chosen to approximate the true underlying function f. Examples of commonly used approximators are *linear basis expansions*, the sigmoid transformation, *maximum likelihood estimation*, etc., and they are described in details in Hastie et al. [2009]. To estimate the parameters of these approximations that best fit the data, learning algorithms usually perform an optimization over a specific *loss function* (Kuhn and Johnson [2013]) which depends on the chosen approximator.

2.2 Validation and Evaluation

The basic supervised learning problem usually consists of two sets of data: **training set** and the **test set**. A model is trained using data from the training set, resulting in a model $\hat{f}$. This model then predicts the data in the test set, and with the predicted $\hat{y}^{(i)}$ the model generalization error (prediction error) can be estimated.

The **Bias-Variance Tradeoff** (Hastie et al. [2009]) in predictive modeling impacts the ability of a learning method to generalize. High variance indicates that the model fits random noise in the training set, which usually results in low generalization power (overfitting). On the other hand, a model with high bias has a very low difference in prediction error between the training set and test set, but usually has poor performance (underfitting). In the model training and evaluation, its important to assess generalization performance, and this is usually done using the test set. In this assessment process **overfitting** and **underfitting** can be detected by comparing the model's prediction error on training versus test set. For more information on this topic consult Chapter 7 of Hastie et al. [2009].

2.3 Classification Task

When the codomain of f is finite, i.e. y can assume a finite set of values, the learning problem is called **classification**. Specifically, when the cardinality of the codomain can assume only two values the problem is called **binary classification**. An example of the decision boundary of a binary classifier can be seen in Figure 2.1.

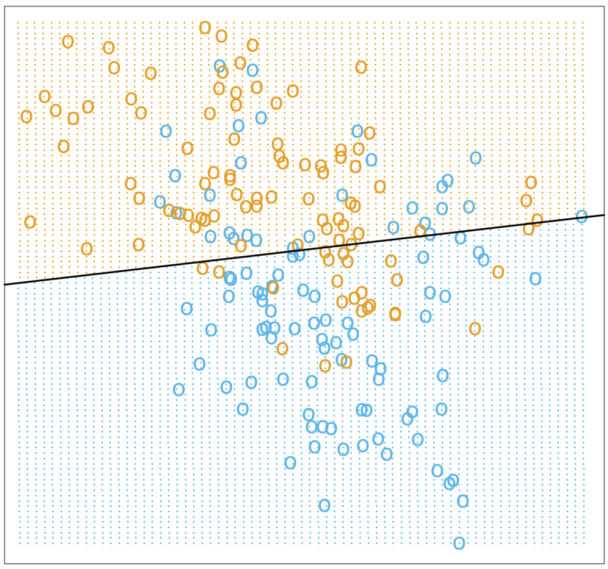


Figure 2.1: *Example of decision boundary for a binary classifier, from Hastie et al. [2009]*

The Email Spam detection problem described in the last section is a binary classification problem since the output is binary: the email is either spam, or it is not. In this work, the focus will be on **binary classification** problems and their specific metrics.

2.4 Classification Metrics

There are many metrics for model performance and evaluation of classification models. In this study, the metrics evaluated are the *AUC* (or AUROC), which stands for Area Under the Receiver Operating Characteristic Curve; The logarithmic loss or *Logloss*, which is a commonly used loss function for classification problems; *Brier Score*, a score function which measures the accuracy of probabilistic predictions. A good overview of these metrics is found in Kuhn and Johnson [2013].

In a binary classifier, each prediction of the model ouputs a number between 0 and 1. For simplicity, assuming that the binary labels are 0 and 1, the ouput can be interpreted as the predicted probability of a given instance being of class 1. In applications where the expected usage is the class itself instead of a probability, a **threshold** needs to be set, i.e by setting a threshold τ, the predicted labels will be equal to 1 if the outputed value is $\geq \tau$, and 0 otherwise.

The intuition behind choosing these specific metrics are:

- AUC is widely used in research, and it is a good metric insensitive to *class imbalance*, having a bounded value between 0 and 1;
- Logloss is the direct function optimized in the experiments, i.e. the gradient boosting steps are trying to reduce the logarithmic loss in each step. The logloss takes into account the uncertainty of the prediction and how much the actual label differ from it;
- Brier Score has good statistical properties, according to Rufibach [2010] it also addresses simultaneously probability calibration, consistency between predicted probabilities and the observation as well as *sharpness*, which is related to the concentration of the predictive distribution;

Below an overview about each metric is given. For more information on specific characteristics and mathematical formulation of each evaluation metric one can refer to Brown and Davis [2006], Rufibach [2010], Kuhn and Johnson [2013] and Hastie et al. [2009].

2.4.1 AUC

The Receiver Operating Characteristic curve is a probability curve that measures the power of prediction of a binary classifier given a threshold. The outcomes of a binary classification can be summarized in a *confusion matrix*, illustrated in Table 2.1. The TP, FP, TN and FN stands for *True Positives*, *False Positives*, *True Negatives* and *False Negatives*. They are calculated by simply distributing all the data into each part of the matrix depending on the predicted and actual class of each data point.

		Actual Class	
		True	**False**
Predicted Class	**True**	TP	FP
	False	FN	TN

Table 2.1: *The confusion matrix for binary classification tasks*

The ROC curve is calculated by plotting the *True Positive Rate* (TPR) against the *False Positive Rate*, as seen in Figure 2.2. In the figure, two points of the ROC curve are highlighted to illustrate a difference in the metrics of the ROC curve. The definition of a ROC curve is

$$Roc_x(\tau) = TPR_\tau = \frac{TP_\tau}{TP_\tau + FN_\tau}$$

$$Roc_y(\tau) = FPR_\tau = \frac{FP_\tau}{FP_\tau + TN_\tau}$$

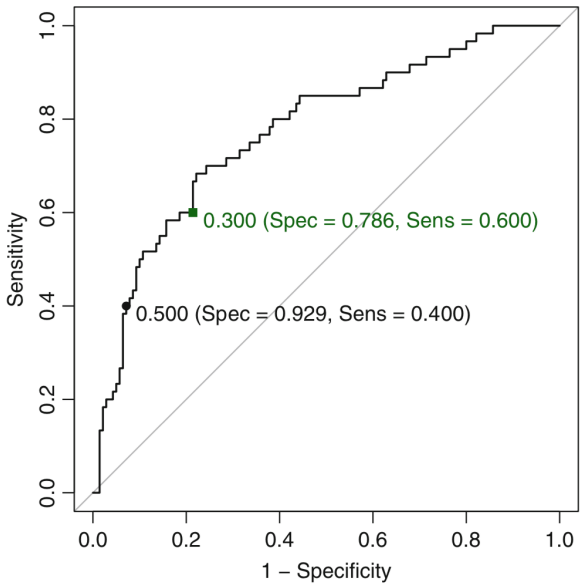


Figure 2.2: *ROC curve of a logistic regression classifier, from Kuhn and Johnson [2013]. Two points of the ROC curve are highlighted, showing different values of Specificity and Sensitivity.*

All of these metrics depend on the threshold τ. The **AUC** is defined as the area under this curve and provides an aggregate measure of the classifier performance against all possible thresholds. It ranges from 0 (worst performance) to 1 (perfect classifier).

2.4.2 Brier Score

The **Brier Score** is the mean squared difference between the predicted probabilities and the actual labels.

Similar to the AUC, the Brier Score is a bounded measure from 0 to 1, but its interpretation is different: 0 is the best possible Brier Score, while 1 is the worst. The actual equation is demonstrated in 2.1.

$$Brier = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2 \tag{2.1}$$

Brier Score can sum up different components related to forecasting, especially taking into account the calibration of the predicted probabilities. In Murphy [1973] there is an insightful partition of the Brier Score in three components, according to the article: "a measure of the uncertainty inherent in the events, or states, on the occasions of concern (...); a measure of the reliability of the forecasts; and a new measure of the resolution of the forecasts". The mathematical formulation of this partition is omitted here as it is out of the scope of this study, but can be found in the works of Rufibach [2010] and Murphy [1973].

2.4.3 Logloss

Logarithmic Loss or **Logloss** measures the accuracy of a classifier by penalizing mistakes depending on how uncertain a prediction is from the actual label. Basically, this encapsulates the intuition that giving a highly confident prediction (e.g. $\hat{y} = 0.95$) but being very far from the actual

label y = 0 is worse then being uncertain about it (e.g. $\hat{y} = 0.5$). The logloss equation for binary classifiers is shown in Equation 2.2.

$$Logloss = -\frac{1}{N}\sum_{i=1}^{n}[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \tag{2.2}$$

2.5 Model Tuning

Many models in supervised learning have parameters that cannot be directly estimated from the available data. For example, in a decision tree algorithm, the max depth or the criterion to measure the quality of a split needs to be chosen before the algorithm learning process. According to Kuhn and Johnson [2013] this type of parameter is called a *tuning parameter* because there is no analytical formula available to calculate an appropriate value for it. This parameter is also called a **hyperparameter** (Kuhn and Johnson [2013]).

Hyperparameters control the complexity of a model, controlling bias and variance of an algorithm. In the decision tree example, increasing the max depth the learning algorithm can use will increase the complexity of the model, as the tree can learn more complex patterns in data. In the logistic regression model, the regularization parameter (usually denoted λ) is a hyperparameter of the model, and also controls the complexity of the model. For a more complete explanation of regularization and its importance in predictive modeling refer to Chapter 5 of Hastie et al. [2009].

Since the hyperparameters are not directly estimated from the training data, there are different methods to optimize and find the appropriate ones for a given model. In applied machine learning and data science, a common procedure for model tuning (i.e. finding hyperparameters) usually consists of multiple runs of the same algorithm, but changing the hyperparameter(s) in each run, and evaluating the classifier (or another type of model) in a **validation set**. This set is different than the *test set* presented in Section 2.2, as the test set is needed to assess the generalization error of the final model after model tuning.

Besides the traditional separation into train, validation and test set, resampling techniques are also widely used for estimating model performance. The two main flavors of resampling according to Kuhn and Johnson [2013] is the *k-Fold Cross-Validation*, where the samples are randomly partitioned into k sets of approximately equal size, and in each iteration $k - 1$ of these folds are used for training and the remaining fold is used for model evaluation (see Figure 2.3); And another resampling method is a *bootstrap*, a random sample of the data taken *with replacement*, training a model on the selected samples and evaluating to predict the out-of-bag samples.

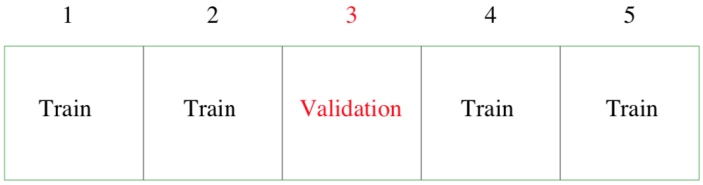


Figure 2.3: *Schema of a single 5-fold Cross-Validation iteration, from Hastie et al. [2009]*

Model tuning is a critical part of the workflow when building machine learning models, and there is vast literature exploring different approaches to quantify more sensitive hyperparameters given an algorithm (in Probst et al. [2018]), automatic tuning approaches using Bayesian Optimization

(Bergstra et al. [2013]), etc. In this study, the focus is on the hyperparameters of gradient boosting algorithms, specifically gradient boosting with decision trees, explained in Chapter 3.

2.6 Decision Trees

Most of the current gradient boosting algorithms use decision trees as the base learners. Decision trees, or specifically *classification trees* in the case of classification problems, consists of multiple nested conditional statements in its internal nodes, with the predictions in the leaves.

The actual learning process of a decision tree is called an *induction* of the decision tree. The idea of the algorithm is to choose internal tree splits that best explain the data and is a small tree, as described in Russell and Norvig [2010]. Figure 2.4 contains an example of a classification tree for deciding whether to wait for a table.

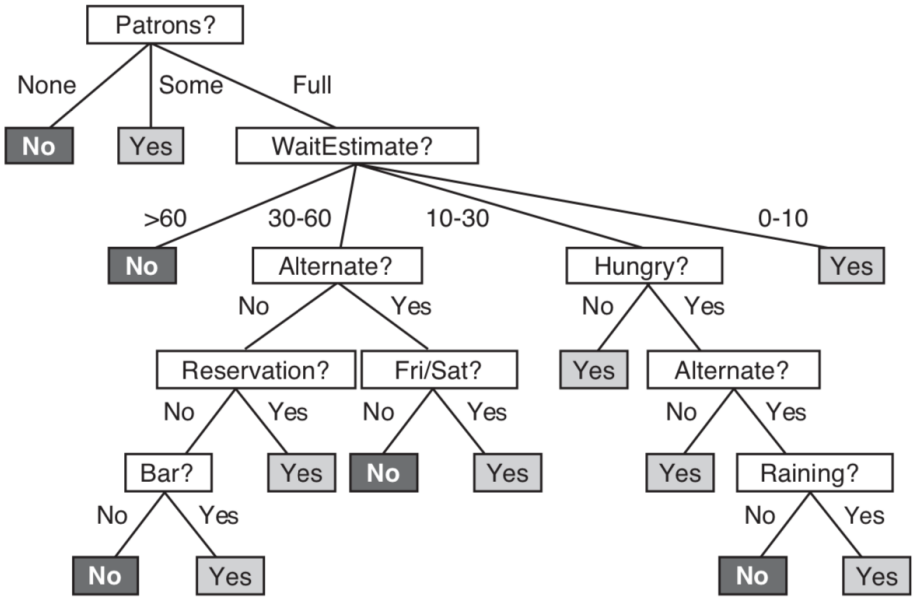


Figure 2.4: *Example of a trained decision tree, from Russell and Norvig [2010]*

The learning process of classification trees consists of multiple iterations to choose optimal splits given the training data provided, according to some criteria. Common optimization criteria are usually derived from Information Theory, like the Gini Index, Cross-Entropy and misclassification error (Hastie et al. [2009]). In practical usages and because of numeric properties (see Hastie et al. [2009]), Gini Index and Cross-Entropy are favored over the misclassification error.

Typical hyperparameters of classifications trees refer to the actual structure of the tree. The *maximum depth* hyperparameter controls how deep the tree can grow, i.e. how many splits will it have, *minimum leaf samples* controls the minimum number of samples each leaf needs to have, etc. For an extensive empirical study of decision tree hyperparameter tuning, one can refer to Mantovani et al. [2018].

Chapter 3

Gradient Boosting Machines

Gradient Boosting Machines(GBM) is a machine learning algorithm, based on the idea of additive models from statistics and gradient descent(Hastie et al. [2009]). GBM works by building a forward stagewise additive model by performing gradient descent in function space, as proposed by Friedman [2000].

Gradient Boosting, especially tree-based gradient boosting, has been achieving state-of-the-art results in many datasets (Li [2012]), especially with structured data. Structured data is any type of data organized in defined data structures, like a *JSON* file or a table with rows and columns. Gradient Boosting is a widely-used machine learning algorithm due to its efficiency, accuracy, and interpretability, according to Ke et al. [2017a]. The *Additive Modeling* technique and the *Gradient Descent* algorithm are key ideas to GBMs and are briefly introduced below.

3.1 Additive Model

An additive model is a regression technique suggested by Friedman and Stuetzle [1981], where the basic idea is to approximate a dataset using a sum of **smooth functions** of the individual features of the observed data, instead of a complex general regression surface.

Consider a supervised dataset is a set of pairs $(\boldsymbol{x}^{(i)}, y^{(i)})$, where $\boldsymbol{x}^{(i)}$ denotes the d-dimensional vector of the i-th observation (d is the number of features), and $y^{(i)}$ is the true outcome of the i-th observation. An additive model is then described as:

$$\mathbb{E}[y^{(i)} \mid \boldsymbol{x}^{(i)}] = \beta_0 + \sum_{j=1}^{d} f_j(\boldsymbol{X}_{:,j}) \tag{3.1}$$

The β_0 is the intercept term, and each function f_j is learned from the respective j-th feature over all the observations in the dataset, i.e. the column $\boldsymbol{X}_{:,j}$. Each of those functions can be estimated using any nonparametric regression technique, such as Gaussian Process Regression, smoothing splines, kernel regression, k-nearest-neighbors, etc., as indicated in Hastie et al. [2009].

3.2 Gradient Descent

Gradient Descent is an iterative optimization algorithm. As explained by Ruder [2016], the basic version of the algorithm is used to **minimize** an objective function $J(\boldsymbol{X}; \theta)$. This is parameterized

by a model's parameters $\theta \in \mathbb{R}^p$, by using the gradient vector w.r.t the parameters θ of the function at a specific point to update the parameters in the opposite direction of this vector. Usually, the parameters θ are updated until a fixed number of iterations is reached, or convergence is achieved. This algorithm uses the learning rate η to control how much the coefficients of θ can change on each iteration.

The basic update rule of the original gradient descent algorithm is defined as

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_{\theta_{t-1}} J(\boldsymbol{X}; \theta_{t-1}) \tag{3.2}$$

A gradient descent execution will run the above update $t = M$ times, and it can be interpreted as generating M vectors p_m of the form $p_m = -\eta \cdot \nabla_{\theta_{m-1}} J(\boldsymbol{X}; \theta_{m-1})$, and this is a key idea which is important in the mathematical formulation of Gradient Boosting Machines. Denoting $p_0 = \theta_0$ (the initial parameters values before optimization), the final optimal parameters $\theta^\star$ can be written as

$$\theta^\star = \sum_{m=0}^{M-1} p_m \tag{3.3}$$

3.3 Boosting and *AdaBoost*

The boosting method is a general technique that attempts to "boost" the accuracy of a given machine learning algorithm, answering the following question: "Can a set of weak learners create a single strong learner?". A *weak learner* is a model that performs just slightly better than random guessing. The purpose of boosting is to sequentially apply the *weak learning algorithm* (i.e. an algorithm that produces *weak learners*) to modified versions of the data, producing a sequence of weak classifiers $f_m, m = 1, 2, ..., M$, as described by Hastie et al. [2009].

On a superficial level, boosting is similar to another technique called *bagging*, used in the famous *Random Forest* learning algorithm, as both are trying to create a strong learner from an ensemble of weak learners. However, they are fundamentally different: In both bagging and boosting, the variance of the estimates is reduced when the ensemble of learners is created, however, the boosting method reduces both the bias and the variance, which makes this technique prone to *overfitting*.

Schapire [1999] gives a brief overview of the boosting method and its applications, especially the famous **AdaBoost** algorithm, which was a major breakthrough in Machine Learning research. The original AdaBoost algorithm is designed for classification problems, where the output is either -1 or 1, and the final prediction for a given instance is a weighted sum of each generated weak classifier:

$$F_m(\boldsymbol{x}) = sign(\sum_{m=1}^{M} \rho_m \cdot f_m(\boldsymbol{x})) \tag{3.4}$$

The weights ρ_m, $\rho_m \in \mathbb{R}$ in equation 3.4 are computed by the boosting algorithm, and the idea is to increase the influence of weak learners that are more accurate, while at the same time penalizing the ones that do not predict very well, and this is illustrated in Figure 3.1.

The data modifications done in each step of the boosting procedure is to assign weights w_i, $w_i \in \mathbb{R}$, for each $\boldsymbol{x}^{(i)}$ in the dataset, based if the instance $\boldsymbol{x}^{(i)}$ was correctly classified or not. The initial weight for all data points is $1/n$, and in the boosting process these weights are adjusted

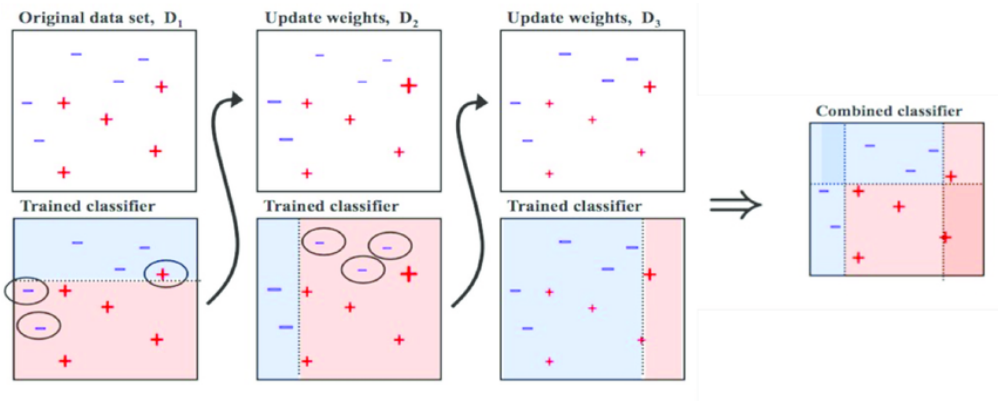


Figure 3.1: *Training of an AdaBoost classifier, image from Marsh [2016]*

following an exponential function using ρ_m and if the point being updated was correctly classified or not in the last update. One can refer to Hastie et al. [2009] for a detailed explanation of the AdaBoost algorithm and its formal mathematical description.

3.4 GBMs

Gradient Boosting Machines were originally introduced by Friedman [2000], in the paper titled "Greedy Function Approximation: A Gradient Boosting Machine". In this work, Friedman makes a connection between stagewise additive expansions (the concept of an additive model) and gradient descent. The GBM algorithm works by optimizing any given differentiable loss function, using gradient descent. However, the optimization is not done in terms of a numeric optimization (i.e. of updating a vector of parameters θ), as described in Section 3.2, but by "boosting" functions in the direction of the gradient of the loss function.

Since GBMs deal with finite data, they optimize the functions in terms of the supervised dataset $(\boldsymbol{x}^{(i)}, y^{(i)})$ inputs, not w.r.t all values of $\boldsymbol{x}$, as they are (usually) infinite. The final model of the GBM will be similar to equations 3.1 and 3.4:

$$F_M(x) = F_0(x) + \sum_{m=1}^{M} F_m(x) \tag{3.5}$$

The $F_m(x)$ functions are boosted functions built in a stagewise fashion, just like the θ_t in gradient descent. The base functions are learners, and they can be parametrized as $\beta_m h(\boldsymbol{x}; a_m)$, where β_m is a weight, and a_m the parameters of the learner h. In most implementations the base functions are tree-based learners, but they could be any learner where it is possible to assign weights. Also, given a loss function $L(y_i, F_m(x_i))$, we would like to find all the optimal values of a_m and β_m that would minimize the loss function, ie., if we have M functions:

$$\{\beta_m, a_m\}_1^M = \underset{\{\beta'_m, a'_m\}_1^M}{\arg\min} \sum_{i=1}^{n} L\left(y^{(i)}, \sum_{m=1}^{M} \beta'_m h(\boldsymbol{x}^{(i)}; a'_m)\right)$$

However in most situations the optimization above is infeasible, so the “greedy-stagewise” approach is to optimize each pair (β_m, a_m) in a stagewise model, that is, for each $m = 1, 2, ..., M$

$$(\beta_m, a_m) = \arg\min_{\beta,a} \sum_{i=1}^{n} L(y^{(i)}, F_{m-1}(\boldsymbol{x}^{(i)}) + \beta h(\boldsymbol{x}^{(i)}; a))$$

using a vectorized notation and replacing $\Delta_m(\boldsymbol{X}) = \rho_m h(\boldsymbol{X}; a_m)$, the update rule of the F_m functions described in Equation 3.5 is

$$F_m(\boldsymbol{X}) = F_{m-1}(\boldsymbol{X}) + \eta \Delta_m(X) \tag{3.6}$$

The learning rate η can also be called the *shrinkage* parameter, as it "shrinks" the influence of the new learner. Friedman [2000] shows that $\beta_m h(x; a_m)$ can be interpreted as the best greedy step towards the optimal estimate, which can be represented as $F^\star(x)$(similarly to Equation 3.3). This can be seen as an update of the gradient descent method, as in Equation 3.2. The analogue of $-\nabla_{\theta_t}$ in the numerical gradient descent from Equation 3.2 is the gradient of the loss function L with relation to the last estimate $F_{m-1}(x)$, in the notation used by Friedman [2000]:

$$-g_m(\boldsymbol{x}^{(i)}) = -\left[\frac{\partial L(y^{(i)}, F(\boldsymbol{x}^{(i)}))}{\partial F(\boldsymbol{x}^{(i)})}\right]_{F(x)=F_{m-1}(x)}$$

In literature, this gradient of the loss function L with respect to the last prediction $\hat{\mathrm{y}}_{m-1}$ (i.e. $F_{m-1}(\boldsymbol{X})$) is sometimes called **pseudo-residual**, and written as $\mathbf{r}_{m-1}$. Using a vectorized notation, the *pseudo-residual* can be written as

$$\mathbf{r}_{m-1} = \nabla_{F_{m-1}(\boldsymbol{X})} L(\mathrm{y}, F_{m-1}(\boldsymbol{X})) = \nabla_{\hat{\mathrm{y}}_{\boldsymbol{m-1}}} L(\mathrm{y}, \hat{\mathrm{y}}_{\boldsymbol{m-1}})$$

The GBM algorithm fits a learner $h(x; a_m)$ with weight β_m using the pseudo-residuals, not the original $\boldsymbol{X}$. The final version of the algorithm is formally defined as

GRADIENT_BOOST$(\boldsymbol{X}, \mathrm{y}, M, \eta)$
1 $F_0(\boldsymbol{X}) = \arg\min_\nu \sum_{i=1}^{n} L(y^{(i)}, \nu)$
2 **for** $m = 1$ **to** M
3 $\quad \mathbf{r}_{m-1} = \nabla_{\hat{\mathrm{y}}_{m-1}} L(\mathrm{y}, \hat{\mathrm{y}}_{m-1})$, where $\hat{\mathrm{y}}_{m-1} = F_{m-1}(\boldsymbol{X})$
4 $\quad$ // Train a base learner minimizing squared error
5 $\quad a_m = \arg\min_{a,\beta} \sum_{i=1}^{n} \left(\mathbf{r}_{m-1}^{(i)} - \beta h(\boldsymbol{x}^{(i)}; a)\right)^2$
6 $\quad \rho_m = \arg\min_\rho \sum_{i=1}^{n} L(y^{(i)}, F_{m-1}(\boldsymbol{x}^{(i)}) + \rho h(\boldsymbol{x}^{(i)}; a_m))$
7 $\quad \Delta_m(X) = \rho_m h(\boldsymbol{X}; a_m)$
8 $\quad F_m(\boldsymbol{X}) = F_{m-1}(\boldsymbol{X}) + \eta \Delta_m(X)$
9 return F_M

3.5 *XGBoost* and *LightGBM*

Even though Gradient Boosting was a known technique used in machine learning, its use became widespread with the development of XGBoost, a scalable end-to-end tree boosting system, by Chen and Guestrin [2016]. The main improvements brought by XGBoost was the development of a sparsity-aware algorithm for sparse data and a "weighted quantile sketch for approximate tree learning". In general terms, the basic greedy algorithm to find the best split at a given tree node in the boosting process is very computationally demanding, especially when finding splits for

continuous variables. In XGBoost an *approximate algorithm* is used for split finding, using quantiles of the feature distribution as split points for continuous variables.

These improvements and a parallel tree learning architecture are the main reasons for the recent spread of gradient boosting techniques. According to Chen and Guestrin [2015], "(XGBoost) has been widely adopted by data scientists and machine learning practitioners. XGBoost has been used as a major system in winner solutions of more than ten machine learning challenges in the past year (2015), most of which are highly competitive. These include highly impactful ones in the field, such as KDDCup. It has also been widely adopted by industry users, including Google, Alibaba and Tencent, and various startup companies."

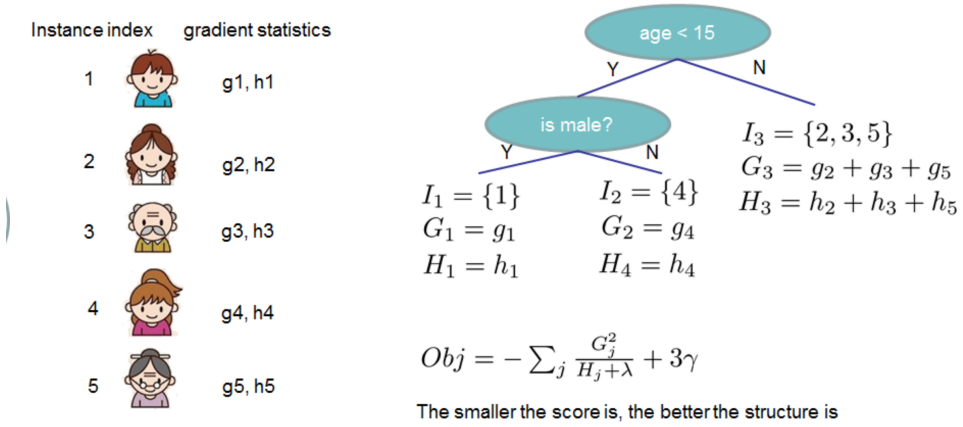


Figure 3.2: *Example of structure score calculation for a single tree, from Chen and Guestrin [2015]*

However, the algorithm used in this study is the novel **LightGBM**, created in 2017 by Microsoft Research in Ke et al. [2017a]. It is important to understand the basic principles and improvements of LightGBM since it is the chosen algorithm for the classification models in this study. There are two main ideas behind LightGBM, both trying to improve the scalability of the gradient boosting process for highly-dimensional data.

First, they propose a new technique for the estimation of information gain called *Gradient-based One-Side Sampling* (GOSS). Since one of the most time-consuming tasks in the learning process in gradient boosting is to find the split for the trees, usually some sort of sampling is done in this step for efficiency purposes. The researchers noticed that the gradient for each data point in the boosting process has useful information for a smarter data sampling strategy. In their paper Ke et al. [2017a] explains that "(...) if an instance is associated with a small gradient, the training error for this instance is small and it is already well-trained". A data sampling strategy is then built based on the rank of the absolute value of the gradient for each instance, sampling instances with higher gradient and subsampling instances with small gradients. The idea is to put more focus on the under-trained instances but keeping the data distribution closer to the original. GOSS has a theoretical proof that its estimation power and generalization performance is close to using the full data and better than random sampling.

To reduce the number of features, LightGBM develops a technique called *Exclusive Feature Bundling*. It makes use of the high sparsity usually present in high-dimensional data, by designing a " (...)nearly lossless approach to reduce the number of features". Since commonly there are many

features which are mutually exclusive (never take nonzero values simultaneously), they can be bundled into a single feature. This reduces significantly the algorithmic complexity of the algorithm since it is basically reducing the number of features.

Since in this work the impact of **hyperparameters** from LightGBM is studied, it is important to understand the basics of hyperparameters in these algorithms, covered in the next section.

3.6 Hyperparameters

One main difference in optimization from other GBM tools is that LightGBM grow trees *leaf-wise* (see Figure 3.3), while most decision tree learning algorithms use a depth-wise approach. This impacts on how each hyperparameter value should be chosen, and which values to optimize.

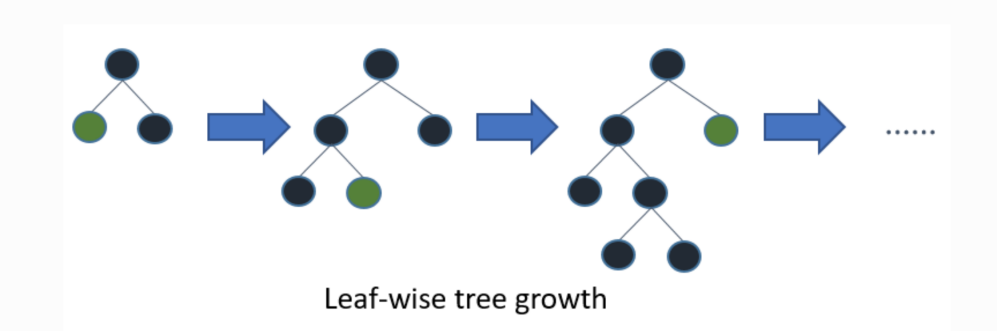


Figure 3.3: *Leaf-wise tree growth, from LightGBM documentation (Ke et al. [2017b])*

In the documentation from Ke et al. [2017b], there is a list of more than 20 hyperparameters one can change and tune when building a gradient boosting model. For tuning the model and controlling complexity, the commonly used hyperparameters to start with are:

- **`num_leaves`**

  This hyperparameter controls the maximum number of leaves to grow on each boosting iteration, and it is the main way to control the model complexity. By default, LightGBM has no tree maximum depth limit, so this is the main way to apply regularization on the structure of the model.

- **`min_data_in_leaf`**

  Controls the minimum number of data points required to grow a new leaf, it is very important since it helps prevent overfitting in leaf-wise trees. From practical experience, this parameter is one of the most sensitive to the number of training samples.

- **`max_depth`**

  Controls the depth of each tree explicitly, if needed. In this study, `max_depth` is one of the hyperparameters studied.

- **`learning_rate`**

  The learning rate is the parameter η from the Equation 3.6, and controls the influence of each new learner. In practice, this is not usually a "tunable" hyperparameter but rather used in combination with other hyperparameters in the tuning process (e.g. higher learning rate

makes the training of the model faster, therefore allowing for faster model tuning). It is one of the hyperparameters analyzed in this study.

- **`n_estimators`**

  The number of estimators or boosting iterations in the gradient boosting process. This is the parameter M in Equation 3.5, and controls how many trees are grown in model training. Usually, the larger the number of iterations, the better the model will get, until it starts overfitting in the training data. This is also another hyperparameter analyzed in this study.

There are many other parameters in LightGBM, but in practice, a model tuning using the models described above usually provides a strong baseline for further tuning.

Chapter 4

Study Structure

The main objective of this work is to study and compare the impact of selected hyperparameters in different models trained on different datasets, using the LightGBM gradient boosting algorithm. Each "type" of dataset will be aggregated in different categories using the values of aggregated characteristics of the dataset itself (e.g. number of features, number of instances), this can be checked on Section 4.3.

In the following sections an overview of the methodology and the experimental setup is given, explaining the datasets, the hyperparameters chosen and its distributions, pipeline for testing and the tools used in each step.

4.1 Tools

All of the study pipeline and analysis is done using Python 3.6. Besides the commonly used data science tools (*numpy, pandas, matplotlib, seaborn*) the most important packages used are:

- **fklearn**: Functional machine learning library, developed by Nubank and built on top of `scikit-learn` and main machine learning implementations, e.g. the `LightGBM`; It has very useful abstractions, used in this study to build a pipeline of multiple learners and applying the same evaluation function;
- **toolz**: Add more functional programming functionality into python, trying to follow the principles of composability, purity and laziness.
- **scikit-learn**: The most widely used machine learning library for python;
- **pingouin**: Statistical package, mainly used for ANOVA tests and Levene's test for homoscedasticity. More details about it can be found in Vallat [2018];
- **scipy**: Scientific computing package, the most used module was `scipy.stats` for statistical tests on the experiment results;
- **scikit-posthoc**: Library from the *scikit* environment, which provides post hoc tests for pairwise multiple comparisons that are usually performed in statistical data analysis to assess the differences between group levels if a statistically significant result of ANOVA (or the equivalent nonparametric test) test has been obtained, more details in Terpilowski [2019].

4.2 OpenML Datasets

The OpenML project has an online service to "(...) share, organize and reuse data, code and experiments. Following best practices observed in other sciences, OpenML allows collaborations to scale effortlessly and rewards scientists for sharing their data more openly" (Vanschoren et al. [2014]).

In the research of previous studies for this work, it was noted that OpenML was being used in different scientific studies related to machine learning; A specific subset of datasets was used in Probst et al. [2018] to conduct a large-scale benchmarking to measure hyperparameter tunability, and it was also used in Couronné et al. [2018] as a benchmark platform for automatically retrieving datasets and comparing different machine learning algorithms.

OpenML provides a python API package for automatically retrieving datasets, tasks, submit customized runs and obtaining OpenML flows (a description of a personalized machine learning task). In this study, the datasets used are retrieved using the python API, and filtered according to some personalized rules, which are:

1. **`classes`** = **2**, the study consists of analyzing binary classification problems;
2. **`min_num_features`** = **3**, the dataset must have at least 3 features;
3. **`min_num_instances`** = **1000**, the dataset must have at least 1000 instances (data points);
4. **`max_num_instances`** = **5000000**, the dataset must have at most 5 million data points (provided just as an upper bound, no dataset used had more then 2 million instances);
5. **`max_nan_percentage`** = **20%**, LightGBM has built-in support for missing values (*NaNs*), but the datasets are filtered to have at most 20% of its total number of rows with some missing values, for more meaningful data and simplicity.

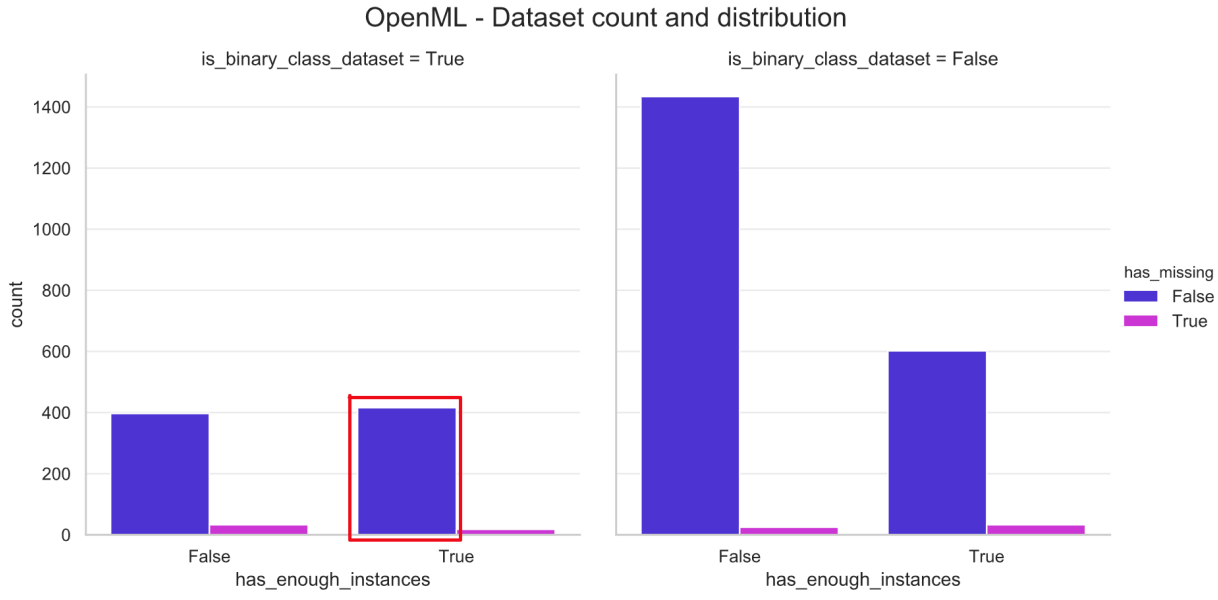


Figure 4.1: *Number of datasets after filtering is close to 400*

After all these filters, the number of valid datasets for the experiment would be close to 400 (figure 4.1), in theory. However, several datasets could not be opened by the python API due to a variety of errors, mostly related to Attribute-Relation File Format (ARFF) in python. This brought the number of possible datasets to experiment close to 200, but since each hyperparameter experiment takes a long time to run (favoring more hyperparameter space covered over the number of datasets, check 4.4.1), the final number of datasets analyzed in this study is 70.

Each dataset retrieved by the OpenML API has a dataset id (`did`) related to it, along with very useful metadata related to it, as it can be seen in Figure 4.2. With a simple exploratory analysis of the datasets, it was observed that a great number of them had a high proportion of categorical variables, and usually a high number of features too. Datasets with only textual data as features were also removed from the study, along with features not explicitly indicated as a categorical feature by the API.

	did	name	NumberOfInstances	NumberOfFeatures	NumberOfClasses	NumberOfInstancesWithMissingValues
1240	1240	AirlinesCodrnaAdult	1076790.0	30.0	2.0	4085.0
179	179	adult	48842.0	15.0	2.0	3620.0

Figure 4.2: *Example of dataset information from OpenML*

Analyzing the number of instances in the subset of datasets used in this study, it can be seen that most of the datasets have a relatively small number of instances, but a portion of them have more than 300000 data points, in hope to cover more data-heavy models (4.3).

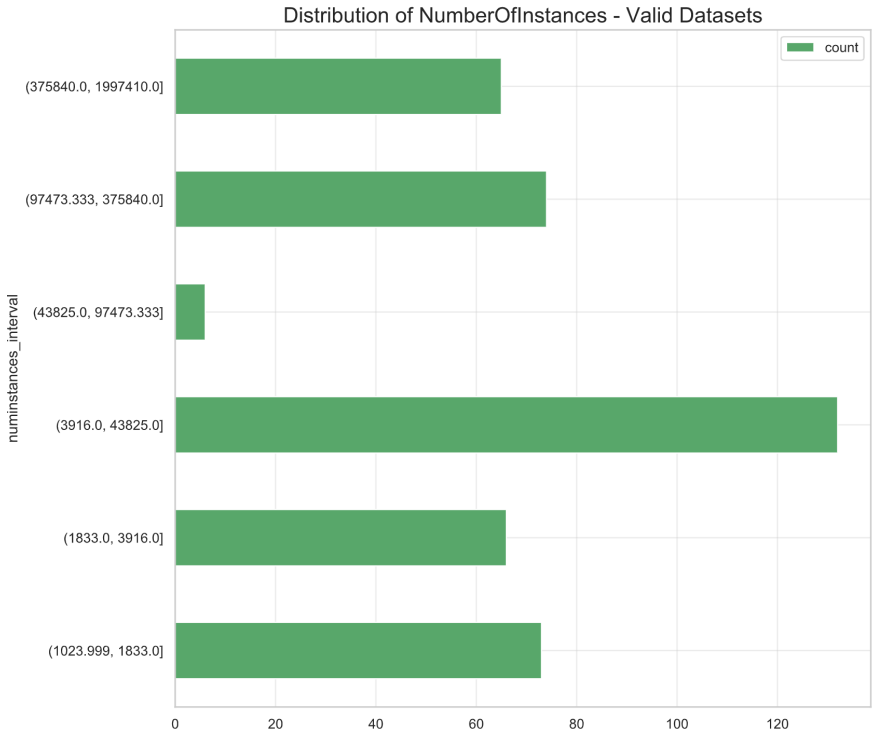


Figure 4.3: *Number of data points distribution*

4.2.1 Data Cleaning

In this study, there are only two data cleaning steps, intending to keep the cleaned dataset as close as possible to the original to not interfere with the experiment results.

1. The OpenML provides an attribute describing which of the features in the dataset are the categorical features; In the dataset building process, the type of each variable is identified (using *pandas-profiling* package), and from each identified feature as "categorical", remove all that aren't explicitly declared by the OpenML attributes;

2. The data in each OpenML dataset does not follow a specific pattern, e.g. some datasets represent the **target** variable as a string (*'y'* and *'n'*), some are represented in floating point, Boolean, etc. In the experiment pipeline, the **target** is converted to an integer, if possible. If not, the **target** is encoded using a *label encoder*.

4.3 Descriptive Dataset Statistics

To analyze subgroups of datasets and categorize each one of them, multiple statistics related to the structure of the dataset are calculated before any machine learning model is trained. Before the training process, the granularity of these statistics is feature-wise, which is then further aggregated in the result analysis step.

Categorical features are known to cause a *prediction shift* in traditional gradient boosting models, as demonstrated in Prokhorenkova et al. [2018]. This shift is identified as a special kind of target leakage, which can also be caused in a preprocessing step of categorical features when using target statistics (e.g. using the target/mean encoding technique)[1]. As noted in the mentioned paper, LightGBM converts categorical features to gradient statistics at each step of the boosting process (as explained in Section 3.5), which results in some information loss. To categorize datasets with relation to categorical features, in this work the following statistics related to categorical variables are calculated:

- **cardinality**, is the number of unique categorical values of a given feature;

- **variance**, is a simple measure of the variability of a specific categorical feature. Let $|top_j|$ be the number of occurrences of the most frequent category in feature j, i.e. the column $\boldsymbol{X}_{:,j}$. Then, this metric is defined as the inverse of the ratio of occurrences of the most frequent category:

$$categorical\_variability = 1 - \frac{|top_j|}{|\boldsymbol{X}_{:,j}|}$$

  As an example, if 90% of the values of a given feature is the same, then the categorical variability of the said feature is 0.1.

For **numeric features**, the main specific statistic calculated is the sample **skewness** of each feature. Skewness is a degree of distribution distortion, and in this study, it is calculated using the

[1] To fix this kind of problem in a dataset with many categorical features a new boosting algorithm was developed by Dorogush et al. [2018] called **Catboost**.

unbiased adjusted Fisher–Pearson standardized moment coefficient from *pandas*. The skewness is also trying to measure the variability of the numeric features and is used in the clustering of each dataset as a feature.

The pipeline also calculates other useful statistics for each feature, like the percentiles, number of NaNs, infinite values, mean, median, etc. They are useful in the exploratory step but are not used when clustering the datasets in the result analysis step.

4.4 Hyperparameter Space

In Section 3.6 the basic hyperparameters used in LightGBM's gradient boosting algorithm is explained. In this section, the *hyperparameter space*[2] of this study is explained, along with a simple overview of the hypothesis behind each distribution choice, related to LightGBM default hyperparameters and dataset size. Let D_i denote the number of instances of the i-th dataset, i.e. the total number of $(\boldsymbol{x}^{(i)}, y^{(i)})$ pairs for the i-th dataset.

4.4.1 Hyperparameters Tree

The intention is to train multiple models changing the hyperparameters values, and calculate the metrics (explained in Section 2.4) for each combination of hyperparameter. In this study three hyperparameters are studied: **`max_depth`**, **`learning_rate`** and **`num_estimators`**.

In this study, the *hyperparameter space* also represents all valid hyperparameters values to run for the datasets, and is a function of D_i. The hyperparameter space is represented as a special tree, where each new leaf of the tree takes its parent node and generate new leafs based on a new hyperparameters. Let LR_i, NE_i and MD_i denote the sets of possible hyperparameter values for learning rate, number of estimators and maximum depth, respectively. Let S_i be the set of all hyperparameter sets, i.e. $S_i = \{LR_i, NE_i, MD_i\}$, and $\mathcal{P}(S_i)$ the *power set* of S_i. The number of values to test for each hyperparameter is denoted with the numbers N_{LR}, N_{NE}, N_{MD}: e.g. if $N_{LR} = 4$ it means 4 values of learning rate will be generated (equivalent to say $|LR_i| = 4$).

For a given tree depth k, the hyperparameter set to be explored at that depth is defined using the rule in equation 4.1, where $\prod$ is the cartesian product operator.

$$HS_k = \left\{ \prod_{X \in Q} X : Q \in \mathcal{P}(S_i), |Q| = k \right\} \tag{4.1}$$

The hyperparameter tree of this study, represented in Figure 4.4, is then defined as different cartesian products of all valid k combinations. Since three hyperparameters are being changed, the maximum is $k = 3$, and the final space is the union of all cartesian products combinations:

$$Hspace_i = HS_1 \cup HS_2 \cup HS_3$$

This structure is generalizable for any number of hyperparameters, and it was designed this way to measure hyperparameter impact depending on the number of combinations. It is expected that changing multiple hyperparameters at the same time have a bigger impact on model performance than changing just one. Each level of the hyperparameter tree represents a "combination effect",

[2] In model tuning, the *hyperparameter space* is the set of all the values each hyperparameter can assume in the tuning process.

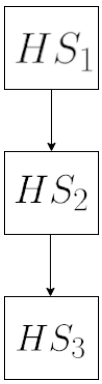


Figure 4.4: *Hyperparameter tree*

i.e. the experiment can measure, for multiple combinations of hyperparameters, which combination effect provides the biggest impact on the metrics (for better or worse) compared with the other combinations.

As an example, consider the three sets of hyperparameters below:

$$LR = \{0.1, 0.2\}$$

$$NE = \{1, 2\}$$

$$MD = \{10, 20\}$$

Then each leaf of the hyperparameter tree will be:

$$HS_1 = \Big\{\{0.1, 0.2\}, \{1, 2\}, \{10, 20\}\Big\}$$

$$\begin{aligned} HS_2 = \Big\{&\{(0.1, 1), (0.1, 2), (0.2, 1), (0.2, 2)\}, \\ &\{(0.1, 10), (0.1, 20), (0.2, 10), (0.2, 20)\}, \\ &\{(1, 10), (1, 20), (2, 10), (2, 20)\}\Big\} \end{aligned}$$

$$\begin{aligned} HS_3 = \Big\{&\{(0.1, 1, 10), (0.1, 1, 20), (0.1, 2, 10), (0.1, 2, 20), \\ &(0.2, 1, 10), (0.2, 1, 20), (0.2, 2, 10), (0.2, 2, 20)\}\Big\} \end{aligned}$$

It can be observed that the number of hyperparameters in each tree node increases multiplicatively with the number of initial hyperparameters and the number of layers k in the hyperparameter tree, and it is one of the reasons why the experiment may take a long time to run depending on the dataset. This was the chosen configuration of the experiment since the objective is not to perform a model tuning, but rather to measure the "magnitude" of the effects in performance metrics. Better strategies could be used for model tuning like Grid Search, Random Search or Bayesian Optimization, as it is stated in Probst et al. [2018].

4.4.2 Maximum Depth

The maximum depth is the hyperparameter that controls the maximum possible depth each boosting tree can have. As explained in Section 3.6, LightGBM default behavior is to grow the trees leaf-wise, having an unbound maximum depth, but the structure is controlled by the number of leaves in the tree. Max depth is a way to apply a constraint on the function space, i.e. the possible hypothesis the learning algorithm can use and is commonly optimized in classical decision trees for pruning.

Since gradient boosting harness the power of weak learners by combining them, usually the depth of a single tree is not as deep as a traditional single classification tree. For example, in XGBoost the default value for max depth is 6, while in the *rpart* package in R (single decision trees) the default max depth is 30. Even though in gradient boosting the max depth can be used to constrain the model, one can overfit in the data given a high enough number of boosting iterations: this can be seen as an "overfit relationship" between `max_depth` and `num_estimators`. For this reason, when defining the maximum depth or which values of it to optimize in a model tuning step there is not a clear rule to choose an optimal value.

In this study, the tested maximum depth values are always the same, independent on the dataset size:

$$MD_i = \{3, 4, ..., N_{MD} + 3\}$$

In the experiments $N_{MD} = 20$, so each dataset is run using max depth from 3 to 22. The intuition behind these values is to neither test with very big depth values, nor very small depth for a weak learner in gradient boosting. The `num_leaves` parameter is automatically set to be $\min(2^{max\_depth} - 1, 2^{15})$, because this way the number of leaves will not influence too much the results when changing the maximum depth and at the same time the number of nodes in each boosting tree will not grow unbound when training with `max_depth` ≥ 16.

4.4.3 Learning Rate

The learning rate distribution is simple, and it depends only on the dataset size D_i. In XGBoost the default value of learning rate (also called *eta*) is 0.3 and in LightGBM it is 0.1, which is also the default learning rate for *scikit-learn* implementation of gradient boosting. Since in the OpenML datasets the dataset sizes aren't too similar (see Figure 4.3), the distribution of learning rate values to generate is a simple function starting from 0.01 for small datasets, changing based on the dataset size.

Premise: Start with a learning rate value between 0.01 for small datasets, and a multiple of 0.1 for bigger datasets (a rule was defined by dividing D_i by 10000). Then, experiment with bigger and smaller values of learning rate around the "middle" value.

First it is defined a "middle" learning rate value ψ_i (defined in equation 4.2). Then the values are generated as a list using N_{LR}, to generate a symmetrical pattern of learning rates (the total length of each "side" of the learning rate list is described in 4.3).

$$\psi_i = \max\left(0.01, 0.1 \times \max(1, \frac{D_i}{10000})\right) \tag{4.2}$$

$$\kappa_i = \cdot \max\left(\lfloor \frac{N_{LR}}{2} \rfloor, 2\right) \tag{4.3}$$

The symmetric pattern and the final LR_i are generated following the rule described in equation 4.4. By changing the maximum length N_{LR} and the dataset size D_i, one can see the learning rate distribution behavior: In Figure 4.5 all the D_i from the study are categorized in percentiles, and depending on the length (N_{LR}) a given distribution of learning rate values LR_i is shown.

$$LR_i = \left\{\frac{\psi_i}{\kappa_i}, \frac{\psi_i}{\kappa_i - 1}, \frac{\psi_i}{\kappa_i - 2}, ..., \psi_i, ..., (\kappa_i - 2) \cdot \psi_i, (\kappa_i - 1) \cdot \psi_i, \kappa_i \cdot \psi_i\right\} \tag{4.4}$$

In this study, 5 unique learning rates are generated.

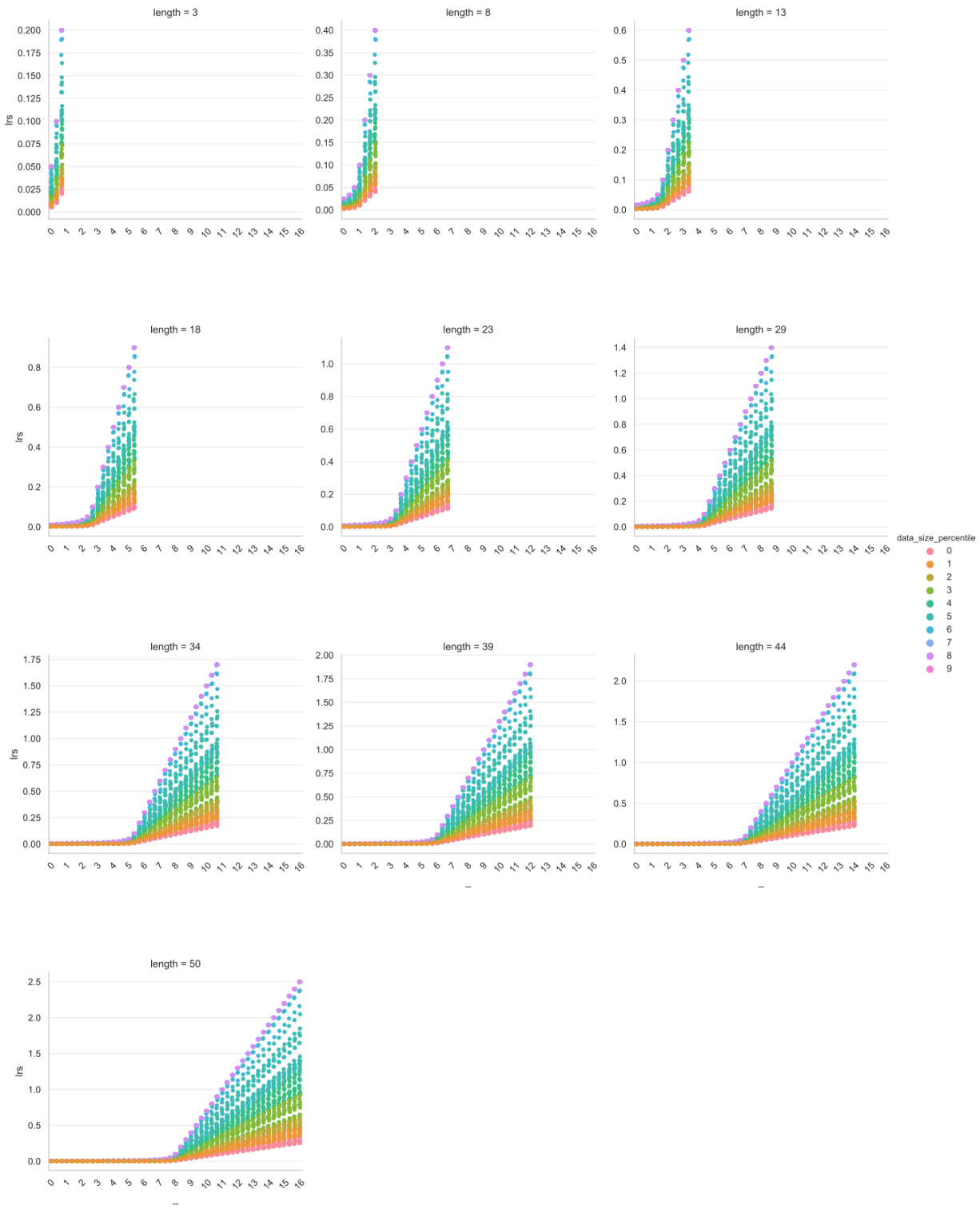


Figure 4.5: *Learning rate distribution, changing D_i and N_{LR}. Bigger dataset sizes generate higher learning rates.*

4.4.4 Number of Estimators

Both in XGBoost (`XGBClassifier`) and LightGBM the default value for the number of estimators is set to 100. The optimal number of boosting iterations can be influenced by the dataset size: by fixing other hyperparameters, a model of a dataset with very few data points can converge much faster (i.e. in few boosting iterations) than a dataset with 10 times its size. Since in the study

the intention is to test different boosting iterations, and the actual distribution ideally depends on D_i, first the maximum value of estimators a given dataset can have is defined, and then multiple values of `num_estimators` are generated.

The definition of this function is a little arbitrary, but the objective is to cover different values and tie together the D_i of the dataset. Using maximum values of 400 estimators for small datasets and 1700 for big datasets, along with the definitions of *small* and *big* datasets being $D_i \leq 10000$ and $D_i > 300000$ respectively, a linear function is then defined to calculate the maximum value to be assigned. After defining the constants in Equation 4.5 the rule for the max value of `num_estimators` is defined in Equation 4.6.

$$a = \frac{1700 - 400}{300000 - 10000}, \; b = 400 - a \tag{4.5}$$

$$max_{ne}(D_i) = \begin{cases} 400 \text{ if } D_i \leq 10000 \\ 1700 \text{ if } D_i > 300000 \\ a \cdot D_i + b \end{cases} \tag{4.6}$$

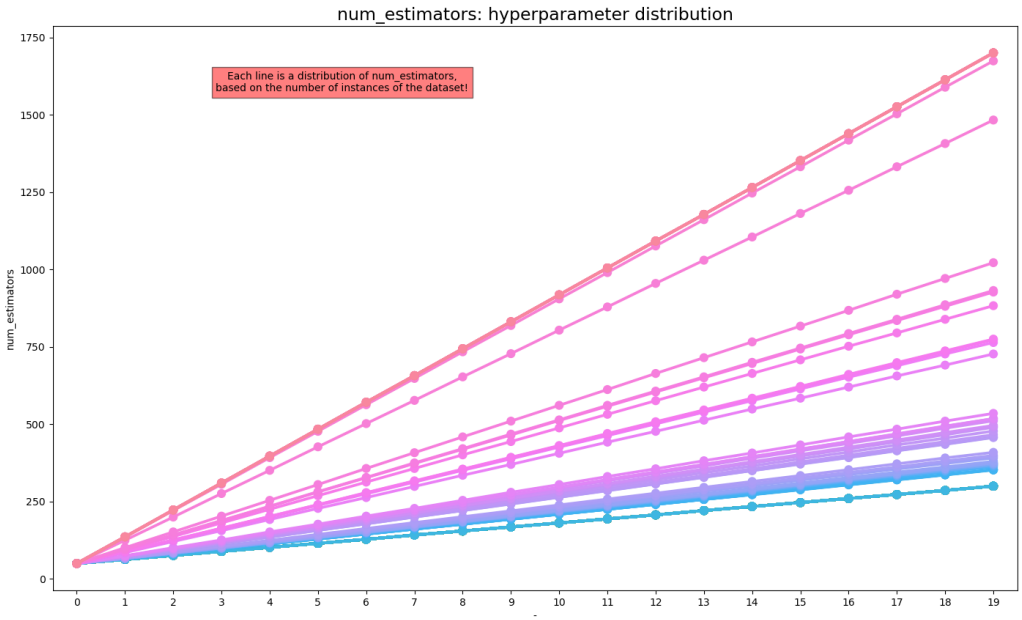


Figure 4.6: *Number of estimators distribution - changing D_i, for a fixed $N_{NE} = 20$. Bigger dataset sizes generate higher num_ estimators values.*

The final list of number of estimators generated is then a linear interpolation starting at $min_{ne}(D_i) = \min(\frac{D_i}{4}, 50)$ to $max_{ne}(D_i)$, in evenly spaced steps (Equation 4.7). An example of multiple NE_i distributions is shown in Figure 4.6 for different D_i values, but with a fixed length $N_{NE} = 20$.

$$step = \frac{\Big(max_{ne}(D_i) - min_{ne}(D_i)\Big)}{N_{NE} - 1}$$

$$NE_i = \left\{ \lfloor min_{ne}(D_i) \rfloor, \lfloor min_{ne}(D_i) + step \rfloor, \lfloor min_{ne}(D_i) + 2 \cdot step \rfloor, ..., \lfloor max_{ne}(D_i) \rfloor \right\} \tag{4.7}$$

4.5 Final Experiment Pipeline

For each dataset considered in this study a pipeline is built for it, following the sequence:

1. Calculate the descriptive statistics for the dataset (see Section 4.3);
2. Build the hyperparameter space $Hspace_i$, for the ith dataset;
3. Split the dataset into train and test set;
4. Build evaluators for each studied metric;
5. Train a LightGBM model for each hyperparameter value $\in Hspace_i$, and evaluate it on the train and test set;
6. Save logs.

The train test split strategy chosen for all datasets is a random split with train size being 70% of the data points and 30% for the test (**70/30**). Given a dataset, the split is fixed, i.e. the train and test are the same in all trained models.

4.5.1 Label Encoding

As stated in LightGBM's documentation, the library has built-in support for categorical features, which uses a specific method from Fisher [1958]. However, in order to make the study more general, it was decided to not use this specific feature of LightGBM. When categorical features are detected, a **label encoding** technique is used on the categorical values of the dataset. The label encoding is just a numeric encoding of the categorical features, i.e. each categorical value is mapped to an integer number.

Chapter 5

Experimental Analysis

Each dataset pipeline result has all evaluated metrics (AUC, Brier and Logloss), the dataset statistics (described in Section 4.3) and the generated hyperparameter space according to Section 4.4 of Chapter 4.

The focus of this Chapter is to give an overview of the main experimental analysis, how the datasets were clustered and a basic explanation of the statistical theory necessary for the analysis and statistical framework used.

5.1 Single Dataset Experiment

Each dataset result in the study has the following files:

1. **`shape`**: The number of data points D_i and the number of features of both train and test set;
2. **`analyzer_info`**: A dictionary with information calculated by the `analyzer`. It contains the feature set, what is the target column, an indicator if the dataset contains any categorical variable, and all the categorical features if it has any;
3. **`openml_object`**: The original *OpenML* object of the dataset, with its name, description (if it has one), basic attribute information, etc;
4. **`hp_tree`**: The generated hyperparameter tree for the dataset, i.e. all nodes of $Hspace_i$;
5. **`final_result`**: All the metric results for all combinations of hyperparameters in the hyperparameter tree; Contains the values used for the main analysis.

Each entry of `final_result` contains the evaluators result for both train and test set for all hyperparameters combinations. An example of a single entry in the `final_result` for a specific dataset (*BNG(kr-vs-kp)*, a chess move classification problem) is shown in Listing 5.1.

```
1 {'learning_rate': 0.30000000000000004,
2 'num_estimators': 1700,
3 'max_depth': 10,
4 'num_leaves': 1023,
5 'seed': 42,
6 'nthread': 32,
```

```
7 'verbose': -1,
8 'train_result': {'auc_evaluator__target': 0.9963452091395292,
9     'logloss_evaluator__target': 0.07025840176806959,
10    'brier_score_evaluator__target': 0.020611611004153745},
11 'test_result': {'auc_evaluator__target': 0.9857593012343584,
12    'logloss_evaluator__target': 0.208979460099794,
13    'brier_score_evaluator__target': 0.03484311126894392},
14 }
```

Listing 5.1: *BNG(kr-vs-kp) experiment result for hyperparam combination (0.3, 1700, 10)*

Since each dataset has multiple results assigned to it, one can do an **individual analysis** looking into a single dataset behavior. In this section, a summarized analysis of the **BNG(kr-vs-kp)** dataset is given as an example. The *BNG(kr-vs-kp)* or "King+Rook versus King+Pawn" is one of the datasets available in the OpenML platform, where the objective is to classify if the white player can win or not (given the board positions determined by the feature values), and it is described in more details in Shapiro [1987]. The "BNG" means this is an artificially augmented/generated dataset.

5.1.1 Individual Hyperparameter Impact

The first node of the hyperparameter tree only defines single hyperparameters values. By keeping the default LightGBM values for the remaining hyperparameters one can observe the impact of changing a single hyperparameter on each metric of interest. The plot of these values is the classical learning curve used in model tuning to assess the hyperparameter effect on model performance. In Figure 5.1 impact of max depth is illustrated and in 5.2 the number of estimators, for the *BNG(kr-vs-kp)* dataset.

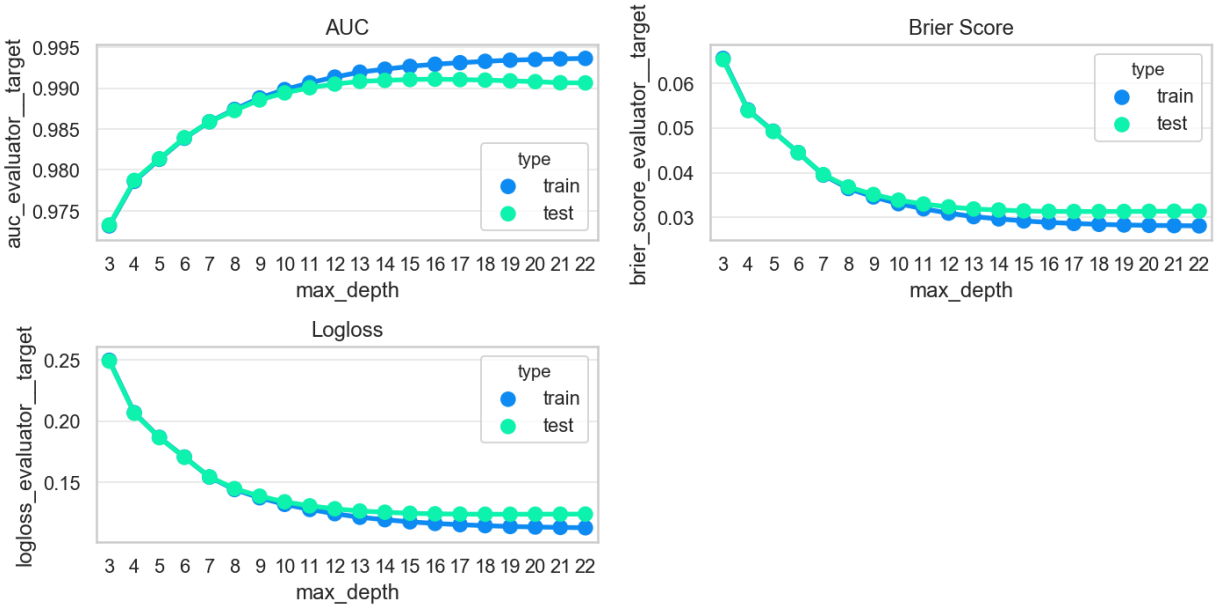


Figure 5.1: *Individual impact of* ***max_depth*** *on the BNG(kr-vs-kp) dataset: one can observe that when max_depth* > 14 *the model starts to overfit.*

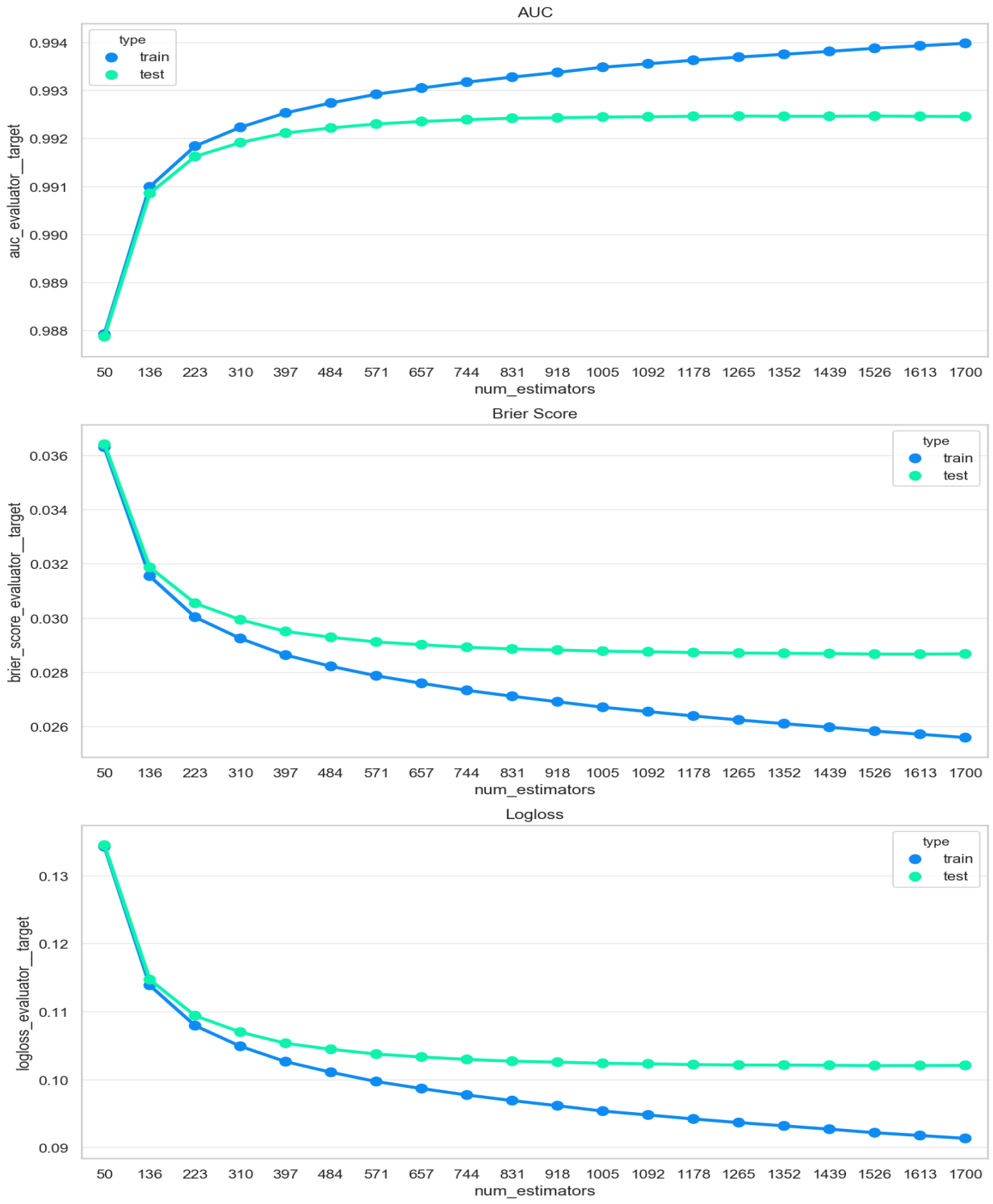


Figure 5.2: *Individual impact of* ***num_estimators*** *on the BNG(kr-vs-kp) dataset*

5.1.2 Hyperparameter Pairs Impact

The second node of the hyperparameter tree defines all pairs of possible combinations of hyperparameters. In this study, these combinations are $MD_i \times LR_i$, $MD_i \times NE_i$ and $LR_i \times NE_i$. For example, in Figure 5.3 one can see at the same time that a high **num_estimators** combined with lower **max_depth** have higher AUC on the test set (on this specific dataset), but at the same time low **num_estimators** combined with high **max_depth** also achieve high AUC. This is an inter-

esting behavior that repeats in different hyperparameters combinations, and are easily detectable in the final general analysis looking at the treatment effect of each hyperparameter combination.

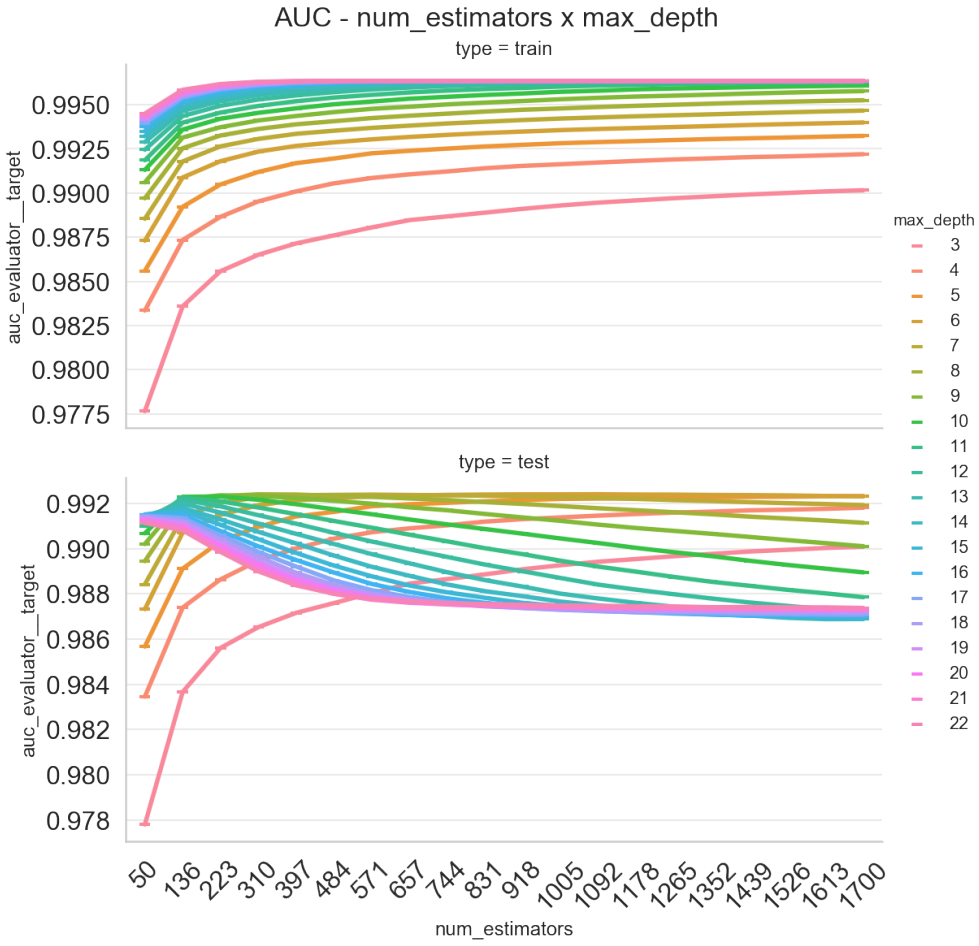


Figure 5.3: *Pair impact of $MD_i \times NE_i$ on the BNG(kr-vs-kp) dataset*

On the other hand, when analyzing Figure 5.4 the pair $MD_i \times LR_i$ does not show a clear inversion of the hyperparameters impact: one can only conclude that for *BNG(kr-vs-kp)* lower **`max_depth`** and high **`learning_rate`** results in higher AUC (keeping the other hyperparameters the same). This is expected since the learning rate does not have the same "overfit relationship" as maximum depth and number of estimators have (explained in Subsection 4.4.2).

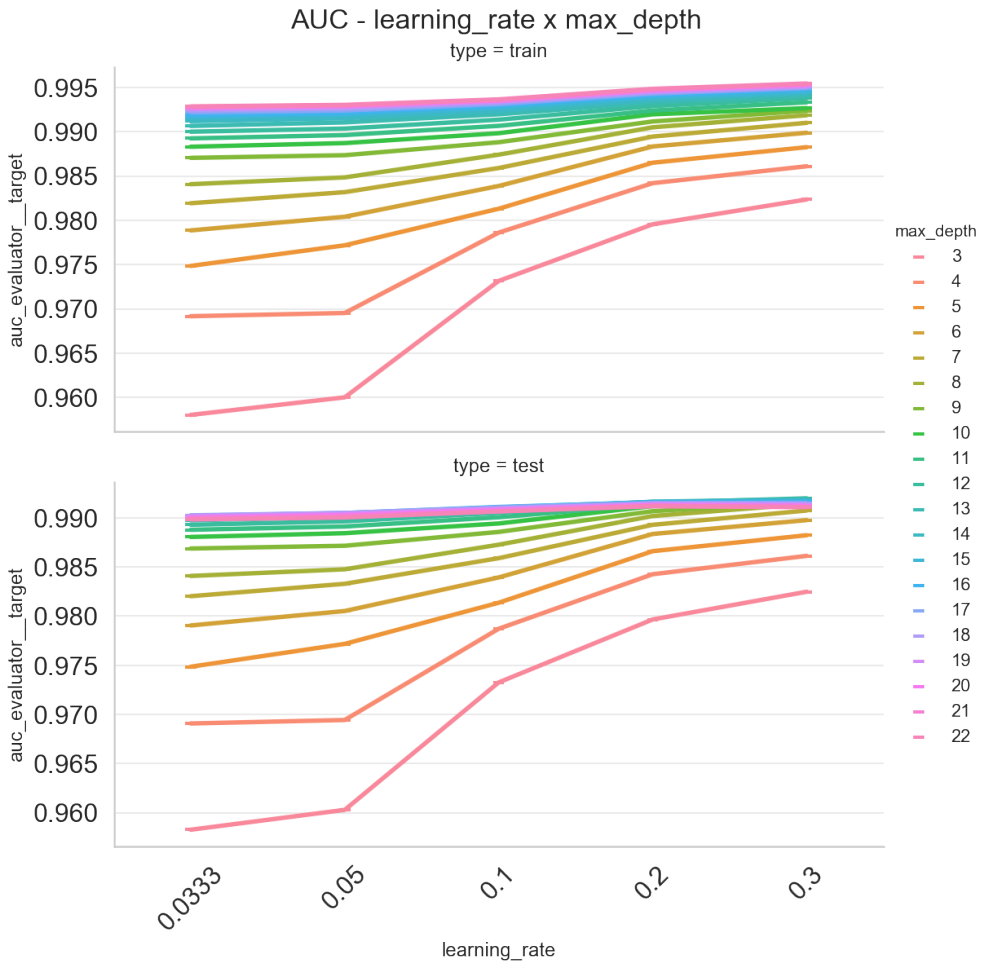


Figure 5.4: *Pair impact of $MD_i \times LR_i$ on the BNG(kr-vs-kp) dataset*

5.1.3 Hyperparameter Triples Impact

Finally, each dataset is also run using all possible combinations of the studied hyperparameters, i.e. a classical Grid Search run using `num_estimators`, `max_depth` and `learning_rate`. Results of AUC of the triple impact for *BNG(kr-vs-kp)* can be seen in Figure 5.5.

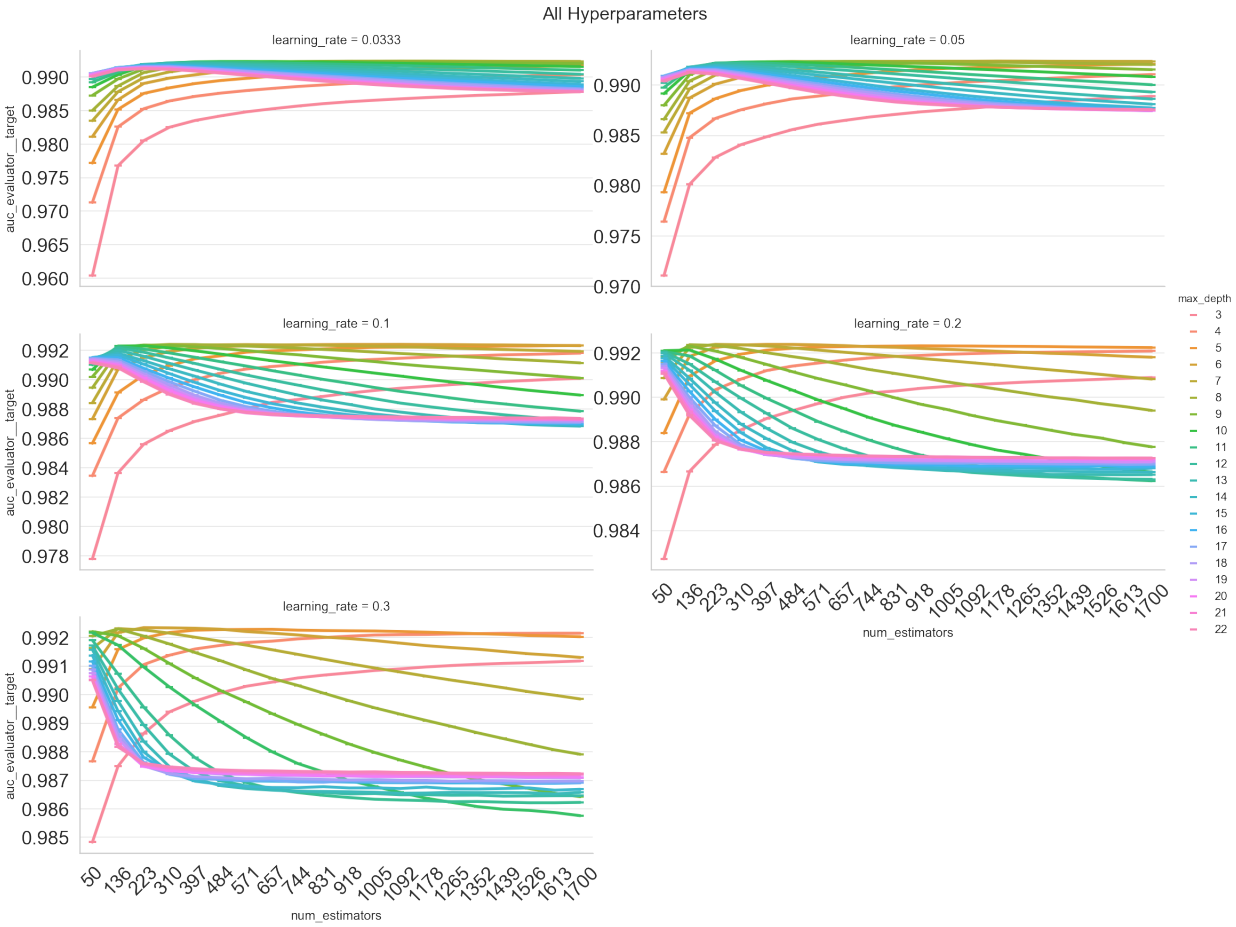


Figure 5.5: *Triple impact of hyperparameters on the BNG(kr-vs-kp) dataset, showing only* ***test*** *set AUC*

One of the inherent difficulties when analyzing these results is due to the high dimensionality of the results: Each experiment has at least three different types of hyperparameters, plus three different metrics for each run. In Subsection 5.3.2 this problem will be tackled, with an overview of the statistical techniques and analysis performed to compare and obtain insights into the impact on performance metrics.

5.2 Dataset Aggregation and Clustering

To tie dataset characteristics to hyperparameters effect in the subsequent analysis, a variety of different measures were calculated in the dataset analysis process, explained in detail in Section 4.3 of Chapter 4. When analyzing all datasets' features characteristics, useful insights into the data can be found on it. First, a clear conclusion is that proportion of categorical features in the dataset is much higher than other types (numeric, Boolean or constants), as demonstrated in Figure 5.6, and that the most common cardinality is two categories for the categorical features[1], while few features have high cardinality (Figure 5.7).

[1]These categorical features could be encoded as the *Boolean* feature type, or vice-versa. In the study it was decided that Boolean features are columns which had explicitly **True** and **False** as values.

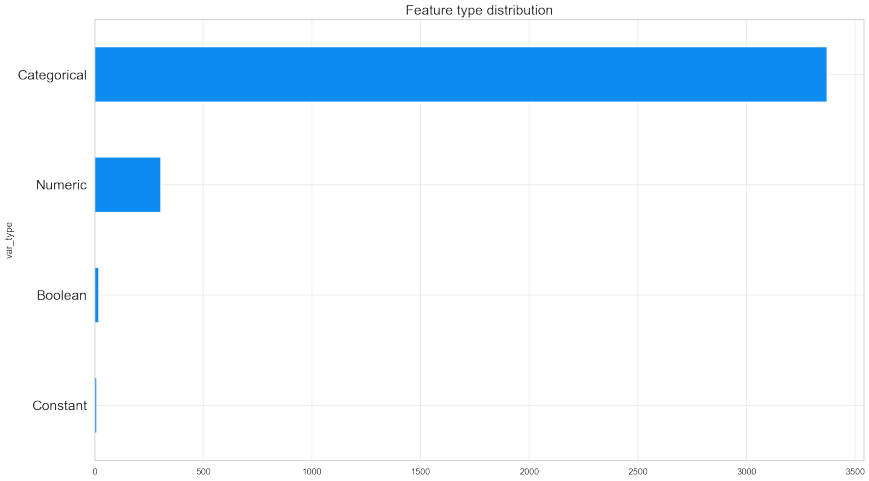


Figure 5.6: *Feature types of the benchmark datasets*

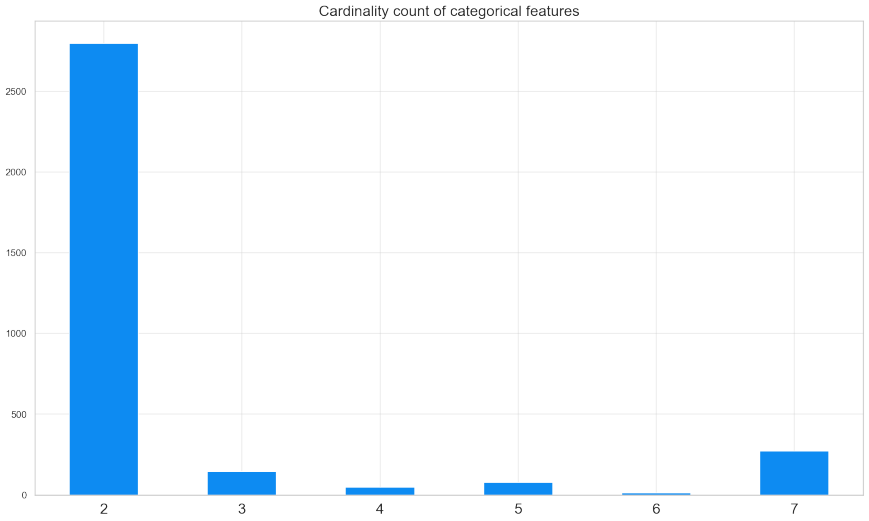


Figure 5.7: *Cardinality of categorical features of the benchmark datasets*

With relation to the skewness of the numeric features, most of them have low skewness, with the majority of them having 0 skewness. This can indicate either that the features themselves do not have innate skewness, or they've been processed by different techniques to normalize and remove skewness, like a logarithmic transformation, cubic root transformation, etc. The distribution can be seen in Figure 5.8.

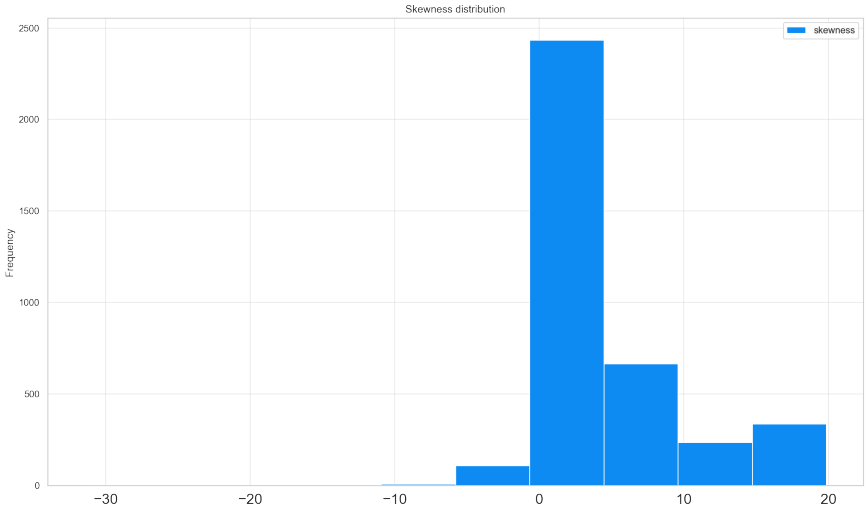


Figure 5.8: *Skewness distribution of the numeric features in the dataset*

5.2.1 Aggregation

Since the granularity of the statistics are feature-wise, all datasets were subjected to an aggregation process which combines the feature-wise statistics into *dataset aggregated characteristics*. The objective of this aggregation is to represent each dataset considered in the study into a point in a high dimensional space, which gives power to perform a multitude of transformations and comparisons, and in the case of this study, a clustering by the similarity of the datasets according to these aggregated values. For each dataset, the following aggregations are made:

- **`num_rows`**: Number of rows in the dataset, i.e. the number of instances of D_i;
- **`num_features`**: Number of features in the dataset;
- **`mean_skewness`**: 0 if there is no numeric features in the dataset, otherwise the mean of the skewness of the numeric features;
- **`mean_variance`**: The mean of the *variance* of categorical features, where the variance concept is the one described at Section 4.3;
- **`num_categorical`**: How many categorical features the dataset has;
- **`sum_cardinality_over_categorical`**: Let K be the set of categorical features of the dataset, and $Cardinality_K$ the sum of the cardinality of all $k \in K$. Then, this aggregation is defined as $\frac{Cardinality_K}{|K|+1}$, which is simply the "*mean*" of cardinality in the dataset. The +1 is to avoid error with datasets which do not have categorical features;
- **`categorical_ratio`**: Proportion of categorical features in the dataset;
- **`numeric_ratio`**: Proportion of numeric features;
- **`boolean_ratio`**: Proportion of Boolean features;
- **`constant_ratio`**: Proportion of constant features.

After this transformation, each dataset D_i is now represented by a $\mathcal{P}_i \in \mathbb{R}^{10}$ where each component of the multidimensional point represents one of the aggregated characteristics calculated above. To visualize these multidimensional points, a t-SNE projection (dimensionality reduction technique, more details in Maaten and Hinton [2008]) on two dimensions is shown in Figure 5.9.

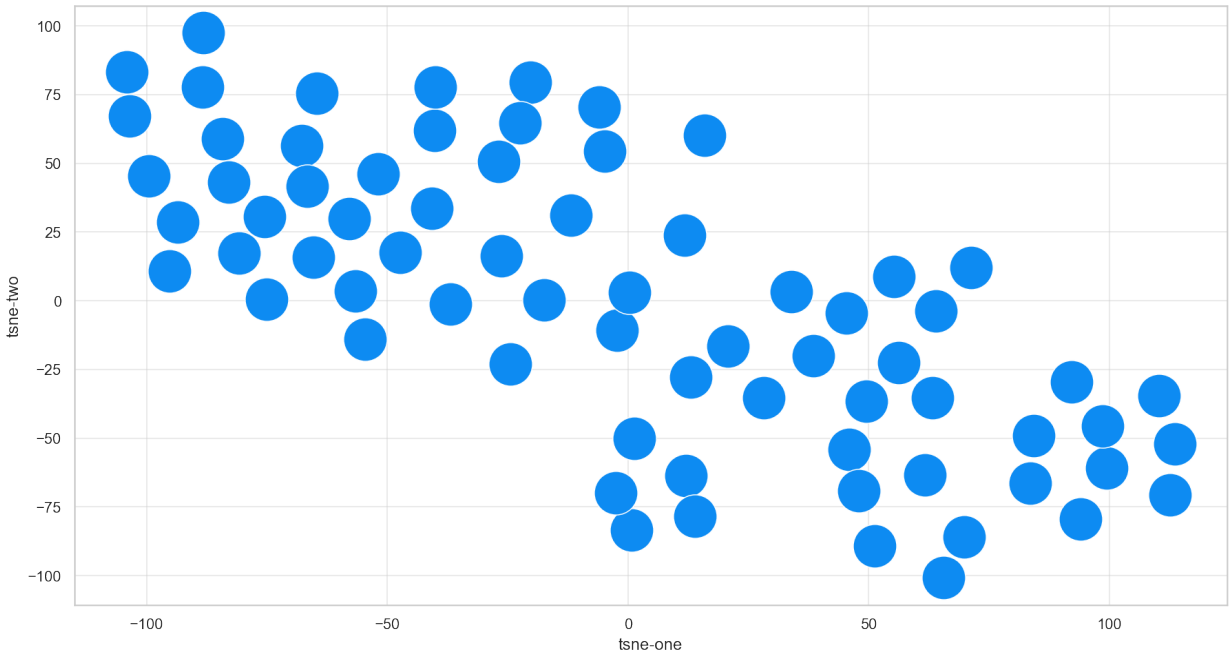


Figure 5.9: *t-SNE projection of all $\mathcal{P}_i$*

The aggregated characteristics were designed in a way to encompass both the proportion of feature types in the dataset and the absolute size of the dataset. The `*ratio` aggregations are done to compare multiple datasets regardless of the absolute numeric value of features it has but on the ratio of the feature types. On the other hand, the absolute features like `num_rows` captures the absolute size of the data. For example, a simple analysis using `num_rows`, `num_features` and `categorical_ratio` could yield interesting insights and a way to bundle similar datasets together based on the proportion of categorical features, controlling for both the number of data points and the number of features.

Finally, there is also an explicit emphasis on characteristics regarding numeric and categorical features, which are of interest in this study (more information about the premisses about it on Section 4.3 of Chapter 4). The mean of skewness aggregation can be used to control (in the experimental design sense) for datasets with a similar distribution of skewed or not numerical features, while the mean of variance, the number of categorical features and the "*mean*" of cardinality describes important information regarding the categorical aspect of the dataset.

5.2.2 Clustering Strategy

To analyze all experiments based on similar dataset characteristics, the decided approach was to perform a simple clustering technique, instead of manually selecting which datasets are similar by looking at the aggregated characteristics by hand. Using the aggregated characteristics explained in the last section, a clustering approach on the points $\mathcal{P}_i$ generates clusters based on those characteristics.

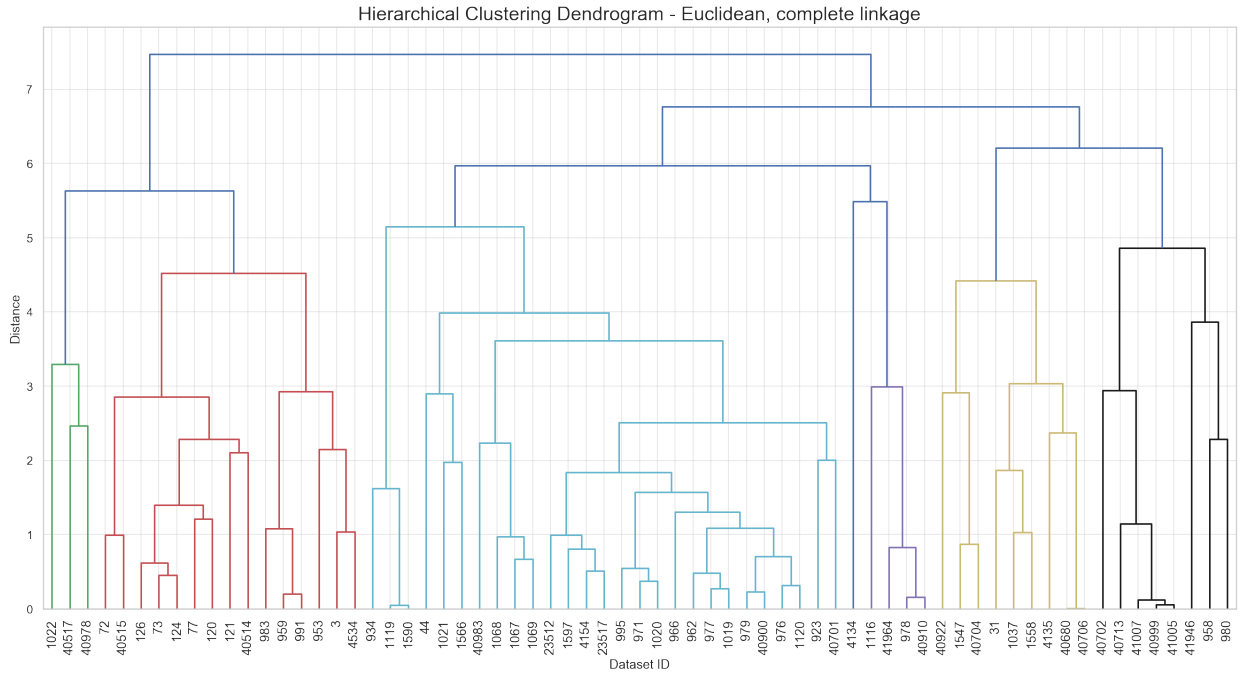


Figure 5.10: *Dendrogram of the old clustering method*

After some initial experimentation, the chosen clustering approach was to use the well known K-means algorithm, with $k = 6$ clusters. K-means is a very simple algorithm that reduces the data to *centroids*, and resulted in reasonable clusters for this study. The other tested technique was to use agglomerative hierarchical clustering and choose the number of clusters by analyzing the resulting dendrogram (the dendrogram for the hierarchical clustering is shown in Figure 5.10). However, this technique generated an uneven number of datasets in each cluster, more than K-means, so the K-means was chosen. It is important to note that the algorithms and the number of clusters chosen here can highly impact the subsequent analysis of the experiment results. The objective is not to create a generic analysis that could work with any possible dataset, but to actually measure the impact of hyperparameters into specific clusters which follow a similar distribution of the aggregated characteristics.

After the K-means procedure, the assigned clusters can be seen in the t-SNE projection in Figure 5.11, where each color represents a different cluster. Even though the number of $\mathcal{P}_i$ in each cluster is not uniform, the clustering assignment is more intuitive, albeit in reduced dimensionality. As an observation note, the two t-SNE projections (Figure 5.9 and 5.11) are different because the embeddings in low dimension change when new data is introduced. In this case, the centroids were also added to the dataset before using t-SNE.

To understand better the behavior and general characteristics of each dataset of clusters, a superficial manual analysis of the datasets in each cluster is performed. As an example, the names of the datasets from Cluster 2 and 3 are described in Table 5.1. From the superficial analysis of the content of each dataset, one can see that most of the datasets in Cluster 2 have a high number of categorical features, while datasets in Cluster 3 are all synthetically augmented, i.e. Bayesian Network Generated (*BNG*) and have a very high number of rows.

Using ridgeline plots is another way to understand the meaning of each cluster. By calculating the distribution of the aggregated characteristics over each cluster and plotting them on the same

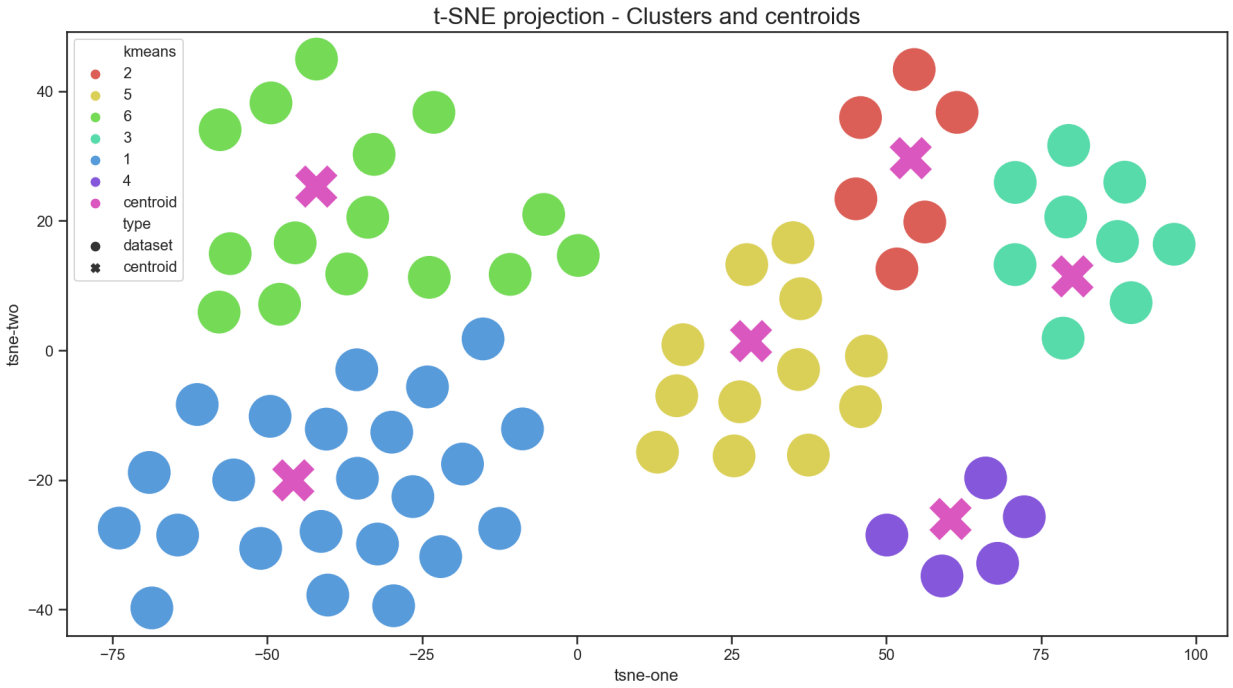


Figure 5.11: *t-SNE projection with the assigned clusters and the centroids*

Cluster 2	Cluster 3
kr-vs-kp	*BNG(kr-vs-kp)*
splice	*BNG(labor,nominal,1000000)*
mfeat-pixel	*BNG(breast-cancer,nominal,1000000)*
PhishingWebsites	*BNG(mushroom)*
20_newsgroups.drift	*BNG(colic.ORIG,nominal,1000000)*
Internet-Advertisements	*BNG(credit-a,nominal,1000000)*
	BNG(credit-g,nominal,1000000)
	BNG(credit-g)

Table 5.1: *Datasets from Cluster 2 and Cluster 3*

scale against each other, it is possible to evaluate if the clustering makes sense. In total there are 10 ridgeline plots, one for each aggregated characteristic. Here two of them are illustrated: In Figure 5.12 and Figure 5.13 the ridgeline plots for `categorical_ratio` and `mean_skewness` can be seen. It is easy to see that Cluster 2 datasets have a very high categorical ratio with very little variance, i.e. almost all of the datasets in this cluster contains a high proportion of categorical features. Clusters 1 and 6 have the lowest proportion of categorical features amongst all Clusters. Another conclusion is that Cluster 6 contains more skewed features, as its distribution is skewed towards high `mean_skewness` values.

All the remaining ridgeline plots are available on the appendix A.

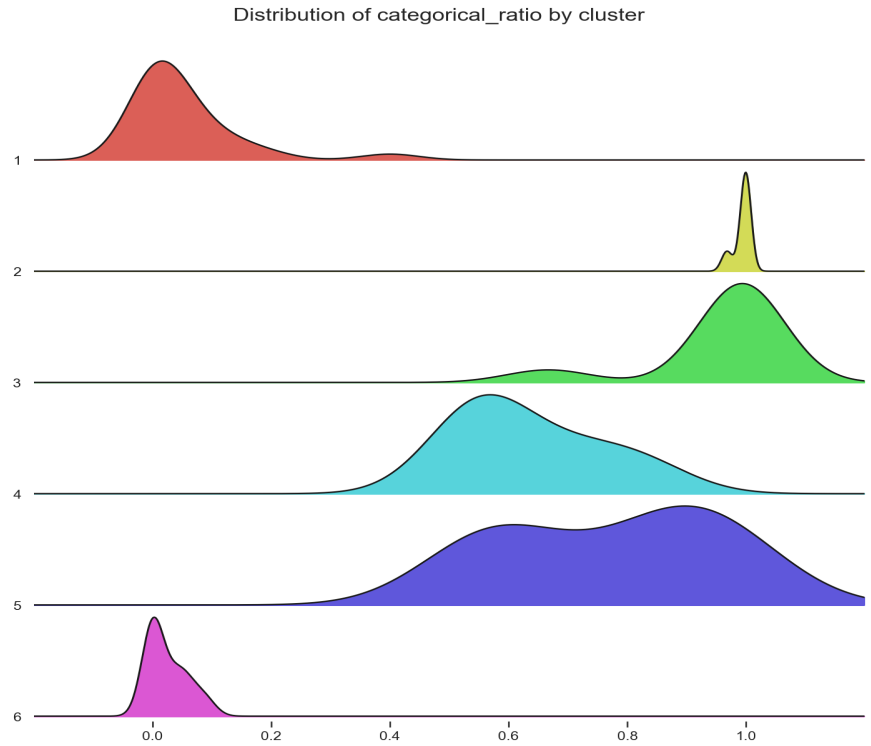


Figure 5.12: *Ridgeline plot of* `categorical_ratio`*. Clusters are represented by different colors*

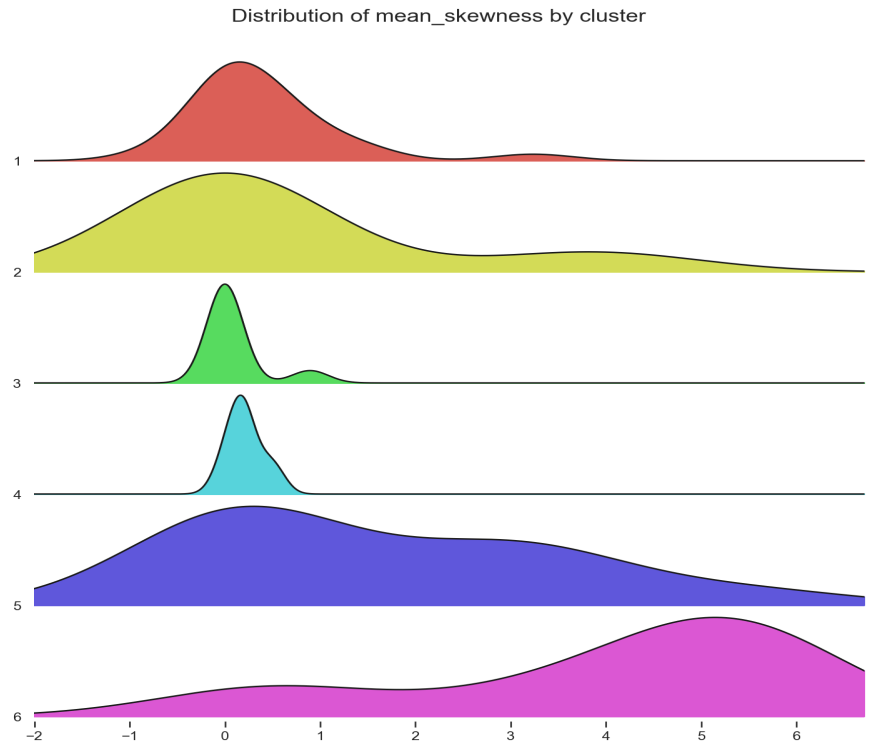


Figure 5.13: *Ridgeline plot of* `mean_skewness`*. Clusters are represented by different colors*

5.3 Experiment Definitions

The experimental analysis is performed on each cluster, therefore a common metric must be used when comparing multiple dataset's results. In fact, one of the objectives is to analyze the

impact of changing selected hyperparameters on the specified performance metrics of classification tasks. For this part of the analysis, instead of looking directly into the performance metrics of AUC, Logloss and Brier Score, a *difference* from the baseline metric δ_{metric} is defined.

5.3.1 δ_{metric} Definition

The reason to not directly use the performance metrics when comparing different models under the same hyperparameters changes is that the objective of the study does not takes into account the actual performance of a given dataset, but actually how much each hyperparameter affects the performance metrics, i.e. the **sensitivity** of a dataset w.r.t a hyperparameter. For example, it does not matter in this case if dataset A has a higher AUC than dataset B under a new hyperparameter value h, but actually if the increase or decrease in performance of both datasets A and B are significant for hyperparameter value h.

Another problem that can be avoided by the following definition of a δ_{metric} is the fact that it captures relative changes, instead of the absolute value of the metric; e.g. a dataset with almost constant 0.9 AUC value will have a δ_{AUC} equal to 0 most of the time, and will be treated as equal as a dataset with constant low AUC (e.g. 0.55): the actual absolute value of the AUC does not matter.

LightGBM has a default set of values for its hyperparameters and the ones analyzed in this study are defined below. As an observation, the correct default value for `max_depth` is -1 because it is unlimited on LightGBM, as mentioned in Section 3.6; However, because the `num_leaves` default is 31, this approximately results in a balanced tree with depth 5.

- LR = {0.1}
- NE = {100}
- MD = {5}

The baseline metrics are defined, for each dataset D_i, as the performance metrics (*AUC, Logloss* and *Brier Score* as explained in Section 2.4) obtained on the **test set** using the LightGBM model trained with the parameters as close as possible to the default hyperparameter values above. The δ_{metric} is then just the difference between a given new metric and the baseline metric. Its interpretation depends on the performance metric being analyzed: in the case of AUC, a positive δ_{AUC} means a higher AUC than baseline (a better result), a negative one means a lower AUC than baseline (worse result), and 0 an equal performance in terms of AUC. For both the Brier Score and Logloss the interpretation is the opposite, as in these metrics the lower the better.

Some of the datasets in the clusters, when analyzed through the δ_{metric} plots, had very erratic behaviors when compared to the majority of the other datasets in the cluster. In the following analysis, some of these datasets were removed from the clusters to avoid an overemphasizing of the outlier behavior.

In Figure 5.14 is displayed the δ_{AUC} w.r.t `max_depth` changes for the datasets in Cluster 3, i.e. one example of a δ_{metric} plot from the experiment result. Each point represents a value of δ_{metric} for a single dataset, and the colors represent different datasets. The δ_{metric} captures the relative increase or decrease in AUC considering 5 the baseline for `max_depth`. By manually analyzing

the results of the δAUC one can determine, for this Cluster, which values of `max_depth` have the highest impact on the AUC metric, both positive and negative.

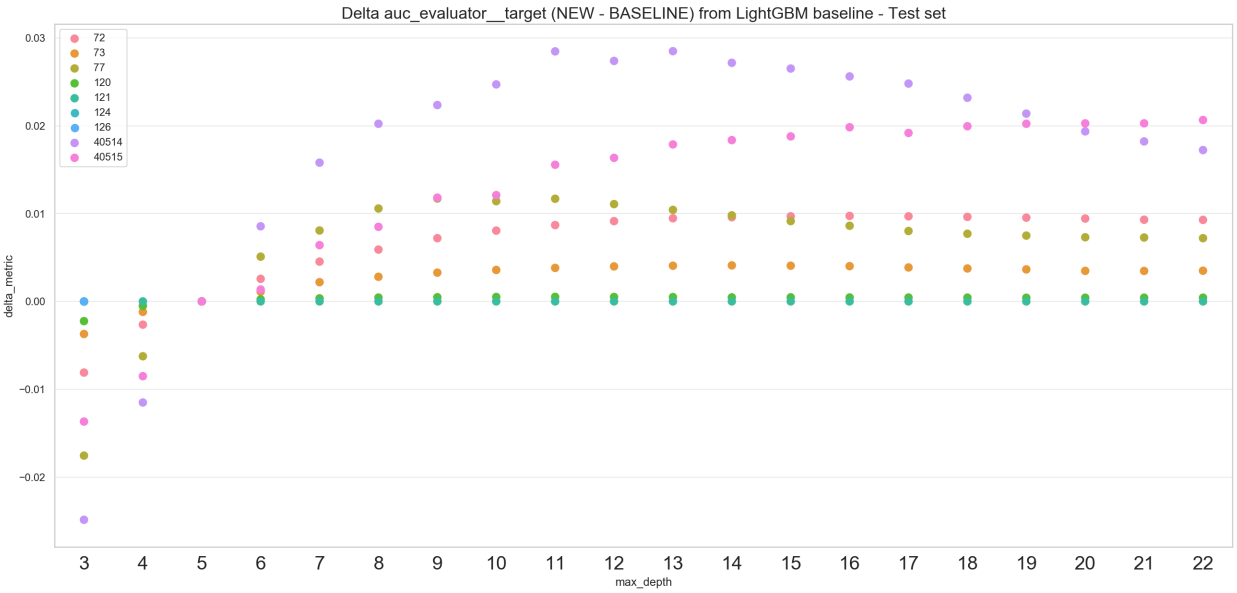


Figure 5.14: δ_{AUC} *for Cluster 3, changing only* `max_depth`*. The colors represent different datasets*

5.3.2 Ordering and Visualization

In Figure 5.15 a different δ_{metric} plot is displayed, where each point is the δ_{brier} w.r.t both `max_depth` and `learning_rate` changes summarized into a single dimension in the *x-axis*, for Cluster 2 experiments. The higher the number of hyperparameters being analyzed at the same time, the more difficult it is to visualize these δ_{metric} plots due to the high dimensionality. For this reason, in the study the hyperparameter values were summarized into a single point that encompasses multiple hyperparameter changes.

In the Subsections 5.1.1, 5.1.2 and 5.1.3 the full results for individual hyperparameter impact, pairs of hyperparameters and finally all three hyperparameters, respectively, are shown. These plots are not possible in this part of the analysis because instead of one individual dataset experiment result being evaluated as in Section 5.1, the final analysis is done with multiple datasets of a cluster.

For visualization purposes, an ordering of the data points in the x-axis was defined to facilitate the visualization and to maintain an order that makes sense, even when reducing the dimensionality. First, the ranking of each hyperparameter is calculated based on the sorted order within that individual hyperparameter distribution, and then the final ranking of these points is based on the average of the individual hyperparameter ranks. This custom ranking has two advantages:

1. Retains the relative value of each hyperparameter into the final ordering. This means that the first points represent the lower values of all hyperparameters and the last points in the x-axis represent the combinations with the highest hyperparameters values;

2. Is robust w.r.t the scale of the absolute values between the hyperparameters. The ordering takes first into account the rank of each hyperparameter to avoid dealing with values in uneven scales. For example, a learning rate of 1.5 is considered to be a high value in practice, but its

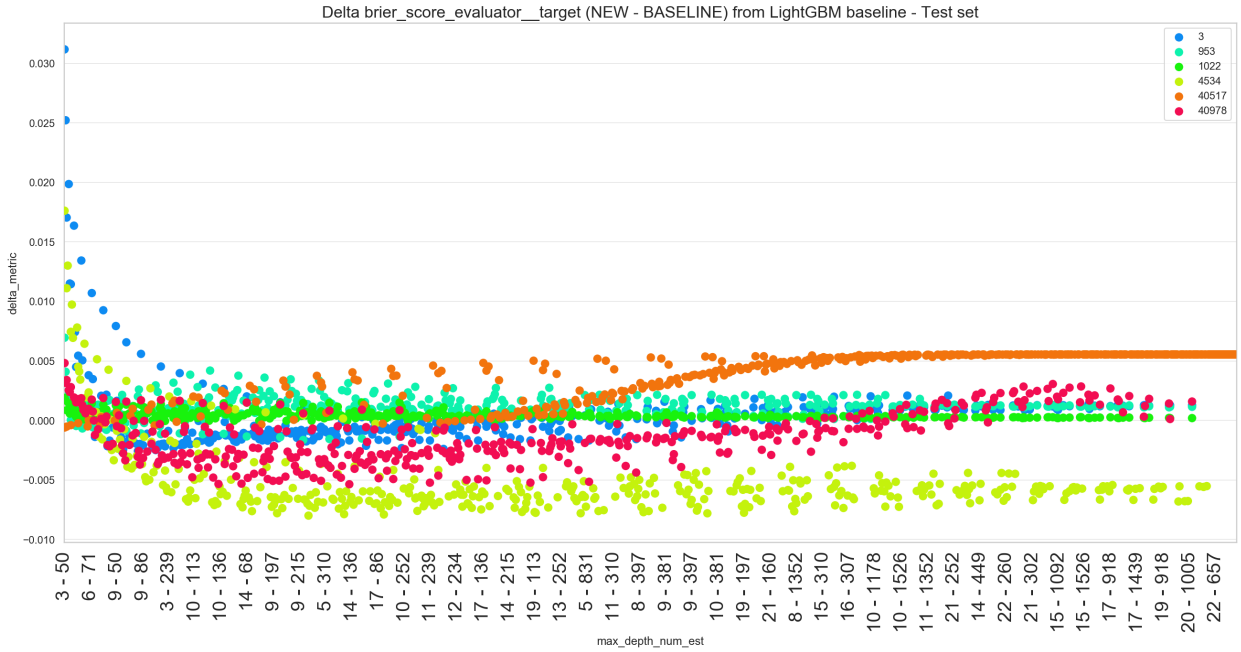


Figure 5.15: δ_{Brier} *for Cluster 2, changing both* `max_depth` *and* `learning_rate`*. The colors represent different datasets*

numeric value is much smaller than a maximum depth of 5, which can be considered a small value of `max_depth`.

As an example of this ordering, if the points were $H = \{(3, 500), (20, 100), (5, 200), (10, 400)\}$, each tuple a pair of maximum depth and number of estimators, the first step of the ordering is to get the ranking of each hyperparameter within its distribution. This would generate a set of the individual ranks $r'(H) = \{(1, 4), (4, 1), (2, 2), (3, 3)\}$, and the final ranking order depends on the average ranks of each tuple, in this case $r_{avg}(H) = \{2.5, 2.5, 2, 3\}$. And using $r_{avg}(H)$ to order the original points H, the final ordering is $H^{\star} = \{(5, 200), (3, 500), (20, 100), (10, 400)\}$.

It is important to note that, because of the statistical framework chosen in the study, the ranking strategy does not change the result of the analysis, but provides a way to visually understand its results.

5.4 Statistical Framework

To generalize and be more certain about the hyperparameter sensitivity analysis, a statistical framework is defined to analyze the δ_{metric} obtained on each cluster and different configurations of hyperparameters. The comparisons include changing different aspects of the experiment, such as the performance metric chosen (AUC, Logloss, Brier Score), the hyperparameter(s) being changed and the clusters being analyzed.

The statistical framework is divided into two main parts: tests for the significance of the changes in the δ_{metric} obtained in the experiment, and the subsequent modeling of the sensitivity factors of the hyperparameters of each cluster. An overview of the statistical tools used is defined in the following subsections, along with examples of real results obtained in the analysis of the experiments.

5.4.1 Experimental Scenarios

An important concept in this study related to the statistical framework is the definition of an **experimental scenario**. In this study the experimental units are the datasets of a cluster, which receive a "treatment" of different hyperparameter values. First, C_k is the set of experimental units (i.e. the datasets) of the kth cluster. The hyperparameters of a scenario are a partition Q of the union of all hyperparameter spaces (as defined in Subsection 4.4.1) of the kth cluster ($\eta^{(k)}$, defined in Equation 5.1) and is referred to as $\eta_Q^{(k)}$. For example, the hyperparameters values of the experiment when changing both maximum depth and learning rate in Cluster 4 can be written as $\eta^{(4)}_{max\_depth,learning\_rate}$.

$$\eta^{(k)} = \bigcup_{d \in C_k} Hspace_d \tag{5.1}$$

Given the C_k experimental units, the hyperparameters configuration $\eta_Q^{(k)}$ and a performance metric m, the experimental scenario can be represented as is Equation 5.2:

$$\mathcal{S}(C_k, \eta_Q^{(k)}, m) \tag{5.2}$$

Which means the experiment outputs of δ_m observed when applying the *treatments* $\eta_Q^{(k)}$ to all experimental units in the kth cluster. For example, Figure 5.14 represents the experimental scenario $\mathcal{S}(C_3, \eta^{(3)}_{max\_depth}, AUC)$ and Figure 5.15 is a visualization of $\mathcal{S}(C_2, \eta^{(2)}_{max\_depth,num\_estimators}, brier)$.

5.4.2 Analysis of Variance

The typical scenario of an analysis of variance, according to Montgomery [2017], is an experimental setting where there are k treatments or different *levels* of a factor that one wishes to compare. The response observed from each of the k levels of treatment is a random variable. In general, there are n observations under the same treatment, which can be seen in Table 5.2. In this study, the *treatment* is the hyperparameter, and the different values of the hyperparameter are the different treatment levels in the analysis of variance.

Treatment	**Observations**				**Totals**	**Averages**
1	y_{11}	y_{12}	$\cdots$	y_{1n}	$\boldsymbol{y}^{(1)}$	$\overline{\boldsymbol{y}^{(1)}}$
2	y_{21}	y_{22}	$\cdots$	y_{2n}	$\boldsymbol{y}^{(2)}$	$\overline{\boldsymbol{y}^{(2)}}$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$
k	y_{k1}	y_{k2}	$\cdots$	y_{kn}	$\boldsymbol{y}^{(k)}$	$\overline{\boldsymbol{y}^{(k)}}$
					y_Σ	$\overline{\mathrm{y}_\Sigma}$

Table 5.2: *Table exemplifying a general Single-Factor Experiment, based on Montgomery [2017]*

There are two main models in the literature when dealing with Single-Factor experiments, the **means model** and the **effects model**. Even though in this study the observations from the experiments are described using the *effects model*, it is useful to comprehend the idea behind both models to understand why one was chosen over the other.

In the *means model* each observation (i.e. each point in the δ_{metric} plot for example) is defined as a function of the mean *within treatment* and a random error component that incorporates "all other sources of variability in the experiment including measurement, variability arising from uncontrolled factors, differences between the experimental units to which the treatments are applied, and the general background noise in the process" according to Montgomery [2017]. The structure from the means model is described in Equation 5.3.

$$y_{ij} = \mu_i + \epsilon_{ij} \begin{cases} i = 1, 2, \cdots, k \\ j = 1, 2, \cdots, n \end{cases} \tag{5.3}$$

On the other hand, the **effects model** defines the treatment means μ_i as a deviation from the overall mean $\boldsymbol{\mu}$ of the experiment, i.e. $\mu_i = \boldsymbol{\mu} + \tau_i$ where $i = 1, 2, \cdots, k$. Thus, the experimental results are described according to Equation 5.4. The parameter τ_i is called the $\boldsymbol{i}$**th treatment effect**, and it represents the deviation from the overall mean of the experiment.

For simplicity, all the scenarios were structured using a single-factor model, even the scenarios where the treatment is actually two or more values of different hyperparameters being changed. This means that the results and conclusions from pairs or triples of hyperparameters are interpreted as a single factor that can change the δ_{metric} of the gradient boosting models, instead of decomposing the effects into multiple variables.

$$y_{ij} = \boldsymbol{\mu} + \tau_i + \epsilon_{ij} \begin{cases} i = 1, 2, \cdots, k \\ j = 1, 2, \cdots, n \end{cases} \tag{5.4}$$

One of the reasons why the effects model was chosen to analyze the results of this study is because it has a more intuitive interpretation for the objective of this study. One can interpret the τ_i as the amount of effect a configuration of hyperparameter produces in the performance metrics, i.e. It is a measure of *sensitivity* from the overall case. Before modeling the experimental results and comparing the τ_i for different scenarios, it is important to test for equality of the output of the treatments (in this case, equality of δ_{metric} for different performance metrics) and testing the assumptions of the statistical tests.

5.4.3 ANOVA

The most widely used test in the analysis of variance is the **ANOVA** test or *single-factor analysis of variance*. In the assumptions of the ANOVA model, there is a requirement that the experiments are performed in random order to keep the environment in which the experiments are applied as uniform as possible. According to Montgomery [2017], the objective is to test the appropriate hypothesis about the treatment means and to estimate them. However, there a very strong assumption that must be validated before applying the ANOVA test: the model errors ϵ_{ij} are assumed to be normally and independently distributed random variables with mean zero and variance σ^2; The data must also be *homoscedastic*, i.e. the variance σ^2 is assumed to be constant for all levels of the factor.

These assumptions imply that the observations must follow a normal distribution:

$$y_{ij} \sim \mathcal{N}(\boldsymbol{\mu} + \tau_i, \sigma^2)$$

The ANOVA model has different flavors, and is highly used in the field of applied statistics. Different variations and extensions of the ANOVA model can be found throughout the literature of statistics and machine learning. In the study of hyperparameters, Hoos et al. [2014] describe a method using the functional ANOVA framework (a variation of ANOVA) to quantify the importance of "both single hyperparameters and of interactions between hyperparameters" in a single dataset. In more recent work, van Rijn and Hutter [2017] also extend the functional ANOVA framework for multiple datasets, with an empirical experiment using OpenML datasets.

5.4.4 Testing of Assumptions

However, the ANOVA test **was not** used in this study, because its assumptions were not met in almost all scenarios. A different nonparametric statistical test was used, and since some of the assumptions apply for both the parametric (ANOVA) and nonparametric tests, it is important to understand the assumptions of them. Using Montgomery [2017] definition of the **residuals** for observation j in treatment i as

$$\mathcal{E}_{ij} = y_{ij} - \hat{y}_{ij} \tag{5.5}$$

where $\hat{y}_{ij}$ is an estimate of the respective observation y_{ij}, obtained as follows:

$$\begin{aligned} \hat{y}_{ij} &= \hat{\boldsymbol{\mu}} + \hat{\tau}_i \\ &= \overline{\mathrm{y}_{\Sigma}} + (\overline{\boldsymbol{y}^{(i)}} - \overline{\mathrm{y}_{\Sigma}}) \\ &= \overline{\boldsymbol{y}^{(i)}} \end{aligned} \tag{5.6}$$

This just means that the estimate of any experimental scenario observation in the ith treatment is the corresponding treatment average $\overline{\boldsymbol{y}^{(i)}}$. In the terms of this study, the estimate of relative change in the performance metric δ_{metric} for a given configuration of a value of hyperparameters is the mean of the δ_{metric}'s of all datasets in the corresponding cluster being analyzed. The analysis of the residuals is one of the ways to check for model adequacy, and can be analyzed using a *Normal Q-Q Plot* and using statistical tests to test for the normality and homoscedasticity assumptions.

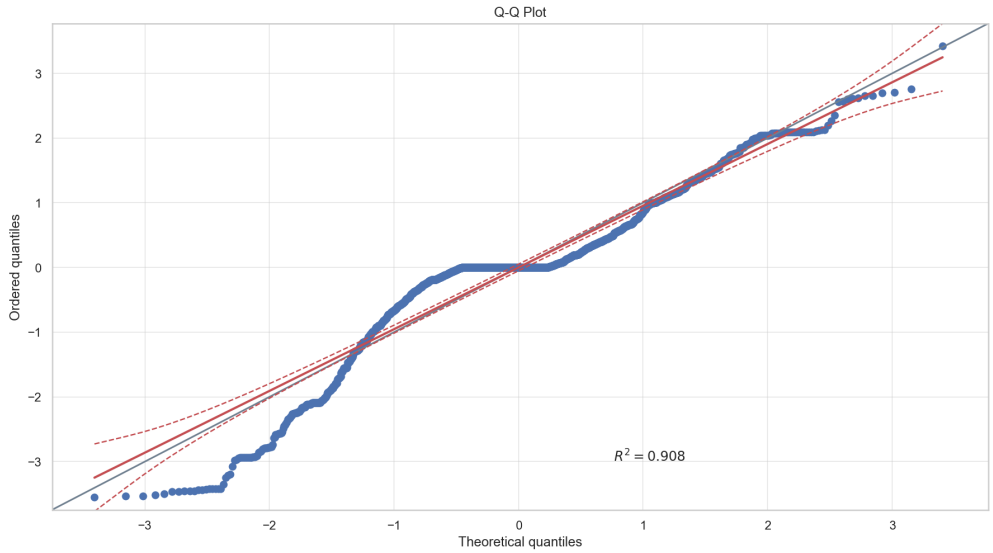
The test of assumptions is applied to every **experimental scenario** $\mathcal{S}$ in this study, in the following order:

1. Fit a single-factor model (SFM) on $\mathcal{S}$, and calculate the residuals according to Equation 5.5. In Table 5.3 the complete SFM model output for $\mathcal{S}(C_1, \eta^{(1)}_{max\_depth}, Brier)$ is illustrated.

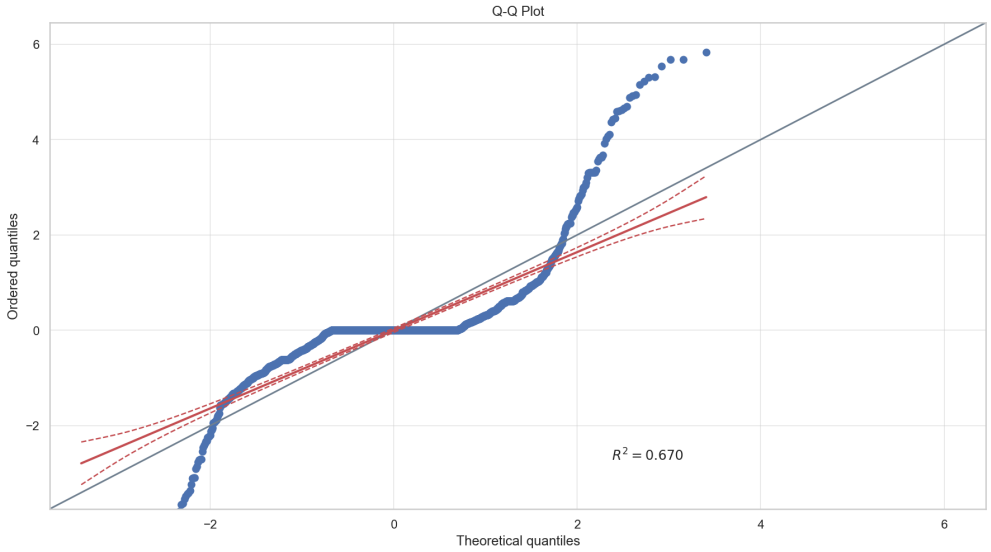
max_depth	$\overline{\boldsymbol{y}^{(i)}}$	$\hat{\boldsymbol{\mu}}$	$\hat{\tau}_i$
3	0.001872	-0.00106	0.002932
4	0.001275	-0.00106	0.002335
5	0.000000	-0.00106	0.001060
6	-0.000835	-0.00106	0.000225
7	-0.001350	-0.00106	-0.000290
8	-0.001588	-0.00106	-0.000528
9	-0.001548	-0.00106	-0.000488
10	-0.001575	-0.00106	-0.000515
11	-0.001558	-0.00106	-0.000497
12	-0.001646	-0.00106	-0.000586
13	-0.001451	-0.00106	-0.000391
14	-0.001456	-0.00106	-0.000396
15	-0.001413	-0.00106	-0.000352
16	-0.001397	-0.00106	-0.000337
17	-0.001391	-0.00106	-0.000331
18	-0.001425	-0.00106	-0.000365
19	-0.001395	-0.00106	-0.000335
20	-0.001383	-0.00106	-0.000323
21	-0.001439	-0.00106	-0.000379
22	-0.001501	-0.00106	-0.000441

Table 5.3: *SFM model for scenario* $\mathcal{S}(C_1, \eta^{(1)}_{max\_depth}, Brier)$

2. Plot a Normal Q-Q Plot to visually observe the structure of the SFM residuals. In this study, the Normal Q-Q Plot will match the SFM residuals quantiles with the theoretical quantiles of a normal distribution, while also fitting a Linear Regression model and calculating the coefficient of determination (R^2), as suggested in Vallat [2018]. Residuals are filtered to be inside the quantiles q_5 and q_{95} to remove outliers. To showcase this, in Figure 5.16a the Normal Q-Q plot for the scenario $\mathcal{S}(C_1, \eta^{(1)}_{max\_depth,learning\_rate}, AUC)$ represents a distribution of residuals close to a normal distribution, with a relatively high R^2. This is a specific case where the ANOVA test could be used because of its robustness to the normality assumption, as stated in Montgomery [2017] "In general, moderate departures from normality are of little concern in the fixed effects analysis of variance". A more drastic departure from normality can be observed in scenario $\mathcal{S}(C_1, \eta^{(1)}_{num\_estimators,learning\_rate}, Logloss)$, in Figure 5.16b.



(a) *Normal Q-Q Plot for scenario* $\mathcal{S}(C_1, \eta^{(1)}_{max\_depth,learning\_rate}, AUC)$



(b) *Normal Q-Q Plot for scenario* $\mathcal{S}(C_1, \eta^{(1)}_{num\_estimators,learning\_rate}, Logloss)$

Figure 5.16: *Two Normal Q-Q Plots from Cluster 1 results*

3. Perform a **Shapiro–Wilk test** for the normality assumption with α-level $= 0.05$. In almost all of the experimental scenarios the test rejected the null hypothesis that the residuals are normally distributed with p-value less than 0.05.

4. To validate the homoscedasticity assumption, a statistical test for equality of variance is also performed on the residuals. As indicated in Montgomery [2017], these statistical procedures are testing the following hypotheses:

$$H_0 : \sigma_1^2 = \sigma_2^2 = \cdots = \sigma_k^2$$
$$H_1 : \text{above not true for at least one } \sigma_i^2$$

A widely used procedure for homoscedasticity is the Bartlett's Test, which computes a statistic whose sampling distribution is, according to Montgomery [2017]: "closely approximated by the chi-square distribution with $k-1$ degrees of freedom when the k random samples are from

independent normal populations.". By analyzing the results of the Normal Q-Q plots and the Shapiro–Wilk tests there is no statistical evidence that the residuals of this study follow a normal distribution, so the Bartlett's test should not be used. The **Levene test** is an alternate procedure that is robust to departures from normality, as explained in Brown and Forsythe [1974], and it is the test chosen to verify the homoscedasticity for all experimental scenarios.

5.4.5 Kruskal–Wallis: A Nonparametric Approach

Given that most of the experimental results failed the normality assumption tests, the Kruskal–Wallis test is used to test for the statistical significance of the treatment means. The Kruskal–Wallis test is applied to all experimental scenarios and is used to test the null hypothesis that the k treatments are identical against the alternative hypothesis that some of the treatments generate observations that are larger than others. There is a convenient interpretation of the Kruskal–Wallis, described in Montgomery [2017]:

> "Because the procedure is designed to be sensitive for testing differences in means, it is sometimes convenient to think of the Kruskal–Wallis test as a test for equality of treatment means. The Kruskal–Wallis test is a **nonparametric alternative** to the usual analysis of variance."

Kruskal–Wallis works by applying the usual analysis of variance to the ranks of the observations. The *rank transformation* consists of ranking the observations y_{ij} in ascending order and replacing each observation by its rank r_{ij}. The results of testing after a ranking transformation are preferred because it is less likely to be distorted by nonnormality and unusual observations, which is the case of the results of the experimental scenarios in the study.

Even though this nonparametric approach does not require the data to be normally distributed, it does, however, assume the same distribution within groups, as explained in McDonald [2009]:

> "(Kruskal–Wallis) does assume that the different groups have the same distribution, and groups with different standard deviations have different distributions. If your data are heteroscedastic, Kruskal–Wallis is no better than one-way ANOVA, and may be worse."

So the homoscedasticity tests are still needed for the assumptions of the Kruskal–Wallis test. In Table 5.4 the statistical results for all the scenarios in Cluster 2 is available, where NE, MD and LR refer to `num_estimators`, `max_depth` and `learning_rate` respectively. The numbers in the table are the p-values of all statistical tests performed in Cluster 2 experiments, and the values in red are the tests that failed the desired hypothesis testing. Some tests can have NaN if there are not enough samples or all values are equal, in the case of the residuals.

Treatment	Metric	Shapiro–Wilk	Levene	Kruskal–Wallis
$\eta^{(2)}_{NE}$	δ_{AUC}	4.059920e-08	9.968117e-01	9.996506e-01
	δ_{Brier}	5.942564e-06	2.105453e-01	9.965572e-01
	$\delta_{Logloss}$	2.502680e-05	9.636175e-03	2.827828e-03
$\eta^{(2)}_{MD}$	δ_{AUC}	2.536182e-10	9.210022e-01	1.860650e-02
	δ_{Brier}	3.926061e-06	2.774635e-01	2.283747e-02
	$\delta_{Logloss}$	2.522154e-05	1.285176e-01	1.764531e-02
$\eta^{(2)}_{LR}$	δ_{AUC}	2.714980e-09	NaN	NaN
	δ_{Brier}	9.808685e-09	NaN	NaN
	$\delta_{Logloss}$	1.213540e-10	NaN	NaN
$\eta^{(2)}_{MD,LR}$	δ_{AUC}	1.406702e-20	9.999939e-01	1.316980e-02
	δ_{Brier}	1.339994e-23	1.000000e+00	6.235464e-12
	$\delta_{Logloss}$	1.334960e-23	1.000000e+00	1.005025e-08
$\eta^{(2)}_{MD,NE}$	δ_{AUC}	2.548462e-22	1.000000e+00	1.000000e+00
	δ_{Brier}	6.494320e-28	9.969882e-01	9.274816e-01
	$\delta_{Logloss}$	3.974093e-21	2.655356e-23	6.191510e-22
$\eta^{(2)}_{LR,NE}$	δ_{AUC}	9.960932e-27	1.000000e+00	3.306332e-06
	δ_{Brier}	2.709085e-26	9.999995e-01	7.035798e-09
	$\delta_{Logloss}$	2.452305e-24	9.999999e-01	1.650304e-05
$\eta^{(2)}_{NE,MD,LR}$	δ_{AUC}	0.000000e+00	1.000000e+00	2.904172e-64
	δ_{Brier}	0.000000e+00	1.000000e+00	4.013155e-111
	$\delta_{Logloss}$	0.000000e+00	1.000000e+00	7.298816e-70

Table 5.4: *Statistical p-values of the experimental scenarios in Cluster 2. All the normality assumptions were rejected in this case*

Chapter 6

Results and Discussion

All the statistical tests shown in the last chapter were applied to all experimental scenarios and then filtered by their significance. In very few scenarios of the study, it was possible to conclude that the residuals followed a normal distribution according to the Shapiro-Wilk test. Even though in these specific instances the typical ANOVA test could be used (given they are also homoscedastic), it was decided to use only the Kruskal–Wallis test so every experimental scenario uses the same statistical framework.

6.1 Statistical Significance

Scenarios with a p-value above the 0.05 significance level in the Levene test means it failed to reject the hypothesis that the group variances are different: in practice, the homoscedasticity assumption is considered to be true in these cases. In the Kruskal–Wallis test if the p-value is less than the 0.05 the null hypothesis of equal treatment means is rejected, i.e. in practice, it is considered at least one treatment mean is different from the other. When an experimental scenario has enough samples (i.e. It won't return NaN), passed the homoscedasticity assumption and reject the equality of treatment means in the Kruskal–Wallis test it is indicated here as ✓. If either the homoscedasticity test failed or Kruskal–Wallis failed to reject the hypothesis of equal means, it is indicated as ×. All the results of the statistical significance of treatment means are summarized in Table 6.1.

An important note about homoscedasticity in these results is that the results can be analyzed even though failing the homoscedasticity assumption. For instance, the treatment means can be individually analyzed taking into the context the experimental scenario it belongs to, and insights can be obtained from it. Even though in this case the Kruskal–Wallis cannot be used, observed behaviors that do not require directly statistical significance are reported if deemed insightful.

Cluster	Metric	η_{NE}	η_{MD}	η_{LR}	$\eta_{MD,LR}$	$\eta_{MD,NE}$	$\eta_{LR,NE}$	$\eta_{NE,MD,LR}$
1	δ_{AUC}	×	×	✓	✓	×	×	✓
	δ_{Brier}	×	✓	×	✓	×	✓	✓
	$\delta_{Logloss}$	✓	✓	×	×	✓	✓	✓
2	δ_{AUC}	×	✓	×	✓	×	✓	✓
	δ_{Brier}	×	✓	×	✓	×	✓	✓
	$\delta_{Logloss}$	✓	✓	×	✓	✓	✓	✓
3	δ_{AUC}	×	×	✓	✓	×	×	×
	δ_{Brier}	×	×	×	✓	×	×	×
	$\delta_{Logloss}$	×	×	×	×	×	×	×
4	δ_{AUC}	×	×	×	×	×	×	×
	δ_{Brier}	×	×	×	✓	×	✓	✓
	$\delta_{Logloss}$	×	×	×	✓	×	✓	✓
5	δ_{AUC}	×	×	✓	✓	×	✓	✓
	δ_{Brier}	×	✓	✓	✓	×	✓	✓
	$\delta_{Logloss}$	×	✓	✓	✓	×	✓	✓
6	δ_{AUC}	✓	×	×	✓	×	×	✓
	δ_{Brier}	×	×	×	✓	×	×	×
	$\delta_{Logloss}$	×	×	×	✓	×	×	×

Table 6.1: *All experimental scenarios and their respective Kruskal–Wallis results for equality of treatment means*

Some interesting insights can be drawn by inspecting the statistical significance results in Table 6.1 and the specific p-values for the tests in the tables from the Appendix B:

1. For instance, Cluster 3 scenarios failed most of the homoscedasticity tests, but by visually analyzing the δ_{metric} plot it displays the same behavior of the treatments as the other clusters. However, Cluster 3 datasets are all synthetically generated (as illustrated in Table 5.1), and this can be a reason why it has so few statistically significant results. In Figure 6.1 one δ_{AUC} plot from Cluster 3 results is illustrated.

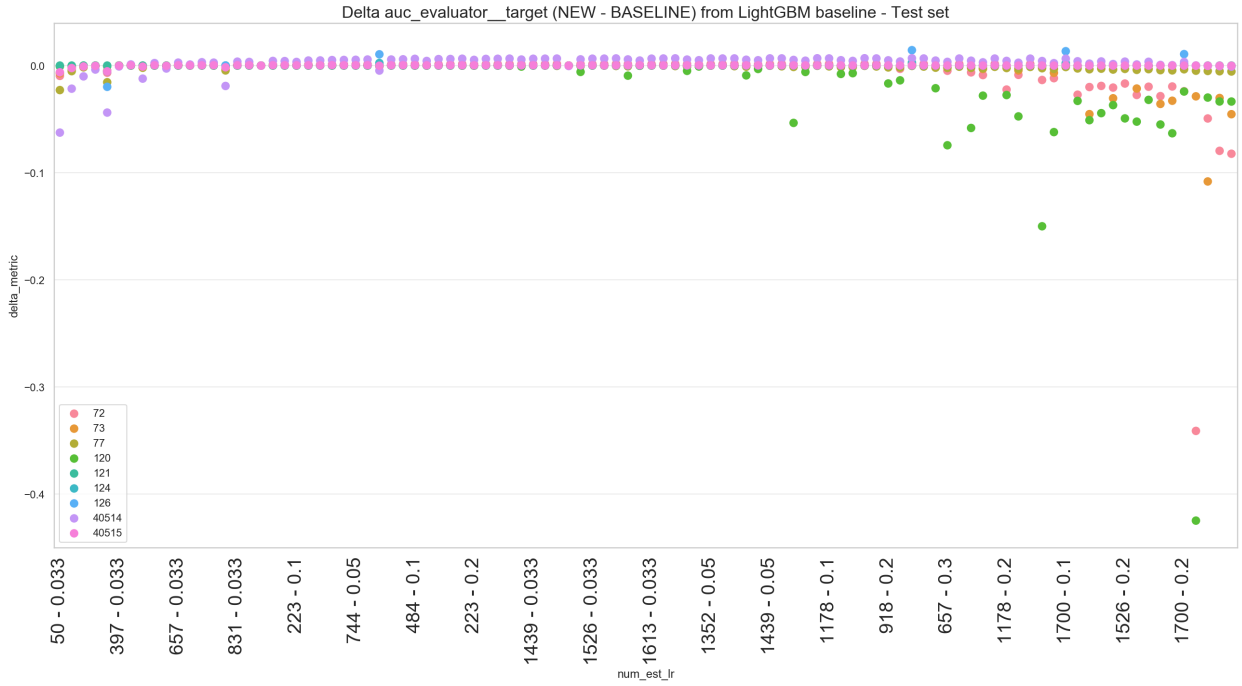


Figure 6.1: δ_{AUC} *for scenario* $\mathcal{S}(C_3, \eta^{(3)}_{LR,NE}, AUC)$*, which failed the Levene Test; Some datasets* δ_{AUC} *increased the variance in treatment groups with very high learning rate and number of estimators*

2. Analyzing Table 6.1 column-wise, the experiments changing the number of estimators (η_{NE}) and changing both the number of estimators and maximum depth ($\eta_{MD,NE}$) have very few scenarios where there is a significant change in the performance metrics.

3. Overall, the Brier Score and Logloss metrics are relatively more sensitive when compared to AUC in the analyzed datasets.

In experimental analysis, if a statistically significant result of parametric or nonparametric analysis of variance is obtained on the experimental data, a *post hoc* test is used to assess the difference between group treatment results, as described in Terpilowski [2019]. In this study, the results of the post hoc tests are not analyzed in this work, although it could be useful in future analysis. For this reason, the analysis pipeline implemented in the code has the option to perform a pair-wise Conover test (using the Holm–Bonferroni method to correct for *family-wise error rate*) on all treatment means. In Figure 6.2 one example of a Conover test of the experimental scenario $\mathcal{S}(C_1, \eta^{(1)}_{LR}, Brier)$ is shown[1].

[1]This is just an example plot that wasn't used in the analysis, since the scenario is not statistically significant according to Kruskal–Wallis.

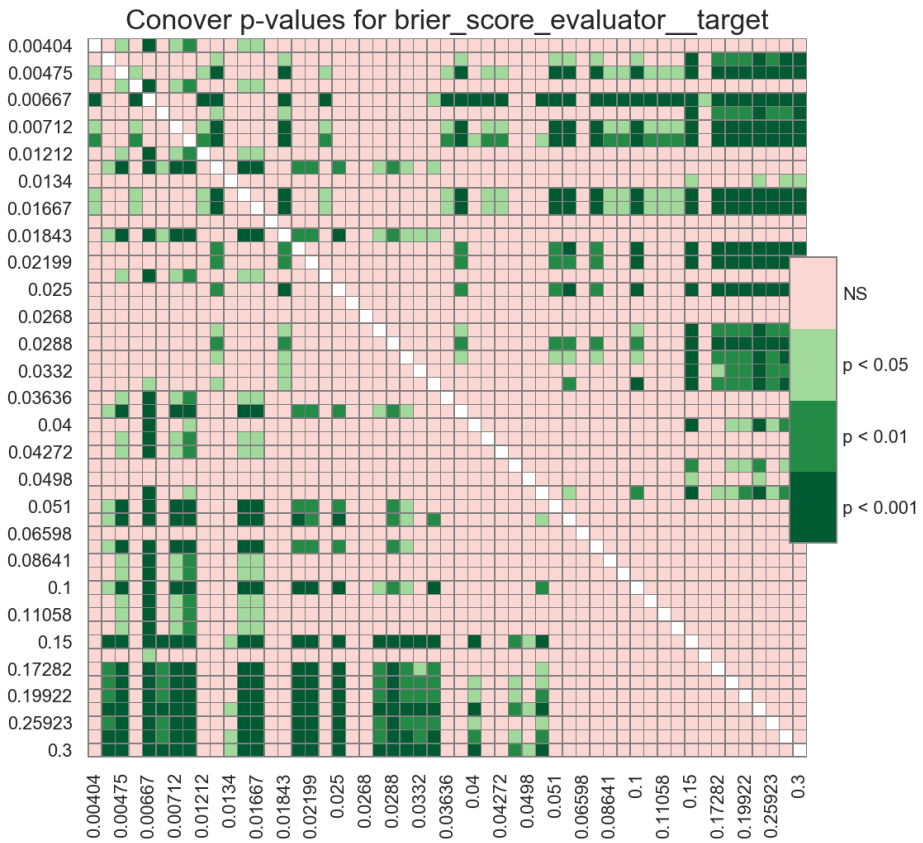


Figure 6.2: *Example of post hoc plot with the pairwise differences in treatment means*

6.2 Treatment Effect and Single-Factor Plots

In the analysis, a SFM model is fitted for every experimental scenario. After the statistical testing, the calculated SFM model is then visualized using the *SFM plot*, a very important plot to understand the overall behavior of the estimated treatment effects $\hat{\tau}_i$ w.r.t the overall estimated mean $\hat{\boldsymbol{\mu}}$. The defined ordering of the treatment values, defined in Subsection 5.3.2 is very useful in this step to grasp the general impact in the metrics of an experimental scenario. To correctly understand the SFM plot, in Figure 6.3 the processed SFM plot for scenario $\mathcal{S}(C_3, \eta^{(3)}_{MD,LR}, AUC)$ is illustrated. All SFM plots have two components:

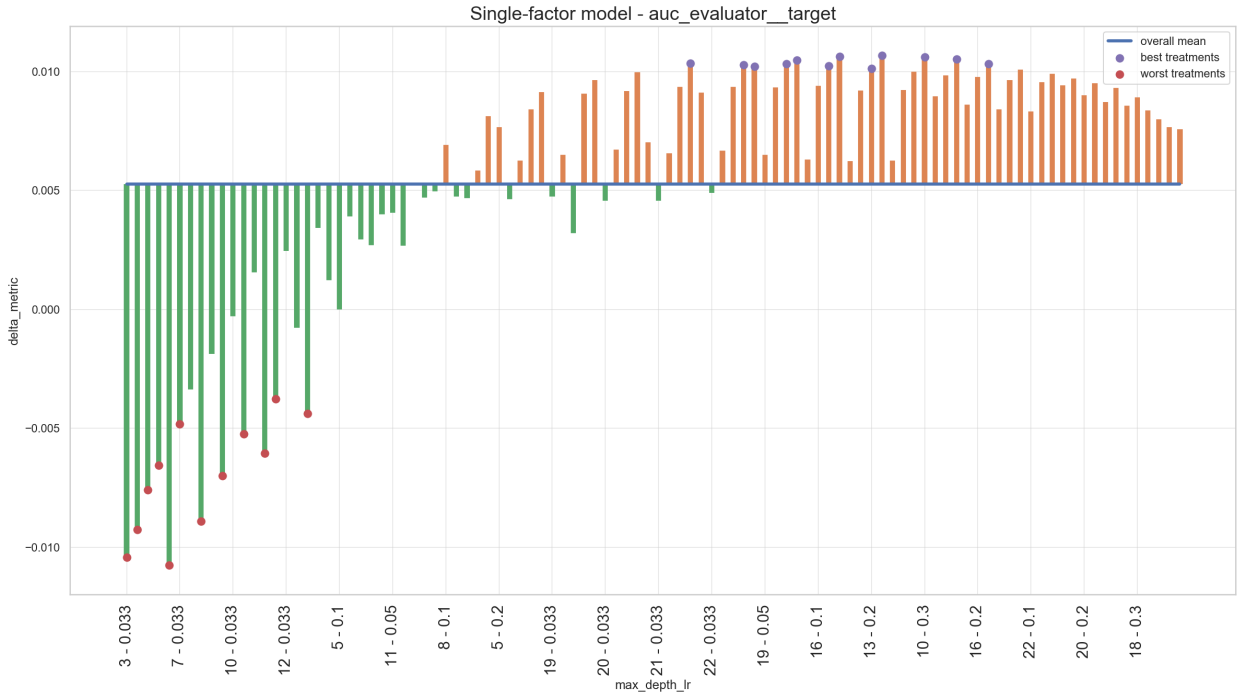


Figure 6.3: *SFM plot of* $\mathcal{S}(C_3, \eta^{(3)}_{MD,LR}, AUC)$

1. The **overall mean** is the estimated average effect $\hat{\boldsymbol{\mu}}$. In practice, it means what is the average change in the performance metric of the LightGBM classification model of the current cluster when subjected to all treatment values $\eta^{(k)}_{Q}$.

2. The vertical lines in the plot are the estimated treatment effects $\hat{\tau}_i$ of each treatment value (shown in the x-axis). The colors orange and green distinguish treatments higher and lower than $\hat{\boldsymbol{\mu}}$, respectively.

3. For each SFM plot, the $\hat{\tau}_i$ with the highest and lowest value are calculated and highlighted in two groups:

   (a) The **best treatments** are the treatments that provide the highest improvement on the performance metric. As explained on Subsection 5.3.1, this depends on the metric being chosen: in Figure 6.3 a positive δ_{AUC} means a model with a higher AUC than baseline, so the highest orange lines are the best treatments in this case.

   (b) The **worst treatments** are the treatments that have the worst impact on the performance metrics, i.e. treatments that generate models with worse metrics than the baseline. Again, it depends on the metrics being chosen: in the case of AUC the worst treatments are the ones where the δ_{AUC} is the lowest in absolute value, but for Logloss and Brier Score this interpretation is reversed. In Figure 6.3 the worst cases for this specific scenario are low values of both learning rate and maximum depth, i.e. the lowest green lines.

The SFM plots are the final plots for the analysis done in this study. They become increasingly difficult to read the higher the dimensionality of the treatments, i.e. the higher the number of different hyperparameters being changed at the same time.

6.3 Result by Hyperparameters

In this part, the hyperparameters combinations are analyzed individually for their impact obtained with the experimental results.

6.3.1 η_{NE}

By analyzing the results of the Kruskal–Wallis tests, without considering the homoscedasticity test, the number of estimators is the hyperparameter that has the least impact on the metrics, on all clusters. The proportion of times this hyperparameter actually provided a significant difference for δ_{AUC}, δ_{Brier} and $\delta_{Logloss}$ were approximately 16%, 16% and 50% respectively. That is, from all the experimental scenarios with treatments of only changing the number of estimators (η_{NE}), almost none of them had a significant difference in AUC and Brier Score.

When analyzing the SFM plots, a peculiar behavior was observed on the results of η_{NE} experiments. The treatment effects are very noisy, with many intercalating positive and negative results. In many of the SFM plots, there is a middle region where the worst treatments appear. In Figures 6.4 and 6.5 two SFM plots exemplify this behavior, one with a Logloss metric and the other with AUC.

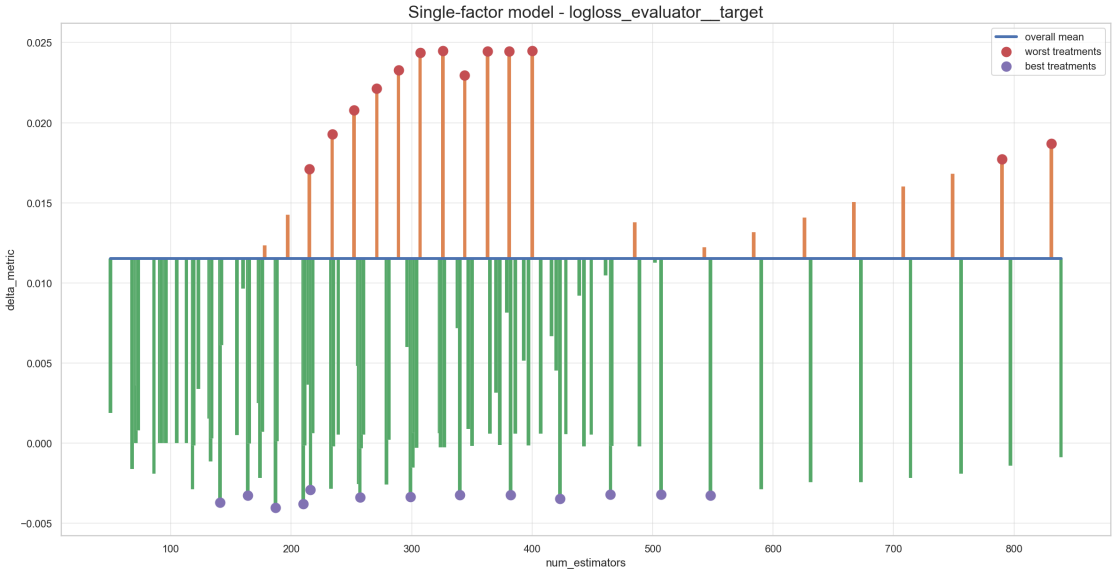


Figure 6.4: *SFM plot for* $\mathcal{S}(C_1, \eta_{NE}^{(1)}, Logloss)$

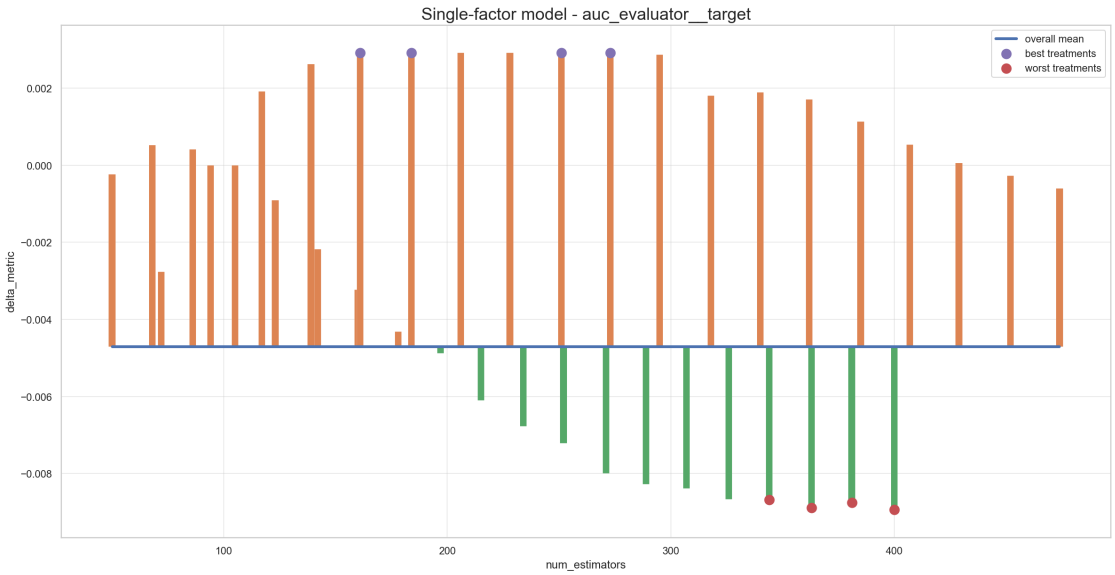


Figure 6.5: *SFM plot for* $\mathcal{S}(C_6, \eta_{NE}^{(6)}, AUC)$

One possible explanation for these results can be that the hyperparameter space chosen for `num_estimators` (see Subsection 4.4.4) didn't cover the most significant values for it. In the results of Probst et al. [2018], the optimal default value for the number of estimators (in their case `n_rounds`) is 4168, and the optimal tuning quantiles q_5 and q_{95} for the number of boosting iterations were 920.7 and 4550.95, respectively. This means that a good improvement to assess the number of estimators impact is to increase the upper limit in the hyperparameter space of this study to at least 4550 estimators, using the values from Probst et al. [2018].

6.3.2 η_{MD}

The maximum depth is the individual hyperparameter (i.e. treatments with only one type of hyperparameter being changed) that had the highest significant changes in the performance metrics over all analyzed clusters. The treatment effects are very intuitive and confirm the expected behavior of changing the maximum depth of the trees in machine learning models. This behavior is that small values of maximum depth will limit the model complexity (i.e. It is a form of regularization) and higher values will increase it, which can be seen in Figure 6.6.

Very low values of maximum depth generate models with poor performance, which then increases as the maximum depth rises, achieving an "optimal point". Then it starts decreasing, probably because the models start to overfit due to high complexity.

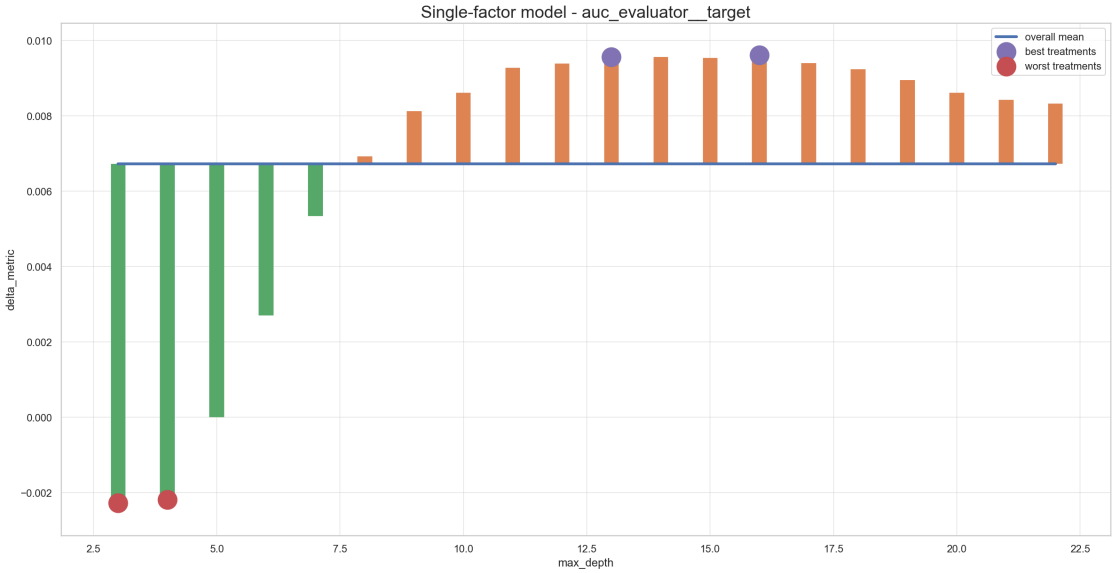


Figure 6.6: *SFM plot for* $\mathcal{S}(C_3, \eta^{(3)}_{MD}, AUC)$

6.3.3 η_{LR}

The learning rate hyperparameter had a small number of significant results, but when analyzing the absolute value of the experiment is $\hat{\tau}_i$, their overall absolute impact is smaller than η_{MD}. Increasing the value of the learning is a way to increase the "steps" in the *optimization space* of an algorithm. Both the δ_{AUC} plot and the SFM plot in Figures 6.7 and 6.8 show a similar pattern, where low values of learning rate usually have a worse performance metric than high values of learning rate. This probably happens because all other hyperparameters are kept the same, and the learning rate increases the effect of each boosting iteration, which can yield a better model if the number of boosting iterations (`num_estimators`) is relatively low.

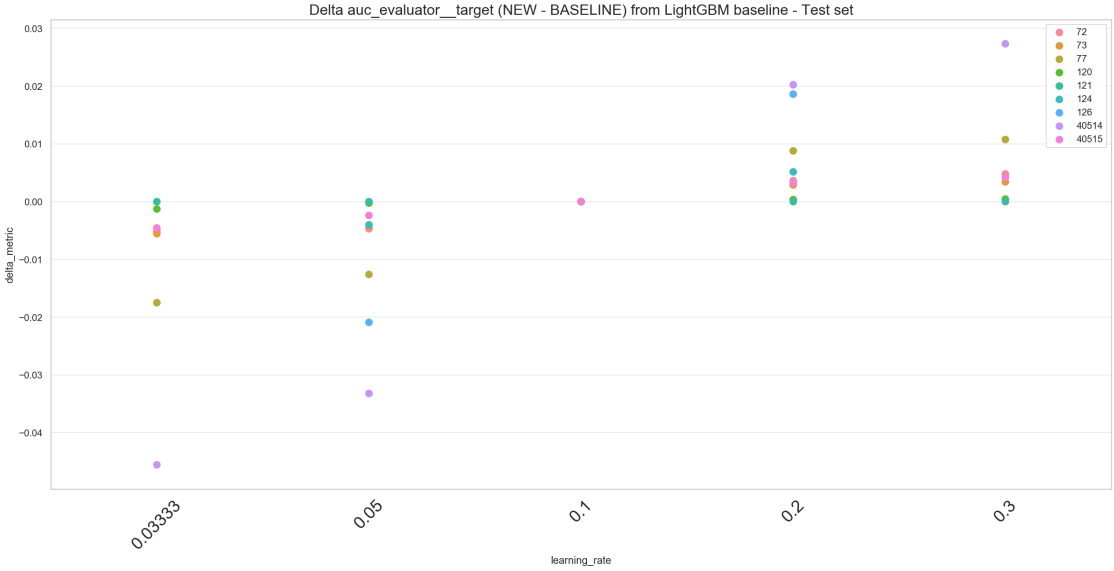


Figure 6.7: δ_{AUC} *plot for* $\mathcal{S}(C_3, \eta^{(3)}_{LR}, AUC)$

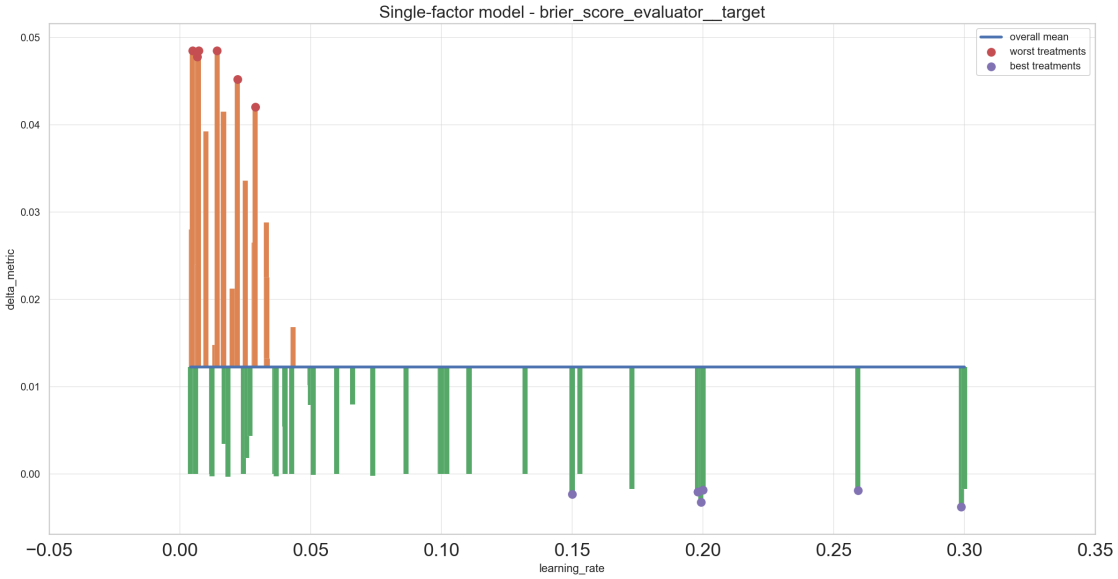


Figure 6.8: *SFM plot for* $\mathcal{S}(C_1, \eta_{LR}^{(1)}, Brier)$

Comparing these results with van Rijn and Hutter [2018], it would be better to use a log scale in this hyperparameter space, instead of a uniformly spaced learning rate values: probably the magnitude of the learning rate values was too small to really observe the full impact of the learning rate individually. Another fact is that some of the learning rate experiments didn't have enough samples when testing their significance, so a possible improvement for future work is to increase the number of learning rates values considered.

6.3.4 $\eta_{MD,LR}$

The combination of changing both the maximum depth and learning rate ($\eta_{MD,LR}$) in the machine learning models is the one combination that had the highest number of significant changes in performance metrics, according to Kruskal–Wallis tests. For instance, 100% of the scenarios with $\eta_{MD,LR}$ and δ_{Brier} have had a significant difference in the treatment means, the only hyperparameter combination to do so.

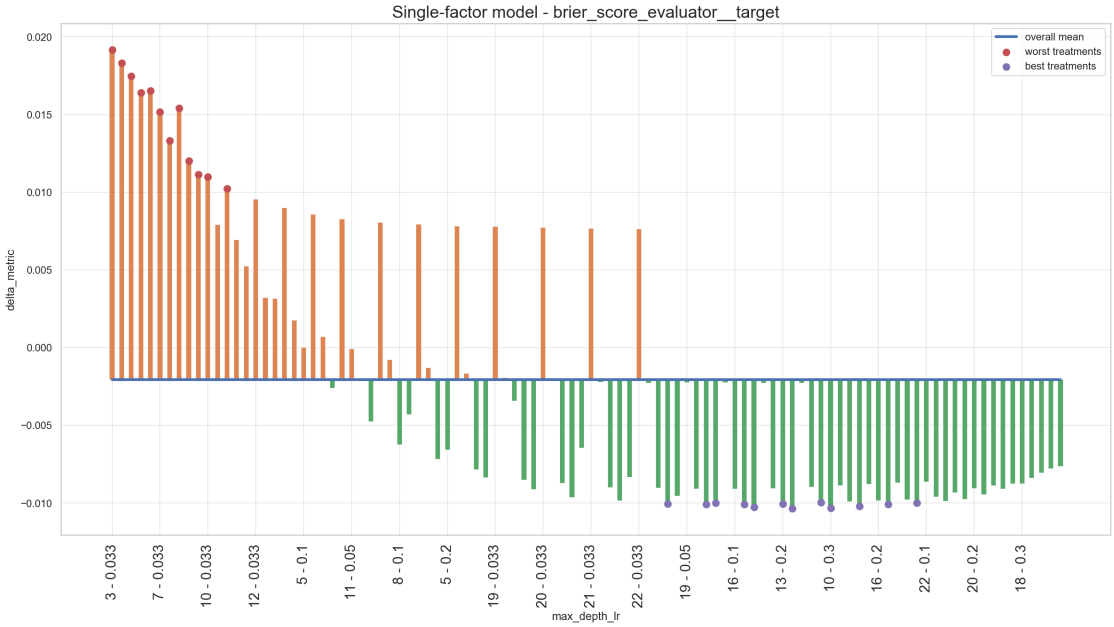


Figure 6.9: *SFM plot for* $\mathcal{S}(C_3, \eta^{(3)}_{MD,LR}, Brier)$

As explained in the last Subsection, the learning rate can increase (or "accelerate") the search over the gradient boosting optimization space, and the results showed here confirmed this intuition. First, we can observe in Figure 6.9 and Figure 6.10 the same behavior found in the SFM plots from the experimental scenarios where the treatments are only the max depth η_{MD}, albeit reversed because the figures showed here display the Brier Score instead of AUC. The other evidence is that by looking at the statistical tests results in Table 6.1, one can see that all significant experiments for treatment η_{MD} are also significant for $\eta_{MD,LR}$, but there is a number of experiments from column η_{MD} which aren't significant that became significant when the treatment is $\eta_{MD,LR}$.

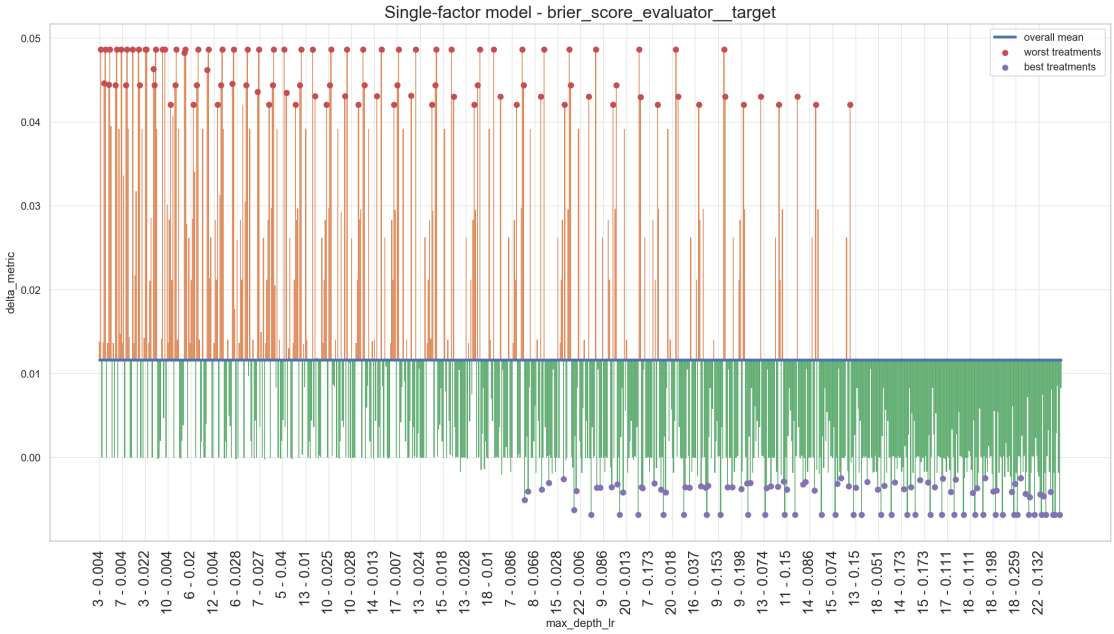


Figure 6.10: *SFM plot for* $\mathcal{S}(C_1, \eta^{(1)}_{MD,LR}, Brier)$

6.3.5 $\eta_{MD,NE}$

This combination of maximum depth and number of estimators had the lowest number of statistically significant experiments: only in two experimental scenarios the null hypothesis of the Kruskal–Wallis test was rejected. This combination, when seen on an individual level (as in Figure 5.3), reveals that possibly treating both maximum depth and number of estimators as a single factor does not capture the overall impact of each hyperparameter. Different approaches might be more useful here, like the *Functional ANOVA* used in van Rijn and Hutter [2018]. Higher limits of the number of estimators distribution could also give a better overall picture of the $\eta_{MD,NE}$, as explained in Subsection 6.3.1.

However, when analyzing some of the SFM models (even the ones which are heteroscedastic), there is a common pattern in the SFM plots structure. It was explained in 4.4.2 a relationship between max depth and the number of estimators, which describe different ways to change the performance of a classifier based on the magnitude of each hyperparameter.

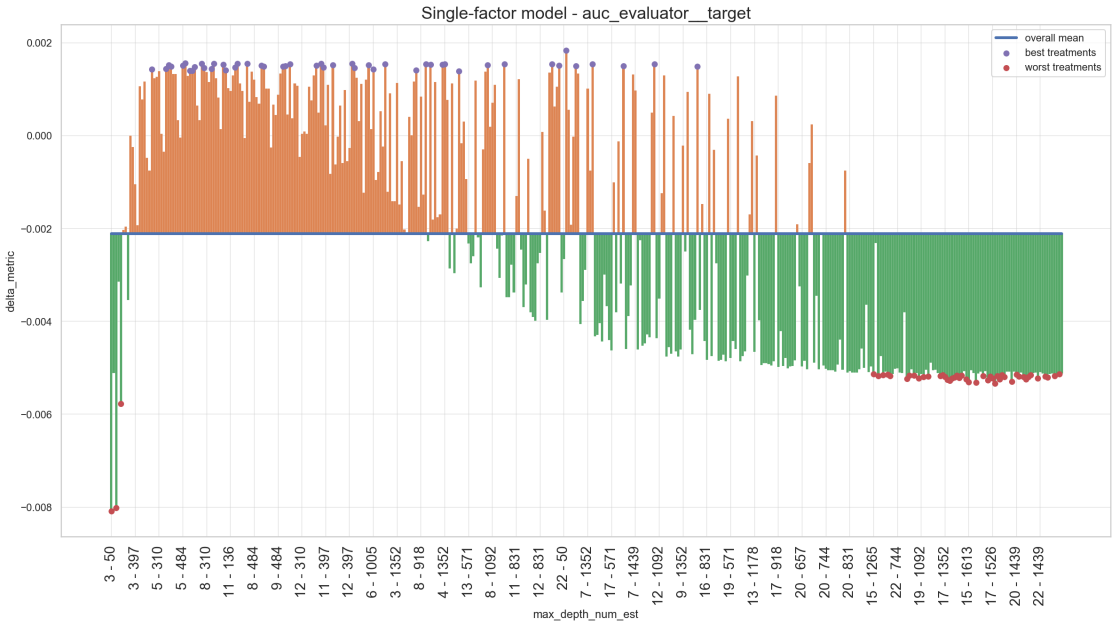


Figure 6.11: *SFM plot for* $\mathcal{S}(C_3, \eta^{(3)}_{MD,NE}, AUC)$

The same pattern observed in Figure 5.3 can be recognized in the SFM plots of the experiments with treatments $\eta_{MD,NE}$. In Figure 6.11 for example, lower values of both `max_depth` and `num_estimators` decreases the AUC, then the AUC increases when there is an equilibrium between both hyperparameter values (the orange section). Finally, the AUC starts decreasing again when both hyperparameter values have high values. There is also a "middle ground" where the estimated treatment effects are intercalating, which is an effect of the ordering defined in Subsection 5.3.2: combinations with an average value of maximum depth or number of estimators can increase or decrease the AUC with high variance, which produces the intercalating effect clearly seen in Figure 6.12.

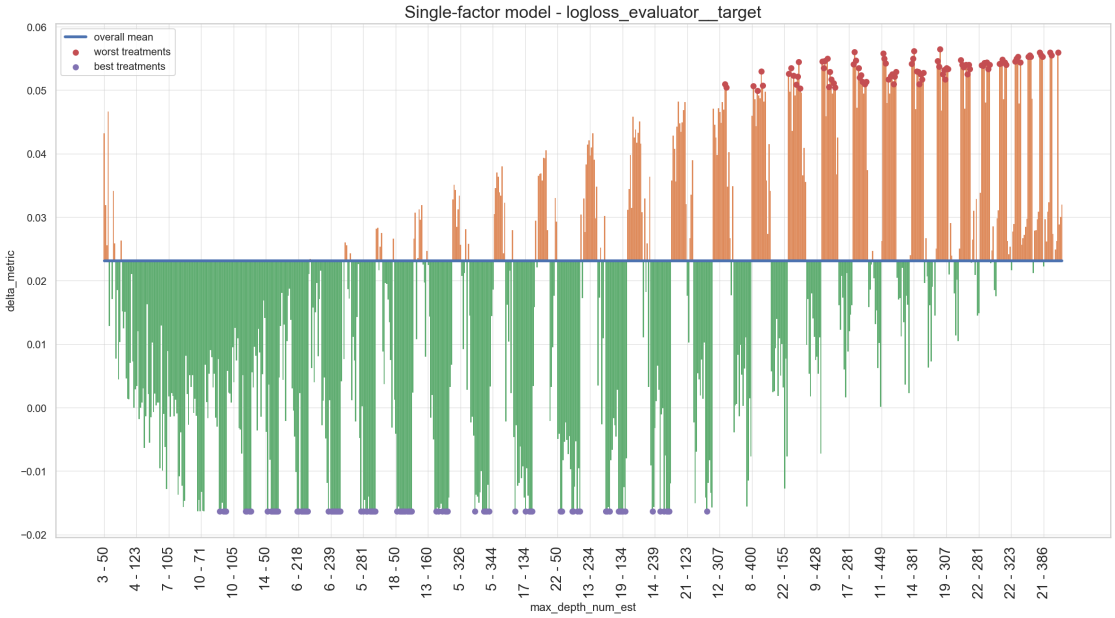


Figure 6.12: *SFM plot for* $\mathcal{S}(C_2, \eta^{(2)}_{MD,NE}, Logloss)$

6.3.6 $\eta_{LR,NE}$

The combination of learning rate and the number of estimators yield a similar result to $\eta_{MD,LR}$, with more statistically significant results than individually changing either learning rate or the number of estimators. The SFM plots from these scenarios follow a ∪-shaped pattern for AUC, and the opposite ∩-shaped pattern for Brier Score (Figure 6.13) and Logloss results.

The result is also intuitive because, on one hand, a small number of boosting iterations coupled with a small learning rate will likely lead to a very simple gradient boosting model, with low performance on the test set. On the other hand, high values of both the learning rate and the number of estimators can also generate bad classifiers. In the middle section of the SFM plot there is the highest density of the best treatments, which equilibrate both hyperparameter values.

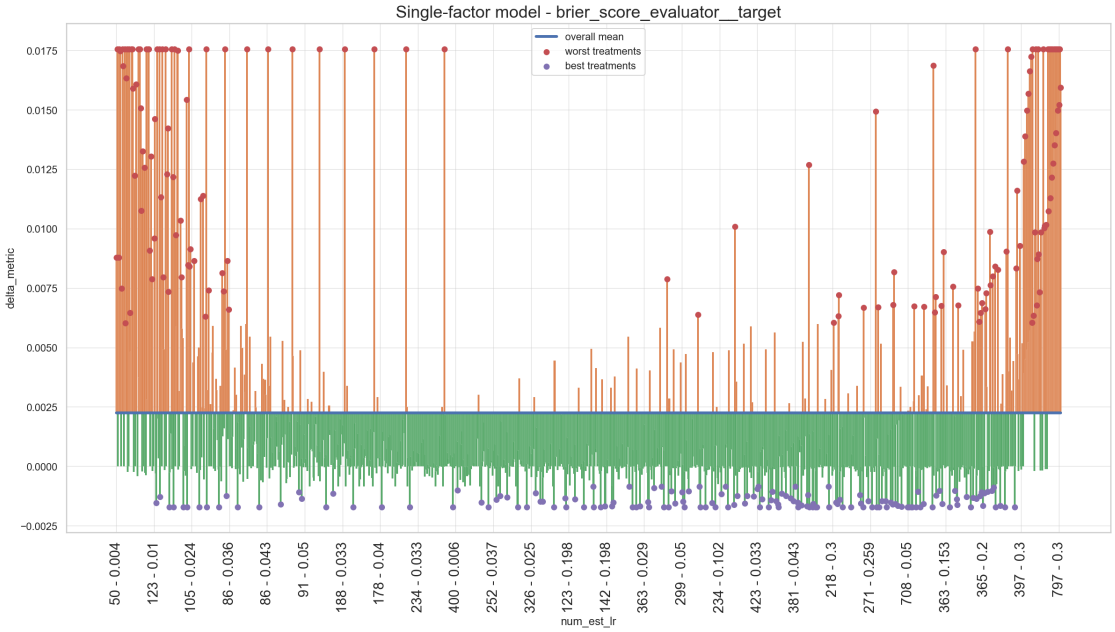


Figure 6.13: *SFM plot for* $\mathcal{S}(C_1, \eta^{(1)}_{LR,NE}, Brier)$

6.3.7 $\eta_{NE,MD,LR}$

There is a high proportion of single-factor models of experimental scenarios with $\eta_{NE,MD,LR}$ that are significant. If not taking into account the homoscedasticity violations from Cluster 3 data, this would be the combination with the highest number of significant experiments.

Even though this combination has a high impact on the performance metrics, it is harder to visualize a pattern in the SFM plots due to the high dimensionality of the data. There is a clear pattern throughout all cluster results, although some behaviors can be interpreted from it. Figure 6.14 illustrates the $\delta_{Logloss}$ plot for all treatment values of $\eta_{NE,MD,LR}$ for Cluster 2, with the respective SFM plot in Figure 6.15: there is a similar ∪-shaped pattern as in the $\eta_{LR,NE}$ combinations, but in this case this pattern does not consistently appear in the other clusters experimental results.

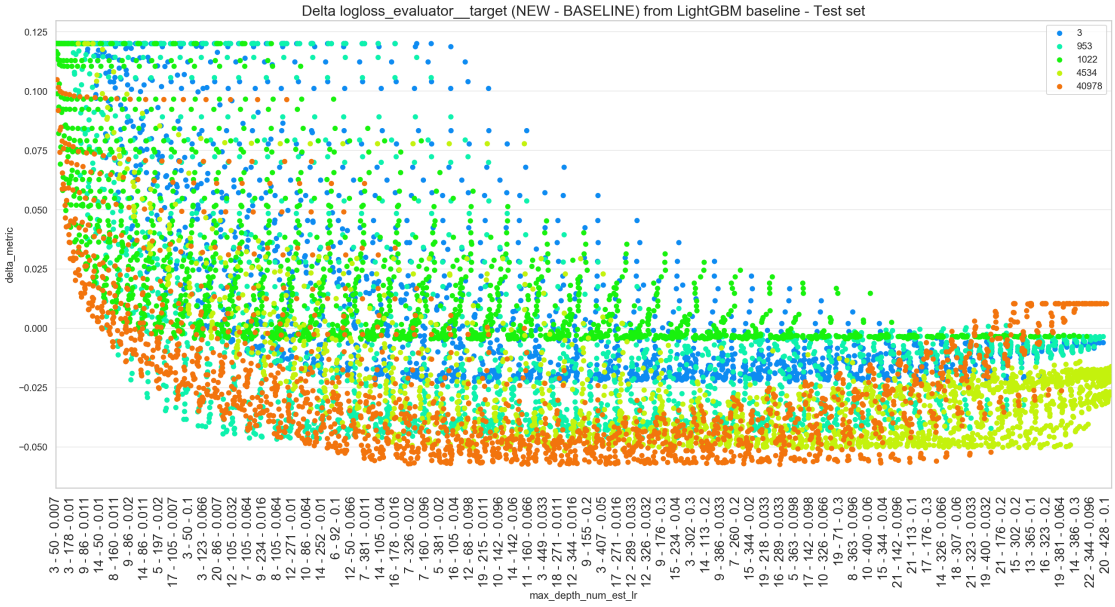


Figure 6.14: *$\delta_{Logloss}$ for scenario $\mathcal{S}(C_2, \eta^{(2)}_{NE,MD,LR}, Logloss)$*

Figure 6.15: *SFM plot for $\mathcal{S}(C_2, \eta^{(2)}_{NE,MD,LR}, Logloss)$*

In some metrics and clusters, the SFM plot for $\eta_{NE,MD,LR}$ is very noisy, probably due to a high dimensionality reduction in the x-axis, which makes it even more difficult to interpret it, as illustrated in Figure 6.16.

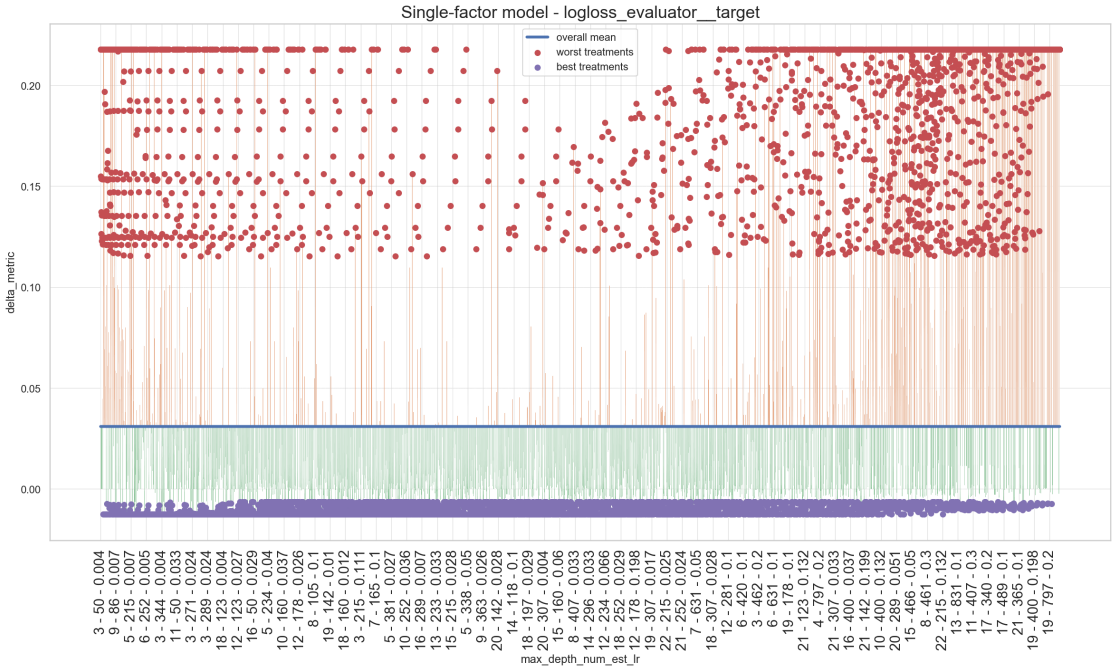


Figure 6.16: *SFM plot for* $\mathcal{S}(C_1, \eta^{(1)}_{NE,MD,LR}, Logloss)$

6.4 Results by Hyperparameter Combination Effect

Another perspective in the analysis of the results of the experiments is about the "combination effect" of the hyperparameters, i.e. how changing one hyperparameter individually, in pairs or in triples differs in terms of significant difference in the performance metrics δ_{metric}.

Theoretically, training models while changing multiple hyperparameters in a model tuning process means more hyperparameter space is covered. For example, the Random Search algorithm (see Probst et al. [2018]) typically change more than one hyperparameter value at a time, searching for a more "extensive" hyperparameter space than changing just a single hyperparameter value each time. This fact can be confirmed in the study, at least for the clusters analyzed here. Table 6.2 show the percentage of scenarios in which a significant change in the metrics was observed in the study.

	Metric		
Combinations	δ_{AUC}	δ_{Brier}	$\delta_{Logloss}$
Individual	22.2%	22.2%	38.8%
Pair	38.8%	55.5%	50%
Triple	66.6%	66.6%	66.6%

Table 6.2: *Percentage of statistically significant results for each comparison and metric*

d

In practice, changing several hyperparameters at the same time can be costly, which is one of the reasons a Bayesian Optimization or multi-stage optimization can be performed, e.g. the multi-stage algorithm described in Wang et al. [2015]. Using the results of this study, the typical trade-off between computational complexity and search space in the model tuning can be better evaluated

depending on the case. For instance, one could start a gradient boosting model tuning process by changing `max_depth` and `learning_rate` at the same time, as it is a pair of hyperparameters (it is faster to run than a triple or higher number of combinations) that resulted in a high percentage of statistically significant SFM models (the models are highly sensitive to this combination of hyperparameters).

6.5 Results by Cluster

Each cluster result can also provide useful insights related to the datasets inside the cluster and how sensitive they are w.r.t different combinations of hyperparameters. In this section, the main observations from each cluster results are addressed, along with possible explanations for their behavior. To understand this analysis, it is useful to refer to the cluster's ridgeline plots in Appendix A.

By using table 6.1 from Section 6.1, it is possible to summarize the results of the experiments grouping by the cluster number. Table 6.3 describes the proportion of statistically significant experiments for each metric on each cluster. This table gives an idea of how sensitive each cluster is to changes in the performance metrics when subjected to all treatments of this study.

	Cluster					
Metric	**1**	**2**	**3**	**4**	**5**	**6**
δ_{AUC}	42.8%	57.1%	28.5%	0.0%	57.1%	42.8%
δ_{Brier}	57.1%	57.1%	14.2%	42.8%	71.4%	14.2%
$\delta_{Logloss}$	71.4%	85.7%	0.0%	42.8%	71.4%	14.2%
Overall	57.1%	66.6%	14.2%	28.5%	66.6%	23.8%

Table 6.3: *Percentage of statistically significant results in each cluster, by metric*

The most **insensitive cluster**, i.e. the cluster with the least number of statistically significant experiments is **Cluster 3**. As said in Subsection 5.2.2, all the datasets of this cluster are synthetically generated datasets, and all of them have $\geq$ 100000 rows in them. However, when looking at the statistical test results of Cluster 3, it is the cluster that rejected the most the null hypothesis of equal variance, i.e. It is the most heteroscedastic cluster. With further analysis, some of its datasets also have a very constant δ_{metric} in the metrics, another evidence or the insensitivity of this cluster: in Figure 6.17 the δ_{AUC} for scenario $\mathcal{S}(C_3, \eta^{(3)}_{MD,NE}, AUC)$ shows that most of the datasets have a constant δ_{AUC} regardless of the changes in maximum depth and number of estimators.

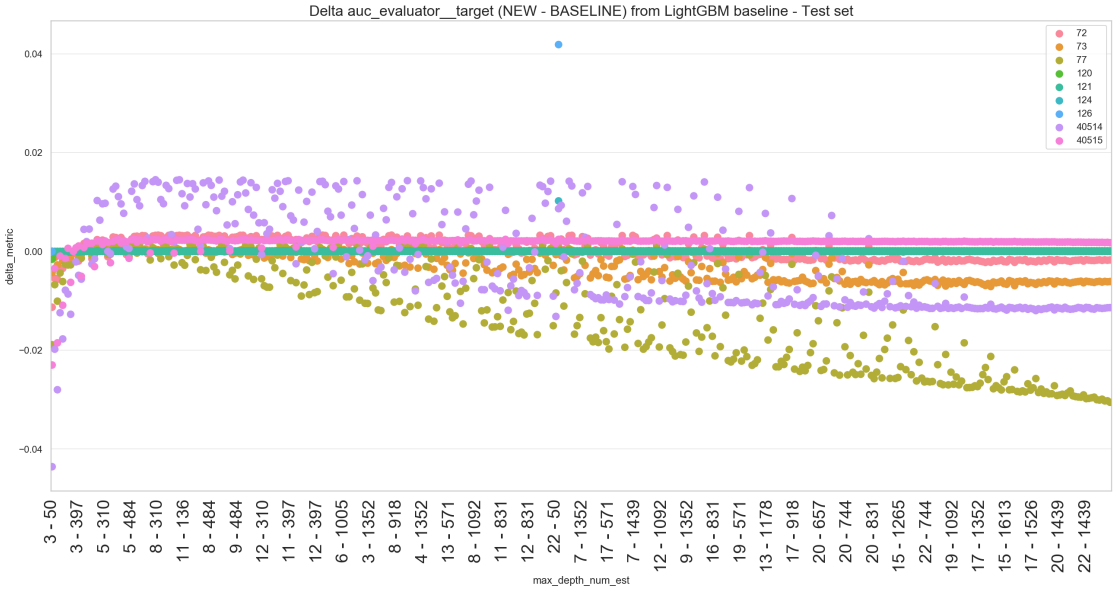
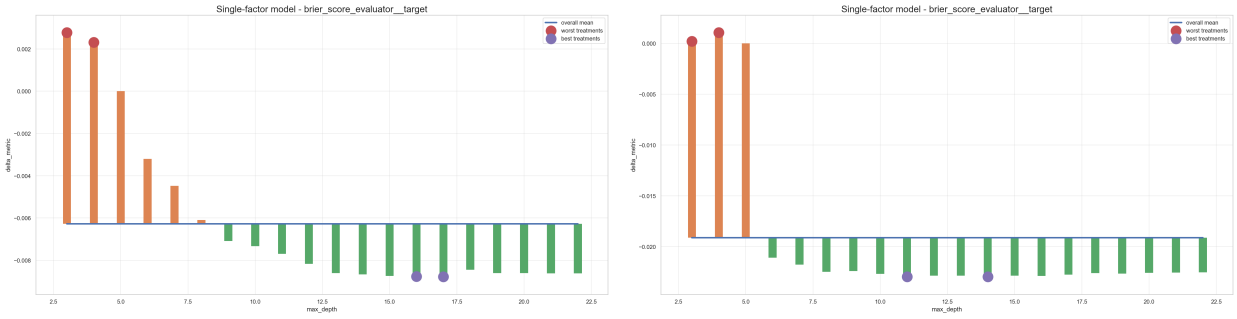


Figure 6.17: *δ_{AUC} plot for $\mathcal{S}(C_3, \eta^{(3)}_{MD,NE}, AUC)$ - only two datasets seem to change the AUC*

On the other hand, the most **sensitive clusters** are **Cluster 2** and **Cluster 5**. Analyzing the datasets aggregated characteristics (more details about them in Subsection 5.2.1) of both clusters, they are fairly similar: both have almost no numerical features, with a high proportion of categorical features; the number of rows is also similar, but Cluster 2 datasets typically have more rows than Cluster 5. The main difference between them is the number of features, where Cluster 2 datasets can have from 2 to 200 features and Cluster 5 datasets typically range from 10 to 20.

In Figure 6.18 two SFM plots of statistically significant experiments showcase both the behavior of max depth explained in Subsection 6.3.2, and also the impact of the number of features in max depth performance. One can see that the amount of tree depth needed to capture more information — reflected here as a lower Brier Score — is smaller when the datasets have fewer features. In Figure 6.18b max depth of 6 is enough to have a negative treatment effect (in this case, an improvement over the baseline Brier Score), while for Cluster 2 which has more features this value is 9, as illustrated in Figure 6.18a.



(a) $\mathcal{S}(C_2, \eta^{(2)}_{MD}, Brier)$ **(b)** $\mathcal{S}(C_5, \eta^{(5)}_{MD}, Brier)$

Figure 6.18: *Two SFM plots of η_{MD} and δ_{Brier} on Cluster 2 and 5, respectively*

Finally, it is interesting to note that Cluster 4 does not have a single statistically significant experiment when the performance metric is δ_{AUC}, and when an experimental scenario with δ_{Brier} rejected the null hypothesis of equality of treatment means the corresponding scenario with $\delta_{Logloss}$

also rejected it, and vice versa.

6.6 Results by Metrics

Regarding the performance metrics of the study, described in Section 2.4, there are some insights about their results from the experimental scenarios. Table 6.4 illustrates the proportion of scenarios with a significant change in the δ_{metric}: The Logloss metric is the most sensitive in this study, changing in almost 50% of the experiments. It is closely followed by the Brier Score, and the least sensitive metric in the study is the AUC.

δ_{AUC}	δ_{Brier}	$\delta_{Logloss}$
38.1%	42.7%	47.6%

Table 6.4: *Percentage of statistically significant results of each metric*

The higher percentage of statistically significant experiments regarding the logarithmic loss is expected since it is the metric being optimized in all LightGBM binary classification models of the study. Logloss is not bound to the $[0, 1]$ interval as AUC and Brier Score, which means in practice the treatment values of the Logloss aren't as interpretable when compared to other metrics. The similarity of the percentage of Table 6.4 between the Brier Score and Logloss is partially expected, as both metrics assess the value of the predicted probabilities of the classifier.

At the same time, the AUC does not directly evaluate the performance of the predicted probabilities themselves, but it actually assesses the relative ordering of the classifier. For this reason, results like the one observed in Cluster 4 can happen, i.e. the relative ordering performance can be kept the same while the forecasted probabilities can be better. The highest shift of the δ_{AUC} was in the experimental scenario $\mathcal{S}(C_5, \eta^{(5)}_{NE,MD,LR}, AUC)$ illustrated in Figure 6.19, albeit mostly worse AUC than baseline — probably because Cluster 5 datasets have relatively few instances that make the models easy to overfit.

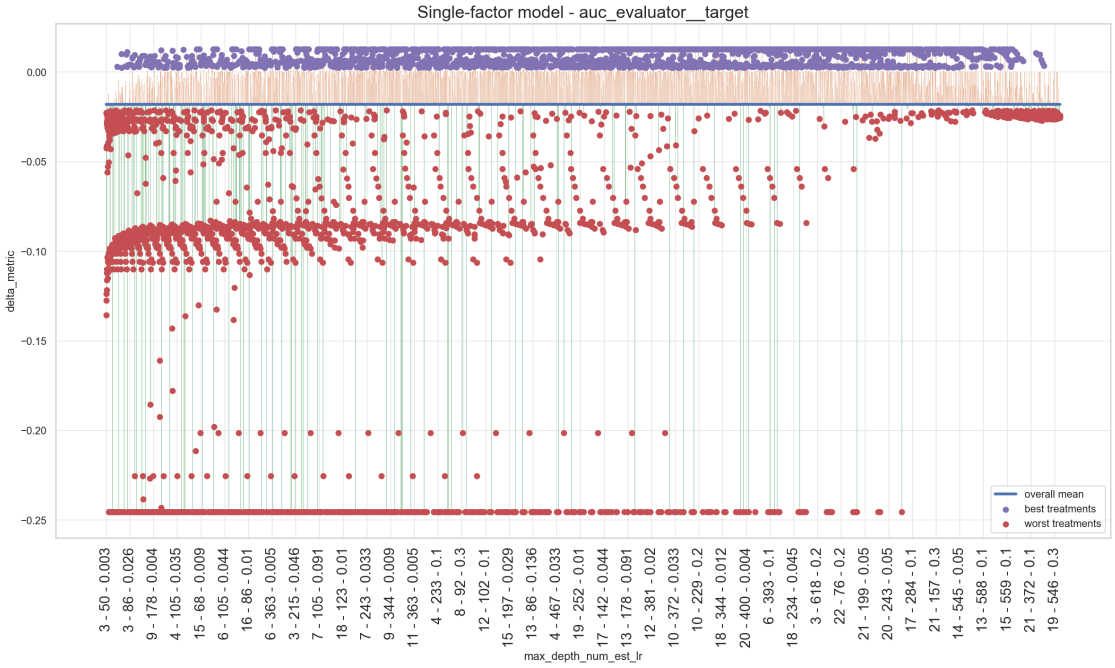


Figure 6.19: *δ_{AUC} plot for $\mathcal{S}(C_5, \eta^{(5)}_{NE,MD,LR}, AUC)$*

Several clusters in the study have imbalanced datasets, and it is known that AUC isn't the most informative metric in imbalanced situations, as described in Saito and Rehmsmeier [2015]. In future work, a possible improvement is to use the Precision-Recall curve for imbalanced datasets, as it provides a better interpretation and of the classifier performance in these cases.

Chapter 7

Conclusion

The comprehension of hyperparameters effects and its impacts on performance metrics is very important in the process of building a good machine learning model. Throughout this study, different aspects of the interaction between gradient boosting hyperparameters and performance metrics were observed and analyzed, providing an insight into how they affect a model.

In this work, a large-scale benchmark was done using 70 different binary classification datasets, which provided enough results to observe how each combination of hyperparameter affected the metrics. Using a solid analysis of variance statistical framework, the experimental results of this work also take into account the statistical significance of each experimental scenario. It was found that some combinations of hyperparameters highly impacted the machine learning models, like the combination of maximum depth and learning rate, while others had a small impact.

The final results also present a variety of single-factor models that estimate the impact effect of each hyperparameter(s) value(s), while at the same time providing useful information about the overall behavior of the hyperparameter in the datasets of the study. Some of the effects observed could be isolated due to the characteristics of some datasets, like the number of features and proportion of categorical features. These insights may prove useful when building new gradient boosting models from scratch, by giving a better idea of which hyperparameters to tackle first depending on the characteristics of the dataset.

7.1 Limitations and Future Work

Given the amount of computational time taken to completely run the study (approximately one month, sharing the workload in parallel between 4 different machines), the scope of this study had to be reduced. First, it only considers binary classification problems, even though the experiment comprises a fairly decent number of 70 datasets. Future work could expand the experiment to analyze the hyperparameters interactions on multiclass classification, regression, ranking, survival models, etc.

The experiment pipeline is rather solid, so it could also be extended with different values, or even with different techniques other than gradient boosting. For instance, different types of hyperparameters could be analyzed in future work, or different combinations of hyperparameters. In this regard, a different framework could be used, like a Functional ANOVA or even a surrogate model to assess the importance of the hyperparameters.

The hyperparameter space of the study could also increase the gaps between the hyperparameter

values, alongside with the maximum level for some of them. A specific limitation found was that the maximum number of estimators generated was probably too small to observe more effects of this hyperparameter. The learning rate could also be generated using a logarithmic scale instead of being uniformly distributed, to generate more different levels of magnitude.

Finally, the clustering strategy could be enriched a lot by adding new information for the datasets. For example, the OpenML platform provides many more characteristics of datasets than the ones used here: entropy of the classes, dimensionality, kurtosis of the features, autocorrelation, percentage of missing values, etc. These statistics could lead to other useful conclusions, depending on the objective of the research.

Appendix A

Cluster Ridgelines Plots

In this appendix all the distributions from the aggregated characteristics of each cluster are summarized using ridgeline plots, with the exception of the `constant_ratio`, because only one cluster has datasets with constant features.

Each cluster is represented by a different color, with the number of the cluster displayed in the y axis. The x axis is the value of the variable, and the height of each distribution is the frequency, like in a histogram.

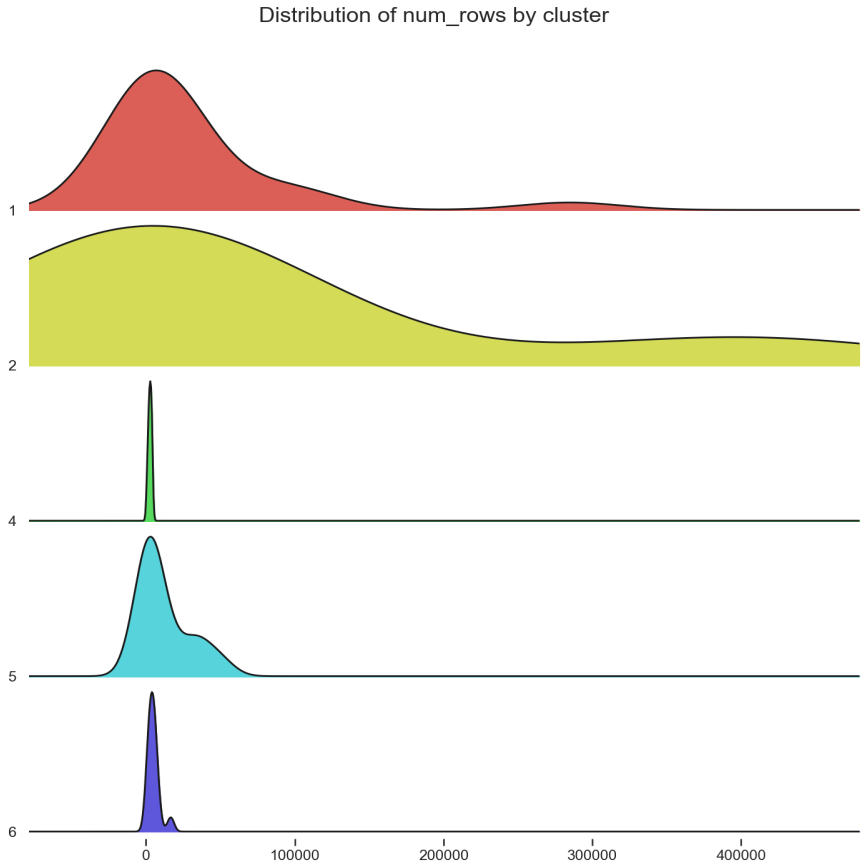


Figure A.1: *Ridgeline plot of* `num_rows`

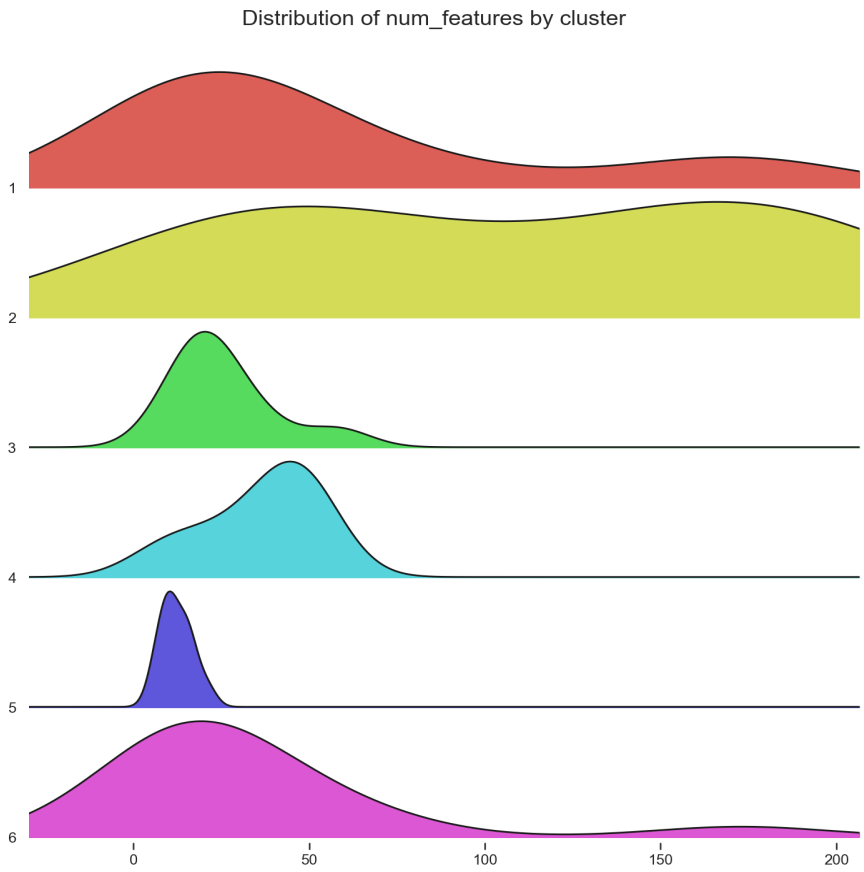


Figure A.2: *Ridgeline plot of* `num_features`

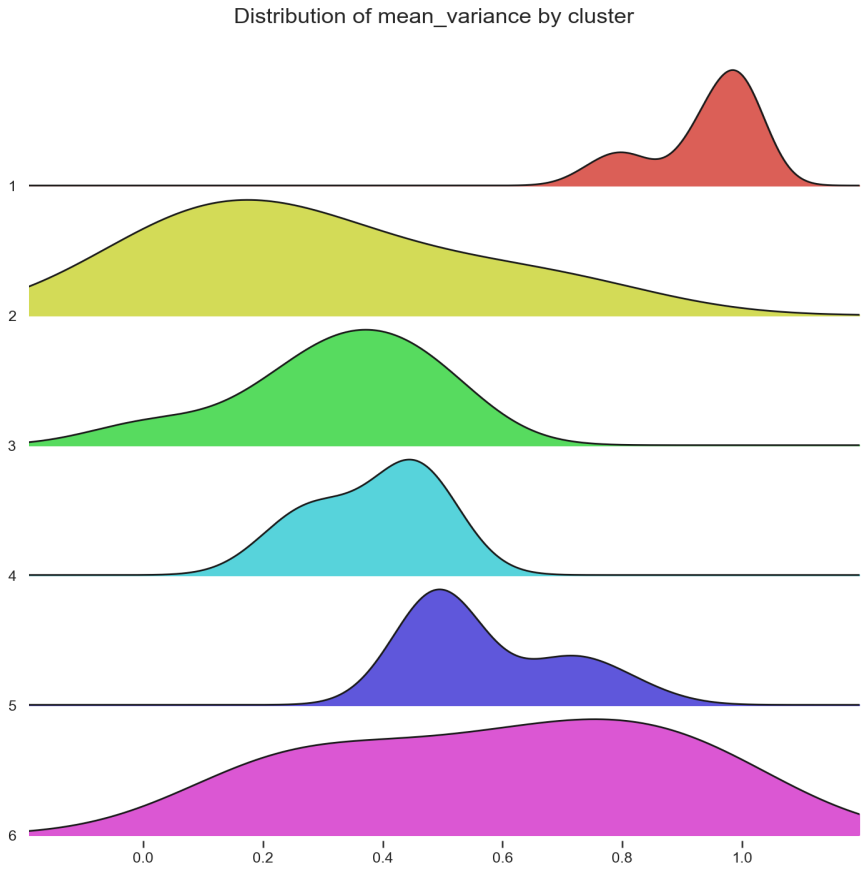


Figure A.3: *Ridgeline plot of* `mean_variance`

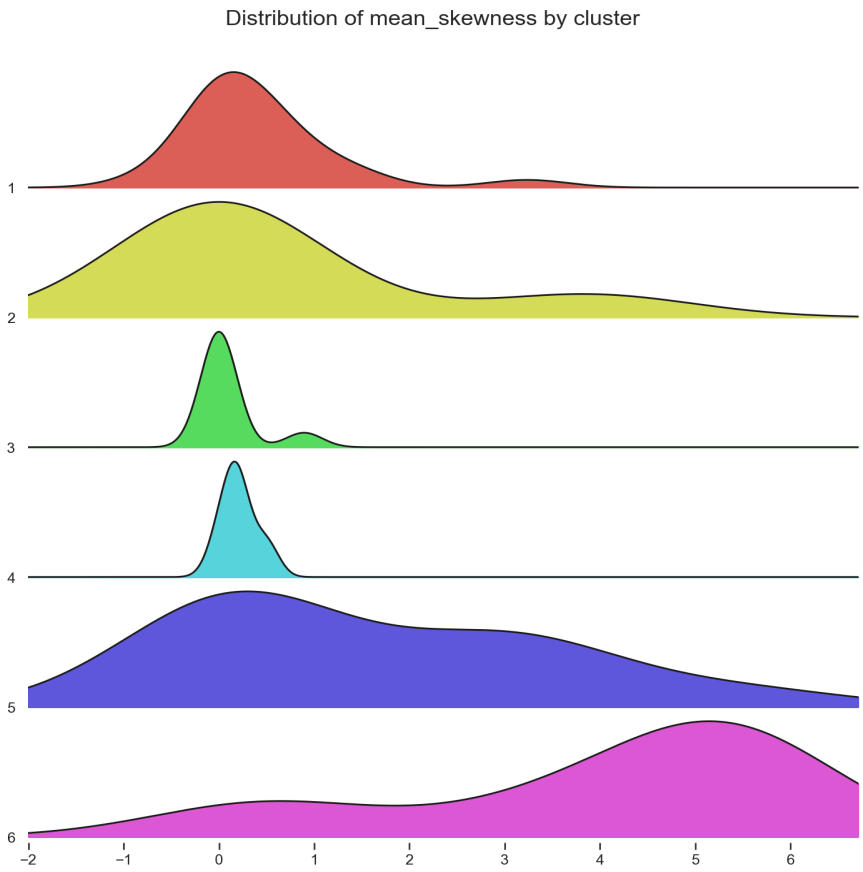


Figure A.4: *Ridgeline plot of* `mean_skewness`

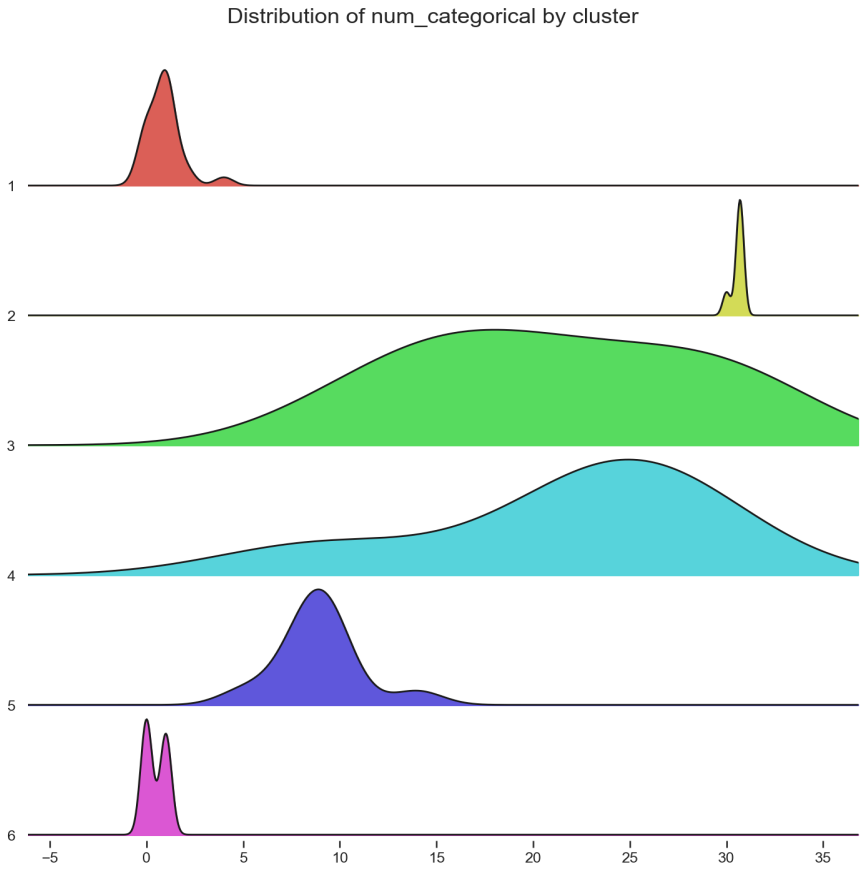


Figure A.5: *Ridgeline plot of* `num_categorical`

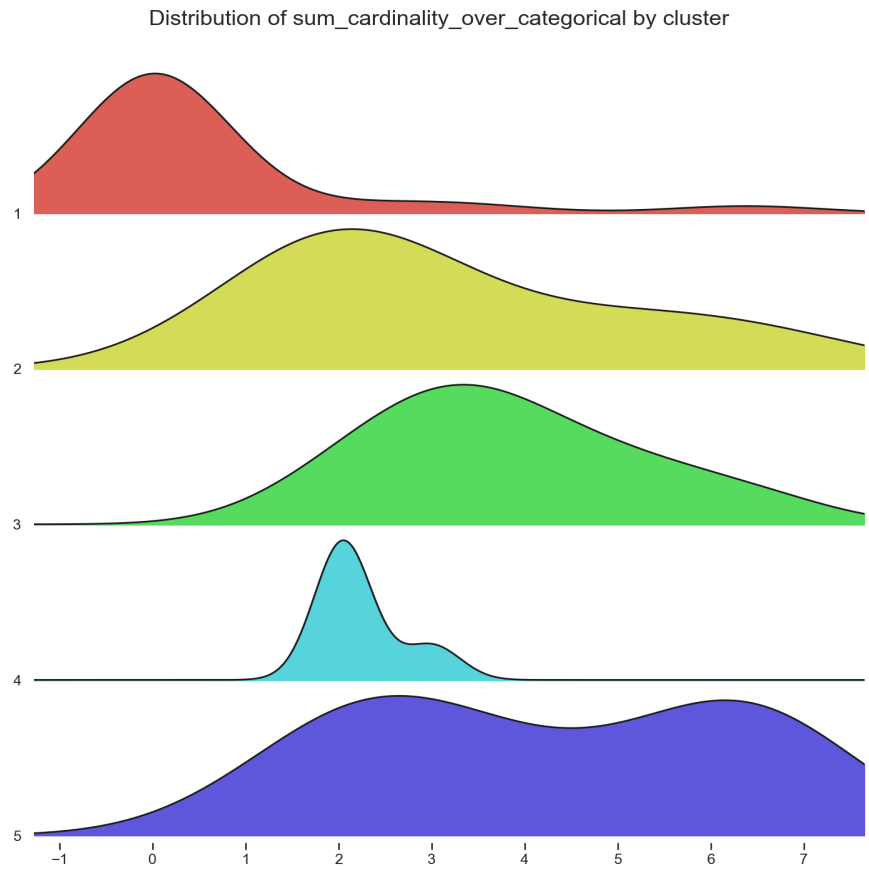


Figure A.6: *Ridgeline plot of* `sum_cardinality_over_categorical`

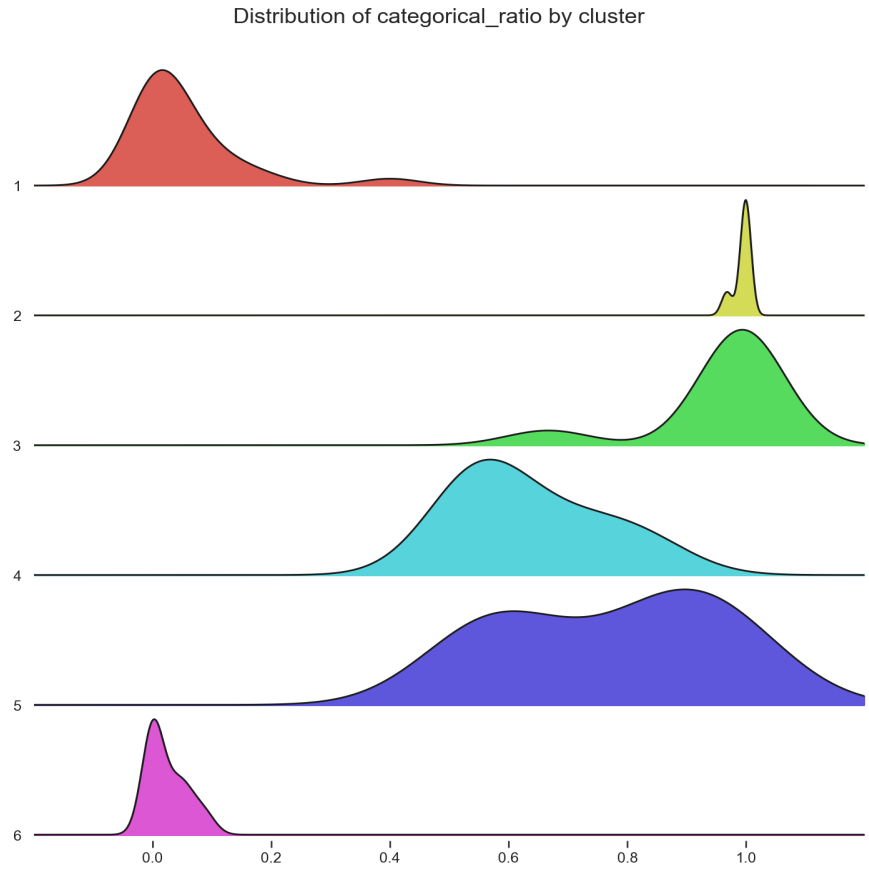


Figure A.7: *Ridgeline plot of* `categorical_ratio`

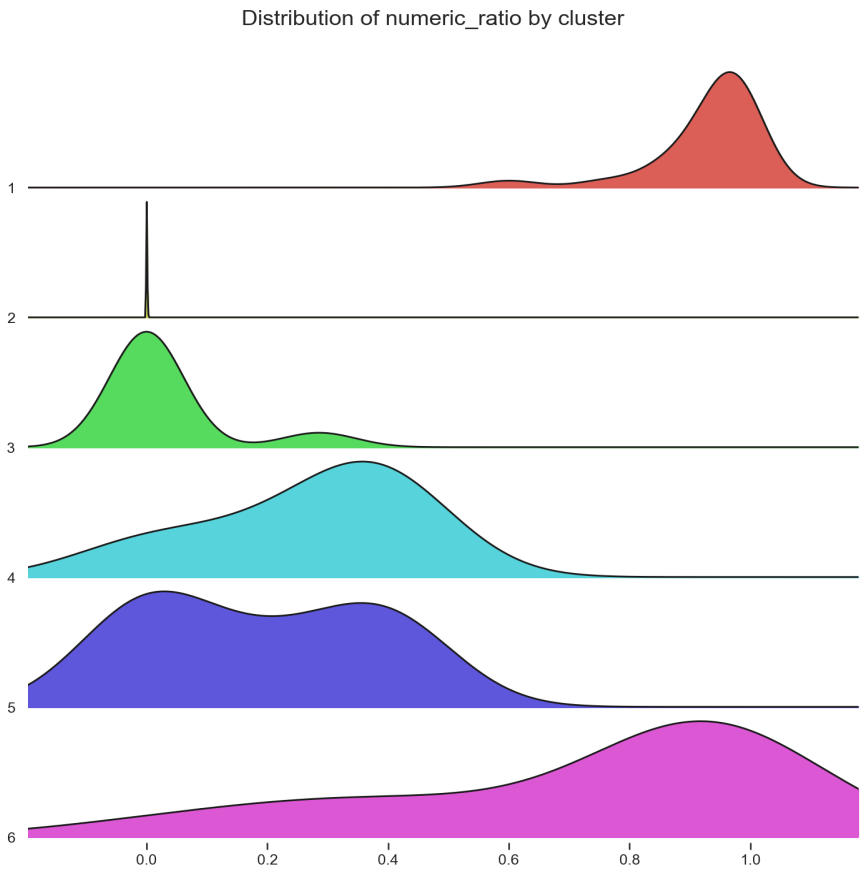


Figure A.8: *Ridgeline plot of* `numeric_ratio`

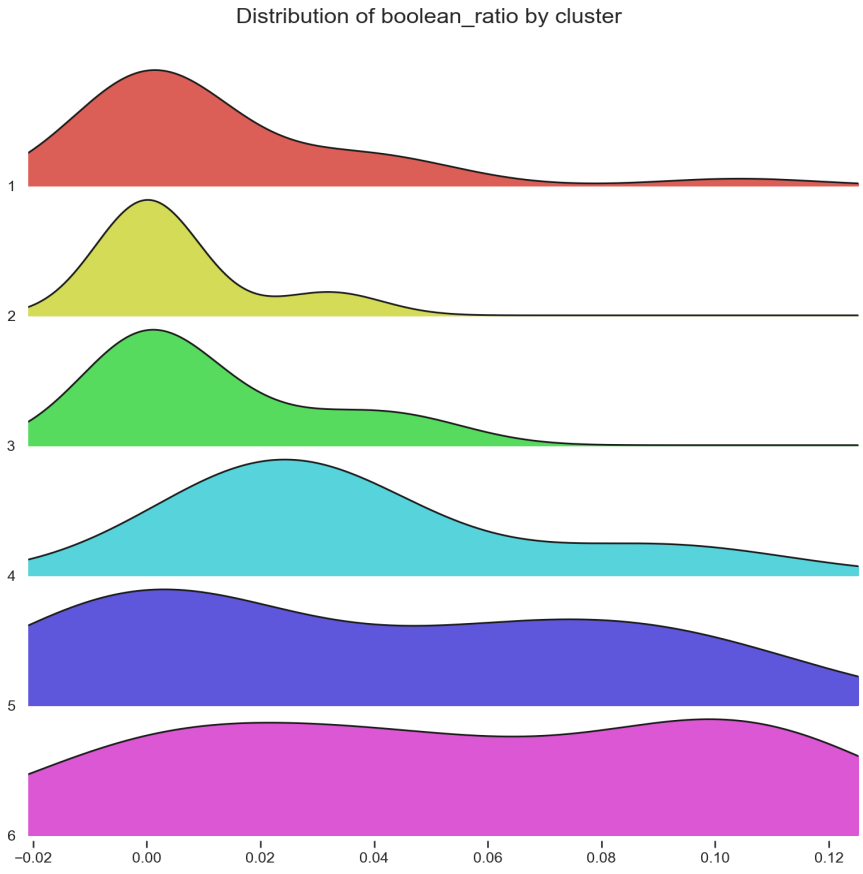


Figure A.9: *Ridgeline plot of* `boolean_ratio`

Appendix B

Statistical Tests Results

Treatment	Metric	Shapiro–Wilk	Levene	Kruskal-Wallis
$\eta^{(1)}_{NE}$	δ_{AUC}	4.965994e-19	9.630606e-01	9.951832e-01
	δ_{Brier}	3.710903e-24	9.999938e-01	9.793033e-01
	$\delta_{Logloss}$	6.203025e-23	9.828542e-01	3.733919e-04
$\eta^{(1)}_{MD}$	δ_{AUC}	3.388219e-21	8.590251e-01	2.901450e-01
	δ_{Brier}	8.838670e-15	2.941775e-01	4.174048e-04
	$\delta_{Logloss}$	9.113385e-14	1.231119e-01	1.246510e-02
$\eta^{(1)}_{LR}$	δ_{AUC}	1.013322e-07	9.811205e-01	2.878008e-02
	δ_{Brier}	5.617271e-14	1.408917e-03	2.329982e-07
	$\delta_{Logloss}$	7.958839e-14	1.050014e-03	3.927624e-07
$\eta^{(1)}_{MD,LR}$	δ_{AUC}	2.401885e-32	9.999982e-01	6.661712e-08
	δ_{Brier}	1.177091e-43	3.361336e-01	1.067366e-90
	$\delta_{Logloss}$	6.678448e-41	9.901324e-19	8.193802e-91
$\eta^{(1)}_{MD,NE}$	δ_{AUC}	0.000000e+00	1.000000e+00	1.000000e+00
	δ_{Brier}	0.000000e+00	1.000000e+00	1.000000e+00
	$\delta_{Logloss}$	0.000000e+00	1.000000e+00	1.025601e-04
$\eta^{(1)}_{LR,NE}$	δ_{AUC}	0.000000e+00	6.240957e-01	2.852539e-01
	δ_{Brier}	0.000000e+00	9.999999e-01	4.590395e-17
	$\delta_{Logloss}$	0.000000e+00	1.000000e+00	1.088594e-33
$\eta^{(1)}_{NE,MD,LR}$	δ_{AUC}	0.000000e+00	1.000000e+00	1.089024e-10
	δ_{Brier}	0.000000e+00	1.000000e+00	6.880061e-293
	$\delta_{Logloss}$	0.000000e+00	1.000000e+00	0.000000e+00

Table B.1: *Statistical p-values of the experimental scenarios in Cluster 1*

Treatment	Metric	Shapiro–Wilk	Levene	Kruskal-Wallis
$\eta^{(2)}_{NE}$	δ_{AUC}	4.059920e-08	9.968117e-01	9.996506e-01
	δ_{Brier}	5.942564e-06	2.105453e-01	9.965572e-01
	$\delta_{Logloss}$	2.502680e-05	9.636175e-03	2.827828e-03
$\eta^{(2)}_{MD}$	δ_{AUC}	2.536182e-10	9.210022e-01	1.860650e-02
	δ_{Brier}	3.926061e-06	2.774635e-01	2.283747e-02
	$\delta_{Logloss}$	2.522154e-05	1.285176e-01	1.764531e-02
$\eta^{(2)}_{LR}$	δ_{AUC}	2.714980e-09	NaN	NaN
	δ_{Brier}	9.808685e-09	NaN	NaN
	$\delta_{Logloss}$	1.213540e-10	NaN	NaN
$\eta^{(2)}_{MD,LR}$	δ_{AUC}	1.406702e-20	9.999939e-01	1.316980e-02
	δ_{Brier}	1.339994e-23	1.000000e+00	6.235464e-12
	$\delta_{Logloss}$	1.334960e-23	1.000000e+00	1.005025e-08
$\eta^{(2)}_{MD,NE}$	δ_{AUC}	2.548462e-22	1.000000e+00	1.000000e+00
	δ_{Brier}	6.494320e-28	9.969882e-01	9.274816e-01
	$\delta_{Logloss}$	3.974093e-21	2.655356e-23	6.191510e-22
$\eta^{(2)}_{LR,NE}$	δ_{AUC}	9.960932e-27	1.000000e+00	3.306332e-06
	δ_{Brier}	2.709085e-26	9.999995e-01	7.035798e-09
	$\delta_{Logloss}$	2.452305e-24	9.999999e-01	1.650304e-05
$\eta^{(2)}_{NE,MD,LR}$	δ_{AUC}	0.000000e+00	1.000000e+00	2.904172e-64
	δ_{Brier}	0.000000e+00	1.000000e+00	4.013155e-111
	$\delta_{Logloss}$	0.000000e+00	1.000000e+00	7.298816e-70

Table B.2: *Statistical p-values of the experimental scenarios in Cluster 2*

Treatment	Metric	Shapiro–Wilk	Levene	Kruskal-Wallis
$\eta^{(3)}_{NE}$	δ_{AUC}	5.611898e-16	9.988256e-01	1.147338e-01
	δ_{Brier}	2.050222e-10	8.124238e-01	9.418066e-02
	$\delta_{Logloss}$	8.273422e-11	6.302806e-01	2.663393e-01
$\eta^{(3)}_{MD}$	δ_{AUC}	3.139396e-07	2.645190e-02	2.556998e-05
	δ_{Brier}	3.959432e-05	3.419306e-02	4.180864e-06
	$\delta_{Logloss}$	6.005177e-04	3.551900e-02	2.702174e-06
$\eta^{(3)}_{LR}$	δ_{AUC}	3.676430e-01	1.621118e-01	1.767196e-06
	δ_{Brier}	5.871765e-03	6.829990e-04	1.551529e-06
	$\delta_{Logloss}$	1.938849e-02	6.932106e-06	1.598158e-06
$\eta^{(3)}_{MD,LR}$	δ_{AUC}	8.839741e-14	5.496276e-01	2.534254e-20
	δ_{Brier}	5.927692e-07	7.132355e-01	8.655283e-48
	$\delta_{Logloss}$	3.870745e-10	3.646846e-02	1.441531e-51
$\eta^{(3)}_{MD,NE}$	δ_{AUC}	3.996994e-40	3.524884e-03	6.177612e-25
	δ_{Brier}	2.490148e-35	2.958452e-02	6.911108e-22
	$\delta_{Logloss}$	2.679125e-35	1.031149e-169	3.001168e-100
$\eta^{(3)}_{LR,NE}$	δ_{AUC}	8.223911e-28	3.639094e-15	4.084732e-19
	δ_{Brier}	4.965566e-27	1.425324e-08	6.548453e-23
	$\delta_{Logloss}$	1.446958e-34	5.364655e-11	1.063454e-24
$\eta^{(3)}_{NE,MD,LR}$	δ_{AUC}	0.000000e+00	2.067193e-23	2.847486e-123
	δ_{Brier}	1.401298e-45	1.522520e-09	1.016972e-125
	$\delta_{Logloss}$	1.401298e-45	0.000000e+00	0.000000e+00

Table B.3: *Statistical p-values of the experimental scenarios in Cluster 3*

Treatment	Metric	Shapiro–Wilk	Levene	Kruskal-Wallis
$\eta^{(4)}_{NE}$	δ_{AUC}	3.616501e-13	9.873830e-01	2.111212e-01
	δ_{Brier}	2.482372e-13	9.980555e-01	9.970748e-01
	$\delta_{Logloss}$	1.336703e-11	9.292868e-01	9.770873e-01
$\eta^{(4)}_{MD}$	δ_{AUC}	2.537239e-11	9.982780e-01	9.797863e-01
	δ_{Brier}	1.019241e-06	9.964629e-01	9.957778e-01
	$\delta_{Logloss}$	6.850468e-10	9.998679e-01	9.999136e-01
$\eta^{(4)}_{LR}$	δ_{AUC}	1.831653e-07	0.000000e+00	1.000000e+00
	δ_{Brier}	3.636517e-09	0.000000e+00	1.024704e-01
	$\delta_{Logloss}$	3.636517e-09	6.581044e-29	1.024704e-01
$\eta^{(4)}_{MD,LR}$	δ_{AUC}	9.831066e-33	9.999617e-01	7.098868e-01
	δ_{Brier}	9.176373e-28	1.000000e+00	2.899170e-10
	$\delta_{Logloss}$	2.758601e-30	9.999158e-01	5.548296e-09
$\eta^{(4)}_{MD,NE}$	δ_{AUC}	3.372529e-39	1.000000e+00	9.862816e-01
	δ_{Brier}	0.000000e+00	1.000000e+00	1.000000e+00
	$\delta_{Logloss}$	4.203895e-45	1.000000e+00	1.000000e+00
$\eta^{(4)}_{LR,NE}$	δ_{AUC}	1.439362e-30	9.994980e-01	1.000000e+00
	δ_{Brier}	1.217919e-29	9.984107e-01	1.066378e-08
	$\delta_{Logloss}$	2.404549e-26	9.772312e-01	4.468989e-09
$\eta^{(4)}_{NE,MD,LR}$	δ_{AUC}	0.000000e+00	1.000000e+00	1.000000e+00
	δ_{Brier}	0.000000e+00	1.000000e+00	2.812655e-136
	$\delta_{Logloss}$	0.000000e+00	1.000000e+00	2.501047e-153

Table B.4: *Statistical p-values of the experimental scenarios in Cluster 4*

Treatment	Metric	Shapiro–Wilk	Levene	Kruskal-Wallis
$\eta^{(5)}_{NE}$	δ_{AUC}	2.727656e-13	9.999921e-01	8.138489e-01
	δ_{Brier}	2.316378e-11	5.085153e-01	9.999896e-01
	$\delta_{Logloss}$	5.203110e-10	5.821598e-03	9.999997e-01
$\eta^{(5)}_{MD}$	δ_{AUC}	1.056675e-27	9.999371e-01	3.299262e-01
	δ_{Brier}	5.635529e-28	9.999508e-01	1.229116e-04
	$\delta_{Logloss}$	1.100347e-27	9.999037e-01	1.275362e-02
$\eta^{(5)}_{LR}$	δ_{AUC}	6.518040e-13	9.998573e-01	1.734229e-02
	δ_{Brier}	3.448695e-14	9.983008e-01	1.984047e-03
	$\delta_{Logloss}$	1.160934e-13	9.997163e-01	2.143289e-03
$\eta^{(5)}_{MD,LR}$	δ_{AUC}	2.371575e-30	1.000000e+00	3.281449e-28
	δ_{Brier}	1.079978e-39	1.000000e+00	7.005511e-07
	$\delta_{Logloss}$	3.393346e-36	1.000000e+00	1.115711e-06
$\eta^{(5)}_{MD,NE}$	δ_{AUC}	0.000000e+00	1.000000e+00	1.000000e+00
	δ_{Brier}	2.802597e-45	9.981370e-01	1.000000e+00
	$\delta_{Logloss}$	4.049753e-43	6.341819e-85	1.000000e+00
$\eta^{(5)}_{LR,NE}$	δ_{AUC}	0.000000e+00	1.000000e+00	1.663213e-08
	δ_{Brier}	0.000000e+00	1.000000e+00	7.634533e-16
	$\delta_{Logloss}$	1.401298e-45	1.000000e+00	1.683672e-15
$\eta^{(5)}_{NE,MD,LR}$	δ_{AUC}	0.000000e+00	1.000000e+00	1.252096e-92
	δ_{Brier}	0.000000e+00	1.000000e+00	1.095053e-251
	$\delta_{Logloss}$	0.000000e+00	1.000000e+00	8.787394e-273

Table B.5: *Statistical p-values of the experimental scenarios in Cluster 5*

Treatment	Metric	Shapiro–Wilk	Levene	Kruskal-Wallis
$\eta^{(6)}_{NE}$	δ_{AUC}	5.555299e-15	7.331148e-02	9.810169e-05
	δ_{Brier}	4.446020e-13	1.695424e-02	1.426687e-07
	$\delta_{Logloss}$	5.699214e-15	1.302647e-04	1.309669e-17
$\eta^{(6)}_{MD}$	δ_{AUC}	8.323186e-18	9.659329e-01	2.275562e-01
	δ_{Brier}	8.587676e-18	8.808170e-01	3.733077e-02
	$\delta_{Logloss}$	8.382108e-16	7.293658e-01	1.738130e-01
$\eta^{(6)}_{LR}$	δ_{AUC}	8.634467e-17	NaN	NaN
	δ_{Brier}	1.000000e+00	NaN	NaN
	$\delta_{Logloss}$	5.394157e-16	NaN	NaN
$\eta^{(6)}_{MD,LR}$	δ_{AUC}	8.691273e-41	9.990496e-01	3.351863e-04
	δ_{Brier}	7.470919e-39	1.000000e+00	6.020948e-12
	$\delta_{Logloss}$	1.597480e-43	9.999978e-01	1.947958e-15
$\eta^{(6)}_{MD,NE}$	δ_{AUC}	0.000000e+00	6.248752e-01	9.997299e-01
	δ_{Brier}	6.459986e-43	9.233229e-71	1.014506e-50
	$\delta_{Logloss}$	0.000000e+00	3.935870e-148	5.383524e-198
$\eta^{(6)}_{LR,NE}$	δ_{AUC}	1.401298e-45	9.999975e-01	2.575427e-03
	δ_{Brier}	1.916556e-41	2.863830e-01	1.070646e-03
	$\delta_{Logloss}$	0.000000e+00	6.187968e-16	2.237852e-06
$\eta^{(6)}_{NE,MD,LR}$	δ_{AUC}	0.000000e+00	1.000000e+00	2.705704e-150
	δ_{Brier}	0.000000e+00	5.205595e-22	6.798990e-02
	$\delta_{Logloss}$	0.000000e+00	0.000000e+00	6.851411e-86

Table B.6: *Statistical p-values of the experimental scenarios in Cluster 6*

Bibliography

Bergstra, J., D. Yamins, and D. D. Cox (2013). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. Citeseer. 9

Brown, C. D. and H. T. Davis (2006). Receiver operating characteristics curves and related decision measures: A tutorial. *Chemometrics and Intelligent Laboratory Systems 80*(1), 24 – 38. 6

Brown, M. B. and A. B. Forsythe (1974). Robust tests for the equality of variances. *Journal of the American Statistical Association 69*(346), 364–367. 51

Chen, T. and C. Guestrin (2015). Xgboost: Reliable large-scale tree boosting system. xi, 15

Chen, T. and C. Guestrin (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794. ACM. 14

Couronné, R., P. Probst, and A.-L. Boulesteix (2018). Random forest versus logistic regression: a large-scale benchmark experiment. *BMC bioinformatics 19*(1), 270. 20

Dorogush, A. V., V. Ershov, and A. Gulin (2018). Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*. 22

Fisher, W. D. (1958). On grouping for maximum homogeneity. *Journal of the American statistical Association 53*(284), 789–798. 29

Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. *Annals of Statistics 29*, 1189–1232. 11, 13, 14

Friedman, J. H. and W. Stuetzle (1981). Projection pursuit regression. *Journal of the American Statistical Association 76*(376), 817–823. https://www.tandfonline.com/doi/abs/10.1080/01621459.1981.10477729. 11

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org. ix

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer. xi, 3, 4, 5, 6, 8, 9, 11, 12, 13

Hoos, H., U. Ca, and K. Leyton-Brown (2014). An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*, pp. 754–762. 48

Kaggle (2019). State of data science and machine learning 2019. https://www.kaggle.com/kaggle-survey-2019. 1

Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu (2017a). Lightgbm: A highly efficient gradient boosting decision tree. pp. 3146–3154. 11, 15

Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu (2017b). Parameters tuning. xi, 16

Kuhn, M. and K. Johnson (2013). *Applied predictive modeling*, Volume 26. Springer. xi, 4, 5, 6, 7, 8

Li, P. (2012). Robust logitboost and adaptive base class (abc) logitboost. *arXiv preprint arXiv:1203.3491*. 11

Maaten, L. v. d. and G. Hinton (2008). Visualizing data using t-sne. *Journal of machine learning research 9*(Nov), 2579–2605. 39

Mantovani, R. G., T. Horváth, R. Cerri, S. B. Junior, J. Vanschoren, A. C. P. d. de Carvalho, and L. Ferreira (2018). An empirical study on hyperparameter tuning of decision trees. *arXiv preprint arXiv:1812.02207*. 9

Marsh, B. (2016, 09). Multivariate analysis of the vector boson fusion higgs boson. xi, 13

McDonald, J. H. (2009). *Handbook of biological statistics*, Volume 2. 51

Montgomery, D. C. (2017). *Design and analysis of experiments*. John wiley & sons. xiii, 46, 47, 48, 49, 50, 51

Murphy, A. H. (1973). A new vector partition of the probability score. *Journal of applied Meteorology 12*(4), 595–600. 7

Probst, P., B. Bischl, and A.-L. Boulesteix (2018). Tunability: Importance of hyperparameters of machine learning algorithms. *arXiv preprint arXiv:1802.09596*. 1, 8, 20, 24, 59, 67

Prokhorenkova, L., G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin (2018). Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, pp. 6638–6648. 22

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR abs/1609.04747*. https://arxiv.org/abs/1609.04747. 11

Rufibach, K. (2010). Use of brier score to assess binary predictions. *Journal of clinical epidemiology 63*(8), 938–939. 5, 6, 7

Russell, S. J. and P. Norvig (2010). *Artificial Intelligence: A Modern Approach* (Third ed.). Prentice Hall. xi, 3, 9

Saito, T. and M. Rehmsmeier (2015). The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one 10*(3), e0118432. 71

Schapire, R. E. (1999). A brief introduction to boosting. *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, 1401–1406. http://dl.acm.org/citation.cfm?id=1624312.1624417. 12

Shapiro, A. D. (1987). Structured induction in expert systems. 32

Terpilowski, M. (2019). scikit-posthocs: Pairwise multiple comparison tests in python. *The Journal of Open Source Software 4*(36), 1169. 19, 55

Vallat, R. (2018, November). Pingouin: statistics in python. *The Journal of Open Source Software 3*(31), 1026. 19, 49

van Rijn, J. N. and F. Hutter (2017). An empirical study of hyperparameter importance across datasets. 48

van Rijn, J. N. and F. Hutter (2018). Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2367–2376. ACM. 1, 61, 63

Vanschoren, J., J. N. van Rijn, B. Bischl, and L. Torgo (2014, June). Openml: Networked science in machine learning. *SIGKDD Explor. Newsl. 15*(2), 49–60. 20

Wang, L., M. Feng, B. Zhou, B. Xiang, and S. Mahadevan (2015, September). Efficient hyper-parameter optimization for NLP applications. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, pp. 2112–2117. Association for Computational Linguistics. 67