PEDRO HENRIQUE CARVALHO DOS REIS

MACHINE LEARNING APPLIED TO CARD PAYMENT FRAUD DETECTION

São Paulo
2022

PEDRO HENRIQUE CARVALHO DOS REIS

MACHINE LEARNING APPLIED TO CARD PAYMENT FRAUD DETECTION

Work presented to the Polytechnic School of the University of São Paulo to obtain the Computer Engineer Title

São Paulo
2022

PEDRO HENRIQUE CARVALHO DOS REIS

MACHINE LEARNING APPLIED TO CARD PAYMENT FRAUD DETECTION

Work presented to the Polytechnic School of the University of São Paulo to obtain the Computer Engineer Title

Major:

Computer Engineering

Advisor:

Prof. Dr. Reginaldo Arakaki

São Paulo
2022

Prof. Reginaldo Arakaki

ACKNOWLEDGMENTS

This project was only possible because of the effort, directly or indirectly, of a group of people who supported me during this journey.

To my advisor, Prof. Dr. Reginaldo Arakaki, who spared no efforts to give me all possible support for the project realization. Thus thank you so much professor for the days in the Digital Systems Laboratory where we spent the mornings discussing the improvements and next steps to conclude the project.

To the professionals from bank industry, who, through professor Reginaldo, I had contact with and helped me clarify doubts and also suggested improvements. Your ideas and clarifications were of great help for me.

To the countless friends who, even without knowing or having knowledge of the project's subject, did not spare any effort to help me in the spelling maintenance of the monograph.

To my family members for staying with me during these last intense months of project and giving me all the support to keep me focused on the experiments to finish this work.

RESUMO

As apostas financeiras envolvidas na detecção de fraudes com cartões de pagamento são enormes e é um campo onde o desempenho e a decisão em tempo real são primordiais. Por enquanto, os algoritmos Gradient Boosted Decision Trees (GBDT) são os mais utilizados para este tipo de problema, pois é o algoritmo de aprendizado de máquina mais bem sucedido para dados tabulares. Entretanto, tem sido sugerido em trabalhos recentes que os métodos de aprendizagem profunda poderiam superar o GBDT em dados tabulares. A maioria dos estudos sobre ele compara os resultados em uma ampla gama de conjuntos de dados, mas nenhum deles se concentra na detecção de fraude. A questão da detecção de fraudes é particularmente complicada de ser abordada do ponto de vista do aprendizado de máquina, pois os bancos de dados disponíveis são compostos principalmente de pagamentos não-fraudulentos: isto é chamado de aprendizado desbalanceado. Como a detecção de fraudes é uma questão muito específica, foi decidido testar se os resultados obtidos de trabalhos recentes podem ser aplicados a este problema.

Palavras-Chave – detecção de fraude, aprendizado profundo, aprendizado de máquina, dados tabulares, dados desbalanceados, redes neurais, árvores de decisão, pagamentos fraudulentos.

ABSTRACT

The financial stakes involved in detecting card payment fraud are enormous and it is a field where performance and real-time decision are primordial. For now, Gradient Boosted Decision Trees (GBDT) algorithms are the most used for this kind of problem, as it is the most successful machine learning algorithm for tabular data. However, it has been suggested in recent papers that deep learning methods could outperform GBDT on tabular data. Most of the studies about it compare the results on a wide range of datasets, but none of them focus on fraud detection. The issue of fraud detection is particularly complicated to address from a machine learning perspective, as the available databases are composed mostly of non-fraudulent payments: this is called Imbalanced Learning. As fraud detection is a very specific issue, it was decided to test if the advanced results of recent papers can apply on this problem.

Keywords – fraud detection, deep learning, machine learning, tabular data, imbalanced data, neural network, decision tree, fraudulent payment

LIST OF FIGURES

LIST OF TABLES

CONTENTS

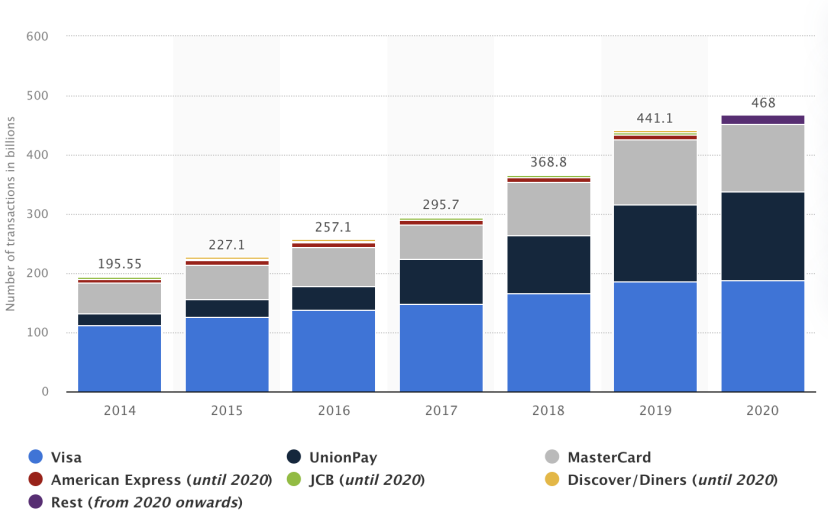
PART I

INTRODUCTION

1 INTRODUCTION

These days, payments using cash are increasingly rare and, although solutions such as checks have appeared it is inevitable that the majority of purchases are made from a bank card. According to a study published by Statista [1], 468 billion bank card transactions were made in 2020, a 6% growth over 2019.

Figure 1: Number of purchase transactions on global general purpose card brands from 2014 to 2020



Source: Study published by Statista [1].

Even though it is already a global trend, credit card payments are not completely secure. A study by NilsonReport [2] estimates that in the next 10 years, $408.5 billion will be due to bank card payment fraud. These frauds are made possible by several factors, including card loss, improper sharing of card data, or even cardholder misuse. Therefore, financial corporations must have a fraud detection system in place before validating payments.

Historically, anti-fraud systems were based on a pre-programmed set of rules that highlights a payment as fraudulent, e.g, in the past, payments made on holidays, where

the user did not inform the bank that he would be traveling, were considered as a fraud. However, nowadays with online shopping, fraudsters have much more flexibility when it comes to fraud, making these ruleset systems weak to detect frauds.

With computational advancement in recent decades, computer data processing has increased significantly, allowing technologies such as machine learning (ML) to enter the domain of bank fraud detection. In this type of system, a ML model analyzes historical data and learns from it in order to be more efficient in fraud detection. Some benefits of this type of system are:

- **Speed of detection:** fraud is never an expected event, so the system needs to detect it quickly. ML enables this detection in real time;
- **Manages mass amounts of data:** with traditional systems, fraud detection can be a laborious task as agents may have to wade through data manually to discover discrepancies. Credit card fraud detection using ML is able to analyze mass amounts of data quickly so inconsistencies or connections between various transactions can be found faster;
- **Adaptation to market changes:** unlike traditional models that need to rewrite their rules from time to time, with ML it is possible to monitor new market trends, learning to better detect the latest fraudsters practices.

This project will study and implement ML models for fraudulent credit card payments. It will consider recent findings and practices in two types of algorithms in the ML field, Gradient Boosted Decision Trees (GBDT) and Deep Learning (DL).

PART II

RELATED WORKS

2 RELATED WORKS

As mentioned in Section 1, the types of models considered in this project will be GBDT and DL. The first one, as we will see, is considered the state-of-art when dealing with tabular datasets, however recent papers have challenged this statement by proposing neural network (NN) based models that outperform GBDT. In this section it goes a short description of each considered paper and its results.

2.1 Well-tuned Simple Nets Excel on Tabular Datasets

Focusing on the domain of tabular datasets, [3] studied improvements in deep learning with better regularization techniques. It shows that the key to improving neural network performance on tabular data lies in exploiting the joint and simultaneous application of a large set of modern regularization techniques such as batch normalization, data augmentation, weight decay, dropout, look-ahead optimizer and so on. Thus, even a simple multilayer perceptron (MLP) achieves state-of-the-art results when conditioned by several modern regularization techniques applied simultaneously. Then [3] provides a method for regularizing a simple MLP network by searching for the best combination/cocktail of 13 regularization techniques for each dataset and providing an understanding of which regularizers to apply.

With 13 regularizations, a large-scale experiment with 40 datasets show that modern deep learning regularization methods developed in the context of raw data (e.g., vision, speech, text) can significantly improve the performance of deep neural networks on tabular data and outperform XGBoost, the current state-of-the-art method for tabular datasets. The regularized MLP beat recent deep learning architectures on 38 of 40 datasets and 26 of 40 if compared to XGBoost.

2.2 Revisiting Deep Learning Models for Tabular Data

[4] describes the state-of-the-art in deep learning for tabular data and improves the state of baselines by identifying several simple yet powerful deep architectures.

In fact, a large number of deep learning models for tabular data have been developed in recent years. However, well-studied DL architectural blocks have not been fully explored in the context of tabular data, so they are rarely used to design better baselines. With this, the authors build on well-established architectures that are well known in other fields and obtain two simple tabular data designs. The first is a residual network (ResNet) that can serve as an efficient baseline, and the second is the feature tokenizer transformer (FT-T), a simple adaptation of the transformer architecture that outperforms other DL solutions on tasks like natural language processing and computer vision.

Comparisons of these models are performed on a wide range of public datasets using the same training and hyperparameter tuning protocols in order to investigate their relative performance.

First, it was found that none of the others DL models could systematically outperform the ResNet model. Due to the ResNet model simplicity, it can serve as a solid foundation for future work. It has actually the best results for 6 datasets out of 11. Furthermore, FT-T shows the best performance in most tasks, becoming a powerful new solution in this field. Interestingly, FT-Transformer turns out to be a more general architecture for tabular data because it performs well on a wider range of tasks compared to more traditional ResNet and other DL models. The new baseline is also compared with GBDTs such as XGBoost and CatBoost, showing that GBDT still performs better on some tasks. Overall, FT-Transformer provides competitive performance on all tasks, while GBDT and ResNet perform well only on some subsets of the tasks. Concluding that there is still no general solution for tabular data among GBDT and deep models.

PART III

PRELIMINARY CONSIDERATIONS

3 PRELIMINARY CONSIDERATIONS

Before moving on to the experimental and procedural part, in order to improve the reader's understanding of the project, this section will define and clarify some key points, as well as justify some decisions made and list the used technologies.

3.1 Programming Language, Frameworks and Technologies

The project was developed in python, a programming language widely used by the academic and professional environment for data acquisition, processing and analysis. Being a high-level language and providing open-source libraries, python is a powerful tool for implementing the data pipelines required for training machine learning models.

The main libraries and frameworks used with python are:

- **Pandas:** a library for data manipulation and analysis. In particular, it provides structures and operations for manipulating numeric tables;
- **Scikit Learn:** an open source machine learning library. In the project scope, it was used for encoding the numerical features and evaluating the models;
- **Category Encoders:** using the same syntax pattern as scikit-learn, it is a library with specific transformation for categorical features;
- **PyTorch:** an open-source framework optimized for prototyping and implementing deep learning models. Representing matrices, tensors are the basic unit of PyTorch and their computation can be accelerated in GPU-enabled environments.

3.1.1 Optuna

Most machine learning models have dozens of parameters that can be optimized in order to improve its performance. In several occasions in this project it will be necessary to tune model's hyperparameters and when the number of parameters is large enough, making a grid search impossible, the optuna [5] library will be used for this purpose.

Optuna [6] is one of the optimization libraries that use sequential model based optimization (SMBO). Conventional optimization approaches like grid search and random search do not make use of the information available regarding the previously explored hyperparameter search space. Unlike those techniques, SMBO use the historical exploration information to make a more informed decision regarding the direction in which the optimizer should explore next in the search space. Such a mechanism gives SMBO an edge over the conventional approaches in terms of convergence speed.

3.1.2 Computational Platform

The computational platform used to develop the project was the Google Colaboratory, an open-source machine learning environment that combines, in an interactive way, executable python code with rich text and charts, the well-known notebooks. Running a stable python version, it also helps the process of managing the different libraries used. In addition, Google Colaboratory provides the allocation of GPUs to its notebooks. The GPUs coupled with PyTorch accelerate the training of deep learning models compared to a CPU-only environments.

On the other hand, because it is a free platform, Colab allocates the resources dynamically [7], not allowing the user to customize his environment, and does not guarantee the constancy of resources over time, so two runs of the same process may vary in terms of execution time.

3.2 Dataset

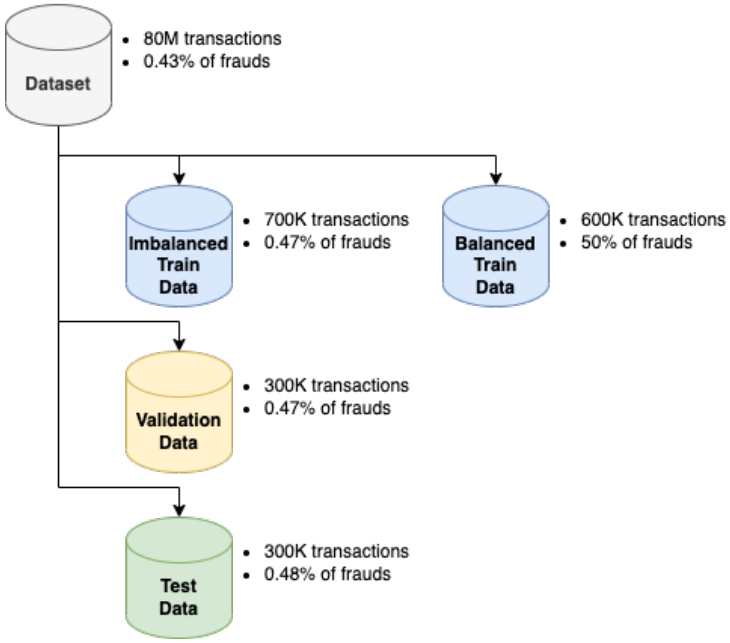
The dataset is composed by 80 million of credit card transactions labeled as fraudulent or not. Also, it is very imbalanced as less than 1% of the transactions are fraudulent. The Table 1 shows the dataset properties.

#samples	80M
#variables	142
#numerical variables	134
#categorical variables	8
imbalance rate	0.43%

Table 1: Dataset properties

The computational resource used did not support using all 80 million samples to train and validate the models. So, based on Figure 2, four datasets were generated from the original one.

Figure 2: Dataset split configuration



Source: Author's compilation.

- **Imbalanced Train Data:** used to train the models as the data is in the real world, imbalanced;
- **Balanced Train Data:** used to train the models in a fully balanced way;
- **Validation Data:** used to validate the models, i.e, check overfitting/underfitting and tuning the hyperparameters;
- **Test Data:** used to test the models. After training and validation, all the metrics will be calculated using this dataset.

It is important to mention that both the validation and testing data are imbalanced, as they need to represent the behavior of the real data.

Little effort was required to preprocess the data because the dataset was already optimized, i.e, there were no missing values and all variables shared the same universe of measurement units. Thus, preprocessing was based on just encoding the numerical and categorical features before training the models.

For the numerical features, four encoders from the *scikit-learn* library were considered:

1. **None:** x
2. **Standard:** $\frac{x-x_{mean}}{x_{std\_deviation}}$
3. **MinMax:** $\frac{x-x_{min}}{x_{max}-x_{min}}$
4. **Robust:** $\frac{x-x_{median}}{x_{IQR}}$, which suffers less influence from outliers

For the categorical features, three encoders from the *category_encoders* library [8] were considered: *Target*, *CatBoost* and *MEstimate*. It was decided to focus on target encoding (catboost and mestimate being derived from it) for two reasons. First, one hot encoding wasn't an option because the number of features would have been huge since the categorical features took too many different values. We also considered using a label encoding, but after a quick test we decided to put this option aside as it wasn't showing great results.

3.3 Evaluation Metrics

In order to evaluate and compare different models when predicting the credit card frauds, it's necessary to establish a evaluation metric to rank these models. Considering the dataset properties from Table 1, we are dealing with an Imbalanced Binary Classification problem because it's needed to classify the input data into two possibilities, fraud or non fraud, being the frauds representing 0.43% of total data. Thus, the first metric that came in mind is the confusion matrix that in this case it is a 2x2 matrix representing all classified instances of the test dataset.

	Predict 0	**Predict 1**
Actual 0	True Negative	False Positive
Actual 1	False Negative	True Positive

Table 2: Confusion matrix

Translating the true/false and positive/negative nomenclature to the project, it gives:

- **True Negative:** the transaction is not a fraud and it was classified as non fraud;
- **False Positive:** the transaction is not a fraud, but it was classified as a fraud;
- **False Negative:** the transaction is a fraud, but it was classified as non fraud;
- **True Positive:** the transaction is a fraud and it was classified as a fraud;

Based on the confusion matrix, a straight to go metric would be the accuracy that is defined as $\frac{TN+TP}{TN+FP+FN+TP}$. However, for imbalanced problems, it does not represent a good metric, why?

Considering the test dataset, from 300K transactions it has 1370 frauds. Now imagine a poorly performing model that predicts every transaction as not fraudulent, so you would have the following confusion matrix:

	Predict 0	Predict 1
Actual 0	298630	0
Actual 1	1370	0

And the accuracy will be $\frac{298630+0}{298630+0+1370+0} = 0.995$, a really good score. It happens because the accuracy gives equal priority predicting both classes, the positive and negative.

Other metrics based on the confusion matrix that most rely on the minority class are the precision and the recall, that are defined as:

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

Defined as the harmonic mean between the precision and the recall, the F1 score [9] is a better metric when dealing with imbalanced problems.

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

PART IV

EXPERIMENTS

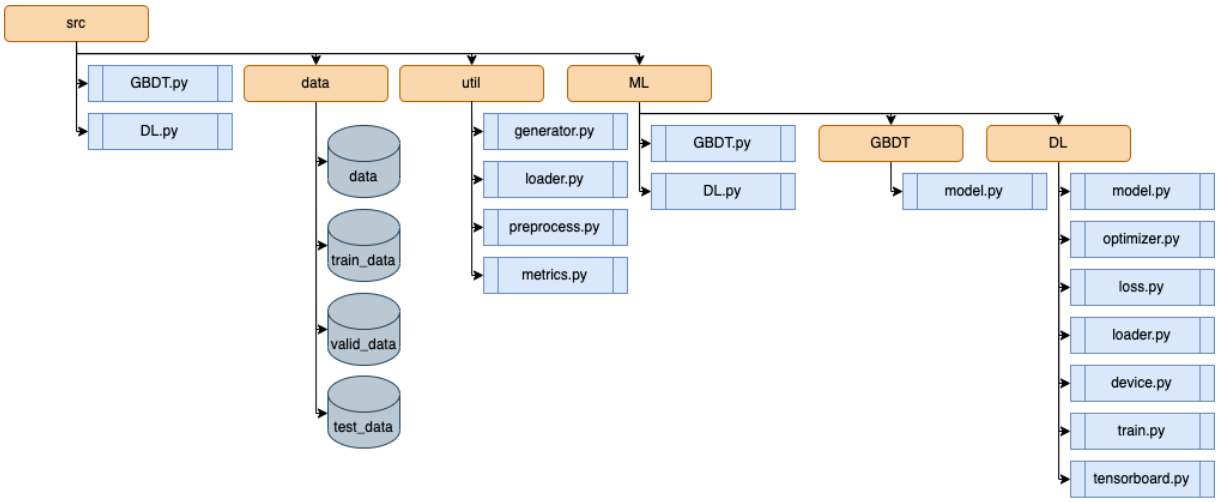
4 EXPERIMENTS

In this section, based in [3] and [4], for each considered model, it follows a brief introduction of it and a step by step description of the experiments and decisions that were made. The goal is to find the most performing model for this dataset and also compare GBDT and DL structures. For GBDT, XGBoost and LightGBM will be implemented and for DL, it was chosen MLP and ResNet as classical models and FT-T and XBNet (more later) as suitable models for tabular data.

4.1 Code Structure

The code was developed in a modular way in order to reuse as many functions as possible. For each type of model architecture studied, GBDT and DL, packages were created to facilitate the reuse and improve code understanding. The Figure 3 shows the hierarchical structure of the written code.

Figure 3: The code structure



Source: Author's compilation.

- **src/data:** contain all datasets, the original, train, validation and test ones;

- **src/util:** contain the utility functions used for all models. The *generator.py*, *loader.py* and *preprocess.py* files are reponsible for, respectively, generating, loading and preprocessing the training, validating and testing datasets. And the *metrics.py* file contain the functions to evaluate the models, calculating metrics like the confusion matrix and the F1 score;

- **src/ML:** contain the scripts to execute one training process of a GBDT or DL model. It's the core files of the project;

- **src/ML/GBDT:** contain the *model.py* file that select from the XGBoost or LightGBM model;

- **src/ML/DL:** contain all the files related for training a neural network model. For example, in the *model.py* file it's possible to select from MLP, ResNet, FT-T or XBNet, while the *loader.py* prepares the data using tensors, PyTorch's basic unit;

- **src:** contain the root files *GBDT.py* and *DL.py* that configures the optuna hyperparameter tuning. It essentially has a dictionary with all possible parameters to be set manually and call the train script located in **src/ML** until the tuning is finished.

4.2 Gradient Boosted Decision Trees

GBDT is a machine learning technique for optimizing the predictive value of a model through successive steps in the learning process. Like other boosting methods, gradient boosting combines weak learners (decision trees) into a single strong learner in an iterative fashion. At each iteration, each sample passes through the decision nodes of the newly formed tree until it reaches a given lead. The result of each tree is used to get the GBDT prediction.

It was decided to use two GBDT algorithms, XGBoost and LightGBM (that have python libraries with the same name). This kind of algorithms are known to perform well on tabular data for classifications problems, so it is a great baseline to compare performances.

4.2.1 Preprocessing

Using model's default parameters, it was considered all the 12 combinations of data preprocessing (4 numerical encoders and 3 categorical encoders) and a grid search was

scheduled for both models. The results among the 12 combinations did not change at all, showing that GBDT models do not suffer from these feature encoding. Thus, the combination **no numerical encoder** and **mestimate encoder** was selected. The Tables 3 and 4 summarizes the evaluation of the models on the test data, respectively, using imbalanced and balanced training data.

Model	TN	FP	FN	TP	F1 Score
XGBoost	298525	105	1097	273	31.24
LightGBM	297905	725	1074	296	24.76

Table 3: GBDT preprocessing results using imbalanced training data

Model	TN	FP	FN	TP	F1 Score
XGBoost	255294	43336	179	1191	5.19
LightGBM	252562	46068	182	1188	4.89

Table 4: GBDT preprocessing results using balanced training data

At this point, training the models using balanced data was discarded since the results were much worse than imbalanced training for the context of credit card fraud prediction. Section 5, about results and discussion, will further discuss this project decision. Thus, for the next experiments only imbalanced training data will be considered.

4.2.2 Hyperparameter Tuning

For the hyperparameter tuning, just the best GBDT model from Table 3 was selected, the XGBoost. Based on [3] and community best practices for tuning XGBoost hyperparameters, 50 trials were scheduled in optuna using a predefined search space. The Table 5 shows the best set of hyperparameters.

Hyperparameter	**Value**
n_estimators	355
max_depth	11
alpha	4.36×10^{-3}
lambda	1.89×10^{-5}
min_child_weight	14.33
gamma	9.95×10^{-1}
learning_rate	4.51×10^{-2}
scale_pos_weight	$2\sqrt{\frac{\#positives}{\#negatives}} \approx 30$

Table 5: Best hyperparameter configuration set for XGBoost

In addition, a early_stopping_rounds equal to 20 was set to prevent overfitting and reduce the training time.

The Table 6 shows the results of the tuned model on the test data.

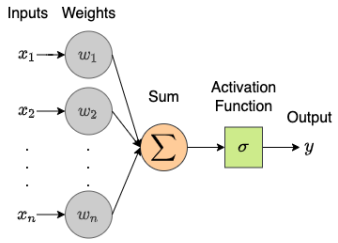
Model	TN	FP	FN	TP	F1 Score
XGBoost	297987	643	846	524	41.31

Table 6: XGBoost result after hyperparameter tuning

4.3 Multilayer Perceptron

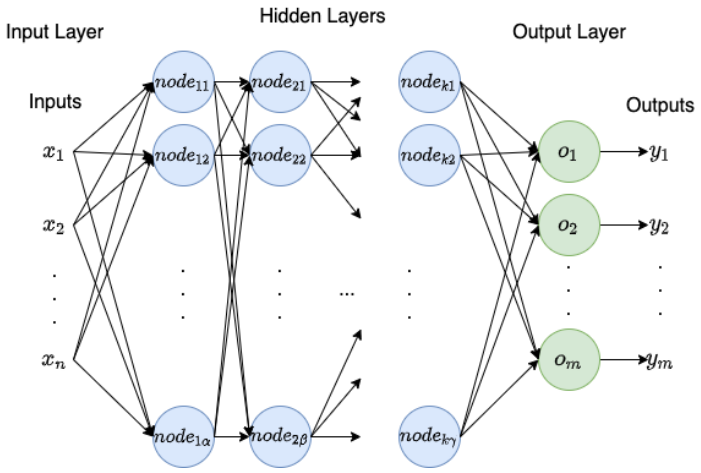
MLP is a deep learning model that combines several neuron units, the perceptron, into layers. This composition allied with activation functions make MLP learn and extract non-linear data relationship.

Figure 4: Perceptron



Source: Author's compilation.

Figure 5: Multilayer Perceptron



Source: Author's compilation.

As showed in Figure 5, the MLP have three types of layers. The input layer where the input feature are placed. Then there are the hidden layers that can be one, two or more. Each node in the hidden layer is a single perceptron (Figure 4) that make the multiplication of each entry, sum up the result and then apply an activation function, having a single output. Finally the output layer is a special perceptron with, in general, an activation function that the result can be interpreted as a probability in classification problems.

The training process is based in two steps. First, a feedforward is conducted where the input goes from the input layer to the output layer, then a predefined cost function

is calculated taking in consideration the prediction and the real value. After that it is possible to perform the backpropagation step where the loss function derivatives is calculated and a gradient descent is performed to update the weights of all nodes. After some iterations the MLP learn the main patterns of a dataset.

Being the simplest NN architecture, this model was developed from scratch using PyTorch, a python framework to implement NN models. The goal of this section is to have a simple DL model to be a baseline for the next ones and, considering paper [4], to see if the regularization cocktail proposed outperform GBDT and a non regularized MLP.

As a initial point, considering [4] default parameters and inspecting the problem, it was decided to start with the parameters expressed in Table 7.

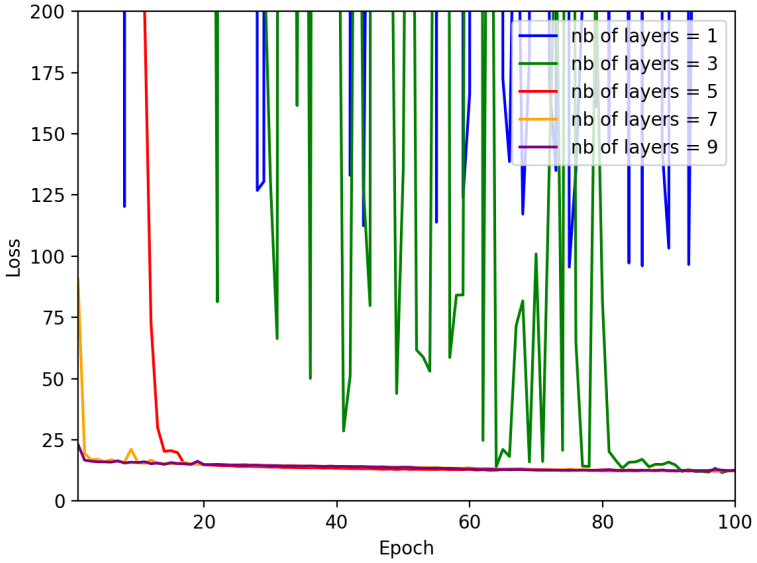
Parameter	**Value**
Epoch	100
Batch Size	512
Number of nodes	512
Learning Rate	10^{-4}
Categorical Encoder	MEstimate
Numerical Encoder	Standard

Table 7: MLP parameters

4.3.1 Number of Layers

The first step was optimizing the number of hidden layers and for that, a grid search was schedule considering {1, 3, 5, 7, 9} hidden layers. The Figure 6 shows, for each number of hidden layers, the loss function curve over the epochs. It's possible to notice that just after 5 hidden layers that the loss function stop to be caotic and start to converge. Thus, 7 hidden layers was selected to continue the experiment because it converges faster than 5 and it is not as complex as having 9 hidden layers.

Figure 6: Loss function over the epochs for each number of hidden layer considered in the tuning

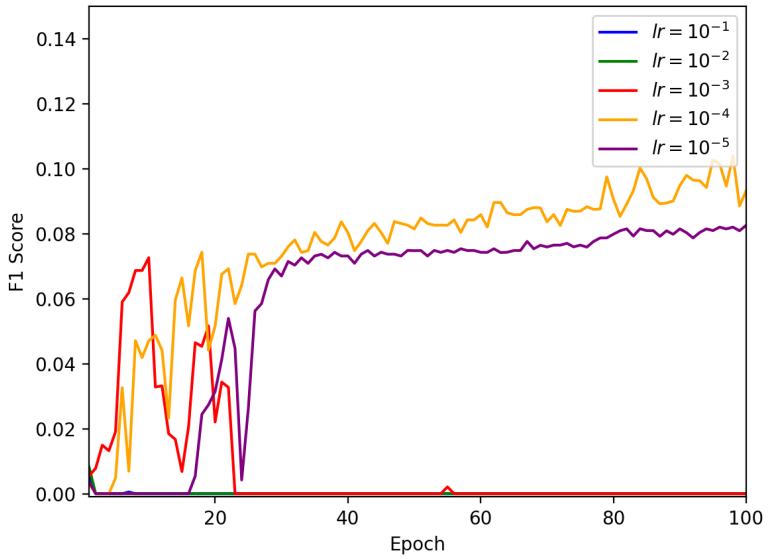


Source: Author's compilation.

4.3.2 Learning Rate

The same kind of search was done with the learning rate in the set $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ and analyzing the F1 score over the epochs in Figure 7, it was decided that 10^{-4} is the best choice since the curve fits better.

Figure 7: F1 score over the epochs for each learning rate considered in the tuning



Source: Author's compilation.

Model	TN	FP	FN	TP	F1 Score
MLP	298614	16	1293	77	10.53

Table 8: MLP result after tuning the number of hidden layers and learning rate

4.3.3 Preprocessing

As a next step, the preprocessing type was decided to be changed and like it was done with GBDT, all the 12 combinations were considered. As result, fixing the numerical encoder, none of the categorical encoder showed improvement from each other, so mestimate transformation remained. However, fixing the categorical encoder, the robust transformation was clearly better than others. The Table 9 show that just changing the numerical encoder from standard to robust, increased model's performance by 10%.

Model	TN	FP	FN	TP	F1 Score
~~MLP~~	~~298614~~	~~16~~	~~1293~~	~~77~~	~~10.53~~
MLP	298535	95	1201	169	20.67

Table 9: MLP - Partial Results II

4.3.4 Regularization Cocktail

Now, based on [3], the regularization cocktail was implemented. However, due to the lack of available documentation, 5 out of 13 regularizers were implemented and instead of the proposed hyperparameter search space, the optuna library was used to do the tuning. With the five regularization techniques, 25 trials were scheduled in optuna using the values suggested by [3].

Regularizer	**Active**	**Value**
Batch Normalization	False	- -
Stochastic Weight Averaging	False	- -
Lookahead Optimizer	True	Step Size = .5 # Steps = 5
Weight Decay	False	- -
Dropout	False	- -

Table 10: Best set of regularizers parameters

Model	TN	FP	FN	TP	F1 Score
MLP	298535	95	1201	169	20.67
R-MLP	298520	110	1199	171	20.71

Table 11: Results of the non regularized (MLP) and regularized (R-MLP) MLP versions

From Table 10, just one out five regularizers was selected to be active and Table 11 show that the regularized version could not improve model's performance. Section 5, about results and discussion, will further discuss about why regularizers could not increase the model's performance.

4.3.5 Loss Function

Considering the non regularized MLP, as another method to improve its performance, the loss function will be modified. Until now the binary cross entropy (BCE) 4.1, that gives the same weight to both positive and negative classes, was used. Two other loss functions will be analyzed, the positive weight binary cross entropy (PWBCE) and the focal loss (FL), which aim to change the penalty of the minority class in order to compensate the imbalancing.

Considering y as the true value and $\hat{y}$ as the prediction probability, it follows:

Binary Cross Entropy

$$BCE(y, \hat{y}) = -ylog(\hat{y}) - (1-y)log(1-\hat{y}) \tag{4.1}$$

Positive Weight Binary Cross Entropy

$$PWBCE(y, \hat{y}) = -pylog(\hat{y}) - (1-y)log(1-\hat{y}) \tag{4.2}$$

The PWBCE is a variant of the BCE that adds a weight, p, to the minority class. According to the documentation [10], $p = \frac{\text{\# negative}}{\text{\# positive}}$, however, for highly imbalanced problems, community best practices suggests $p = \beta\sqrt{\frac{\text{\# negative}}{\text{\# positive}}}$. To tune the β parameter, it was considered the values {0.5, 1, 2, 3} and a grid search was performed.

Focal Loss [11]

$$FL(y, \hat{y}) = -y\alpha(1-\hat{y})^{\gamma}log(\hat{y}) - (1-y)(1-\alpha)\hat{y}^{\gamma}log(1-\hat{y}) \tag{4.3}$$

As another variant of the BCE, FL adds more flexibility to the calculation of the loss function, since it has two parameters, α and γ. According to [11], values of $\alpha = 0.25$ and $\gamma = 2$ are recommended, however we will tune them using a two-step grid search. First, we set $\gamma = 2$ and varied α in {0.6, 0.7, 0.8, 0.9, 0.99}, then with α_{best} value, it was varied γ in {0, 1, 2, 3, 4}.

Loss	TN	FP	FN	TP	F1 Score
BCE	298535	95	1201	169	20.67
PWBCE($\beta = 0.5$)	297250	1380	994	376	24.06
FL($\alpha = 0.9, \gamma = 0$)	296524	2106	899	471	23.87

Table 12: MLP results for each loss function

From Table 12, that shows the best result for each loss function, the PWBCE with $\beta = 0.5$ could improve the standard BCE from 20.67% to 24.06%

4.4 Residual Network

ResNet models are major contributors to the use of very deep neural networks especially convolutional neural networks (CNN), by introducing the concept of residual

learning and devised an efficient method for the training.

As mentioned in [4], being a simple NN model, the ResNet-like architecture achieved good performances, outperforming GBDT in some datasets. Thus, using paper's library, the proposed ResNet model was adapted to our dataset.

The training parameters setting is equal to the ones used for the MLP in the previous section. The Table 13 shows them.

Hyperparameter	Value
Epoch	100
Batch Size	512
Learning Rate	10^{-4}
Categorical Encoder	MEstimate
Numerical Encoder	Robust

Table 13: The ResNet training parameters

Using optuna, 25 trials were scheduled over the predefined configuration set recommended by [4]. Table 14 shows the set of hyperparameters for the best trial achieved.

Hyperparameter	**Value**
# Layers	14
Layer size	566
Hidden factor	1.10
Hidden dropout	0.19
Residual dropout	0.39
Activation	ReGLU
Normalization	LayerNorm

Table 14: ResNet model best trial's hyperparameter set

The model's performance on the test data is summarized in the Table 15.

Model	TN	FP	FN	TP	F1 Score
ResNet	298535	95	1197	173	21.12

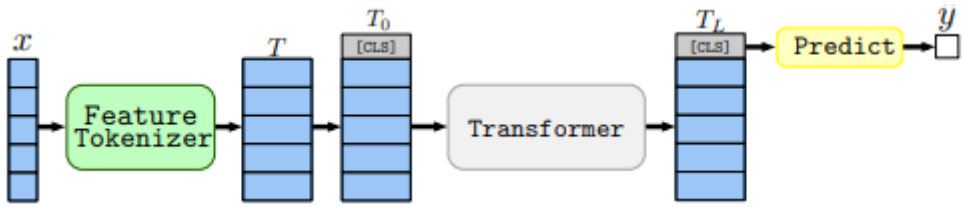
Table 15: ResNet result

4.5 Feature Tokenizer - Transformer

FT-T [4] is a simple adaptation of the transformer architecture [12] for tabular domains. Figure 8 shows the main components of the FT-Transformer. In short, this model converts all features (categorical and numerical) into embeddings and applies several ransformer layers to the embeddings. Therefore, each transformer layer operates on the feature level of the object and the final representation of the [CLS] token is used for prediction.

As a downside, FT-T requires more resources (hardware and time) to train and do not scale easily if the number of features is large.

Figure 8: FT-T architecture



Source: FT-T paper [4].

Using [4]'s library and recommendations, the FTT was implemented using default parameters. It took more than 3 hours to be trained and Table 16 shows its result of the single run on test data.

Model	TN	FP	FN	TP	F1 Score
FTT	298625	5	1277	93	12.67

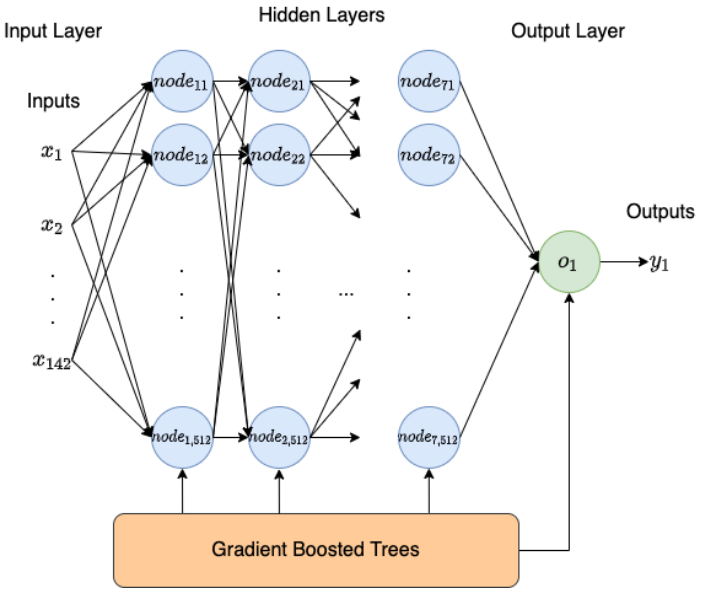
Table 16: FTT result

4.6 eXtremely Boosted Network

As the last model, XBNet [13] [14] is a MLP that the weights update step, besides the standard gradient descent, is also supported by the feature importance of concurrent trained XGBoost models.

[13] compared XBNet with XGBoost and showed that the proposed model beat its adversary in 4 of 7 dataset. A downside is that XBNet is time and resource consuming since it need to train a gradient boosted decision tree in every hidden layer.

Figure 9: XBNet architecture



Source: Author's compilation based in XBNet paper [13].

Based on [13]'s library, the XBNet model was adapted to our dataset using the same configurations from the non regularized MLP already presented. Taking around 8 hours to train, the Table 17 summarize the result of the only run that was made.

Model	TN	FP	FN	TP	F1 Score
XBNet	298421	209	1287	83	9.90

Table 17: XBNet result

PART V

RESULTS & DISCUSSION

5 RESULTS & DISCUSSION

5.1 Imbalanced vs Balanced Training

Before tuning the XGBoost, LightGBM and MLP models, it was checked whether it was better to train them using imbalanced or balanced data. The Tables 18 and 19 show, respectively, the result considering imbalanced and balanced training for each model.

Model	TN	FP	FN	TP	F1 Score
XGBoost	298525	105	1097	273	31.24
LightGBM	297905	725	1074	296	24.76
MLP	298535	95	1201	169	20.67

Table 18: Imbalanced training result

Model	TN	FP	FN	TP	F1 Score
XGBoost	255294	43336	179	1191	5.19
LightGBM	252562	46068	182	1188	4.89
MLP	231081	67549	287	1083	3.10

Table 19: Balanced training result

The discussion of this comparison depends on the domain where the imbalanced dataset is applied. The F1 score shows the imbalanced training as superior, but looking carefully at the confusion matrix it's possible to see that, focusing in the best model, the XGBoost, it could classify correctly $\frac{1191}{1191+179} = 86.9\%$ of all fraudulent payments, in counterpart the number of non fraudulent payments classified incorrectly increase from 105 to 43336.

In the domain of credit card fraud detection the number of frauds that are detected is, obviously, important but not sacrificing the good transactions is almost equally important. In this case, it's possible to interpret that in one hand the system could prevent

86.9% of frauds, but in the other hand others 43336 transactions were badly classified as fraud, blocking client's transactions mistakenly, it means that the true positives do not compensates the false positives.

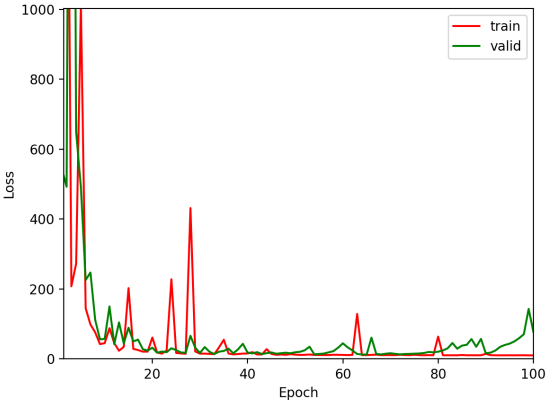
Thus, because of that interpretation, the equally balanced training do not represents a good approach for the business domain of credit card fraud detection.

5.2 Unregularized vs Regularized MLP

As seen previously in Section 4.3.4, the MLP regularizers tuning just activate one regularization technique, the lookahead optimizer, but none considerable improvement was observed.

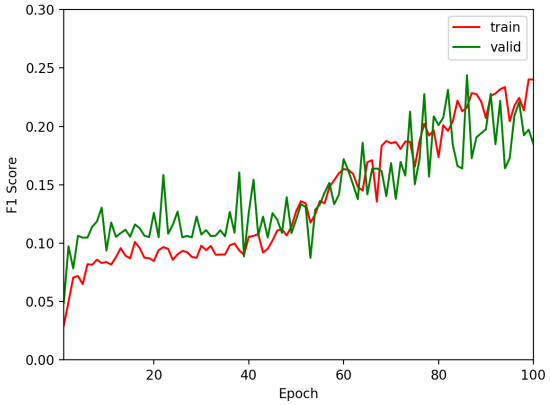
In fact, the regularization techniques are used to control model overfitting [15] [16] [17] and regarding the loss and F1 score curves of the unregularized MLP in Figures 10 and 11, it's possible to see that there is no sign of overfitting. That is why the regularization could not improve the results.

Figure 10: MLP loss curve



Source: Author's compilation.

Figure 11: MLP score curve



Source: Author's compilation.

5.3 GBDT Supremacy

Model	TN	FP	FN	TP	F1 Score	Train Time
XGBoost	297987	643	846	524	41.31	10min 3s
LightGBM	297905	725	1074	296	24.76	6.6s
MLP	297250	1380	994	376	24.06	25min
ResNet	298535	95	1197	173	21.12	57min 13s
FTT	298625	5	1277	93	12.67	3h 13s
XBNet	298421	209	1287	83	9.90	8h

Table 20: Model's result. *t-* means a tuned model

From Table 20 that summarizes all model's best result, it's unquestionable that GBDT is a better choice than DL, especially with XGBoost that achieved a F1 score of 41.31% in comparison to MLP (the best DL model) that scored a F1 of 24.06%.

In addition, the GBDT took less time to be trained than the DL models. Without any tuning, the LightGBM performed better than the MLP and only took 6.6 seconds to be trained.

Another advantage is the implementation: the GBDT models came with well-defined libraries with the same name, *xgboost* and *lightgbm*, and having the *scikit-learn* code style, they are easy to implement. With some lines of code, it's possible to train a GBDT model. In counterpart, despite of DL models having libraries like *pytorch* that provides more flexibility when designing and implementing a model, it's not straightforward as GBDT.

5.4 Deep learning models

It's clear that classical models, the MLP and ResNet, performed better than the tabular data designed DL models, FTT and XBNet. The MLP and ResNet had similar performances, 24.06% and 21.12% of F1 score respectively and reasonable training time with 25 and 57 minutes respectively, what makes the hyperparameter tuning possible.

The tabular data designed models were placed as the worst models, with a score of 12.67% for the FT-T and 9.90% for the XBNet. One big issue is that they took hours to be trained, so both were executed just one time, the FT-T with paper's parameters and the XBNet with the MLP training configuration.

PART VI

CONCLUSION

6 CONCLUSION

First of all, this project showed how important it is to analyze and adapt the training parameters of the models to the dataset used, for example, in the MLP, simply changing numerical variable encoder from standard to robust had a gain of 10% in F1 score. Furthermore, when optimizing the XGBoost and MLP models, it evinced that adapting the loss function is a crucial factor to compensate the data imbalance.

In addition, when comparing GBDT with NN, the results showed that DL, whether classical or adapted to tabular data, could not defeat GBDT for the imbalanced tabular data used. Thus, GBDT remains the state-of-art for this type of data.

Another important factor is that the best model, XGBoost, after the optimization and tuning runs, improved its performance considerably, going from 31.24% to 41.31% of F1 score, proving to be a significant model for card fraud detection, as it managed to keep false positives low and increase its true positives. However, it can't be said that it detects most frauds (recall of 38.25%), so to put this system in production one can take into consideration the mutual action of this proposed model with the already established rule-based models installed in the banking institution, increasing the redundancy in fraud detection.

Finally, as next steps, it remains improving the XGBoost performance by considering some techniques that were outside the scope of this project:

- **Feature Selection:** the dataset contains 142 variables, however are all of them relevant to the problem? Irrelevant variables lead to redundant and noisy data which, besides increasing training time, reduces model performance. More specifically for XGBoost, the models contain a piece of information called feature importance, which measures the degree of how relevant each feature is to the model. Thus, the less important features could be remove based on a pre-established threshold;
- **Data Oversampling:** the project showed that fully balanced data is not a good practice for the fraud detection domain, however what if an imbalance of 1%, 2%,

5% or 10% is considered instead of the actual 0.5%? Decreasing the data imbalance together with adapted cost functions could increase the model's performance.

REFERENCES

[1] "Visa, mastercard, unionpay transaction volume worldwide 2014-2020." [Online]. Available: https://www.statista.com/statistics/261327/number-of-per-card-credit-card-transactions-worldwide-by-brand-as-of-2011/

[2] "Credit card fraud." [Online]. Available: https://nilsonreport.com/upload/content_promo/NilsonReport_Issue1209.pdf

[3] A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka, "Well-tuned simple nets excel on tabular datasets," 2021.

[4] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," 2021.

[5] "Optuna." [Online]. Available: https://optuna.org/

[6] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019. [Online]. Available: https://arxiv.org/abs/1907.10902

[7] "Google colaboratory." [Online]. Available: https://research.google.com/colaboratory/faq.html

[8] "Target encoding." [Online]. Available: https://contrib.scikit-learn.org/category_encoders/targetencoder.html

[9] "Good f1 score." [Online]. Available: https://stephenallwright.com/good-f1-score/

[10] "Bce documentation in pytorch." [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

[11] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," 2018.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[13] T. Sarkar, "Xbnet : An extremely boosted neural network," 2021.

[14] ——, "Library xbnet for tabular data which helps you to create a custom extremely boosted neural network," https://www.codeocean.com/, 6 2021.

[15] "Regularization in deep learning — l1, l2, and dropout." [Online]. Available: https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036

[16] "Regularization techniques for training deep neural networks." [Online]. Available: https://theaisummer.com/regularization/

[17] "An overview of regularization techniques in deep learning (with python code)." [Online]. Available: https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

[18] G. Somepalli, M. Goldblum, A. Schwarzschild, and C. B. Bruss, "Saint: Improved neural networks for tabular data via row attention and contrastive pre-training," 2021.

[19] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1706.02515

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[21] S. Sun and M. Iyyer, "Revisiting simple neural probabilistic language models," 2021. [Online]. Available: https://arxiv.org/abs/2104.03474

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805

[23] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao, "Learning deep transformer models for machine translation," 2019. [Online]. Available: https://arxiv.org/abs/1906.01787

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[25] L. Liu, X. Liu, J. Gao, W. Chen, and J. Han, "Understanding the difficulty of training transformers," 2020. [Online]. Available: https://arxiv.org/abs/2004.08249

[26] X. S. Huang, F. Perez, J. Ba, and M. Volkovs, "Improving transformer optimization through better initialization," pp. 4475–4483, 13–18 Jul 2020. [Online]. Available: https://proceedings.mlr.press/v119/huang20f.html

[27] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," 2020. [Online]. Available: https://arxiv.org/abs/2009.06732

[28] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon, "Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020," 2021. [Online]. Available: https://arxiv.org/abs/2104.10201

[29] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017. [Online]. Available: https://arxiv.org/abs/1711.05101

[30] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M.

Rush, "Huggingface's transformers: State-of-the-art natural language processing," 2019. [Online]. Available: https://arxiv.org/abs/1910.03771

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015. [Online]. Available: https://arxiv.org/abs/1502.01852

[32] "Rtdl python library." [Online]. Available: https://yura52.github.io/rtdl/stable/index.html

APPENDIX A – HYPERPARAMETER TUNING CONFIGURATION SET

A.1 XGBoost

Hyperparameter	**Distribution**
n_estimators	Integer(50, 500)
max_depth	Integer(1, 12)
alpha	LogUniform(10^{-8}, 1)
lambda	LogUniform(10^{-8}, 1)
min_child_weight	Float(0.1, 20.0)
gamma	LogUniform(10^{-8}, 1)
learning_rate	LogUniform(10^{-5}, 1)
scale_pos_weight	{7.5, 15, 30, 45}

Table 21: XGBoost hyperparameter tuning configuration set

A.2 MLP

Regularizer	**Active**	**Distribution**
Batch Normalization	[True, False]	- -
Stochastic Weight Averaging	[True, False]	- -
Lookahead Optimizer	[True, False]	Step Size = Float(0.5, 0.8, 0.1) # Steps = Integer(5, 10, 1)
Weight Decay	[True, False]	LogUniform(10^{-5}, 10^{-1})
Dropout	[True, False]	Float(0.1, 0.5)

Table 22: Regularized MLP hyperparameter tuning configuration set

A.3 ResNet

Hyperparameter	Distribution
# Layers	Integer(1, 16)
Layer size	Integer(64, 1024)
Hidden factor	Float(1.0, 4.0)
Hidden dropout	Float(0.1, 0.5)
Residual dropout	Float(0.1, 0.5)
Activation	{Swish, ReGLU}
Normalization	{LayerNorm, BatchNorm}

Table 23: ResNet hyperparameter tuning configuration set

APPENDIX B – HOW DOES TARGET ENCODING WORKS?

To answer this question, let's take an example. Consider the following dataset:

	Color	**Target**
0	RED	0
1	RED	0
2	RED	1
3	BLUE	0
4	BLUE	1
5	BLUE	1
6	GREEN	1
7	GREEN	1
8	GREEN	1

Table 24: Example dataset to learn how Target Encoder works

To encode the categorical variable *Color*, the target encoder strategy considers, for each unique categorical value, the count of samples having target = 1 over the total number of samples, e.g:

$$\forall x \in CategoricalVariables, TE(x) = \frac{count(x, target = 1)}{count(x)} \tag{B.1}$$

For RED color, $\frac{1}{3} = 0.333$. For BLUE color, $\frac{2}{3} = 0.666$ and for GREEN, $\frac{3}{3} = 1.0$. Thus, the encoded data will be:

	Color	**Target**
0	0.33	0
1	0.33	0
2	0.33	1
3	0.66	0
4	0.66	1
5	0.66	1
6	1.0	1
7	1.0	1
8	1.0	1

Table 25: The encoded example dataset

The real implementation of the encoders considered in this project, i.e, target, catboost and mestimate, is based in the previous explanation, but they introduce smoothie factors or others strategies to change a bit the pure target encoding behavior. For more information, see the reference [8].