CentraleSupélec

université PARIS-SACLAY

Decision Support and Business Intelligence

Master Thesis

Buse OZER

Predicting the Success Rate of Video Ads

Prepared at Smart AdServer's Research and Development Department

Defended on September 2020

Advisor:
Boris PRODHOMME - Smart AdServer bprodhomme@smartadserver.com
Tutor:
Nacéra SEGHOUANI BENNACER - CentraleSupelec nacera.bennacer@centralesupelec.fr

smart

BDMA

Acknowledgments

This master's thesis is written in cooperation between Smart AdServer's Research and Development Department and CentraleSupelec.

First of all, I would like to thank the committee of Big Data Management and Analytics Erasmus Mundus Joint Master Degree and all the professors of the master's program for their incredible wisdom, support, and compassion throughout the program. BDMA has been an amazing and unforgettable experience. I am honored to be a part of this program.

Furthermore, I appreciate that Smart AdServer gave me a great opportunity to work with the latest technology with state-of-the-art algorithms. Also, a special thanks to my supervisor Boris Prodhomme for his excellent guidance and supervision.

I am grateful to my family for being always there for me, believing in me and giving me the power to keep going.

Abstract: With the extensive usage of video ads in the digital advertising industry, achieving a high success rate of video-based advertisements has gained significant importance in the ad tech industry. Hence it is essential to deliver video advertisements to end-users seamlessly. Ad servers need to choose the best advertiser per auction whose ads can be delivered without having problems on the corresponding publisher's website. These problems could be raised due to time-out, internet connection problem, non-compliant video format, and incompatibility between publishers and advertisers.

However, ad tech companies want to prevent this obstacle in order to serve the best experience to end-users, to give substantial benefit to advertisers, to increase the revenue of publishers and the regarding ad server company because if an ad is served on the server-side, though it fails on the client-side, it is not exposed to an end-user. Consequently, there is no revenue either for a publisher or an ad server. Moreover, the advertiser is not able to show its content to a user. This situation is undesirable for all parties. Therefore this paper proposes a methodology that builds a machine learning model in order to predict the success rate of video ads for given constraints. This method helps ad servers to calculate the expected revenue of each bidder in an auction. By doing so, the ad server chooses a bidder with the highest expected revenue. Consequently, the ad server can increase the success/delivery rate of video ads and its revenue.

Keywords: binary classification, ad tech, delivery rate, success rate, video advertisements

Contents

Chapter 1

Introduction

This chapter is divided into four primary sections. Section 1.1 provides the motivation and the scope of the study. Section 1.2 introduces the general context of online advertising ecosystem and the current problem with video advertisements. Section 1.3 describes the objective, the missing points in the industry, and the paper's main contributions. Section 1.4 explains the organization of the thesis.

1.1 Motivation

The digital display advertising industry has gained substantial importance over the years. According to the Interactive Advertising Bureau (IAB) research report, the revenue of digital advertising has surpassed $100 billion for the first time is $107.5 in 2018. The total revenue for the full year of 2018 was increased by 21.8% over the full year of 2017 [IAB b]. The reason behind this increase is that most brands have been accelerating their digital commitment to reach the target consumers. In addition to this engagement, the wide usage of mobile phones, social media activities, and the advancements in the adtech field helped increase growth. Therefore, the digital advertising ecosystem will continue to generate growth in the future. This growth attracts many businesses; therefore, it has been an active research field, including a wide range of topics and challenges to solve.

The devices used to serve ads to customers are desktops, smartphones, tablets, TVs, and Set-Top Box. An ad can be served as an image, text, or video format. Each format has its specificity, such as display configurations. However, most advertisers prefer image and video ads to get customers' attention. When we compare video against image-based ads, video-based ads have gained significant popularity over the last years because most marketers believe that video tends to drive more engagement. According to the research done on Facebook ads that 59.3% of the ad clicks come from video on Facebook, whereas 29.6% of the ad clicks come from images, and 11.1% come from text-based ads, specifying that the number of impressions is the same for different formats. The survey conducted by databox shows that the number of clicks on a video-based ad doubled the number of clicks on an image-based ad. They also observed an increase in conversion by 20-30%. Video ads are favored advertisers because they want to increase their brand awareness, recognition, CTR, and purchases depending on an advertiser's target. Video ads have a higher impact, but their price is higher than other ad types, therefore on publishers side, video ads have substantial economic importance over the other formats because it generates more revenue. Moreover, video advertising is the key driver in the online advertising industry. It is clearly seen that ad spending in the video advertising industry has increased every year in figure 1.1. The ad spending on video advertisements was 21,721 million$ in 2017, 26902 million$ in 2020 and 35,048 million$ is expected in 2024

according to Statista statistics [Statista]. Since the video advertising ecosystem gained substantial importance over the past years, and video ads have a deficient delivery rate problem that has not presented before, this study spouts only video ads.

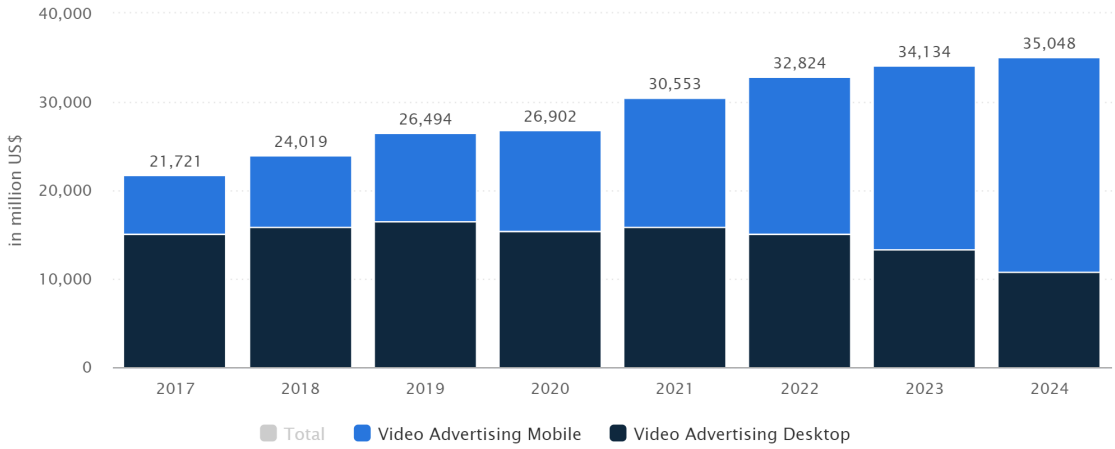

Figure 1.1: Video Adspending, taken from [Statista]

1.2 General Context and Problem

Ad tech solves numerous problems that publishers and advertisers face and improve their programmatic buying and selling process via Real-time bidding (RTB), which is a way of displaying advertisements. They play a vital role in this environment because they are advertising technology (AdTech) that brings publishers, advertisers, ad agencies, and ad networks together to manage and run online advertising campaigns. They are in charge of making instantaneous decisions about what ads to show on a website and serve ads to end-users. It usually handles a dozen billion auctions per day. As described before, an ad can be an image, text, or video. However, only video ads are addressed for the rest of the study, as explained in the Motivation section. An ad exchange can run first-price or second-price auction among multiple advertisers to sell an impression in RTB. It chooses an advertisement as a winner who gives the highest price. However, this approach causes a problem because having the highest price does not mean meeting all the requirements and displaying the video ad successfully to an end-user. It might lead to a failure in displaying the corresponding video advertisement. This problem might occur due to some reasons as follows:

- User might stop video session during ad loading
- The creative is malformed, or the file is corrupted
- Incompatibility between ad creative/media file format and the video player
- VAST response is not well-formed
- Timeouts of media files

- Problem in internet connection / disconnection

All the errors that are listed above cannot be detected on the server-side; it can only take place in the browser, client-side.

1.3 Objective and Main Contribution

This thesis concentrates on a problem that Smart AdServer faces currently. Smart is an advertising technology company. "It connects a supply-side platform (SSP) where publishers' advertising inventory is sold programmatically via Real-time bidding (RTB). The platform handles a dozen billion auctions per day, and this results in tens of billions of bid-requests per day" [Smart]. Smart takes a small percentage of the revenue on the publisher's side for each impression; therefore, ads should be served seamlessly to end-users. Therefore it is essential to have a high delivery rate as explained in detail in the subsequent sections.

This paper aims to build a model that predicts the probability of a video advertisement to be displayed. This helps to increase the delivery rate, increase the revenue of publishers and consequently increase the revenue share of Smart by choosing ads that have the highest expected revenue based on our methodology. The contributions have broad applicability in the ecosystem because it brings high value to ad servers, advertisers, publishers, and improves user experience. Therefore further steps are required to decide which video ad would be delivered to end-users. However, to our knowledge, no study has considered addressing the problem of choosing the best ad that is most likely to be served by considering the limitations that are discussed in the General Context and Problem section. Two gaps are observed in online advertising related to this specific problem. First of all, the delivery rate of video advertisements is insufficient compared to banners; therefore, it does not answer the need for publishers, SSPs, and advertisers. Another gap is the lack of comprehensive research attempts that focus on this problem. One way to overcome the issue is calculating the delivery/success rate of each bid in an auction given the prominent importance of an ad server and video ads in the entire chain of ad market ecosystems.

This paper's main contribution is to fill the gap mentioned above by highlighting the current problem and implementing a model to predict the delivery/success rate of video ads and compare the results of Decision Tree, Random Forest, AdaBoost, Extreme Gradient Boosting, aka XGBoost and Artificial Neural Network and improve progressively best performing one.

1.4 Outline

The thesis is organized into eight chapters. The rest of the thesis contains information as follows. Background and Terminology, chapter 2 gives a general overview of the advertising technologies and introduces some essential notions in the industry. Chapter 3, Related Work, provides the current research topics in the adtech field and the related work with our problem. The Data chapter 4 discusses the data flow of the system, introduces the data description and analysis. The Methodology and Development chapter 5 describes

the used methodology, the selected machine learning algorithms and the architecture. Results and Evaluation chapter 6 elaborates on the selected evaluation metrics, performed experiments and the comparison of different models. The last chapter 7, which is the Conclusion provides the overall summary, the limitations, and the future work.

Chapter 2

Background and Terminology

This chapter provides a general overview of the platforms and systems within the advertising ecosystem, and introduces some required terminology to have a better understanding of the study.

2.1 General Overview

This section provides a general overview of the platforms and systems within the advertising ecosystem. The entire process of header bidding starts with a request of accessing a web page by a user. The entire process can be followed through the figure 2.1 and all the steps are as follows.

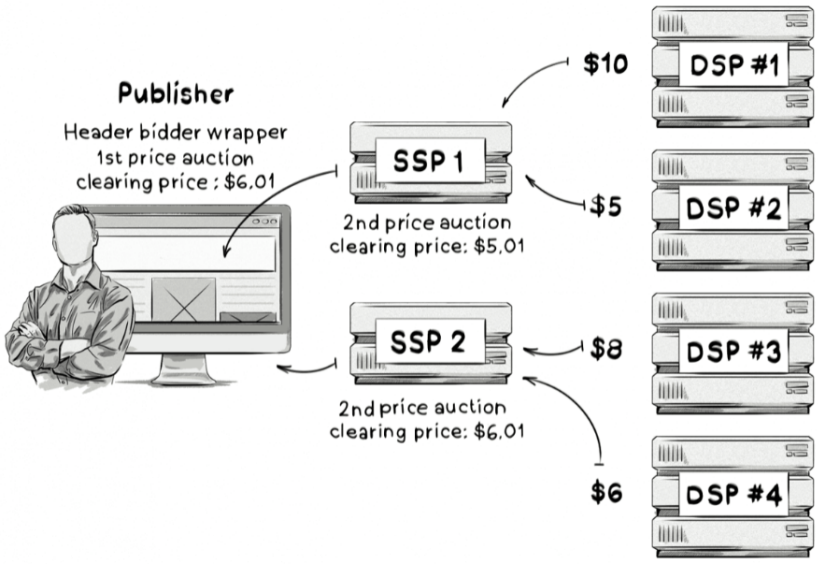

Figure 2.1: The entire process of advertising ecosystem, taken from [ClearCode c]

- When there is a user that visits a web page, a request is sent to all SSPs in parallel.
- Each SSP calls different DSPs
- Each DSP calls different buyers who are interested in displaying an advertisement
- The buyers, who are interested in the corresponding end-user, send a positive response to DSPs, as mentioned in section 2.9

- Based on some criteria, each SSP makes a decision of choosing its winner
- After that the last competition takes place between different SSPs in order to choose the final winner whose ad would be displayed
- The advertisement of the winner is passed to the web page of the publisher and it is displayed to the corresponding user.

2.2 Impression

An impression is a measurement of response from a web server to a page request from the user browser. An impression is counted each time when an ad is shown successfully on a publisher's website to an end user [IAB a]. There are two types of impressions which are client-side and server-side impressions. Server-side impressions are the impressions that won auctions whereas client-side impressions are the impressions that won auctions and are displayed successfully to end-users. Each server-side impression does not turn to a client-side impression due to some technical issues.

2.3 Auction

An auction takes place in order to decide which ad will be shown on the website of a publisher. When a visiter accesses a publisher's website, an auction takes place. Inside an auction, there are some steps that are discussed previously in 2.1 section.

At the end of an auction, there are two possible outcomes. One of them is that the ad of a chosen winner is shown to an end-user which means that it is counted as an impression on the client-side. Therefore, the publisher gets paid and consequently the ad server too. Another possible outcome is not being able to show the winner's ad on the publisher site due to some technical issues. In this case, there is no displayed ad therefore publisher will not get paid by the corresponding advertiser, consequently, there is no revenue for the ad server.

2.4 Delivery Rate

Delivery rate, also known as, display and success rate is the ratio of client-side impressions to server-side impressions. In other words, delivery rate is the ratio of the number of impressions that are shown to an end user to the number of won auctions. Having a higher ratio is the target of ad servers because it is simply how they make money. In contrast, if there is no displayed ad, consequently, there is no revenue. Therefore it is crucial to resolve discrepancies.

$$Delivery\ Rate = \frac{Client\ Side\ Impressions}{Server\ Side\ Impressions} \tag{2.1}$$

In order to increase the high delivery rate, ad servers should only choose an ad that is most likely to deliver without any issue thus the number of client-side impressions would be closer to the number of won auctions.

2.5 CPM

Cost per impression is the ratio of the advertising costs to the number of impressions. Cost per impression is often signified as Cost per Thousand Impressions (CPM) to make it easier to manage. [AllBusiness]

$$Cost\ per\ impression(\$) = \frac{Advertising\ Cost}{Number\ of\ Impressions} \tag{2.2}$$

2.6 Ad Server

An ad server can be thought as a web server that includes all the required information about a relevant ad and it manages to place the ad on a website. Most of the time ad server refers to an ad tech platform where one launch and manage online ad campaigns [Epom].

There are two kinds of ad servers. Although they have the same concept, they serve different purposes. The fist one, known as ad server for advertisers is used to store, manage, optimize advertisers' campaigns and provide reporting. The second type of ad server known as ad server for publishers is used to deal with all the resources of publishers to sell and provide necessary information to ad server of advertisers.

2.7 DSP

A demand-side platform known as DSP is an advertising technology platform that allows programmatic media buying on each impression via Real-time bidding (RTB), which means that publishers and advertisers do not handle the sales manually [ClearCode a]. The DSP platforms automate the process of buying inventories from publishers and selling to advertisers. Criteo and The Trade Desk are well-known examples of DSPs [ClearCode b].

There are a few steps taken in DSP as follows.

- An advertiser wants to publish an ad campaign.
- The advertiser signs in its DSP account create the new campaign content, specifies their target audience, the target location, and target device etc.
- DSP connects to an associated ad exchange server which is responsible for offering opportunities to DSPs from various publishers
- If the user's information matches the advertiser's criteria, DSP bids on that impression on behalf of the advertiser.
- If the DSP wins, the advertiser's ad is displayed on the associated publisher's page to the specific user.
- After that the last competition takes place between different SSPs in order to choose the final winner whose ad would be displayed.

2.8 SSP

A supply-side platform known as SSP is an advertising technology platform that allows to sell available inventories of publishers. Smart and Google Ad Manager can be given as an example of SSPs.

2.9 Bidder

A bidder is a component of the DSP platform in charge of placing bids on a publisher's website through RTB in case if a buyer is interested in the corresponding inventory, the buyer bids and it is called bidder. There can be many bidders in an auction, each of them makes an offer and based on the auction policy, the winning bidder is chosen as shown in 2.2.

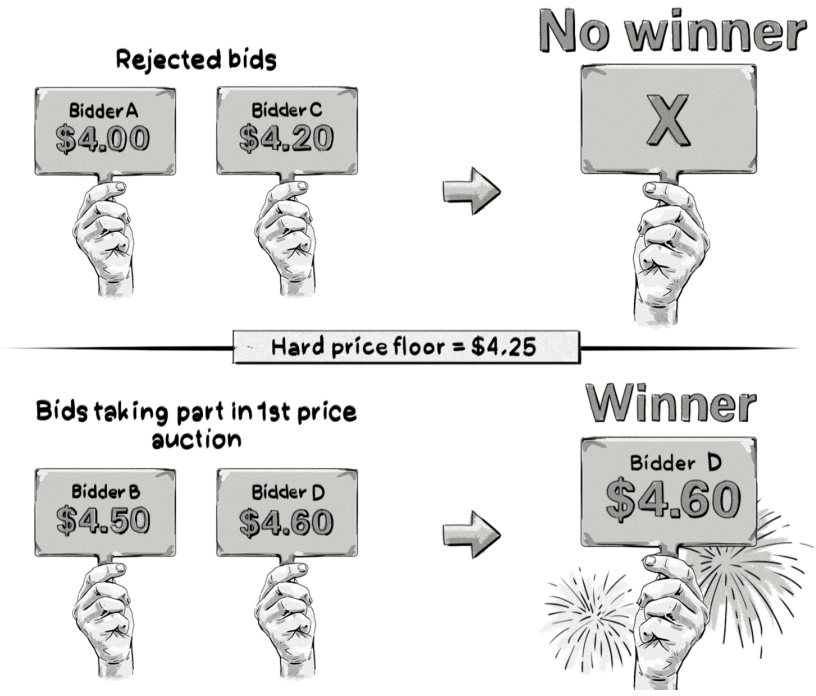

Figure 2.2: Auction, taken from [ClearCode b]

2.10 VAST

VAST is an acronym for Video Ad Serving Template which is a template for structuring ad tags that serve video and audio ads to media players. It was initially launched in 2008 by IAB [IAB c]. Using an XML schema, VAST transfers important metadata about an ad from the ad server to a media player. It includes information about which ad to play, the length of a video ad, type of ad, how the ad will show up on the screen, and whether users can skip the ad or not.

VAST has played a crucial role in the growth of the digital video marketplace because the number of video-based ad platforms has significantly increased and it brings some complexity within. There are different specifications of a video ad in terms of a variety of devices, connection speeds, video players, and ad servers therefore it requires a common protocol for all different integration to simplify the process. VAST provides a common protocol that allows ad servers and media players to speak the same language by using a single ad response format for video players across compliant video players.

Chapter 3

Related Work

In this chapter, we explore various problems in online advertising industry and discuss the state of the art methods of binary classification problems. This section allows us to explore the theory behind the methods that we would like to use. As discussed in Chapter 1, the goal of the master thesis is to develop a method that is able to predict the success rate of video advertisements to be served. It is a binary classification problem which composes of being displayed or not. However, what we interest is the probability in order to maximize the expected revenue of each bidder as explained in section 5.1.1.

3.1 Ongoing Research in Ad Tech

In online advertisement industry, there are many challenges to solve and different problems to address such as click through rate (CTR) prediction, finding a better floor-price in order to maximize the revenue. Researchers have done many studies regarding online advertising. There has been numerous studies to address CTR prediction by using logistic regression, ensemble learning and various deep learning architectures [McMahan 2013], [Yan 2014], [Zhai 2016]. According to our knowledge, no study aims the same target as our project. Therefore, we examine various problems in the online advertising industry because it is still valuable to discover ongoing researches. One of the most extensive research areas is the prediction of Click-Through-Rate (CTR), the proportion of the number of clicks to the impression count(the number of times an ad is served on the client-side). This metric has one of the most significant business value because it is aligned with the commercial value. [Cakmak 2019] provides a comparative analysis of the CTR by applying state-of-the-art prediction algorithms, which are Random Forest, Gradient Boosting, AdaBoost, Support Vector Regression, and Extreme Gradient Boosting algorithms. Even though Support Vector Regression and Random Forest are known to be quite successful, the outcomes confirm that decision tree-based boosting algorithms defeated the other algorithms. XGBoost reached the highest score for the prediction of the number of clicks.

Besides, [Wang 2015] proposes a model that predicts the probability of an ad being viewed by end-users depending on the position of the ad on a page. This paper solves a binary classification problem, which consists of in-view or not in-view and being interested in the probabilities of each advertisement. It is relatively similar to our problem. Another research investigates the relationship between pixel percentages of an ad creative to be viewed and exposure time from the human-computer interaction perspective in order to prevent the advertisers from paying from the ads that are not be seen by the users, and it does not bring any revenue [Zhang 2015]. In detail, when an ad is loaded to the browser, it is immediately counted as one impression, though it might not be viewed by an end-user or not be exposed to the viewport. In this case, advertisers pay for ads that are not disclosed to end-users. This situation is a waste of money.

In conclusion, as far as we know, no previous research has studied the prediction of an ad to be delivered on the client-side. Therefore, the literature review is done for binary classification problems with state-of-the-art algorithms in a general context.

3.2 Ensemble Learning

Enhancing the predictive performance of classification problems is an open-ended research challenge. There are numerous successful machine learning algorithms for classification problems such as SVM, K-Nearest-Neighbour, and Decision Trees, aka base classifiers. On top of that, several studies have demonstrated that ensemble techniques improve predictive performance when compared to base classifiers for both classification and regression problems [Breiman 1996] [LeBlanc 1996]. Therefore we concentrate on ensemble learning in this section.

An ensemble learning is a machine learning technique that converts numerous weak learners into a strong ensemble learner that beats the weak learners, which can be explained as any machine learning algorithm performs just slightly better than chance (50%). An ensemble approach consolidates weak learners by combining the output of the weak learners. The output integration can be accomplished by majority voting, simple/weight averaging, or using machine learning algorithms [Breiman 2001] [Wolpert 1992]. It has been proved that ensemble learning enhance the performance in many applications [Banfield 2006], [Zhu 2002] and [Rodriguez 2006].

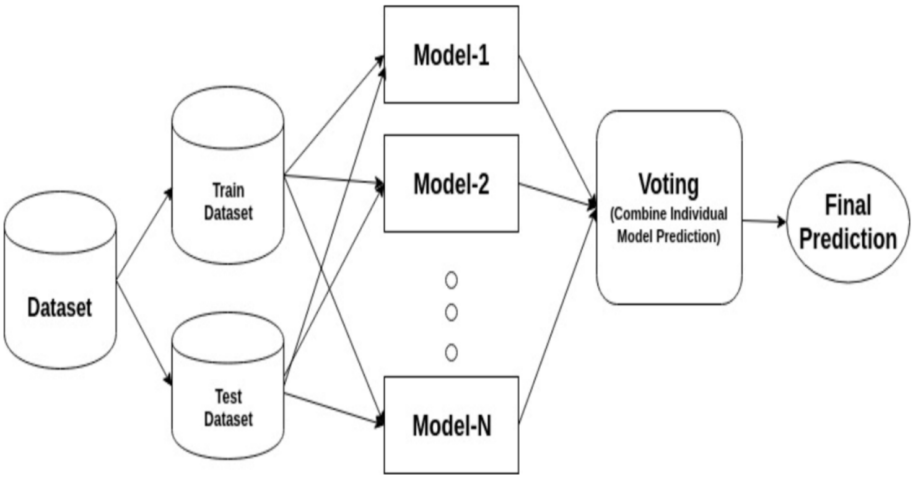

Figure 3.1: An example of ensemble approach, taken from [datacamp]

The reason for the high demand in ensemble learning is having a high bias and low variance when a single classifier is employed compared to the ensembled classifiers [Webb 2005]. The study was done by [Bauer 1999] points out that ensemble models can lower variance and bias. For instance, a decision tree classifier has a high variance for choosing attributes and splitting nodes [Breiman 1984]. Therefore, it has a great inclination of overfitting due to the high variance. Hence, it is vital to decrease its variance in order to achieve better predictive performance. The studies by [Chiu 2002] and [Dietterich 2000] prove that decision trees present a high level of performance when they are employed as base classifiers for ensemble models. Random Forest, Extremely Randomized Trees, AdaBoost, Gradient Boosting Trees, and Extreme Gradient Boosting trees are the widely used ensemble approaches that perform well.

The intuition behind the ensemble approach is to diminish the variance between training and newly seen data. The diversity between the base learns helps to overcome the high variance problem. There are several methods of enhancing variety amongst base models as follows:

- Employing different machine learning algorithms
- Sub-sampling of the training set
- Sub-sampling of dimensions while creating base learners

Each point in the list above can be merged or used independently. If an ensemble model employs weak learners that use the same algorithm and differ in terms of the of training, it is known as a homogeneous ensemble model. Whereas if an ensemble model employs weak learners that consists of different algorithms, it is known as a heterogeneous ensemble model. Moreover, there are three approaches to consolidate weak learners.

- Bagging
- Boosting
- Stacking

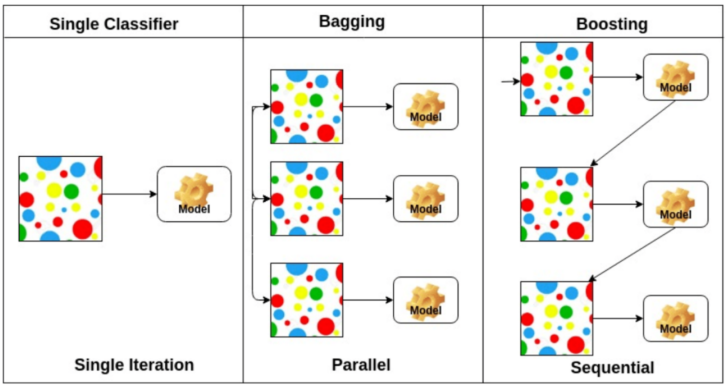

Figure 3.2: Bagging and Boosting, taken from [datacamp]

Bagging and boosting are the most extensively applied strategies, as illustrated in figure 3.2. They fall into a homogeneous ensemble model category. However, they differ in terms of combining single type base learners.

The bagging approach is described as constructing independent base learners in parallel and aggregating their results. In contrast, the boosting technique builds weak learners sequentially, which empowers learning from the mistakes of previous weak learners and combining them. Furthermore, stacking falls into a heterogeneous ensemble learning category and it allows base classifiers to learn individually by employing diverse machine learning algorithms in parallel and consolidate them in order to produce the final output.

3.2.1 Bagging

Bagging stands for bootstrap aggregation, which generates a set of samples that are randomly picked with replacement, training of these subsets, and combining the output of the base classifiers. The bootstrap process is shown in figure 3.3.

Figure 3.3: Illustration of boostrapping process, taken from [towardsdatascience a]

The study of [Breiman 1996] and [Freund 1996] designate that the bagging approach increases the diversity between base learners with the help of manipulating the distribution of training data set. It is time-efficient due to the highly parallelized nature of the process. Random Forest is one of the popularly used examples of bagging approach that performs very well for classification and regression problems.

3.2.2 Boosting

Freund and Schapire in 1996 suggest an ensemble method that consists of having various weak learners sequentially in an adaptive way. Due to the nature of the boosting method, the training of each model depends on the previously fitted model [Breiman 1996]. As a matter of fact, boosting consists of many iterations that can be described as the number of trees to be built in figure 3.2. At each iteration, the next model gives more importance to observations in the data set where it performs poorly in the previously fitted model. Weights of each model are calculated based on their performance. Eventually, all the base models are multiplied by their weights in order to obtain the final output. To sum up, the boosting approach attempts to improve the model progressively. Extreme Gradient

Boosting Trees, Gradient Boosting Trees, AdaBoost are one of the broadly practiced examples of boosting approach that performs very well for classification and regression problems. In addition, XGBoost is realized that it outperforms against multiple machine learning algorithms in the competitions and industry.

3.2.2.1 XGBoost Algorithm

This section is dedicated to the theory behind the XGBoost because it beats the other tree-based algorithms according to the literature review and many Kaggle competitions. XGBoost employs CART trees, which are identified as regression trees for classification and regression problems. Unlike decision trees, regression trees contain continuous values that indicate a score on each leaf. Furthermore, the ultimate outcome is measured based on the scores, as displayed in figure 3.4, which can be observed that decision ruling is involved in each level. The example is based on two shallow trees. Each tree yields a score for the prediction of a given value. After that, all the predictions are consolidated by summing up scores for the corresponding leaves given by the weights of each tree.

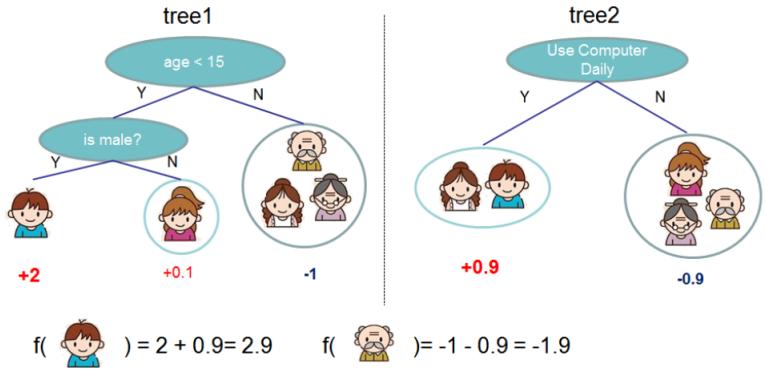

Figure 3.4: Calculation of the final predictions, from [Chen 2016]

XGBoost strives to minimize the error between the prediction and true value by preventing over-fitting. Hence, it is vital to know the objective function of XGBoost. The loss function of XGBoost $\mathcal{L}(\phi)$ consists of 2 main components. The first component is a convex loss function for the training loss, which signifies the difference between the prediction value $\hat{y}_i$ and the target value y_i. It measures how well a model fits on training data. The second component Ω is for regularization. It measures the complexity of a model. The regularization component penalizes complex models and advocates to have a simpler model. The objective of the additional regularization term is smoothing the final learned weights to limit over-fitting. If the regularization term is set to zero, it becomes the objective function of a normal gradient boosting tree. The objective function of XGBoost enables the model to find the trade-off between bias and variance.

$$\begin{aligned} \mathcal{L}(\phi) &= \textstyle\sum_i l\left(\hat{y}_i, y_i\right) + \sum_k \Omega\left(f_k\right) \\ \text{where } \Omega(f) &= \gamma T + \tfrac{1}{2}\lambda\|w\|^2 \end{aligned} \tag{3.1}$$

Additional to the regularization component, there are other ways to control the model complexity explained below.

- Max Depth
- Number of Trees
- Shrinkage
- Sub-sampling

To begin with, the max depth, which is the maximum allowed tree depth for each base learner. By keeping the depth low, we employ simple trees. The higher the max depth, the more complex the base learners are. The second hyper-parameter is the number of gradient boosted trees to be built, which is equivalent to the number of boosting rounds. The more trees to be built, the more iteration to make, the more time to take. The third parameter is shrinkage, which is the weighting factor for the improvements from the existing trees each time when a new tree is constructed. It is similar to the learning rate in stochastic optimization. If we keep the learning rate high, it leads to less iteration but does not reach the optimal point. As we are taking big steps, there would be frequent oscillations and no stability. If we keep the learning rate low, we need more iterations to converge. A low learning rate enables the model to reach the optimal point and achieve a robust model. The last hyper-parameter is sub-sampling, which can be applied for rows and columns. According to the [Chen 2016], column sub-sampling is suggested more than row sub-sampling if one wants to prevent over-fitting. Column sampling is the number of features to be chosen randomly to build the trees. It also speeds up the computations since there are a few features used.

3.2.3 Stacking

Stacking deviates from bagging and boosting because it falls into a heterogeneous ensemble learning category. Stacking enables weak learners to be trained individually by applying different machine learning algorithms and unite their results in order to produce the ultimate output. As an example of stacking, Support Vector Machine and K-Nearest Neighbors and Decision Tree can be used as base learners, and a neural network can be employed to consolidate the results of base learners in order to make the final decision.

3.3 Neural Networks

This section is devoted to the principles behind the Artificial Neural Networks, aka Neural Networks. It is based on the notion of perceptron [Rosenblatt 1962]. The structure and function of artificial neural networks are inspired by the human brain, and it consists of many neurons. They are connected to each other in order to carry out the learning process. There are two kinds of architecture in artificial neural networks. Single Layer perceptron is the first proposed model that consists of an input layer and an output function. It is also known as a Linear Binary Classifier. The architecture of the single-layer perceptron is shown in figure 3.6. Single Layer perceptron is used when the data

is linearly separable with a binary target 3.5. However, when the data points are not linearly separable, consequently, more complicated architecture is needed.

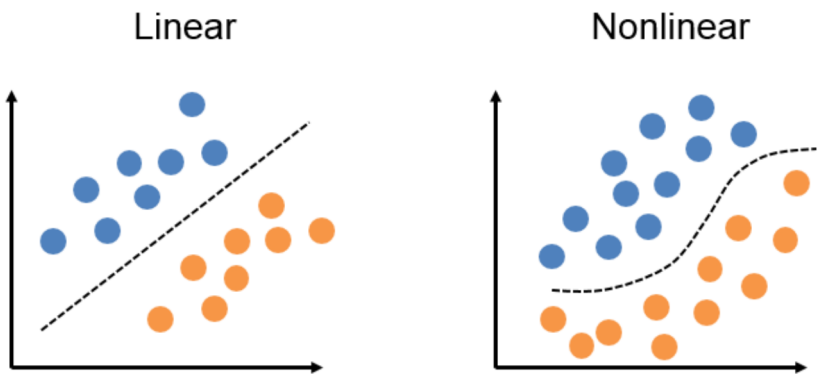

Figure 3.5: Linearly separable data, taken from [jtsulliv]

x1, x2, x3 are the inputs of the network and they are located in the input layer.

$$\sum x_i w_i \tag{3.2}$$

where x_i in a feature and w_i is a weight of the corresponding feature.

$$\text{output} \begin{cases} 1 & \text{if } \sum w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

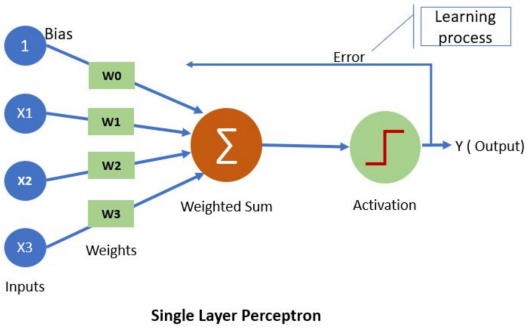

Figure 3.6: Single Layer Perceptron, taken from [sumanthrb]

The output of the network is computed based on the weights of the corresponding neurons and adding the bias parameter. The output of the network is either 0 or 1, depending on the calculated output. If the calculated value is higher than the defined threshold value, it equals one; otherwise, it is zero. In the absence of bias, the neuron

may not be initiated by only the weighted sum of the inputs. Thus, the presence of a bias parameter is important to activate a neuron in order to pass the information.

When data points are not linearly separable, a single-layer perceptron cannot distinguish the instances accurately. Hence, it performs poorly. Multi-layer perceptron architecture is introduced in order to resolve this problem [Rumelhart 1986]. Multi-layer perceptron has the same architecture as the single-layer perceptron with one or more than one hidden layer, as shown in figure 3.7. It is used in order to solve various problems in prediction, recognition, optimization, computer vision, and pattern recognition. Having fully-connected architecture implies that each neuron in a layer is connected with all the neurons in the subsequent layer. Multi-layer perceptron (MLP) comprises at least one hidden layer with non-linearly activating nodes in the architecture. Therefore, it is capable of finding a pattern to classify data that is not linearly-separable.

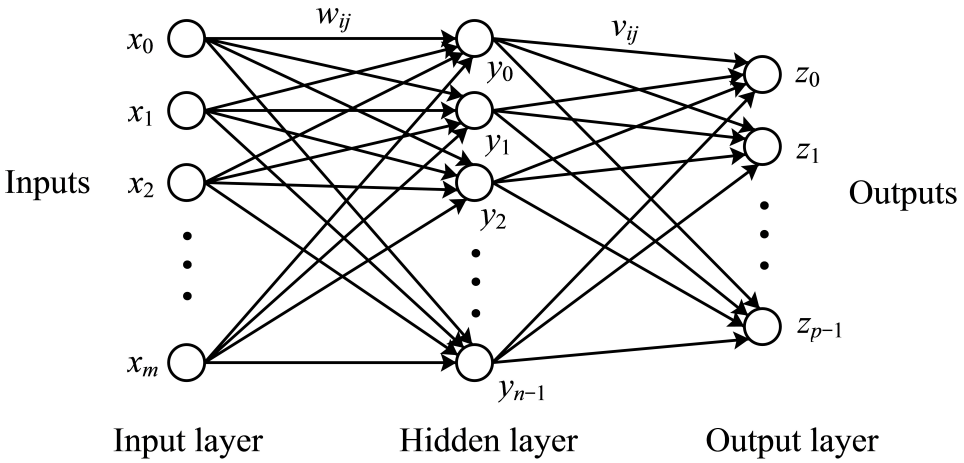

Figure 3.7: Multi-Layer Perceptron,taken from [groundai]

Chapter 4

Data

This chapter consists of three primary sections. Section 4.1 elaborates on the data flow, how the data is stored, and how it is accessed. The next section 4.2 provides the definition of each dimension. The last section 4.3 elaborates on the data analysis and some insights about the data.

4.1 Data Flow

This section gives an idea about how the data flow inside the system and get the input data for the modeling purpose. There is Hadoop ecosystem where HDFS and HBase are used in order to store the logs. The data is inserted to HBase in batches in a key-value format. Also, HDFS is used in order to store the Parquet files contain pre-computed data by using map-reduce operations. There is a reporting API that is implemented by data engineering team in order to extract data from HBase. Reporting API allows us to get the raw logs from HBase by specifying the required columns in a given time period. For all the experiments, the data size is reduced for the PoC by asking 17 days of data starting from 2020-02-01 till 2020-02-17. The number of days is selected based on the need for different experiments throughout the study.

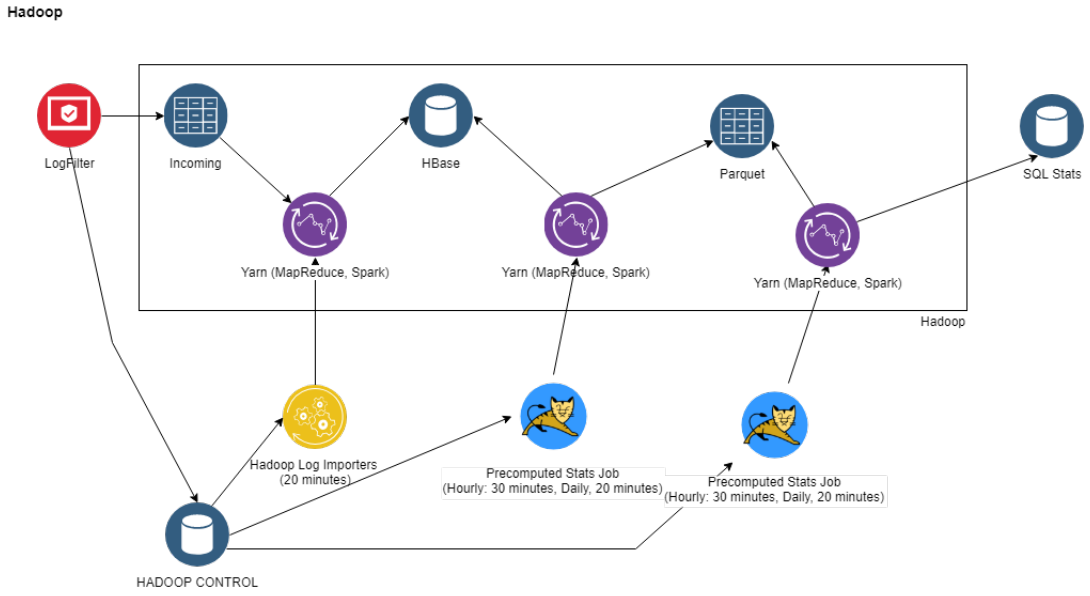

Figure 4.1: Overview of Logging System

4.2 Data Description

The data set contains 30 dimensions which indicates the various information about demand and supply side (publisher and advertiser) and the metric columns which will be transformed to a target column that includes a binary value which indicates whether an ad is delivered to an end user or not.

Each line has information that is briefly indicated as follows.

- the end-user visit which publisher, which site, which page
- the end-user use which channel to see the website (web, mobile web, mobile app)
- the end-user use which platform to see the website (desktop, tablet, smartphone)
- which advertiser wins the auction, which DSP is used
- information about the type of advertisement (video, banner)
- specific information about video type
- information about the number of impressions on the server-side and client-side

Field Name	**Type**	**Definition**
IsHeaderBidding	Integer	Boolean true when it's header bidding, it is true when it's header bidding
HeaderBiddingTypeId	Integer	Specifies the header bidding type
ImpressionType	Integer	Type of the impression Banner = 0, VideoInStream = 1, Native = 2, VideoOutStream = 3
Inventory	Integer	An identifier representing any of Web=0, Mobile Web=1 or Mobile App=2
WinnerDealTypeId	Integer	This is the Id of the winning deal; Direct Deal, Private Auction and Guaranteed
VideoPlacementType	Integer	An identifier representing where InStream = 1, InBanner = 2,InArticle = 3, InFeed = 4, Interstitial = 5
PlatformId	Integer	Platform is the environment where the ad is displayed (Desktop, Table, Smartphone...)
BrowserParentId	Integer	Identifier of the browser (Google Chrome, Firefox, Safari...)
BrowserId	Integer	Identifier of the browser along with the version
OsParentId	Integer	Identifier of the operating system (Windows, Linux, iOS...)
OsId	Integer	Identifier of the operating system along with the version
PartnerId	Integer	Identifier of Dsps such as Criteo, Smart Demand, Google DBM, LoopMe
CountryId	Integer	Country identifier

NetworkId	Integer	Identifier of the publisher
FormatId	Integer	Identifier of the format of an advertisement
InsertionId	Integer	Insertion is an unique Id which consists of placement type, creative and targetting criteria
InsertionScriptId	Integer	Aka template it indicates different ad format options like floor, side, in an image. it helps customers choose the way that they want to display their ads
InputBundleId	String	Identifier of the app version
BuyerId	Integer	Identifier of the advertiser who bought the ad
RtbVideoPlayerWidth	Integer	Specifies the width of a video player reserved on the publisher side
RtbVideoPlayerHeight	Integer	Specifies the height of a video player reserved on the publisher side
SiteId	Integer	Identifier of the site
PageId	Integer	Identifier of the app version
Referrer	String	Base URL of the site displaying the ad, the same as domain
CreativeHashCode	Integer	It is an ID that Smart creates from other IDs to identify uniquely the video ads sent by DSPs and consists of dspId, smartBuyerId, campaignId, creativeId and adId
AdBreakId	Integer	Identifier of instream video ad break (1=preroll, 2=midroll, 3=postroll)
AdPositionId	Integer	Identifier of the position of the ad in the web page (header, footer, middle)
PositionInAdBreak	Integer	Identifier of the order of an ad in Ad Break
TotalPaidPrice	Float	Bidding Price
RtbImpressions	Integer	Server-side RTB impressions. It represents the number of bids that have won the competition, counted as server side
ImpressionsTrueCount	Integer	RTB impressions that is served successfully to the client-side

Table 4.1: Real Time Bidding (RTB) features and their descriptions

4.3 Exploratory Data Analysis

The goal of this section is to make the exploratory data analysis of the features, see their cardinality, missing values, distributions, and correlations. The analysis was made with the data starting from 2020-02-01 till 2020-02-04 on four days.

4.3.1 Uni-variate Analysis

As a part of exploratory analysis, a univariate analysis was performed in order to see the frequency of distinct values in each category and identify their cardinality. Since most of the data have high cardinality, two features are selected to show the frequency distribution over unique values within a column.

The frequency of different inventory types is given in Figure 4.2 and the mappings of Inventory Ids are provided in Table 4.2. As it can be seen, mobile web has the highest frequency compared to web and mobile app, which means users most of the time visit mobile web and it is when most of the auctions take place. Moreover, the frequency of different platform types is given in figure 4.2, and the mappings of Platform Ids are provided the table in 4.3.

In the next chart, figure 4.3 represents us the total number of auctions, RTB impressions, impressions true count, delivery rate, and daily revenue of in-stream and out-stream videos. First of all, the data is filtered when there is a win; therefore, the number of auctions equals RtbImpressions. If it were not filtered, then the number of auctions would be higher then RtbImpressions. In addition, it can be seen that both impression types have a similar number of total auctions and RTB Impressions, which indicates that these inventories are winning on the Smart sides because they are bidding high prices compared to other bidders, whereas the number of ImpressionTrueCount is much lower in the out-stream video, this could be due to the incompatibility between supply and demand-side when it comes to out-stream video. Because the number of Impression True Count is different, the delivery rate of out-stream videos is much lower. Since the number of impressions that are delivered to client-side is much higher in in-stream videos, consequently, it leads to higher revenue.

Inventory	Name	Frequency
0	Web	28,848,527
1	Mobile Web	44,590,099
2	Mobile App	11,840,051

Table 4.2: Mapping of Inventory

Platform Type	Name	Frequency
1	Desktop	9552817
2	Tablet	33186449
4	Smart Phone	1977716
8	TV	22011
16	Set Top Box	1046
32	Game console	3751
64	Other Devices	4600

Table 4.3: Mapping of Platform

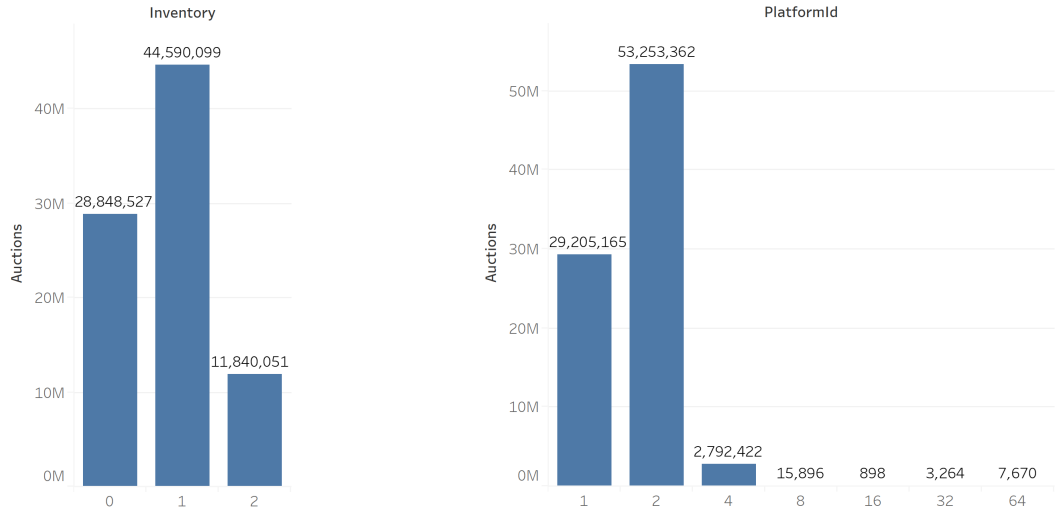

Figure 4.2: Inventory and Platform distributions

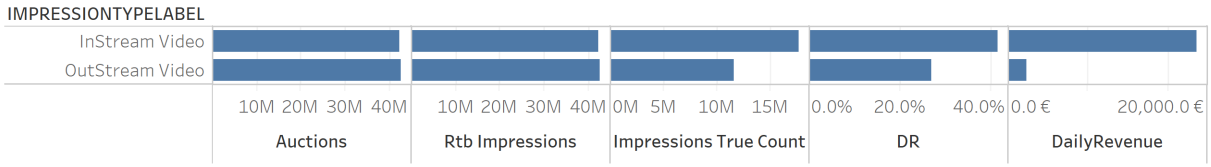

Figure 4.3: Impression types in terms of different metrics

Figure 4.4 represents the DSPs and their daily revenue and the top 5 DSP in terms of their daily revenue. In each circle represents a DSP, and the size of the circle depends on the daily revenue of the corresponding DSP. It can be seen that adform, Trade Desk, Google DV360, and Mediamath have the highest daily revenue, respectively. It is also worth reviewing the total number of true impressions along with the daily revenue values, as can be observed in figure 4.5. In each circle represents a DSP, the first appeared value represents the total number of impression true counts, which means the total number of served ads to end-users, the final number in the circle represents the daily revenue of corresponding DSP. Moreover, the size of each circle depends on the total number of ads served to end-users. The bigger the size, the higher the total number of served ads to end-users. The visualization displays that Criteo has the highest number of served ads. However, the daily revenue is much lower compared to the other DSP, whereas, Google has a similar number of served ads as Criteo. However, the daily revenue is sixteen times higher, and Google appears on top 5 DSP in terms of Daily Revenue 4.4. In addition, Active Agent has six times lower than the number of served ads in Criteo. However, the daily revenue is seven times higher than Criteo. It is important to analyze these values separately without hypothesizing that a higher number of true impression count leads to higher revenue due to the variation in price.

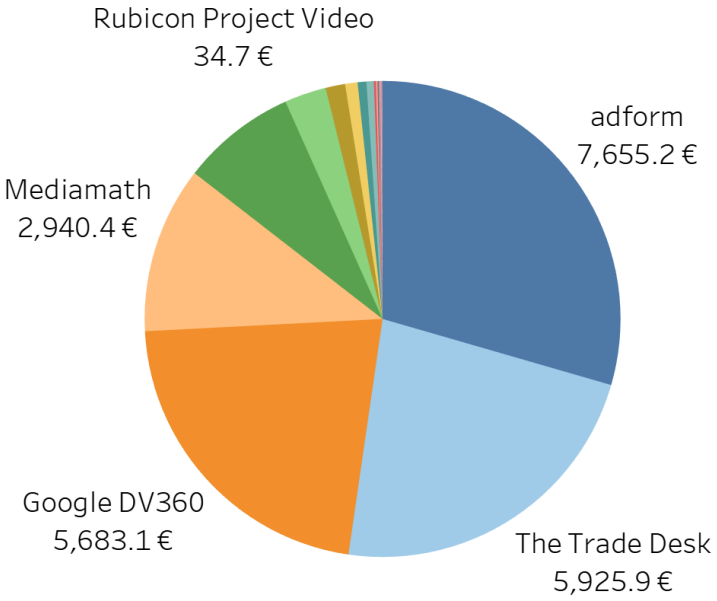

Figure 4.4: DSP revenue distribution

As can be seen from the table 4.4, each feature is distinguished in terms of its cardinality. The idea of this chart is to demonstrate the cardinality of each feature and detect the percentage of missing values.

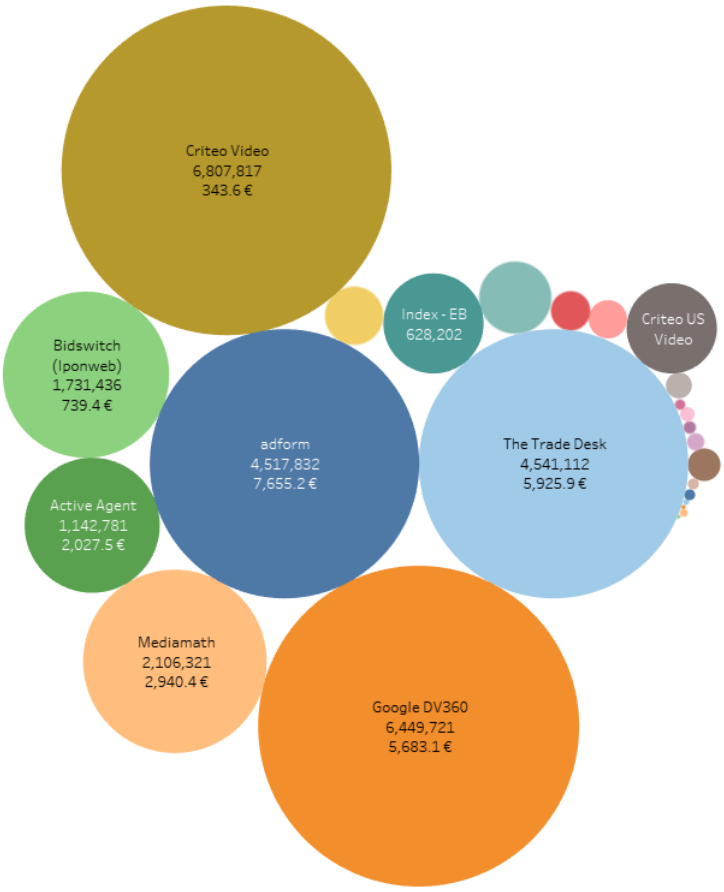

Figure 4.5: DSP True Count and the daily revenue

Feature Name	Cardinality	Distinct values	Missing Values
IsHeaderBidding	Low	2	
HeaderBiddingTypeId	Low	3	
ImpressionType	Low	2	
Inventory	Low	3	
WinnerDealTypeId	Low	3	
VideoPlacementType	Low	6	3.05%
PositionInAdBreak	Low	7	
PlatformId	Low	8	
BrowserParentId	Low	8	5.25%
BrowserId	High	267	5.25%
OsParentId	Low	11	5.25%
OsId	High	137	5.25%
PartnerId	Medium	34	
CountryId	High	221	0.009%
NetworkId	High	384	
FormatId	High	1015	18.43%
InsertionId	High	2243	18.43%
InsertionScriptId	High	103	18.43%
InputBundleId	High	1259	95.30%
BuyerId	High	2124	
RtbVideoPlayerWidth	High	3069	14.74%
RtbVideoPlayerHeight	High	2605	14.74%
SiteId	High	10771	
RtbAdvertiserId	High	13087	
PageId	High	15373	
Referrer	High	59350	4.95%
CreativeHashcode	High	84312	

Table 4.4: Cardinality and Missing Values

The next analysis is the distributional range of different values over delivery rate. As a reminder, the delivery rate is our target and it has continuous values. Therefore it is necessary to see a box plot of different categories over delivery rate. The size of each section in a box plot shows how widely spread a data range is, and it says nothing about the quantity of the group. In figure 4.7, two features are used as an example. Each BrowserParentId is mapped to a unique browser such as Opera, Chrome, and Edge, whereas each OsParentId is mapped to a unique operating system such as Windows, Unix, and macOS.

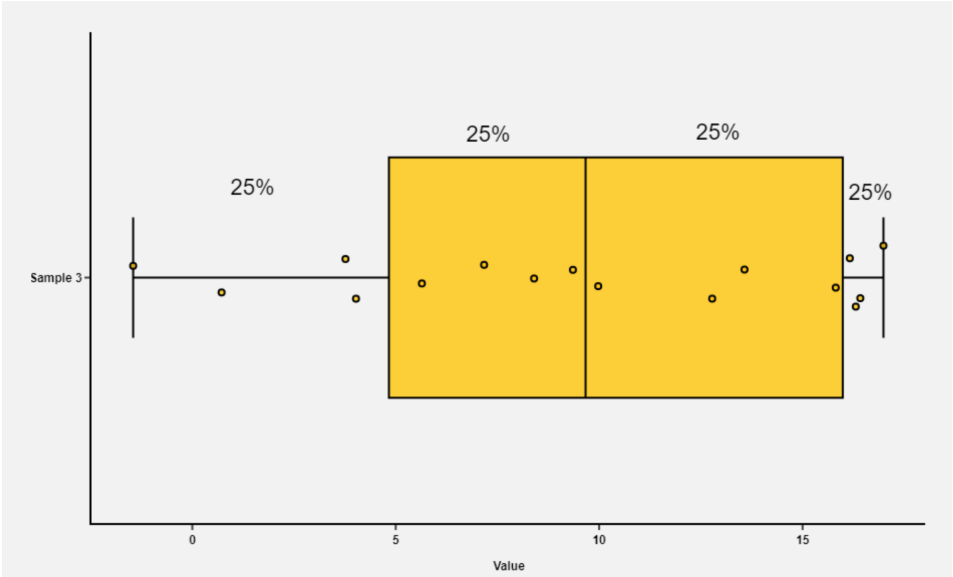

Figure 4.6: Box Plot, taken from [bioturing]

However before analyzing the chart, it is important to refresh the knowledge about how to interpret the box plot in figure 4.6.

Each section contains an identical number of data points, which is a quarter of the entire group. The different sizes come from how variable the values are in each section. If two boxes do not overlap with each other, it can be concluded that there is a difference between the two groups. If they overlap, it is better to examine the lines inside the boxes. If the median line of one box lies outside of the other box entirely, then there is likely to be a difference between the two groups. In case of overlapping of median lines, whiskers show how big a range there is between those two extremes. Larger ranges designate wider distribution. Taller boxes imply more variability in data. Therefore it can be concluded from figure 4.7, there is no direct difference between different values over the delivery rate, so both features do not have a big impact on the delivery rate.

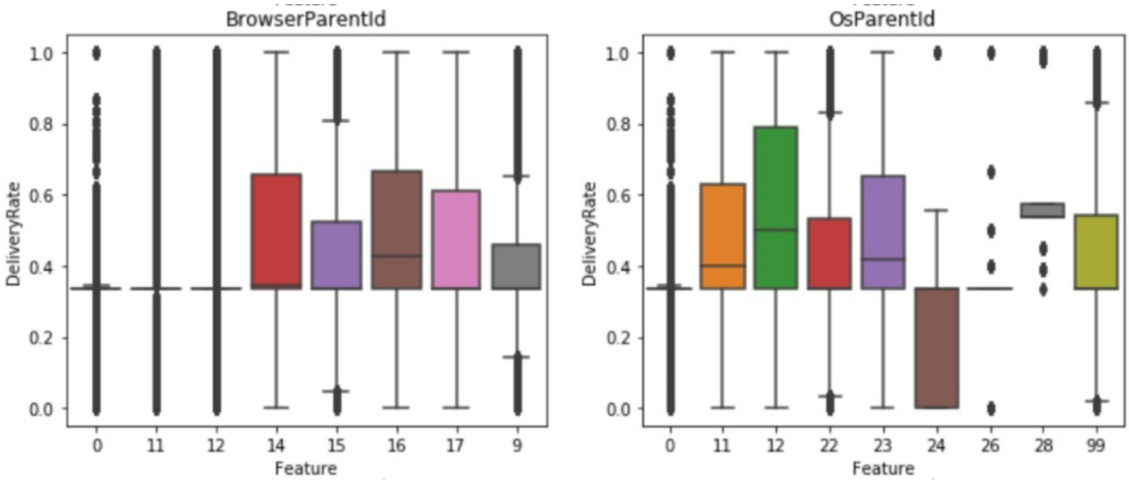

Figure 4.7: Box Plot of BrowserId and OsId

4.3.2 Multi-variate Analysis

In this section, multivariate analysis was performed by applying Cramer's V method, which helps us to measure the correlation between two categorical variables. The correlation metric varies between 0 and 1. Value 0 indicates that there is no correlation between the variables, whereas 1 designates the strong correlation.

Table 4.5 displays the correlation between each feature and the target variable, which is DeliveryRate. As can be seen from the figure, the most correlated features are PageId, SiteId, Referrer, CreativeHashCode, RtbAdvertiserId, and BuyerId. However, we should bear in mind that correlation does not imply causation [Aldrich 1995]. Having a high correlation does not imply high feature importance. Thus the best way of getting the feature importance through the tree-based models as is discussed in section 6.2. For instance, InsertionId is highly correlated with the target columns, whereas the feature importance of InsertionId is relatively low. Moreover, figure 4.8 illustrates the correlation between each feature. The brighter the color is, the more the correlation is. There are highly correlated features like OsId and ParentOsId because OsId is more granular version of OsId. Therefore it is expected to have a high correlation. The same logic is applied for PageId and SiteId because PageId is the granular version of SiteId.

Features	Cramer's V
PageId	0.304
SiteId	0.298
Referrer	0.289
CreativeHashcode	0.275
RtbAdvertiser	0.249
BuyerId	0.233
InsertionId	0.226
FormatId	0.206
RtbVideoPlayerHeight	0.194
RtbVideoPlayerWidth	0.176
InsertionScriptId	0.171
NetworkId	0.160
PartnerId	0.142
CountryId	0.128
VideoPlacementType	0.111
HeaderBiddingTypeId	0.101
ImpressionType	0.090
BrowserId	0.089
WinnerDealTypeId	0.082
BrowserParentId	0.078
OsId	0.076
IsHeaderBidding	0.074
OsParentId	0.061
InputBundleId	0.055
Inventory	0.049
PositionInAdBreak	0.005

Table 4.5: Correlation between each feature and target column

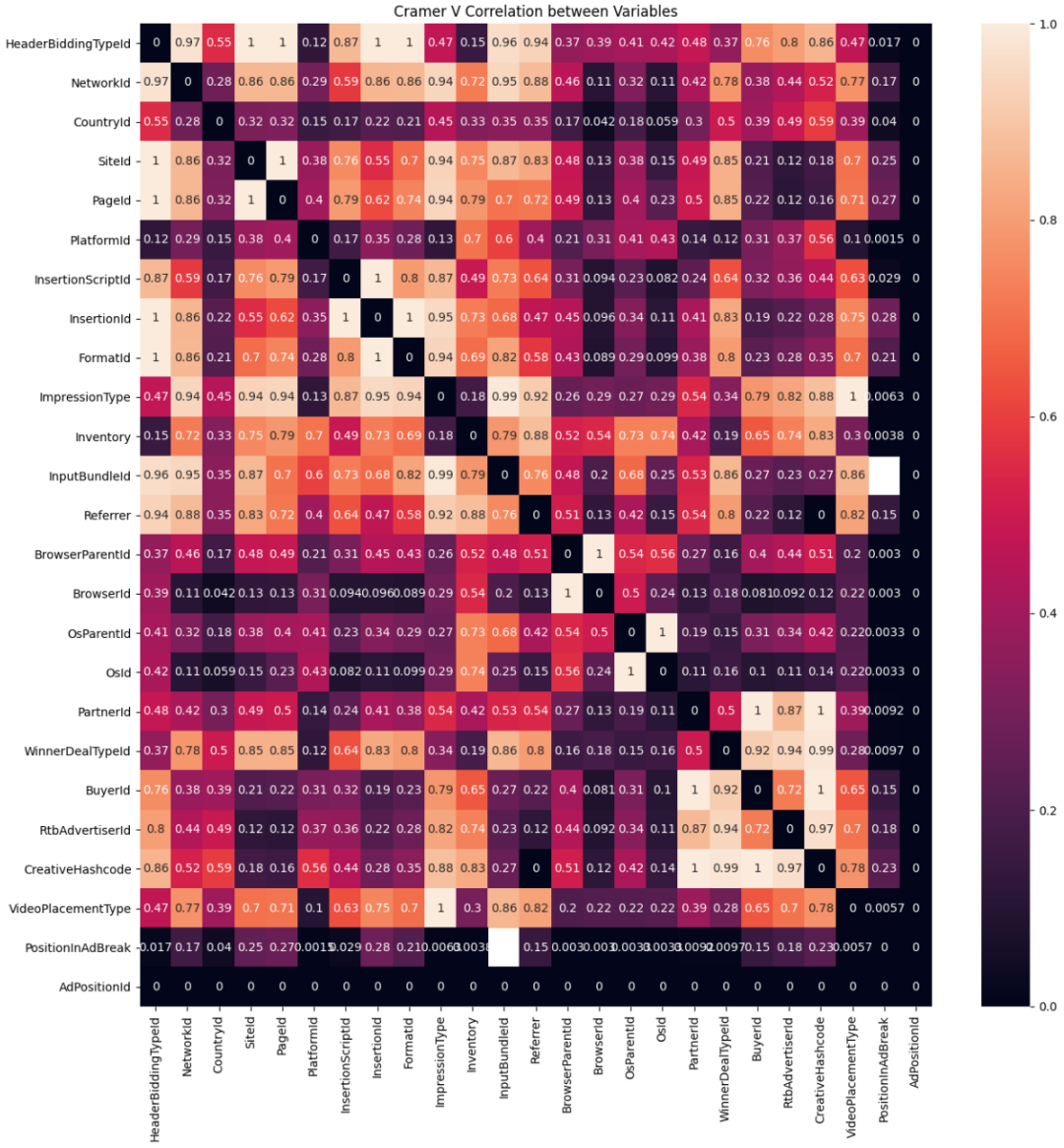

Figure 4.8: Heat Map of all the features

Chapter 5

Methodology and Development

This chapter explains the methodology, architecture, and implementation details. The methodology section explains the overview of the methodology, the reasons behind the chosen algorithms, and the required pre-processing steps. The implementation section provides the architecture details and implementation details of the chosen algorithms.

5.1 Methodology

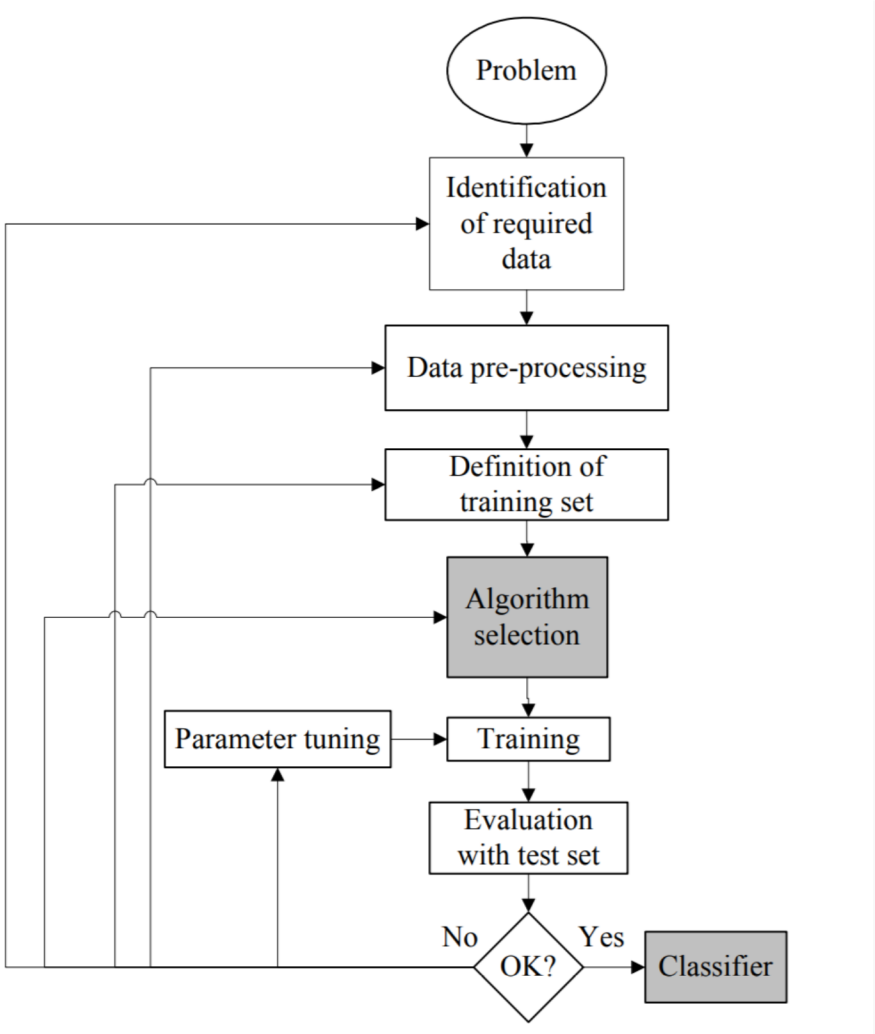

Figure 5.1: The life cycle of the project, taken from [Osisanwo 2017]

CRISP-DM, which stands for Cross Industry Standard Process for Data Mining framework, is applied since the beginning of the project. All the steps are followed along with the project, as displayed in figure 5.1. First of all, the first weeks are spent to know the problem, understand the key activities in the online advertising industry. This stage includes the basic understanding of the ecosystem, what the problem is with the current situation, how we can solve the problem, what the impact would be on the company, and on the client's side. The second step is data understanding. Data analysts, business- side help us to have a better understanding of the data. Also, data engineers support us in knowing the data flow in the system, how the data is stored, how different services interact with each other, and how to get the data from the database. Lastly, data exploration is performed and analyzed the correlation between different features. The next step is data preparation, which includes variable transformation, missing value treatment, and feature extraction. After that, the modeling part is implemented to apply the algorithms and predict the probability of each bid. This task includes separating the training, dev, and test sets, implementing the model, and fine-tuning. The evaluation step contains the comparison of different models based on the appropriate metric, which is discussed in section 6. The last stage is the deployment of the model. Once the model is decided, and it gives good results based on the evaluation step, the model is put in production and kept on monitoring the results. Moreover, finding better ways to improve the model. Once there is adequate progress in terms of performance metrics in the experiments, the improved model is put in production. The life cycle of a data science project is iterative and incremental; thus, it is open to new modifications if there is any need. These are all the steps that we follow throughout the project.

Figure 5.2: CRISP-DM

5.1.1 Overview

This section elaborates on the logic of our methodology. We would like to build a model that computes the probability of each bid response given by a DSP being delivered on the client-side. Consequently, we would like to maximize the expected revenue of a given SSP. Before the implementation of this project, the winner was chosen based on the bid price. Who bids the highest price becomes the winner regardless of their delivery rate. Based on

figure 5.3, without our methodology, we would have chosen Bid A as the winner because it is bidding the highest. However, there is something wrong with this logic because even though we choose a winner with the highest price, there is only 10% probability that this bidder can deliver the ad. Hence, it will fail to deliver the ad with 90% probability. The expected revenue is pretty low for bidder A, and most probably, the ad will not be exposed. Consequently, there is no revenue for an SSP and publisher.

Bidder	Bid price	Delivery rate (predicted)
A	$10	10%
B	$4	50%

Figure 5.3: Winner Selection

However, with the help of our model, we can predict the delivery rate of each bidder in an auction and calculate the expected revenue by multiplying bid price and predicted delivery rate. The new logic is based on maximizing the expected revenue and choosing the winner based on the highest expected revenue. Bidder A is bidding 10 € with a probability 10% of Delivery, and the expected revenue is 1 €. Bid B is bidding 4 € with a probability 50% of Delivery and the expected revenue of 2 €. In this case, we prefer the bidder B as a winner with the new methodology.

5.1.2 Machine Learning Algorithms

This section introduces the chosen machine learning algorithms, the intuition behind these algorithms, their advantages and disadvantages. Moreover, the advantages and disadvantages of different tree-based models are analyzed to decide which one to pick for further experiments. After that, it concentrates on the chosen tree-based and neural network models because these algorithms are used widely in the literature and produce tangible results.

First of all, a decision tree is a series of consecutive steps designed to address a specific question. They are intuitive and straightforward to interpret because they provide a clear visual guide about how to make a decision at each step. However, it has a high tendency to overfit, in case of having a complex problem. Over-fitting occurs when a model perfectly fits on a training set, and it fails to generalize on a test set, as displayed in figure 5.4. Over-fitting can happen for a few reasons, including the presence of noise, lack of samples, lack of representative instances, and having a complex model.

Therefore, it is better to have a look at tree-based ensemble models, it was already discussed in section 3.2.1 and 3.2.2. Even though building a single decision tree takes less time than ensemble models, the more trees to build, the more robust the model is. Consequently, a model generated by multiple trees and averaging the results diverge significantly from a model generated by only one tree. Having various trees and averaging the results would be closer to the correct answers. However, they are not as intuitive as single decision trees because it is more difficult to understand the decision ruling. Hence, the interpretability of complex models is still an open-ended research field. Random Forest

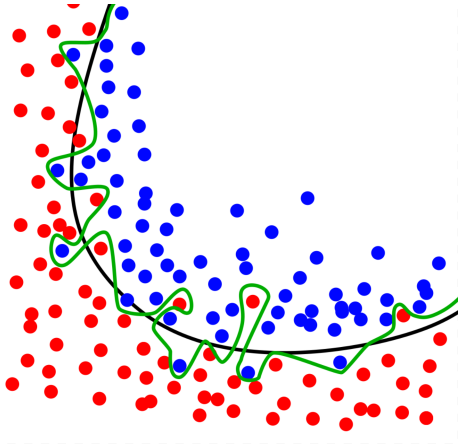

Figure 5.4: Illustration of over-fitting, from Wikipedia

and Extreme Gradient Boosting Trees are one of the most widely used state-of-the-art models. Both models can be used for classification and regression problems. They create multiple trees and aggregate the results. However, they vary in the way of constructing the trees and the way of combining the results.

It is easier to tune Random Forest than Gradient Boosting Trees because it has a few numbers of hyperparameters to tune. Moreover, each tree can be built in parallel because there is no dependency between them; thus, it enhances parallelism and performance. However, there are a few drawbacks of the Random Forest. One of the weaknesses of the Random Forests algorithm is that a large number of trees may make the algorithm slow for real-time prediction. Another disadvantage of using RF is biased feature importance for high cardinal features. It gives high cardinal features to higher importance even though it does not reflect reality. To sum up, the feature importance score is not reliable in Random Forest.

Gradient Boosting Trees constructs trees sequentially by learning from the mistakes of previous trees. It addresses to reduce the bias than the variance. Besides, Gradient Boosting Trees are slower than Random Forest because it builds one tree at a time, and each tree is dependent on the prior trees. Therefore, it becomes way slower when the number of trees to be built increases. Therefore [Chen 2016] introduced a scalable and highly effective algorithm called Extreme Gradient Boosting Trees, aka XGBoost, which addresses the drawbacks of Gradient Boosting Trees. XGBoost deals with sparse data and instance weights. It enables parallel and distributed computation; therefore, it makes the learning performant and resource-efficient.

Besides tree-based models, Neural Networks work better with bigger data size, as illustrated in figure 5.5. It enables scalability and parallelization; thus, it is highly flexible and performant. It allows us to build a better pipeline with TensorFlow and to apply continuous learning, as described in section 5.2.4. However, it is harder to tune neural networks compared to tree-based models and requires more expertise and effort.

All the tree-based models are compared and the results are evaluated in the Experiments section. Furthermore, the most performant tree-based algorithm, which is XGBoost, is compared with the Neural Network model, as discussed in section 6.2.1. To sum

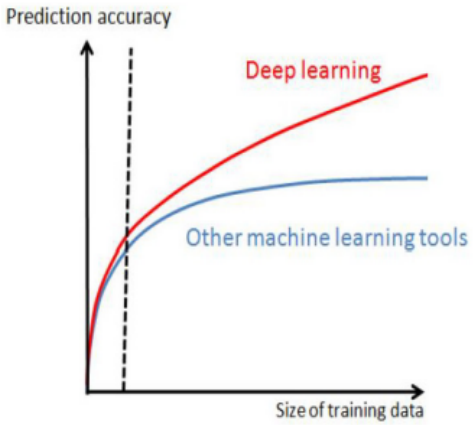

Figure 5.5: Prediction accuracy versus size of training data, from [D'Agaro 2018]

up, Extreme Gradient Boosting and Neural Networks are chosen as the main algorithms because they are highly scalable and the most performant state-of-the-art algorithms.

5.1.3 Pre-Processing

Tree-based and Neural Network models take numerical values as input. However, our data mostly consists of categorical and high cardinal features. Therefore, pre-processing operations are performed in order to prepare the data in an appropriate format for the machine learning models. Categorical and high cardinal data needs to be handled, aka categorical encoding, which refers to a transformation of categorical values to numeric values. One-Hot encoding, target encoding, and embeddings are the most used techniques for the encoding of the categorical data. Consequently, the target encoding is chosen for tree-based models, and embedding is chosen for the neural network model as discussed the advantages and disadvantages in the following sections.

5.1.3.1 One-Hot Encoding

One-Hot encoding is a sparse representation that consists of a single 1 and a bunch of 0s. It creates as many columns as you have cardinalities (unique values) in the categorical feature. This operation causes the feature space to grow quickly and the curse of dimensionality thus there is a decision to make for the representation of one-hot encoding as listed below.

- Dense Representation: 0s are stored in memory, which blows up the memory a lot if there are many features with high cardinality.
- Sparse Representation: Only 1s are stored in memory, which prevents us from memory wastage even though there are many features with high cardinality. However, the support of sparse representation is not common; therefore, it is crucial to find algorithms to support sparse representation.

In addition to the problem of the curse of dimensionality, each vector is equally distant to each other because one-hot encoding is not capable of capturing any rela-

tion between different categorical values. Moreover, there is an experiment made by [TowardsDataScience b], which compares the feature importance of each feature where uses one-hot encoding and does not use one-hot encoding. The finding of this experiment shows that one-hot encoding with high cardinal features causes inefficiency in tree-based ensembles, and the model conceals the feature importance of high cardinal features compared to continuous features, which result in poorer performance. In our case, there are many high cardinal features. For instance, CreativeHashCode consists of 84312 unique values and Referrer comprises 59350 unique values as shown in 4.4. Consequently, it is decided not to use one-hot encoding because it is more convenient since our data contains mostly high cardinal features.

5.1.3.2 Target Encoding

Target encoding is another approach of categorical encoding preferred by many Kaggle competitions. It is used to convert categorical values to numerical values by replacing a categorical value with the mean of the target variable. This approach is fast, intuitive, and requires less effort compared to embeddings. For every distinct value in a column, the average of the corresponding target values is computed. However, there is a disadvantage of this approach in the case that only a few samples appear of a corresponding category in the data set. This leads to over-fitting because the computed means relies on only a few samples. It has a high probability that different values are encountered in the test set. Hence, the computed mean from a few samples does not reflect the real value; it might be misleading. There are possible ways of overcoming this issue. The first approach is using cross-validation and computing the mean across different folds. The second approach is applying additive smoothing. The intuition behind this approach is smoothing the computed mean by the global mean. For instance, in our case, the target is the delivery rate, and we are interested in encoding the networkId column. Assuming that there is a new networkId that is contained by only three samples in the data set and it gives 0.9 of target mean for the new networkId whereas the target mean of all networkIds hover around 0.3. Therefore, it is better not to rely on only three values and to smooth the average by including the average delivery rate over all the networkIds. The formula is given below.

$$\mu = \frac{n \times \bar{x} + m \times w}{n + m} \tag{5.1}$$

where

- μ is the mean value that would be replaced the categorical values
- n is the number of values for the corresponding category in a column
- $\bar{x}$ is the estimated average target value by category
- m is the weight assigned for the overall mean
- w is the overall mean

m is the only variable that it is predefined, which indicates how much we rely on the overall average. The higher the value is, it relies on the overall mean, whereas the lower the values are, it relies on the estimated mean. In our case, the value 50 is chosen for the weight as a result of the obtained results of different experiments.

5.1.3.3 Entity Embedding

Entity Embedding maps discrete variables into multi-dimensional space, which consists of continuous values. This method maps categorical values to lower dimensional space; therefore, it allows us to reduce the memory usage. This approach is introduced by Cheng Guo and Felix Berkhahny [Guo 2016]. The main goal of this study is to map similar categories close to each other in the embedding space. These mappings are learned by the neural network during the training process. In the first layer, there is a one-hot encoding layer that maps each discrete value to a vector. In the second layer, there is an embedding layer, which is an extra layer on top of a one-hot encoded layer, as shown in figure 5.6. As can be seen, the mapped embeddings are just the weights of the embedding layer, and it can be learned as the parameters of a standard neural network layer. After embedding each feature into n-dimension, all embedding layers are concatenated and treated as a normal input layer. Moreover, hidden layers can be stacked on top of the merged layer like a standard neural network. The neural network is trained with the back-propagation method. Consequently, the weights of the embedding are initialized randomly in the beginning, and it is updated via gradient-descent during the training till it finds the weights in order to represent the embedding in the best way.

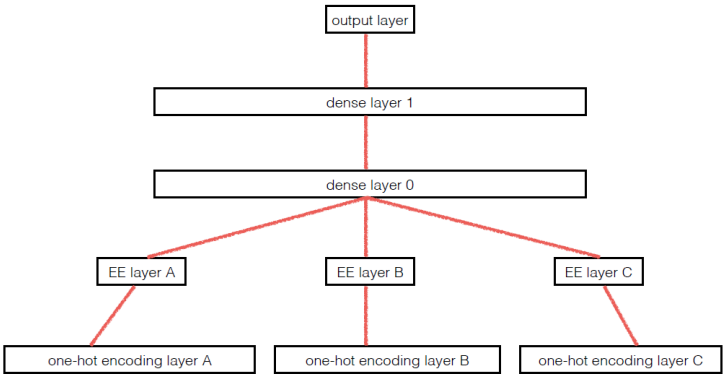

Figure 5.6: Illustration that entity embedding layers are equivalent to extra layers on top of each one-hot encoded input, from [Guo 2016]

Entity Embedding allows us to avoid a lot of pitfalls of one-hot encoding. First of all, it limits the number of columns per category, and it is able to capture similarities between similar variables. It performs well in many benchmarks. However, it requires more tuning compared to the previous methods therefore it takes more time to implement, as explained in the next paragraph. The dimension of the embeddings is the only hyper-parameter to be predefined. The dimension of each embedding layer can be different if it is necessary. The dimension of the layers can be between 1 and $m_i - 1$, m_i is the number of unique values in feature/dimension x_i. As the dimensionality increases in an embedding layer, the number of parameters to be optimized increases. In conclusion, the neural network gets more complex. Therefore, it tends to over-fit. This entails more work to fight against the over-fitting thus, it is crucial to find a balance between representing values in a better way and not making the network so complex. There is a dedicated experiment in the

6.2.5 in order to tune the embedding layer.

5.2 Implementation Details

This section consists of the architecture, the implementation details of XGBoost and neural network models. The architecture part is divided into two main sections. The first section 5.2.1 gives details of the overall project structure in production and the next section 5.2.2 gives information about how experiments run. The reason for dividing them into two sections is that various experiments are applied, and only one algorithm is put in production. Therefore the first section provides the architecture for a chosen algorithm, as explained in the Results chapter. Moreover, the next sections 5.2.3 and 5.2.4 give the implementation details of the XGBoost and neural network models because these two algorithms performed the best, as explained in the Experiments section.

5.2.1 Architecture in Production

This section provides the architecture details in production. Since XGBoost outperforms different tree-based algorithms and performs slightly better than Neural Network model as discussed in the Experiments section, it is chosen to put in production. Furthermore, any model can put in the architecture because it is flexible. Moreover, the entire pipeline is discussed, starting from getting the data, performing pre-processing operations, training, and predicting, as shown in 5.7. The delivery rate project consists of 2 main services; learning and prediction service.

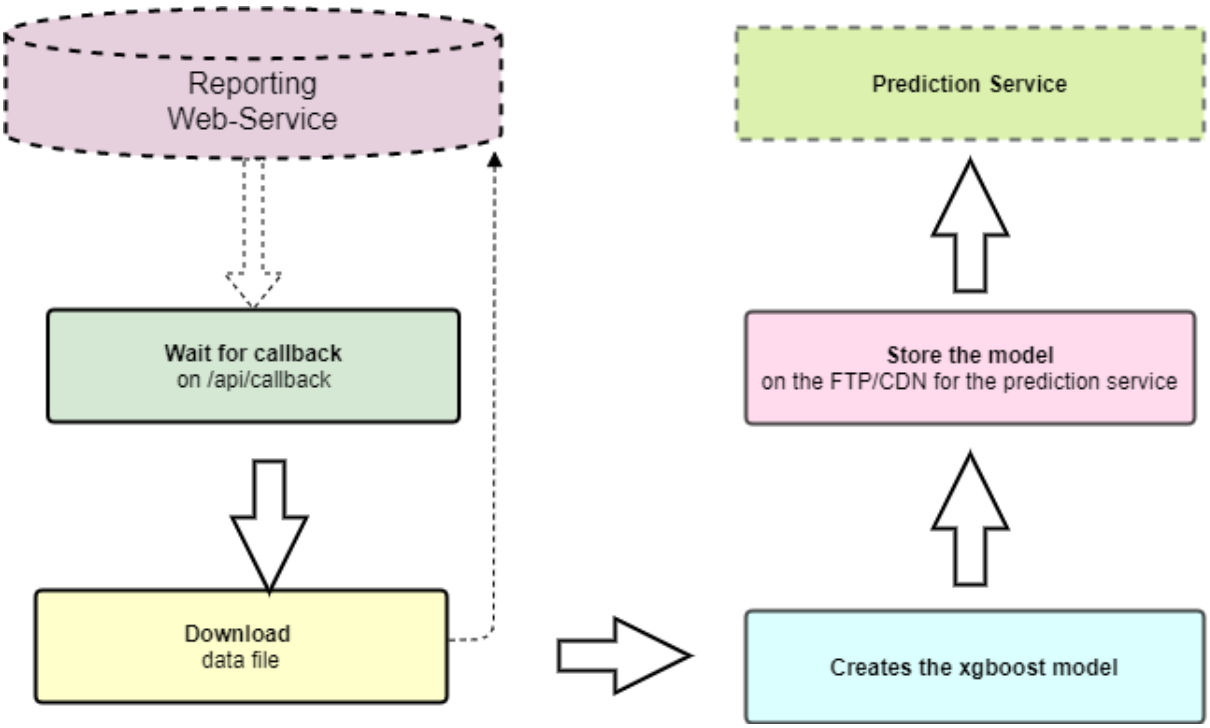

Figure 5.7: The entire pipeline of the delivery rate project

Reporting Service allows us to access the logs in HBase. There is a scheduled job whose task is getting the last 24 hours of data every hour. When the report is ready, there is a callback to our web application and the learning service follows the steps as follows.

- Downloading the calculated report
- Reading the downloaded CSV file
- Performing pre-processing operations to make the data ready for the algorithm
- Applying the algorithm aka learning process
- Storing the model in external storage

Moreover, the prediction service retrieves the model from the external storage, and it applies the trained model and returns a prediction for each bid. This is the structure of the delivery rate optimization project in production, once the algorithm is decided after the comparison of different models.

5.2.2 Architecture for Experiments

This section provides the architecture and implementation details for the XGBoost and Neural Network model for the experiment purposes before deciding which model to put in production.

5.2.2.1 Hardware and Software Environment

This section provides the hardware and software environment for implementing the training process and performing experiments. The hardware specifications of the local machine are listed below.

- System Model: XPS 15 9560
- System Type: x64-based PC
- Processor : Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 4 Core(s), 8 Logical Processor(s)
- System Type: x64-based PC
- Installed Physical Memory (RAM) : 16 GB

However, the local machine is used for limited experience. Most of the experiments are performed on the Google Cloud Platform, depending on the specifications. Since the data size is big, and the implemented model is complex. Consequently, a more powerful machine is required in order to parallelize the training process and load the data in-memory. n1-standard-16 machine containing 16 Virtual CPUs and 60GB of memory is used on GCP.

In addition to the hardware specifications, the operating system, software specifications are as follows.

- Windows 10 Pro
- Python 3.7.4
- PyCharm 2019.3.3 (Community Edition)
- Tableau Desktop 2019.4.3 (Profession Edition)

On top of these, there are several Python libraries are used depending on the task requirements shown in Appendix B.

5.2.2.2 Project Structure of Running Experiments

The master thesis is based on running experiments for various models and making a conclusion. Therefore the presented architecture in this section is used for the entire study before deciding which model to put in production.

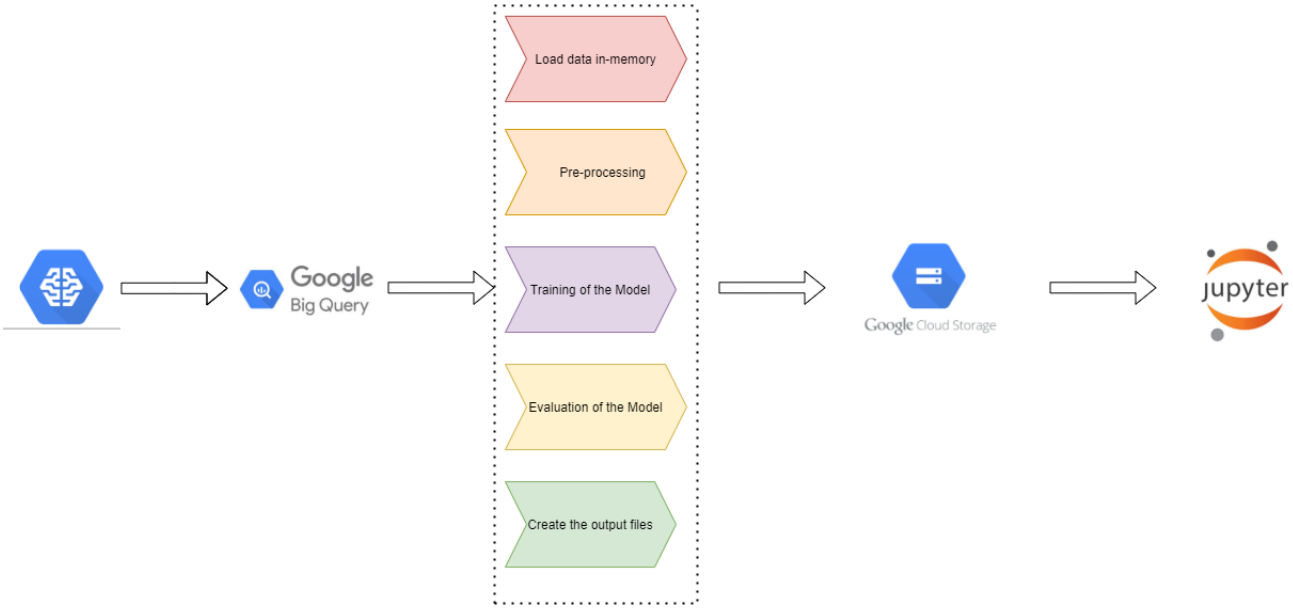

Figure 5.8: The entire pipeline of the delivery rate project

The entire pipeline shown in figure 5.8 is based on Google Cloud Platform because it has various advantages over working on a local machine. These advantages are listed below.

- Allocate resources as much as needed
- Improved Performance
- Allow collaboration, accessible amongst team members
- Provide state-of-the-art security
- Provide monitor the billing in order to have control over the cost

The architecture consists of Google AI platform, Google Big Query, and Google Cloud Storage. First of all, 17 days of data is stored in BigQuery, and the table is partitioned per day because it helps reduce the cost. Otherwise, for each query, it would query the full table, which is a costly operation because there are 24 million rows on average per day. The second component of GCP is Google Storage, where it is used to store the outputs of the model. The outputs of the model are the trained model, metric results of the train and test data in a CSV format, and feature importance of the trained model in a CSV format. All the outputs are stored in a bucket that is dedicated to the delivery rate project. The last component of GCP is Google AI platform, which is used for training, evaluating, and tuning machine learning models. There is a new job initiated when there is a new experiment that needs to be run. After introducing all the components of the architecture, the interaction between different components is done with the help of the python project. First of all, the flow starts with the initialization of a new job of the corresponding experiment, which with a python script, and it follows all the steps that are described below.

- Reading the data from BigQuery
- Performing pre-processing operations and data transformation
- Training of the model
- Evaluation of the training and testing data set
- Putting the output files to the bucket

After the training process, there is another python script run on Jupyter Notebook, which reads the output files on google cloud storage in order to analyze the evaluation metrics and feature importance.

5.2.3 Implementation of XGBoost model

As explained in section 6.2.1, XGBoost outperforms the other tree-based models. Therefore we focus on the implementation of the XGBoost model. This section provides the implementation details of the XGBoost model and some pre-processing operations because some pre-processing steps need to be performed in order to make the data ready. Our input data has missing values for two columns that are strings; thus missing value imputation is required. Also, it consists of mostly categorical data with high cardinality. Since the XGBoost model only works with numerical data, target encoding is used to convert categorical values to numerical values. Moreover, the input data includes the aggregated, and it needs to be transformed into a proper target column format for a binary classification problem.

First of all, our data is aggregated by time, which means that one row represents many samples that have the same combination of values of a given time. There is a small example given to demonstrate the input data structure, as shown in table 5.1. The input data is aggregated by day with only five dimensions.

Day	NetworkId	BuyerId	RtbImpressions	ImpressionsTrueCount
2020-02-01	125	15	3.0	1.0
2020-02-01	126	15	3.0	1.0
2020-02-01	2375	236	15.0	9.0
2020-02-02	126	15	4.0	3.0

Table 5.1: Small example of the structure of the input data

RtbImpression represents how many times the corresponding row win auctions on the server-side of a given time. ImpressionsTrueCount describes how many times the corresponding row delivered to an end-user. After the data is acquired from BigQuery, the data is divided into three as train, dev, and test set. The data is not split randomly because the time constraint plays an important role therefore, it is split based on the time constraint 5.2.

Experiment	Training Set	Dev Set	Test Set
Experiment 1	Day 1	Day 2	Day 3
Experiment 2	Day 2	Day 3	Day 4
Experiment 3	Day 3	Day 4	Day 5
...			
Experiment n	Day n	Day n+1	Day n+2

Table 5.2: Data Split

After missing value imputation and splitting the data, target encoding is applied by replacing a categorical value with the mean of the target variable, as explained in chapter 5.1.3.2. In our case, the target variable is the delivery rate thus, it is calculated by dividing ImpressionsTrueCount to RtbImpressions. Also, additive smoothing is applied in order to overcome the bias of a few samples in the data set. Additive smoothing allows us to smooth the new value by including the global delivery rate. Target encoding is applied for all the categorical features. Consequently, the values are calculated for each distinct value in a column and replaced by the target value. The code is shown as follows. After applying mean encoding, all the categorical values transform into continuous values that vary between 0 and 1.

```
def mean_encoding(dataframe, train_set, weight, features_to_enc):
    encoded_df = dataframe.copy()
    delivery_global = 1-(train_set['ImpressionsTrueCount'].sum() \
    / train_set['RtbImpressions'].sum())
    for f in features_to_enc:
        counts = train_set.groupby(f)['RtbImpressions'].sum()
        means = 1-(delivery_rate(train, f))
        smooth = (counts * means + weight *delivery_global) \
/ (counts + m)
        encoded_df[f] = encoded_df[f].map(smooth, na_action='ignore')
    return encoded_df.fillna(delivery_global)
```

The last operation before the training is to transform the data into a proper format for a binary classification problem because our target column should consist of 0s and 1s as not being delivered and delivered, respectively. It can be seen the format of the data before the transformation, as presented in table 5.1 and the structure of the data after the transformation, as shown in table 5.3. Each ImpressionsTrueCount is a success that belongs to the positive class. The difference between RtbImpressions and ImpressionsTrueCount is a failure that belongs to the negative class. Hence, IsSuccess column is created as the target column, which includes only 0s and 1s. Moreover, since each day has 24 million rows on average, the number of occurrences kept as the weight rather than disaggregating the data set because after this transformation, the number of rows are doubled to make the separation of each class as success and failure. Disaggregation, which means showing each sample in one row, would be so hard to maintain the data set because of its size. As it is shown in 5.3, there are eight rows in this small example of aggregated data. If we would like to disaggregate the data, we would end up with 25 rows, which equals to the number of RtbImpressions because each RtbImpressions can be interpreted as one sample/log.

Day	NetworkId	BuyerId	Weight	IsSuccess
2020-02-01	125	15	1	1
2020-02-01	125	15	2	0
2020-02-01	126	15	1	1
2020-02-01	126	15	2	0
2020-02-01	2375	236	9	1
2020-02-01	2375	236	6	0
2020-02-02	126	15	3	1
2020-02-02	126	15	1	0

Table 5.3: Small example of restructuring the input data

After all the pre-processing steps, the data is ready for the training process. XGBoost library is used to apply Extreme Gradient Boosting algorithm in python. Inside the XGBoost library, scikit-learn API is used to train the model. This API allows us to assign weights for each row in the data set, train the model, apply built-in early stopping method, and make predictions. The training set is used to train the model along with the weights. Development set -dev set- is used to monitor the log loss and helps the early stopping mechanism to decide when the results are converged. A test set is used to evaluate the performance of the trained model. After the training process, we store the best number of trees coming from the early stopping algorithm, re-train the model with the best number of trees and store the final model. After each training, the prediction function is called in order to calculate the different metrics of the associated model. After each training, there are three outputs of the model. The first one is the serialized version of the trained model in order to persist for future use with the joblib library. The second output is the metric results of training, dev, and test set along with the parameters of the model and used features during the training. The last output is a CSV file that contains the feature importance that is calculated by the model.

5.2.3.1 Feature Importance

This section elaborates on the feature importance of the initial model. There were 14 features used in the initial model, as presented in listing 6.1. The chart below demonstrates the feature importance of the model with 14 features as shown in 5.9. As can be observed SiteId, CreativeHashCode, Referrer, and InputBundleId are the most informative features in the model. Moreover, the last two attributes in the chart are the least important features; thus, it can be removed from the used features since the gain is really small.

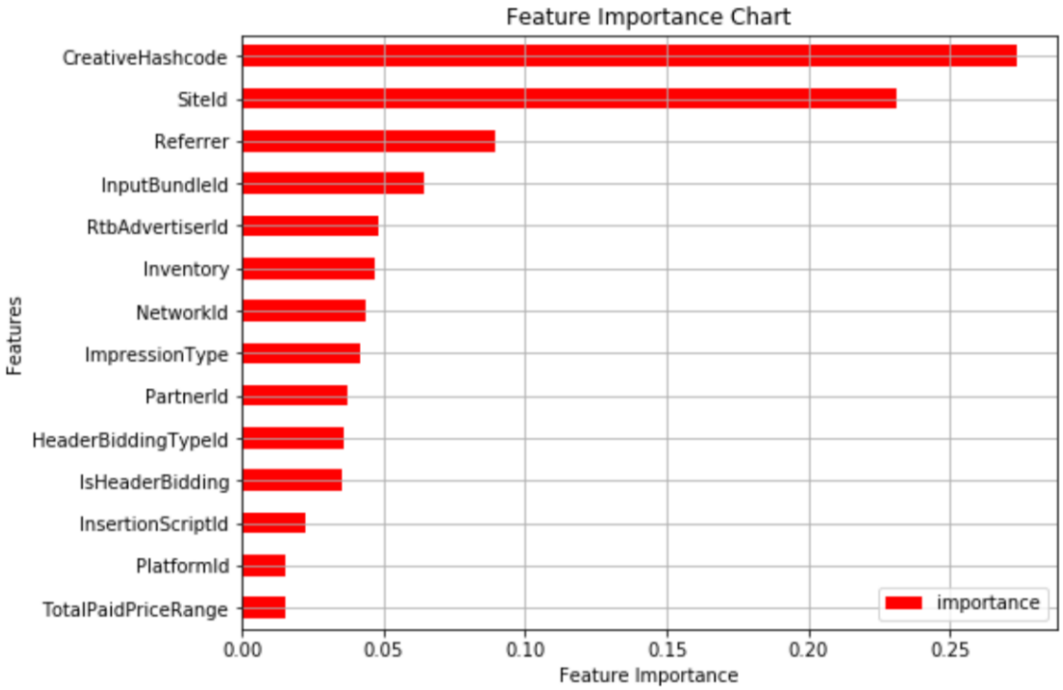

Figure 5.9: Feature Importance of 14 Features

The next figure 5.10 demonstrate the variation of the feature importance across days. Even though, SiteId and CreativeHashCode are the most important features, the importance of both features varies more than the other features across days.

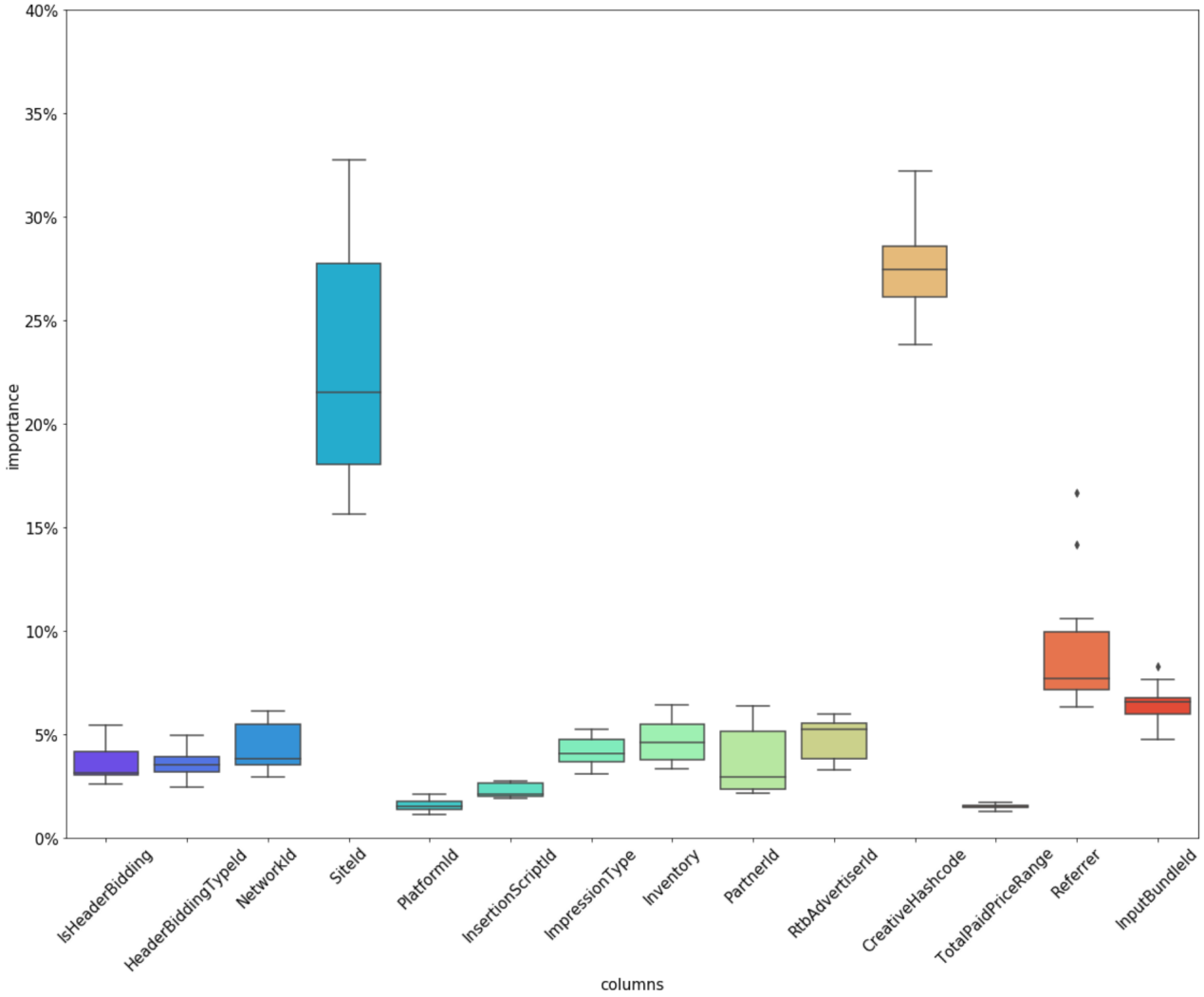

Figure 5.10: Feature Importance Variation of 14 Features

In addition to these features, the next figure 5.11 demonstrates the feature importance of SiteId, CreativeHashCode, TotalPaidPriceRange, and Referrer on 16 days of data. Each day is shown in different colors. It can be obtained that the feature importance of features is similar when the days are closer to each other due to the data variation over days. Also, the feature importance of TotalPaidPrice never changes, and it is very low; thus, it is proof that this feature does not bring any value to the model therefore, it should be removed.

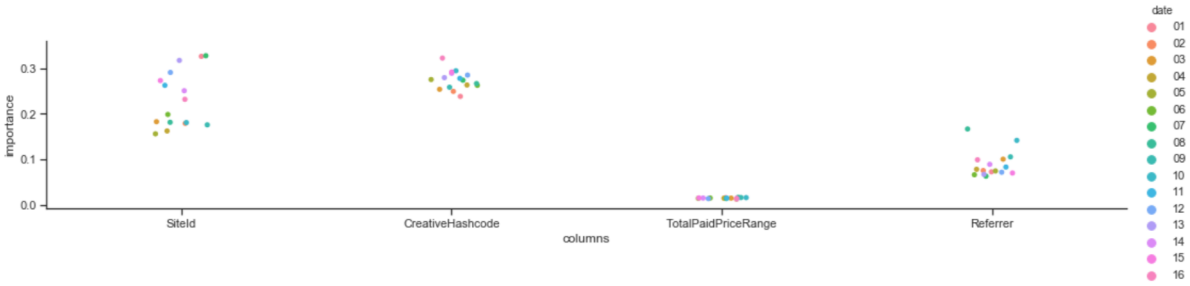

Figure 5.11: Feature Importance for some features on 16 days of data

In addition to the implementation details, some experiments are performed to improve the performance of the model, as described in section 6.2, such as the impact of adding new features to the model and parameter tuning.

5.2.4 Implementation of Neural Network Model

In this section, the implementation details of the neural network model are discussed. The model is implemented by using tf.keras submodule within TensorFlow, which is an end-to-end open-source platform for building machine learning and deploying ML applications. tf.keras is TensorFlow's implementation of the Keras API specification. This is a high-level API to build and train models for neural networks. TensorFlow brings some advantages and flexibility as follows.

- integration with Google Cloud Platfrom
- simplified pipeline management
- efficient usage of hardware, high performance
- a stable solution for serving in production
- possibility for transfer learning

These advantages make the neural network implementation more tempting. In addition to the general features of TensorFlow, embedding dimensions, drop out, number of hidden layers, number of neurons in each hidden layer are the parameters that need to be optimized in order to have the maximum performance from the model.

Before discussing the pre-processing steps, there is an extra step called data disaggregation. Unlike XGBoost, unweighted data is used during experiments. In order to do the first experiments, a sampling method with replacements on weighted data is used

provided by pandas in order to have the same number of rows as XGBoost algorithm. The sampling method is used in the first step of the experiments in order to compare the results, and after that, experiments are performed on the entire data set in order to have more reliable results.

After disaggregation of the data, there are some pre-processing operations that need to be performed before applying model the model, as in the case of XGBoost. There are a few pre-processing steps that are common between XGBoost and Neural Network model with some modifications. Since the input data is the same, the problem that we have is similar. There is a need for imputing the missing values, converting categorical values to continuous values, and transforming the structure of the data for a binary classification problem. The difference is the way of converting categorical values to continuous values.

Since all the features are categorical, the embedding layer is used provided by tf.keras in order to deal with categorical variables. Embedding allows reducing the dimensionality and capturing similarities between different categories. We represent each category by a vector of floating-point numbers. The values of these representations are learned as the network is trained.

The implementation of the Neural Network model is given below. First of all, the embedding layer is added one by one for each feature with a defined dimension size. In the end, all the embedding layers are concatenated. The merged (concatenated layer) is treated as a normal input layer, and the hidden layers are built on top of it. After adding the desired number of hidden layers with neurons, the output layer is stacked on top of the hidden layers with sigmoid function, which allows the model to produce outputs between 0 and 1 as probabilities. As can be seen that, drop-out method is applied after each layer in order to prevent the over-fitting. This part is explained in the Experiments section. In this case, there is no need for early stopping algorithm because if the Neural Network model is well-regularized, even though we train the model for more iterations, it will not overfit at some point, unlike XGBoost.

```
def create_model(columns_unique_values, cat_features, embed_dim, \
    drop_out, first_layer_neurons):
    inputs = []
    outputs = []
    for feature in cat_features:
        num_unique_values = columns_unique_values[feature]
        inp = tf.keras.layers.Input(shape=(1,))
        out = tf.keras.layers.Embedding(num_unique_values + 1, \
              embed_dim, name = feature) (inp)
        out =feature tf.keras.layers.SpatialDropout1D(drop_out)(out)
        out = tf.keras.layers.Reshape(target_shape=(embed_dim, ))(out)
        inputs.append(inp)
        outputs.append(out)
    x = tf.keras.layers.Concatenate()(outputs)
    x = tf.keras.layers.BatchNormalization()(x)

    x = tf.keras.layers.Dense(first_layer_nb_neurons, \
activation="relu")(x)
```

```
    x = tf.keras.layers.Dropout(drop_out)(x)
    x = tf.keras.layers.BatchNormalization()(x)

    y = tf.keras.layers.Dense(1, activation="sigmoid")(x)

    model = tf.keras.Model(inputs=inputs, outputs=y)

    return model
```

In order to build a neural network model, many experiments should be performed to decide which architecture fits the best for our case. All the performed experiments are shown in section 6.2.5 in order to find the final version of the neural network.

Chapter 6

Results and Evaluation

This chapter is divided into two main sections as metrics and experiments. Section 6.1 gives details about the metrics are chosen to evaluate different algorithms. Section 6.2 provides the evaluation of different tree-based models, some experiments in order to improve the models, and the overall comparison between the Extreme Gradient Boosting and Neural Network model.

6.1 Metrics

This section gives details about the metrics are chosen to evaluate different algorithms. Area Under the ROC Curve, aka AUC, precision, and recall, F1 scores are analyzed. However, AUC is the main metric in order to make the final decision in our case because it shows the ability of a classifier to distinguish between classes. Since our problem is a binary classification problem, it is important to have a model that can distinguish between positive and negative classes. AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance [Fawcett 2006].

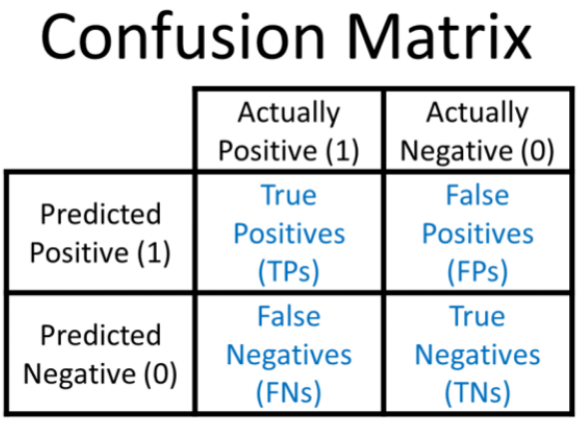

Figure 6.1: Confusion Matrix, taken from [glassbox]

It is also good to analyze the precision and recall of a classifier thus, we need a confusion matrix as shown above. Precision answers that of all the ads that are predicted as success, what fraction is actually success 6.1. Recall can be described as of all the ads that are actually success, what fraction did we correctly detect as being success 6.2. In the context of our problem, True Positive means classifying an ad as a success, whereas it is a success. True Negative is defined as classifying an ad as failure when it is a failure. These

two situations are desirable. However, False Positives and False Negatives are undesired and want to be minimized as much as possible. False Positives is described as classifying an ad as a success, whereas it is a failure. False Negative is defined as classifying an ad as a failure, whereas it is a success. False Positive and False Negative are equally important for our case. Hence, F1-score is added to evaluation metrics 6.2. F1-score will be zero if recall or precision is equal to zero, whereas F1-score will be one if recall and precision are equal to zero. Moreover, it is crucial to know when the recall of a classifier would be zero. If the classifier predicts all ads as failure, it will lead recall to zero.

$$\text{Precision} = \frac{TP}{TP+FP} \tag{6.1}$$

$$\text{Recall} = \frac{TP}{TP+FN} \tag{6.2}$$

$$\text{F1 Score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{6.3}$$

6.2 Experiments

This section includes experiments in order to evaluate the performance between different tree-based models, various experiments to increase the performance of the models, and the final comparison between Extreme Gradient Boosting and Neural Network models to make the final conclusion. As explained above, AUC is used as the main evaluation metric for all the experiments. As a reminder, the methodology is based on the iterative approach. At each step, changes are applied in order to see the impact in a model, and it is kept going further as it improves progressively.

6.2.1 Comparison of Tree-Based Models

The purpose of the experiment is to compare the performance for different tree-based models and choose an algorithm that performs the best in terms of the chosen metrics. The used models are Decision Tree, Random Forest, Ada Boost, and Extreme Gradient Boosting. Initially, 14 features are used, and they trained and tested on a daily basis. The data set is divided with respect to the time constraint, as given in table 6.1. Three data sets are used to run the experiments, and the results of different data sets are given separately.

Experiment	**Training Set**	**Test Set**
Experiment 1	Day 1	Day 2
Experiment 2	Day 2	Day 3
Experiment 3	Day 3	Day 4

Table 6.1: Data Split for Experiment 1

The table 6.2 demonstrates the training data set, used algorithm, and different metric results on the test data. The parameters of the algorithms are optimized in advance. The parameters of each model stay the same for different data sets. Moreover, table 6.3 shows

the fitting time in seconds of each model on different data sets. From the tables below, key findings emerge as follows.

Decision Tree performs the worst compared to ensemble models for each data set in terms of log loss. Even though the model complexity is kept low enough to fight with the variance but high enough to have good results, the variation between training and test is really high on 2020-02-02 for the decision tree model. Since the depth of the decision tree is low, the training takes the least time amongst the others. Once we increase the complexity, over-fitting occurs very easily, and it takes a longer time to build the tree since the process is not parallelizable. Random Forest outperforms AdaBoost in terms of loss, recall, accuracy, AUC, and F1 except for precision. Moreover, the variation between training and test scores of the Ada Boost model is really high on 2020-02-02, alike the Decision Tree model. Also, once we compare the fitting time for different models, Ada Boost takes a substantial amount of time compared to Random Forest due to the nature of the boosting framework. And Random Forest takes less time because the trees do not depend on each other, and it can be built in parallel easily. Lastly, Extreme Gradient Boosting performs the best after averaging the results of the data sets. In addition, Random Forest and Extreme Gradient Boosting are really close to each other in terms of fitting time due to the implementation improvements of Extreme Gradient Boosting.

Data Set	Model	LogLoss	Precision	Recall	Accuracy	AUC	F1
2020-02-01	Decision Tree	0.7477	0.8820	0.6223	0.6726	0.7942	0.7297
2020-02-01	Ada Boost	0.7259	0.8798	0.6099	0.6634	0.7705	0.7204
2020-02-01	Random Forest	0.6130	0.8789	0.6176	0.6678	0.7907	0.7254
2020-02-01	XGBClassifier	0.5960	0.8869	0.6242	0.6766	0.8059	0.7327
2020-02-02	Decision Tree	2.4462	0.7590	0.9019	0.7287	0.6389	0.8243
2020-02-02	Ada Boost	0.7950	0.7621	0.8882	0.7247	0.6562	0.8203
2020-02-02	Random Forest	0.4752	0.7496	0.9541	0.7452	0.7186	0.8396
2020-02-02	XGBClassifier	0.4716	0.7517	0.9358	0.7381	0.7330	0.8337
2020-02-03	Decision Tree	0.4975	0.8563	0.8996	0.8241	0.8728	0.8774
2020-02-03	Ada Boost	0.4750	0.8566	0.8960	0.8221	0.8431	0.8759
2020-02-03	Random Forest	0.3438	0.8548	0.9084	0.8285	0.8741	0.8808
2020-02-03	XGBClassifier	0.3081	0.8586	0.9062	0.8306	0.9042	0.8818

Table 6.2: Results of trained tree-based models on three data sets

In conclusion, the XGBoost model outperforms the other tree-based models; therefore, following sections are concentrated on the XGBoost model for further improvements.

6.2.2 Importance of Freshness of the Model

The goal of this experiment is to analyze the importance of recalculating the model. Since XGBoost gives the best result from the previous experiment, the XGBoost algorithm is used to train the model. The model was trained and tested on a daily basis before. However, we would like to see the impact of training and testing on an hourly basis. In detail, the model was trained on the last day, and the evaluation metric was computed on the next day in the past experiments. However, with the current changes, the model is

Data Set	Model	FitTime
2020-02-01	Decision Tree	11.7987
2020-02-01	Ada Boost	2,295.0980
2020-02-01	Random Forest	242.4515
2020-02-01	XGBoost	210.0059
2020-02-02	Decision Tree	8.2435
2020-02-02	Ada Boost	1,631.6850
2020-02-02	Random Forest	129.5035
2020-02-02	XGBoost	186.1548
2020-02-03	Decision Tree	55.2040
2020-02-03	Ada Boost	11,806.0894
2020-02-03	Random Forest	1,278.4752
2020-02-03	XGBClassifier	1,436.9810

Table 6.3: Results of fitting time in seconds on three data sets

trained in the last 24 hours, and the evaluation metric is calculated in the next 4 hours. The intention of this change is to dissect the freshness of the model. This experiment is done on the first model by using 14 features. The computation of performance metrics in the next 4 hours, and the next day are compared by using three days of test data. The chart below 6.4 shows the average performance metrics on a daily basis, an hourly basis, and the improvement of an hourly basis compared to a daily basis in terms of percentages.

	Hourly Basis	**Daily Basis**
	AUC	AUC
Day 1, Test Data	0.805977	0.805917
Day 2, Test Data	0.751866	0.733012
Day 3, Test Data	0.904297	0.904175
Average on 3 days	0.859501	0.820714
Improvement	4.51%	

Table 6.4: The result of performance metrics on a daily and hourly basis

As can be seen that when the model is trained and tested on an hourly basis, the performance metric is higher than a daily basis. It shows us the importance of recalculating the model as often as possible because the variation decreases between data points when they are closer to each other in terms of time. Therefore it is crucial to compute the model as often as possible in order to have higher the performance of the model because the model is trained and tested with the most current data. It is also important to keep data availability into account. In our system, we can access the last 24 hours of data every hour; therefore, we can train the model every hour.

As described before, our approach is an iterative improvement. Since XGBoost outperforms the other tree-based models and it gives better results with the most current data, it is put in production with 14 features, and it is trained every hour. The architec-

ture of the model is already explained in section 5.2.1. Other improvements are applied in the coming sections.

6.2.3 Experiments of XGBoost Model

This section contains three different types of experiments regarding the XGBoost model. The first experiment is to demonstrate the impact of new features on the model. The second experiment is to show the effect of early stopping method, and the last experiment is the tuning of the model in order to improve the performance.

6.2.3.1 Impact of New Features

With the exploration of new RTB data, new features become available to use; thus, new features coming from RTB dimensions are added to the model in addition to the 14 features as shown in listing 6.2.

Listing 6.1: First features

```
FEATURES = ['IsHeaderBidding',
 'HeaderBiddingTypeId',
 'NetworkId',
 'SiteId',
 'PlatformId',
 'InsertionScriptId',
 'ImpressionType',
 'Inventory',
 'PartnerId',
 'RtbAdvertiserId',
 'CreativeHashcode',
 'TotalPaidPriceRange',
 'Referrer',
 'InputBundleId']
```

Listing 6.2: Additional features

```
['CountryId',
 'PageId',
 'InsertionId',
 'FormatId',
 'BrowserParentId',
 'BrowserId',
 'OsParentId',
 'OsId',
 'WinnerDealTypeId',
 'BuyerId',
 'RtbVideoPlayerWidth',
 'RtbVideoPlayerHeight',
```

```
'CreativeWidth',
'CreativeHeight',
'VideoPlacementType',
'PositionInAdBreak']
```

Also, new features were generated by using width and height dimensions in addition to the new features above because the combination of height and weight would be more informative rather than analyzing individually. For instance, CreativeSize, namely CreativeWidth_CreativeHeight was derived by using CreativeWidth and CreativeHeight features.

The chart below demonstrates the comparison of the model performance for the model with all the features and the model with only 14 features in table 6.5. The experiment is done on three days of data by using the Google AI Platform, and the results below are averaged on three days of data.

	30 Features Model	**14 Features Model**
	AUC	AUC
Day 1, Test Data	0.848858	0.805917
Day 2, Test Data	0.762511	0.733012
Day 3, Test Data	0.912691	0.904175
Average on 3 days	0.841353	0.814368
Improvement	3.21%	

Table 6.5: AUC scores of both model

As can be seen in the table above, the model with the new features has improved by 3.21% compared to the 14 features model since there is a good level of improvement, it is important to analyze the feature importance of the new model.

The chart above shows the feature importance of each feature in the new model, as shown in figure 6.2. As can be seen in the table, newly added dimensions have a great impact on our model such as; PageId, CreativeWidth_CreativeHeigh, WinnerDealTypeId, FormatId, VideoPlacementType and RtbVideoPlayerWidth_RtbVideoPlayerHeight.

6.2.3.2 Early Stopping Experiment

This section provides a discussion about early stopping strategy, the reasons, and the effects of applying it in our case. A set of experiments is performed in order to perform hyper-parameter optimization which will be discussed in the next section. During these experiments, the optimal number of trees are stored in the output file. However, during the experiments, it is observed that there is a high variation in terms of model complexity across different days as can be seen in figure 6.3. By means of this, some days are easy to train therefore a simpler model is needed and some days are harder to find the pattern, and a more complex model is required. This implies that the model should be dynamic in order to answer the needs of data for a given time. Therefore, it shows the importance of applying the early stopping rather than setting a fixed number of trees in order to handle the variance in model complexity over time. It has been a major challenge to decide how

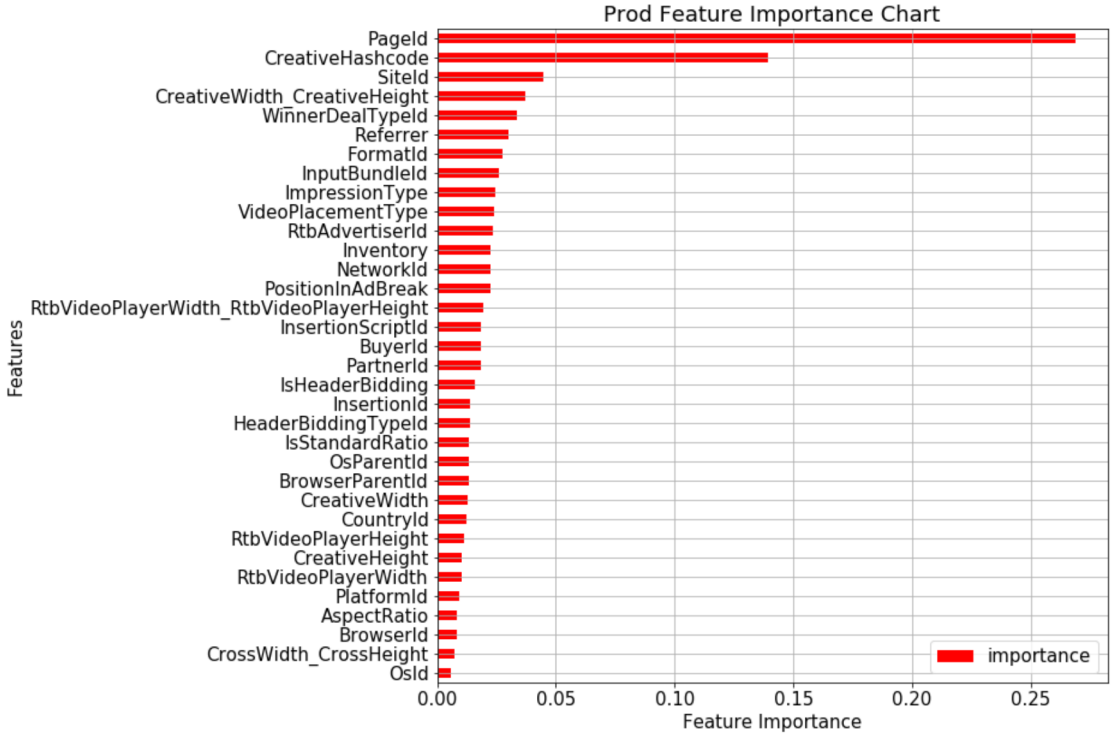

Figure 6.2: Feature Importance of New Features

long a model should be trained for machine learning algorithms. It allows us to train a model well enough and prevent over-fitting. If a model stops the training process earlier than it is required, it leads to under-fitting thus it is not able to capture the substantial patterns and consequently, it performs poorly. Early Stopping method allows us to keep on training a model unless there is no more improvement on dev set. It determines the optimal number of trees to construct before over-fitting. It is an effective and widely used approach in machine learning models. A substantial study by Tong Zhang and Bin Yu showed the importance of early stopping in boosting trees [Zhang 2005]. A large number of existing studies in the broader literature have examined over-fitting in tree-based ensemble learning. For instance [Grove 1998] conducts a study by showing that one of the boosting techniques, aka AdaBoost, overfits eventually, and it performs worse on a test set; therefore, early stopping is required.

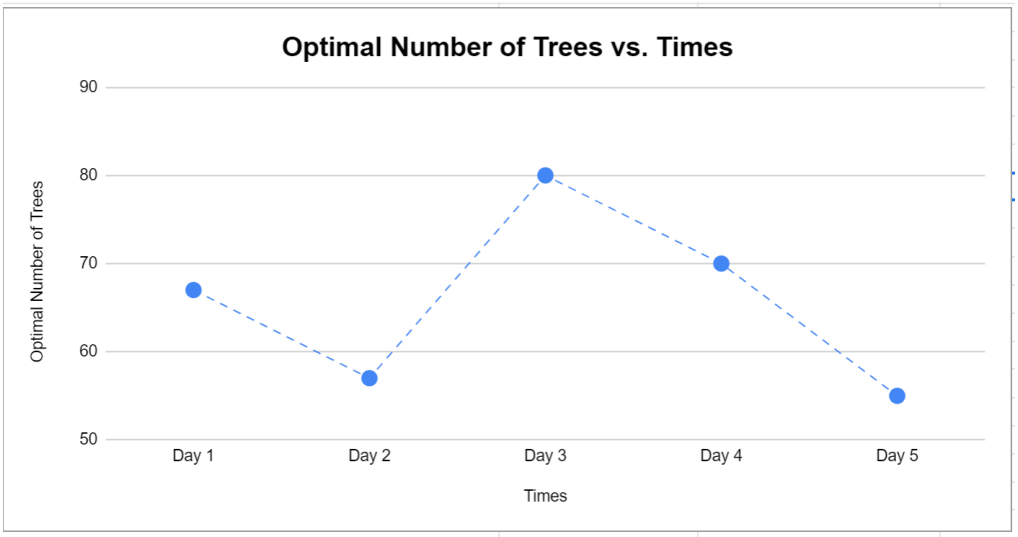

Figure 6.3: Optimal number of trees over days

After analyzing the definition and the usage in the literature review, it is crucial to check the feasibility in our scenario. Thus, two illustrations are given in order to see the effect of the early stopping method during the training process by our model. Log Loss metric is monitored during the training process. As can be seen in figure 6.4, over-fitting occurs after the 60th iteration. As proof of over-fitting, training loss gradually decreases, whereas the dev loss starts to increase slightly at some point. Since there is no early stopping mechanism, the model builds as many trees as the predefined number of estimators regardless of the need of the model. However, in figure 6.5, as can be seen, the early stopping mechanism stops the training after the 70th iteration since there is no more improvement on the dev set in terms of the evaluation metric, which is log loss. Early stopping prevents us from over-fitting and building more trees when it is not necessary. Consequently, the training takes less time in this particular example.

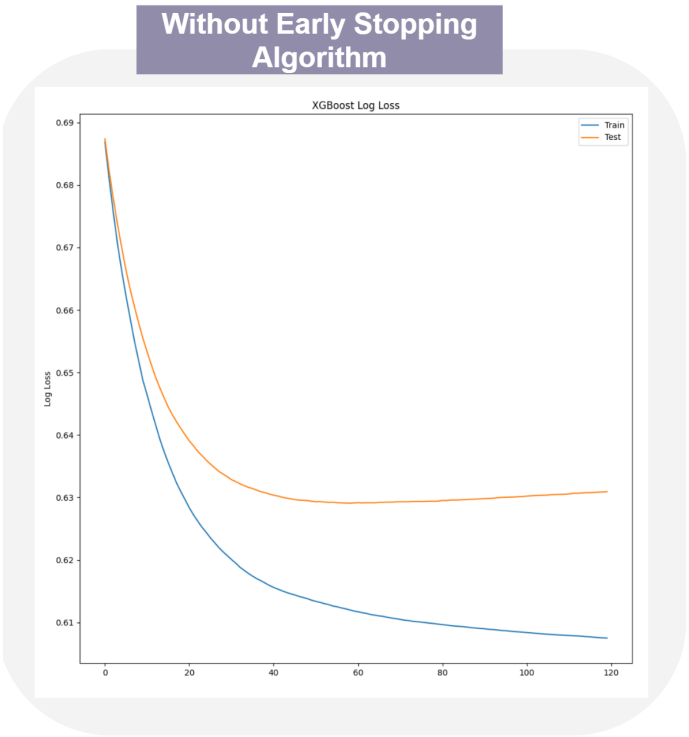

Figure 6.4: Monitoring the training process of the model without early stopping

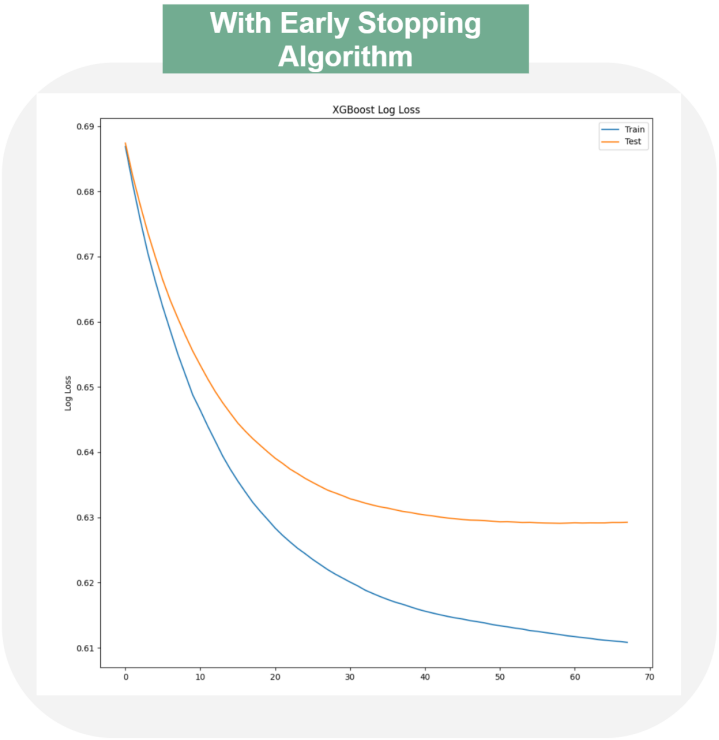

Figure 6.5: Monitoring the training process of the model with early stopping

6.2.3.3 Hyper-Parameter Optimization

The goal of this section is to tune the XGBoost model and compare the results of different experiments since some changes are made to the model; adding new dimensions and applying early stopping method.

We started tuning the model by Bayesian model-based optimization because the search space is huge and we would like to decrease the number of times to run the model and find a set of parameters that give us smaller search space. Bayesian optimization is really efficient because previous tests are used to decide which values to pick for the next experiments. After restricting our search space, grid search is applied. The aim is to find the best combination of all parameters that give the highest AUC score. We keep the number of experiments small for grid search because training on a few days takes a long time and it leads to a high cost on Google Cloud Platform. The reason for running on multiple days is due to the variation of AUC over days as shown in figure 6.6 there is no trend over days thus it is crucial that experiments are performed on multiple days and averaged in order to have more reliable results.

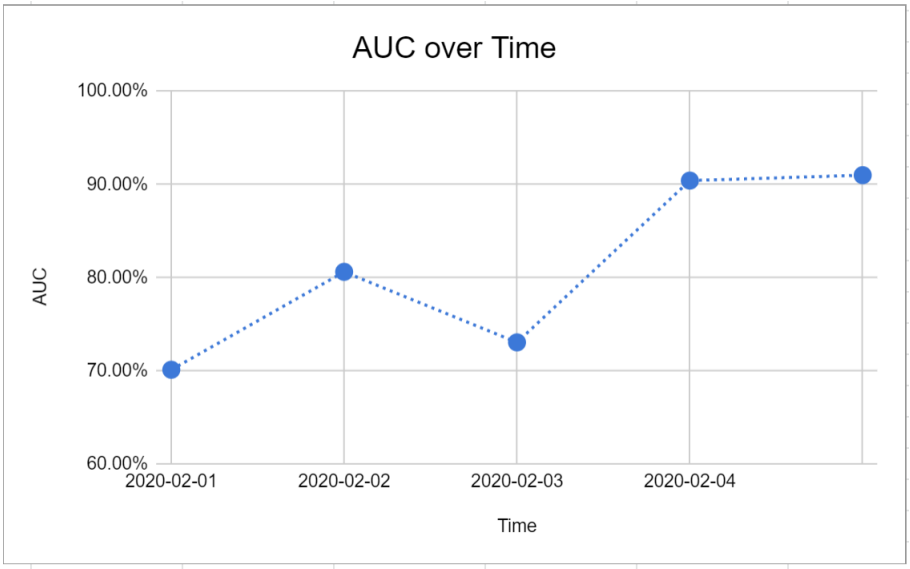

Figure 6.6: AUC variation over days

The most important parameters that should be tuned are as follows.

- Max Depth
- Number of Trees
- Learning Rate
- Column Sub-sampling

Each parameter is explained in section 3.2.2.1 in more detail therefore we go over them quickly. In the previous section, it is observed that there is high variation across different days as shown in figure 6.3 thus early stopping is applied rather than setting a fixed

number of trees in order to address the variance in model complexity consequently there is no need to tune the number of trees. The learning rate is one the most important parameter to tune because it is used in update by shrinking the step sizes in order to prevent the over-fitting. It allows us to update the feature weights in each step in order to make the model more robust. Another important feature is the maximum depth of a tree which signifies the complexity of each base learner. As the depth of the trees is increased, the complexity of the model is increased thus it is more likely to overfit. The reason for causing over-fitting is that the escalation of depth makes a model to learn specific patterns on the training set. Also, as we increase the depth of a tree, memory consumption significantly increases. Another substantial parameter is a column by sampling which is the sub-sampling technique of columns. It represents the proportion of features to be randomly chosen during the construction of each base learner. It is another way of controlling the over-fitting. As colsamplebytree is increased, it is more likely to overfit whereas if it is decreased more than enough, it leads to under-fitting. Also, the chosen value for colsamplebytree directly affects the memory consumption and taken time to train a model.

XGBoost provides a nice way of finding the best number of trees by applying the early stopping strategy which we discuss in the previous section, therefore, we do not need to configure the number of trees. The maximum possible number of trees to build is set to 140 because it never goes beyond it. Also, the numberofrounds is set to ten which is required by early stopping. If the performance does not improve for ten rounds/iterations, the algorithm stops the training process. In addition to the number of trees parameter, a set of values are given for the rest of the parameters.

- Max Depth = [20, 22]
- Number of Trees = [140]
- Learning Rate = [0.05, 0.1, 0.15]
- Column Sub-sampling = [0.6,0.8]

In total, 12 experiments are run on three different data sets. The given results are averaged on the results of different data sets in table 6.6 and 6.7. The table includes the experiment number, learning rate, column sub-sampling ratio, max depth, and averaged AUC and F1 scores, respectively. Table 6.6 represents experiment results on training data and table 6.7 demonstrates experiment results on test data.

Exp	LR	Colsample	Maxdepth	Train AUC	Train FScore
Exp 1	0.1	0.6	20	0.839409	0.83820
Exp 2	0.1	0.8	20	0.837882	0.83835
Exp 3	0.15	0.6	21	0.838511	0.838511
Exp 4	0.15	0.8	21	0.837850	0.837850
Exp 5	0.1	0.6	22	0.839433	0.839433
Exp 6	0.1	0.8	22	0.839319	0.839319
Exp 7	0.15	0.6	22	0.839143	0.839143
Exp 8	0.15	0.8	22	0.838697	0.838697
Exp 9	0.05	0.6	20	0.838417	0.838417
Exp 10	0.05	0.8	20	0.838284	0.838284
Exp 11	0.05	0.6	22	0.839418	0.839418
Exp 12	0.05	0.8	22	0.839113	0.839113

Table 6.6: Hyper-parameter optimization results on Training Date

Exp	LR	Colsample	Maxdepth	Test AUC	Test FScore	Fit Time
Exp 1	0.1	0.6	20	0.839409	0.853428	6.99
Exp 2	0.1	0.8	20	0.837340	0.853650	7.30
Exp 3	0.15	0.6	21	0.840347	0.854269	4.31
Exp 4	0.15	0.8	21	0.838153	0.853014	4.58
Exp 5	0.1	0.6	22	0.841073	0.853609	6.96
Exp 6	0.1	0.8	22	0.8409051	0.852888	8.10
Exp 7	0.15	0.6	22	0.837735	0.853287	4.88
Exp 8	0.15	0.8	22	0.832060	0.852120	5.55
Exp 9	0.05	0.6	20	0.844148	0.853971	8.68
Exp 10	0.05	0.8	20	0.840420	0.853370	9.80
Exp 11	0.05	0.6	22	0.843017	0.853686	12.94
Exp 12	0.05	0.8	22	0.838153	0.852576	14.02

Table 6.7: Hyper-parameter optimization results on Test Data

As can be seen that amongst the 12 experiments, the best result is 0.844 of AUC by Experiment 9. As a result, the best parameters that give the highest score when max depth is 20, the number of estimators is 140, the learning rate is 0.05 and the colsamplebytree is 0.6. Furthermore, the training takes 8.68 minutes in total which is reasonable.

6.2.4 Experiments for Neural Network Model

In order to build a neural network, many experiments should be performed to decide which architecture fits the best for our problem. The parameters need to be optimized as follows:

- Learning Rate
- Embedding dimensions
- Number of hidden layers
- Number of neurons in each layer
- Drop-out probability

Experiments were run in order to find a good learning rate amongst 0.1, 0.01, 0.001, and 0.0001. The learning rate is chosen as 0.001 because it gives the best result. The used technique is that the model starts with a learning rate of 0.001, and it decreases by a factor of 0.1 if there is no more improvement in the model performance for five epochs.

The goal of the first experiment is to show the impact of varying the embedding dimensions and find the best dimension size in our case. In order to perform this experiment, the most basic Multi-Layer Perceptron is used. The architecture consists of an embedding layer, one hidden layer (32 units), and an output layer. As can be seen in table 6.8, train AUC, test AUC, and test loss are given regarding different embedding dimensions.

Dimension of Embeddings	Train AUC	Dev AUC	Test AUC	Test Loss
1 dimension	0.9131	0.8863	0.8777	0.3637
2 dimension	0.9188	0.8686	0.8635	0.3875
3 dimension	0.9271	0.8676	0.8610	0.399
4 dimension	0.9298	0.8673	0.8606	0.3963
5 dimension	0.9308	0.8712	0.8652	0.3931
6 dimension	0.9345	0.8680	0.8621	0.4012
7 dimension	0.9376	0.8684	0.8623	0.4043
8 dimension	0.9381	0.8680	0.8581	0.4120
9 dimension	0.9471	0.8658	0.8588	0.4184
10 dimension	0.9393	0.8679	0.8595	0.4172
14 dimension	0.9415	0.8606	0.8529	0.4279

Table 6.8: Different Dimensions in Embeddings

The difference between train and test scores reflect the variance of the model. There is a trade-off between having higher training and test scores. Having a higher training score indicates a higher potential capacity of the model. However, increasing the potential of the model gives more work to fight against the variance. When we examine the AUC results of the dev set and test set, the best score belongs to the dimension of 1, and the second-best belongs to the dimension of 5. Therefore it is important to check their loss, AUC, and accuracy charts over epochs in order to see if over-fitting occurs during the

training process. Figure 6.7 shows accuracy, loss, and AUC charts over epochs when one is chosen as an embedding dimension. In contrast, figure 6.8 shows accuracy, loss and AUC charts over epochs when five is chosen as an embedding dimension.

As we can see, both charts stay stable over epochs, so over-fitting is not an issue here. However, it occurs slightly, and it can be fixed in the next steps.

As a conclusion of the embedding experiment, since more than half of our features have high cardinality, choosing a dimensionality of 5 would be more promising than the dimensionality of 1. Moreover, the gain is 2 points in training while losing 1 point in the test compared to the dimensionality of 1. After choosing the dimensionality of the embedding layer, a new experiment is performed for the hidden layer optimization.

The goal of this experiment is to show the impact of varying the hidden layer units and decide which one suits better for our problem.

Hidden Layer Units	**Train AUC**	**Test AUC**
64 units	0.9343	0.8606
80 units	0.935	0.861
100 units	0.935	0.8618
128 units	0.9338	0.8639
140 units	0.9349	0.864
180 units	0.9365	0.8635
128-64 units	0.9324	0.8766
64-128 units	0.9281	0.881

Table 6.9: AUC results for different hidden layer units with sampling

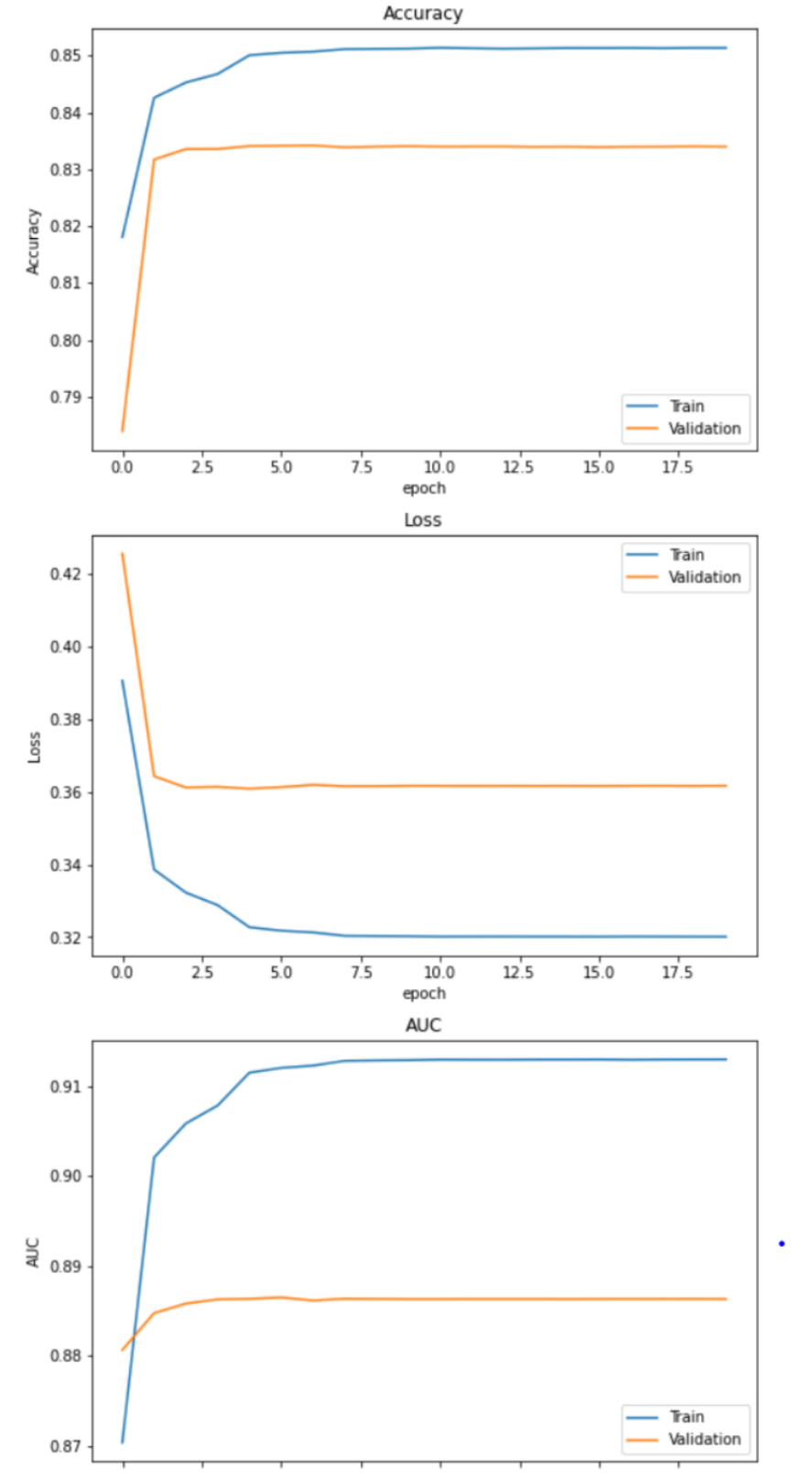

Figure 6.7: Accuracy, Loss and AUC charts for dimension of 1

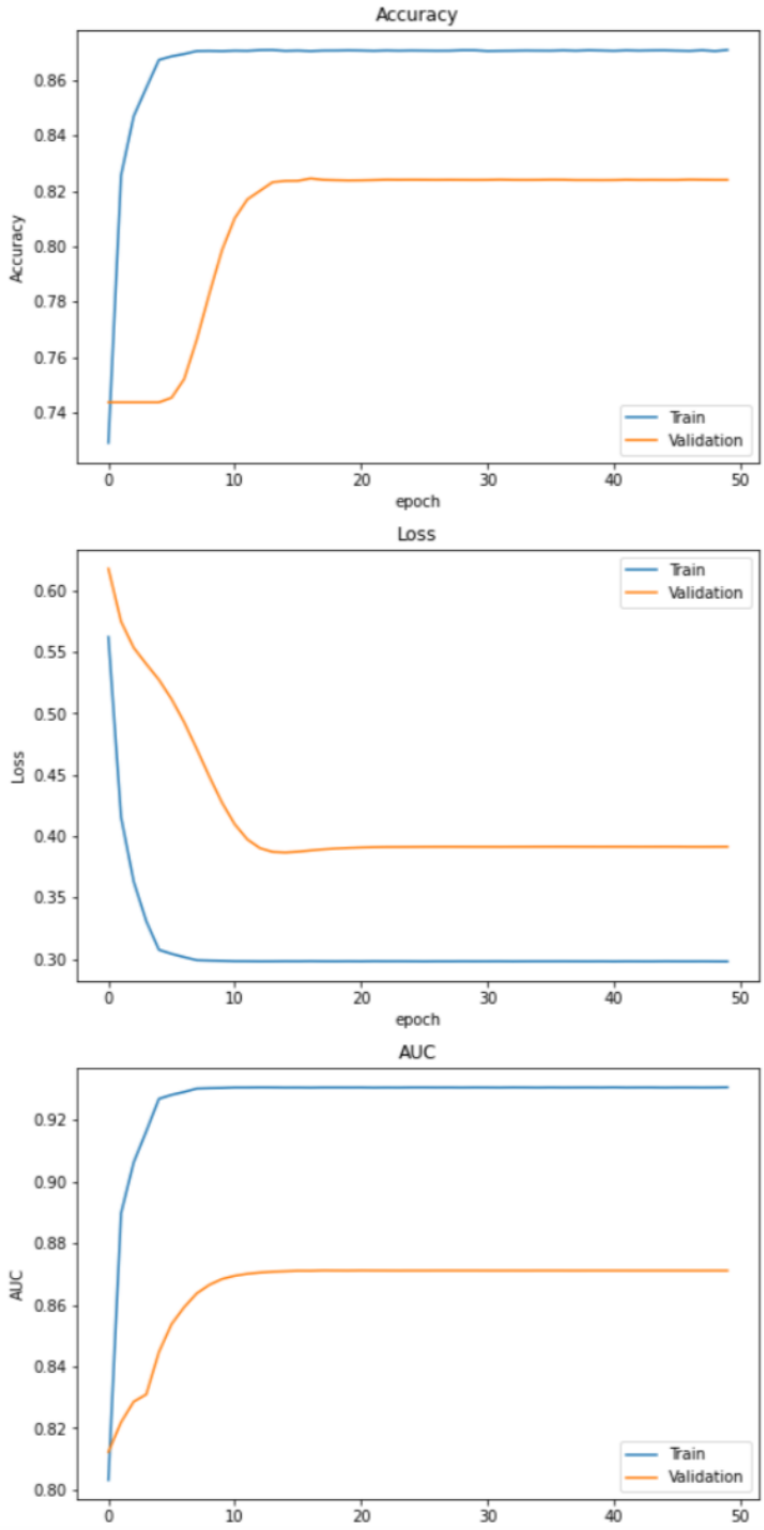

Figure 6.8: Accuracy, Loss and AUC charts for dimension of 5

As can be seen from the table 6.9, different units are used in the hidden layer with the embedding layer, associated train and test AUC scores are given. The best results are acquired by two hidden layers with 64 and 128 neurons, respectively, because it gives the highest AUC on the test data. However, as an alternative, 128 units of hidden layer give good training and test AUC scores without making the model more complex thus, it would be worth investigating the two alternative architecture on the entire data set because sampling is used to perform the neural network experiments until now as it is explained in section 5.2.4. The next coming experiments are performed on the entire data sets in order to have more reliable results.

Architecture	**Data Set**	**AUC**	**Loss**	**Precision**	**Recall**
128 units	train	0.920924	0.305686	0.883677	0.92836
128 units	dev	0.90020	0.338335	0.874471	0.921003
128 units	test	0.890117	0.34535	0.884236	0.912480
64-128 units	train	0.921296	0.304981	0.88412	0.928271
64-128 units	dev	0.899875	0.338452	0.874944	0.919941
64-128 units	test	0.889551	0.346108	0.884898	0.910770

Table 6.10: Evaluation metrics on the entire data set

The tab 6.10 shows the AUC, loss, precision, and recall values of the entire data set for 128 units and 64-128 units. The difference between the last two experiments is the data. As a conclusion of the experiment, the architecture of 1 hidden layer with 128 units and two hidden layers with 64 and 128 units give really close and good results.

After narrowing down the number of architecture possibilities to two, the next step is the regularization which is used to prevent over-fitting. Drop-out method is used proposed by [Srivastava 2014] which an approach to overcome the over-fitting problem. It randomly drops neurons in hidden layers during the training process. This method restrains the model from fitting perfectly on the training data. In the first experiment, drop-out of 0.25 is applied after each layer, and the results of two different architectures are given table 6.11.

	Train Set	Dev Set	Test Set
128 units	0.916178	0.901797	0.892356
64-128 units	0.915301	0.900937	0.891775

Table 6.11: Drop-out of 0.25

In the next experiment, drop-out of 0.3 is applied after each layer and the results of the two different architecture are given table 6.12.

	Train Set	Dev Set	Test Set
128 units	0.914892	0.900694	0.891617
64-128 units	0.913974	0.899825	0.890741

Table 6.12: Drop-out of 0.30

As can be seen clearly from the results, the results are so close to each other. It is important to check the loss, AUC, and accuracy charts over epochs in order to make the conclusion therefore two different data sets are chosen in order to show the results over the epochs.

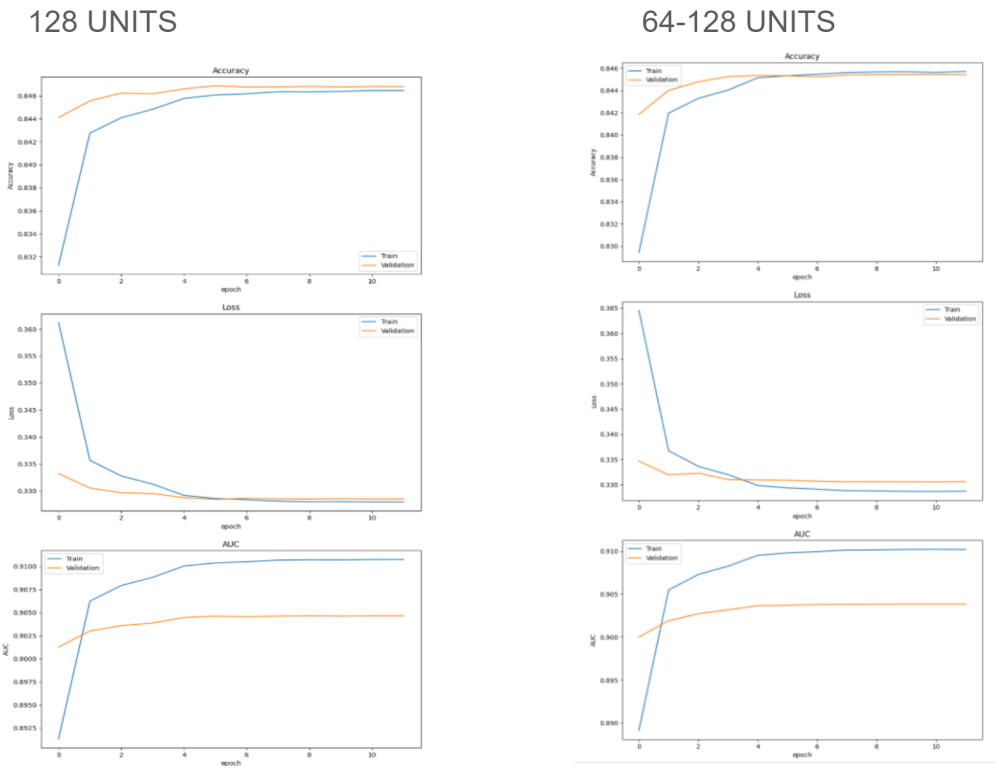

Figure 6.9: Accuracy, Loss and AUC charts with the probability of 0.25 drop-out

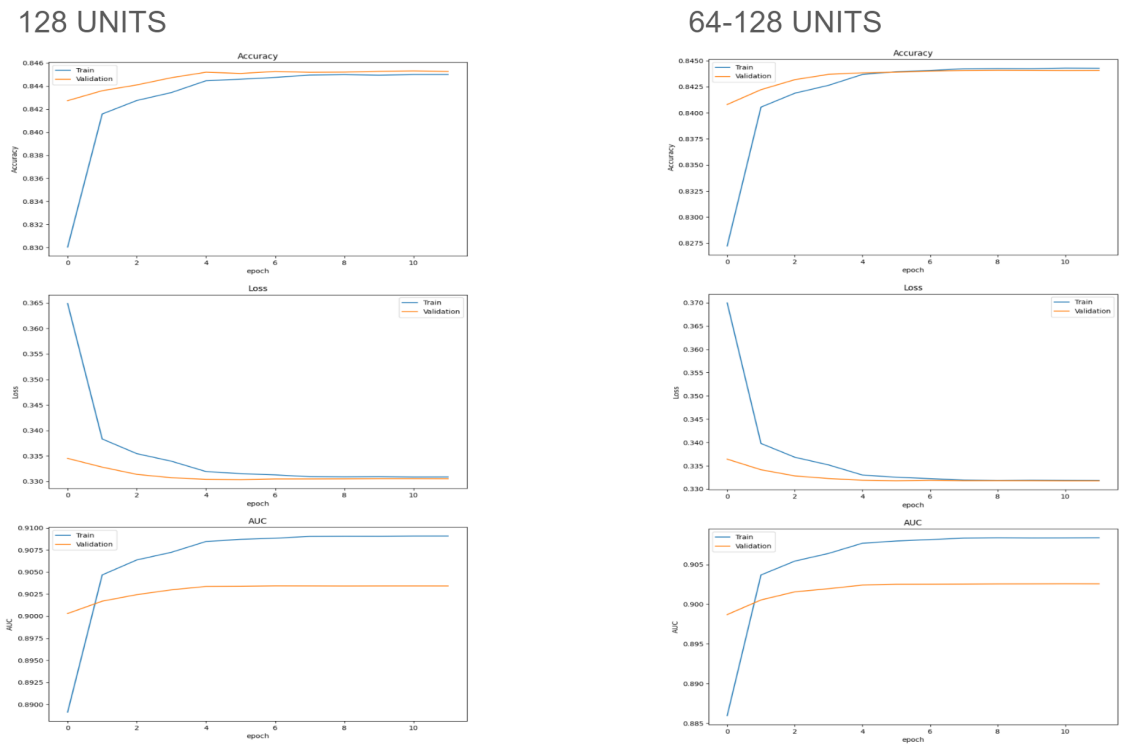

Figure 6.10: Accuracy, Loss and AUC charts with the probability of 0.3 drop-out

As we can see clearly from the results, drop-out method with the probability of 0.3 gives smoother charts over epochs compared to the probability of 0.25 drop-out, and over-fitting no longer occurs for both architectures. To sum up, amongst the possible options, one hidden layer with units of 128 gives the best performance on dev and test set even though the results are too close; therefore, it is chosen to continue our experiments with 128 neurons. Moreover, the next experiment demonstrates the impact of different drop-out values on 128 neurons network.

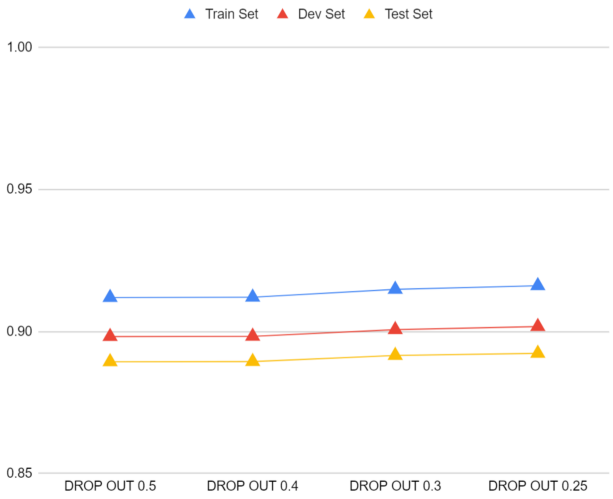

Figure 6.11: The impact of different drop-out values on AUC score

It can be seen from the figure 6.11, different drop-out probabilities applied to our network of 128 units, as we increase the regularization, AUC scores decrease. Therefore drop-out with 0.3 probability gives good trade-off between having smoother charts over epochs and good training and test AUC. Overall conclusion, amongst all the possible options, one hidden layer with units of 128 with 0.3 drop-out and dimension of 5 embeddings gives the best performance on dev and test set. Thus the neural network model is finalized to compare the results with the XGBoost model in the next experiment.

6.2.5 Comparison of Extreme Gradient Boosting and Neural Network Model

In this section, we analyze the comparison between the Neural Network with the Extreme Gradient Boosting model. There are 21 features used to train the models on two different data sets in order to compare the results. The results of the Extreme Gradient Boosting and Neural Networks models are given in figure 6.13.

DATA SET	MODEL	DATA SET TYPE	LOSS	PRECISION	RECALL	AUC
1	Neural Network	train	0.320920	0.878158	0.921203	0.915090
		test	0.345202	0.870642	0.919584	0.894979
	XGBoost	train	0.300656	0.886257	0.929251	0.926710
		test	0.340871	0.872662	0.922020	0.894817
2	Neural Network	train	0.318384	0.878384	0.926440	0.914693
		test	0.345495	0.882674	0.913632	0.888250
	XGBoost	train	0.301633	0.885334	0.933864	0.922418
		test	0.318525	0.88296	0.929344	0.901462

Table 6.13: The results of the XGBoost and Neural Network Model

As can be seen for both data sets, all the metrics of the XGBoost model is better than the Neural Network model, except the AUC score for the first data set. Therefore, it also crucial to see the averaged AUC results of both models. Figure 6.12 demonstrates

Figure 6.12: Averaged results of the XGBoost and Neural Network Model

the averaged results of the AUC score for both models. As can be seen, even though XGBoost performs 0.96 % better on train AUC and 0.67% better on test AUC, both models perform very closely. To sum up, XGBoost model is improved with additional new dimensions, early stopping mechanism and fine tuning. Therefore, the improved XGBoost model is put in production. Since the used algorithm is the same as well as the architecture explained in section 5.2.1, it is straight-forward to put the new model in production.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This chapter provides the overall conclusion of the approach, the main findings, the limitations of the study, and the future work. Section 7.1.1 provides the summary of the project and the obtained results. 7.1.2 highlights the restrictions of the study and proposes solutions to them. Lastly, 7.1.3 gives ideas that can be implemented in order to improve the results in the future.

7.1.1 Summary of Results

In summary, this paper argues different state-of-the-art approaches in order to predict the delivery rate/success rate of video ads. An iterative approach is followed throughout the study. At each step, some changes are applied, and their impact is analyzed on the corresponding model. First of all, different tree-based models are compared, and the best performing algorithm is chosen for further improvements. As a result, Extreme Gradient Boosting gives clearly better results than Decision Tree, Ada Boost, and Random Forest. Besides performing well, Extreme Gradient Boosting is scalable and parallelizable. Although it is harder to tune it compared to the other tree-based classifiers due to the high number of parameters, it allows us to have more control over the model.

During the experiments, it is realized that there is a high variation in data, and the distribution changes over time. Therefore, the model should be adaptive and flexible, depending on the data set of any given time in order to perform well. Consequently, the parameters of the model should not be the same; thus, early stopping strategy is applied in order to adapt the changes over time. Furthermore, due to the data variance in time, the model should be trained with the most current data as frequently as possible in order to get the greatest performance.

Moreover, Extreme Gradient Boosting and Neural Network models are trained to compare their results and analyze their advantages and disadvantages. In conclusion, even though their results are very similar to each other, Extreme Gradient Boosting performs slightly better than the Neural Network model. However, it is difficult to reach a conclusion since there is a slight difference between the two models. There are a few advantages of using neural networks over XGBoost as follows. As a reminder, TensorFlow is used to implement the Neural Network model in the Implementation section. TensorFlow provides the flexibility of building a better pipeline, the stable solution for serving in production, and the possibility for transfer learning for future projects. Another advantage is the ability to apply continuous learning, which is progressive and adaptive as new data is acquired. These advantages make the neural network more suitable. Therefore there is some future work that needs to be done.

7.1.2 Critical Discussion

This section addresses the limitations of the study. First of all, the average bid density is low in auctions. It means that we choose the best ad amongst only a few options. The bid density is the number of bidders in an auction. Our algorithm makes an impact if there are at least two bidders in an auction so we can optimize the delivery rate and the expected revenue. Otherwise, it is trivial to use the methodology. Therefore, the first requirement is to have a high number of auctions with more than two bidders in order to make a significant impact in revenue on a global scale. In this case, we should attract more advertisers to bid.

The second requirement is to have at least one good creative that leads to a high delivery rate in every auction. Unfortunately, it does not always happen in the ecosystem. If all bidders lead to a low delivery rate, it is not possible to increase the delivery rate and propose a better ad that is likely to be delivered. In this case, the required actions are the detection of bad creatives, the investigation of what causes the low delivery rate, and resolving the problem. This part is related to anomaly detection, which is discussed in the Future Work section.

7.1.3 Future Work

There are good results in predicting the success rate of video ads. Nevertheless, various strategies can bring significant improvements. To begin with, the change of the weighting strategy of the XGBoost model might improve the model. Rather than having the number of impressions of a given combination in each row, the revenue of a given combination can be used as weights. By doing so, the model focuses on where the impact is high.

In future work, investigating VAST data might improve the model. As described before, VAST data includes metadata specific to a media player such as the length, the type, and the display method of a video ad. These additional features need to be explored as well as the impact on the model.

More importantly, continuous learning in the Neural Network model might have a great impact on the results. By doing so, the oblivion problem of previously seen data is overcome because the current models are trained only with the last 24 hours of data from scratch. This restricts the ability of continuous learning. Lastly, anomaly detection is an interesting topic for future work, and it should contain the following steps below.

- detecting the high revenue uplift
- finding the network responsible for the anomaly
- getting the responsible dimensions in the data set
- identifying the specific values that cause to lower delivery rate

Appendix A

Extra Experiments

This chapter is the extension of the Experiment section. First experiment provides the model calibration and possible ways of improving the calibration of the model. Second experiment demonstrates the impact of varying the threshold on AUC and F1-score for the XGBoost model.

A.1 Calibration

The goal of this section is to check the calibration of the model and apply some calibration methods to analyze the effects on the model. We predict the probabilities in the delivery rate optimization project therefore calibration of the model is crucial. Calibration is a comparison of the actual output and the expected output produced by our model. The following graph shows the reliability diagram when the model is calibrated perfectly A.1.

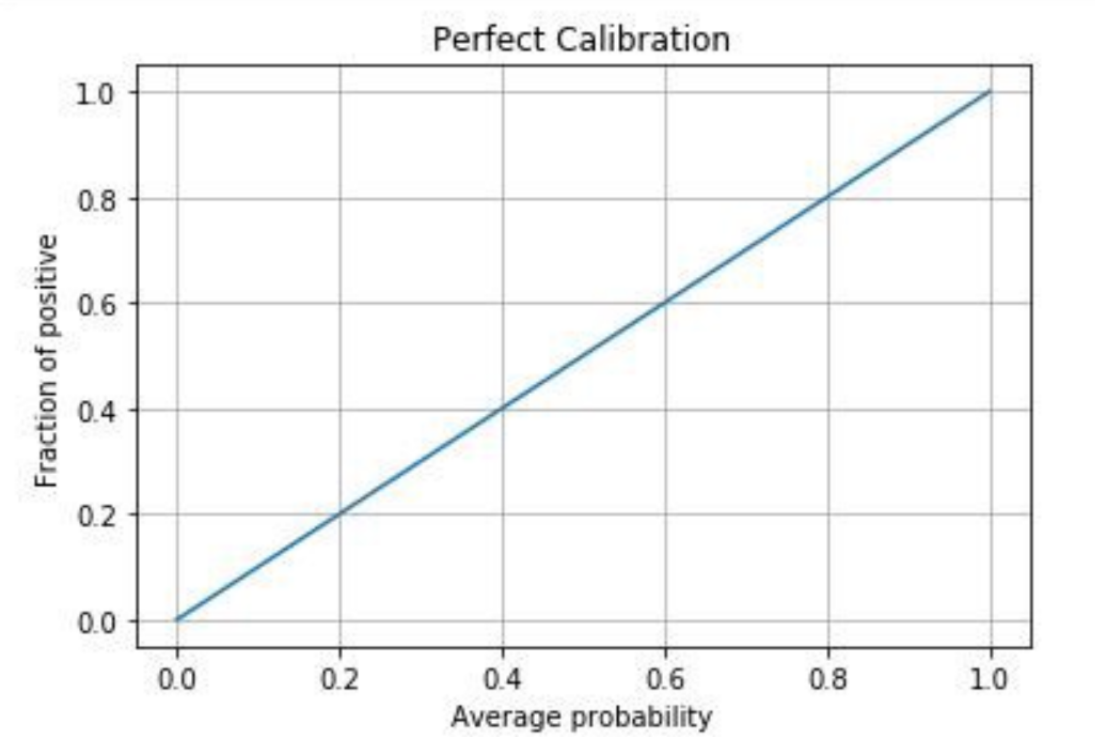

Figure A.1: Perfectly Calibrated Model

In order to check the calibration of a model, the reliability diagram of the actual probability on the y-axis and probability predicted by the model on the test data set on the x-axis is plotted in figure A.2.

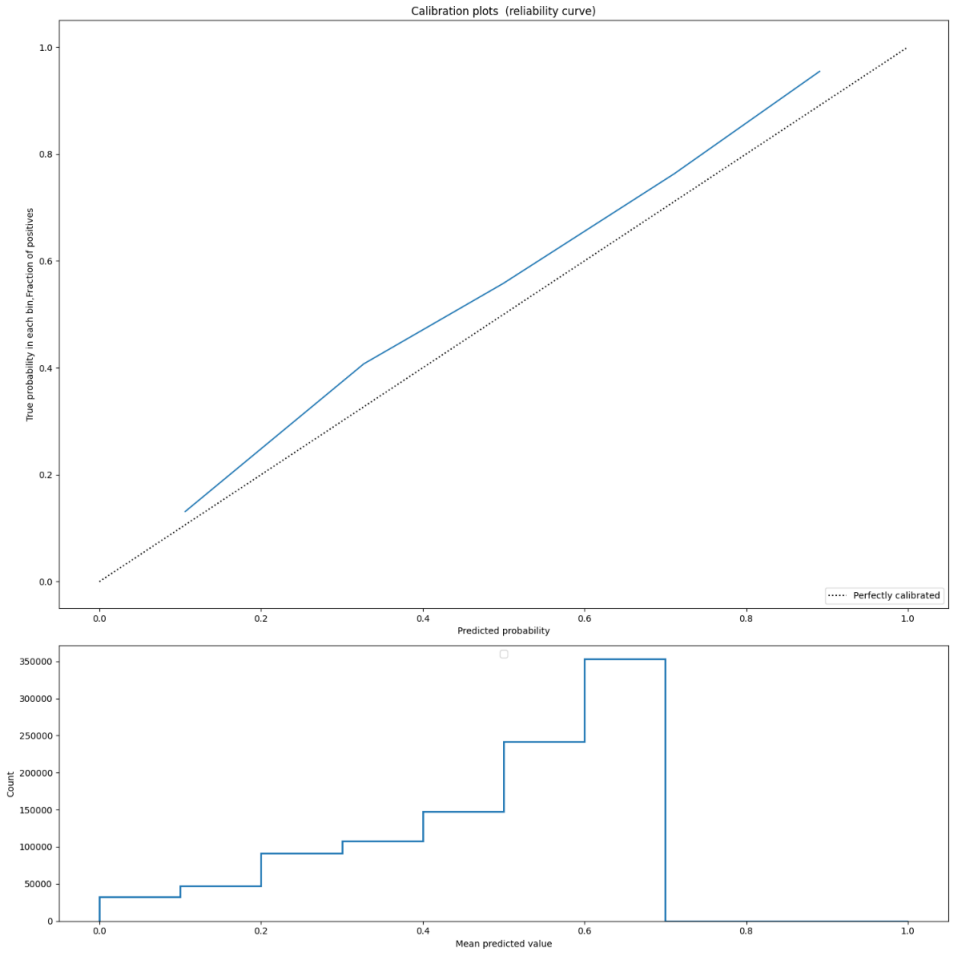

Figure A.2: Model Calibration

There are two methods of diagnosing the calibration of a model.These methods are applied one by one, as a result their reliability diagram are shown in figure A.3. It is crucial to analyze the graphs in order to see whether it diagnoses the model calibration or not. First, we applied the sigmoid method in order to diagnose the calibration on the left side of the figure. On the right side, isotonic method is applied in order to diagnose the calibration. As a conclusion, sigmoid helps to calibrate the model in a better way whereas isotonic method performs poorly on the reliability diagram compared to the current model and the sigmoid method.

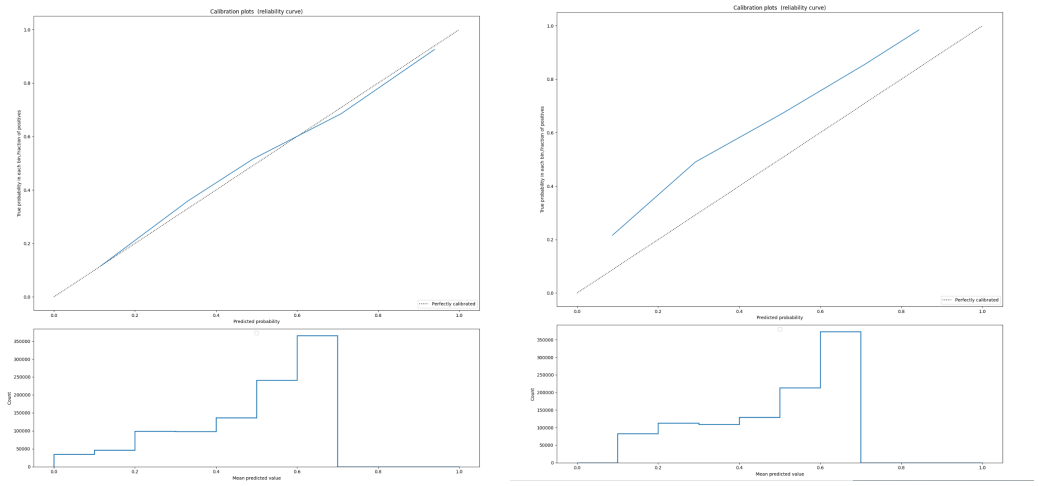

Figure A.3: Reliability Diagram after applying Sigmoid and Isotonic Method

A.2 Threshold Variation

The goal of this experiment is to show the impact of varying the threshold on AUC and F1-score for the XGBoost model.

It is important to remind that the model that we would like to have returns only probabilities, not the class labels. However, this experiment addresses the effect of the variation of different thresholds in case of being interested in having a model that returns 0/1 classes.

The chart in figure A.4 demonstrates the effect of varying the threshold on F1-score. As can be seen, the F1 score is maximized when the threshold is equal to 0.4 and 0.5. In order to get the best F1 score, we should analyze the F1 score for threshold of 0.4 and 0.5 closely A.5 and A.6.

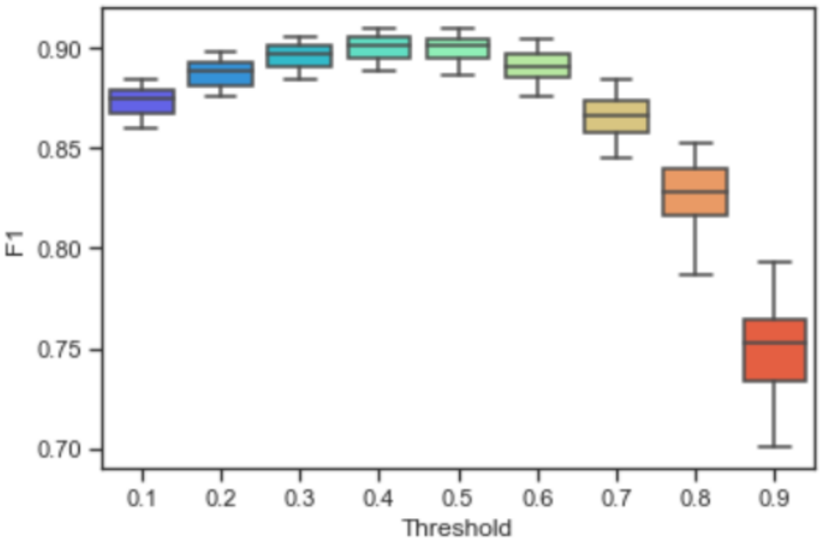

Figure A.4: Effect of varying the threshold on F1 score

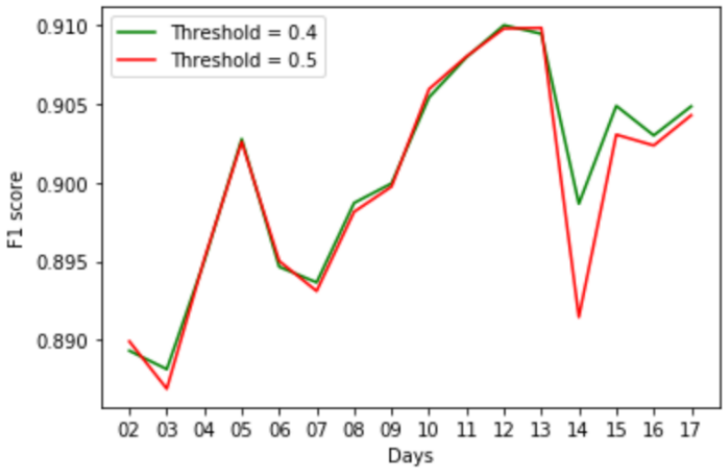

Figure A.5: F1 score on 17 days of data for threshold of 0.4 and 0.5

```
The mean of threshold 0.4 is 0.90040222118125
The std of threshold 0.4 is 0.00658752903151269

The mean of threshold 0.5 is 0.8997063851375
The std of threshold 0.5 is 0.00696074750789785
```

Figure A.6: The mean and variance of F1 score on 17 days of data

After making some analyses which can be seen above, it was observed that the mean of F1 score for the threshold of 0.4 is greater than the mean of F1 score for the threshold of 0.5. Also, the standard deviation F1 score for the threshold of 0.4 is smaller than the standard deviation F1 score for the threshold of 0.5 which gives us a better result and better consistency across different days. Therefore this experiment shows that the threshold should be assigned to 0.4 in case we aim to maximize the F1 score and the model returns only class labels as 0/1 instead of the probabilities.
The chart in figure A.7 demonstrates the effect of varying the threshold on AUC score. As can be seen, AUC score is maximized when the threshold is equal to 0.7 and 0.8. In order to get the best AUC score, we should analyze the AUC score for the threshold 0.7 and 0.8 closely A.8 and A.9.

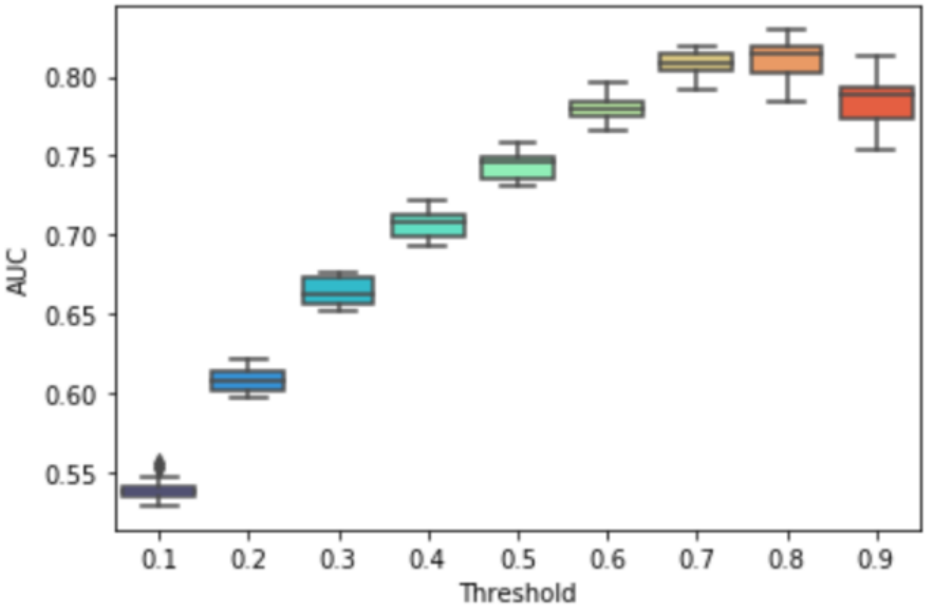

Figure A.7: Effect of varying the threshold on AUC score

Figure A.8: F1 score on 17 days of data for threshold of 0.4 and 0.5

```
The mean of threshold 0.7 is 0.8078434503875
The std of threshold 0.7 is 0.008589543216448236

The mean of threshold 0.8 is 0.81088954605625
The std of threshold 0.8 is 0.011990748511217305
```

Figure A.9: The mean and variance of AUC score on 17 days of data

After making some analysis as it can be seen above, it was observed that the mean AUC score for the threshold of 0.8 is slightly greater than the mean of AUC score for the threshold of 0.7. However, the standard deviation AUC score for the threshold of 0.7 is smaller than the standard deviation AUC score for the threshold of 0.8. Since the difference between the mean values is very small, it is better to choose a threshold of 0.7 which gives us steady results across different days. Therefore this experiment shows that the threshold should be assigned to 0.7 in case we aim to maximize the AUC score and the model returns the class labels as 0/1 instead of the probabilities. It can be concluded that varying the threshold has an opposite effect on different metrics therefore choosing a proper metric has substantial importance on a model.

Appendix B

Requirements

```
APScheduler==3.6.3
attrs==19.3.0
backcall==0.1.0
bleach==3.1.1
cachetools==4.0.0
catboost==0.21
certifi==2019.11.28
chardet==3.0.4
Click==7.0
cloudpickle==1.3.0
colorama==0.4.3
cycler==0.10.0
decorator==4.4.1
defusedxml==0.6.0
entrypoints==0.3
fire==0.2.1
Flask==1.1.1
future==0.18.2
google-api-core==1.16.0
google-auth==1.11.2
google-cloud==0.34.0
google-cloud-core==1.3.0
google-cloud-storage==1.26.0
google-resumable-media==0.5.0
googleapis-common-protos==1.51.0
graphviz==0.13.2
hyperopt==0.2.3
idna==2.8
importlib-metadata==1.5.0
ipykernel==5.1.4
ipython==7.12.0
ipython-genutils==0.2.0
ipywidgets==7.5.1
itsdangerous==1.1.0
jedi==0.16.0
Jinja2==2.10.3
joblib==0.14.1
```

```
jsonschema==3.2.0
jupyter==1.0.0
jupyter-client==6.0.0
jupyter-console==6.1.0
jupyter-core==4.6.3
keras2onnx==1.6.0
kiwisolver==1.1.0
MarkupSafe==1.1.1
matplotlib==3.1.3
mistune==0.8.4
nbconvert==5.6.1
nbformat==5.0.4
networkx==2.2
notebook==6.0.3
numpy==1.18.1
onnx==1.6.0
onnxconverter-common==1.6.0
onnxmltools==1.6.0
pandas==0.25.3
pandocfilters==1.4.2
parso==0.6.1
pickleshare==0.7.5
plotly==4.5.1
prometheus-client==0.7.1
prompt-toolkit==3.0.3
protobuf==3.11.2
pyaml==19.12.0
pyasn1==0.4.8
pyasn1-modules==0.2.8
Pygments==2.5.2
pyparsing==2.4.6
pyrsistent==0.15.7
python-dateutil==2.8.1
pytz==2019.3
pywin32==227
pywinpty==0.5.7
PyYAML==5.3
pyzmq==18.1.1
qtconsole==4.6.0
requests==2.22.0
retrying==1.3.3
rsa==4.0
scikit-learn==0.22.1
scikit-optimize==0.7.4
scipy==1.4.1
```

```
seaborn==0.10.0
Send2Trash==1.5.0
six==1.13.0
skl2onnx==1.6.0
sklearn==0.0
termcolor==1.1.0
terminado==0.8.3
testpath==0.4.4
tornado==6.0.3
tqdm==4.43.0
traitlets==4.3.3
typing-extensions==3.7.4.1
tzlocal==2.0.0
urllib3==1.25.7
waitress==1.4.2
wcwidth==0.1.8
webencodings==0.5.1
Werkzeug==0.16.0
widgetsnbextension==3.5.1
xgboost==0.90
zipp==3.0.0
```

Bibliography

[Aldrich 1995] John Aldrichet al. Correlations genuine and spurious in Pearson and Yule. Statistical science, vol. 10, no. 4, pages 364–376, 1995. (Cited on page 28.)

[AllBusiness] AllBusiness. Web Advertising and CPM. [Online; accessed 4-February-2020]. (Cited on page 7.)

[Banfield 2006] Robert E Banfield, Lawrence O Hall, Kevin W Bowyer and W Philip Kegelmeyer. A comparison of decision tree ensemble creation techniques. IEEE transactions on pattern analysis and machine intelligence, vol. 29, no. 1, pages 173–180, 2006. (Cited on page 12.)

[Bauer 1999] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. Machine learning, vol. 36, no. 1-2, pages 105–139, 1999. (Cited on page 13.)

[bioturing] bioturing. Box Plot. [Online; accessed 1-June-2020]. (Cited on page 27.)

[Breiman 1984] Leo Breiman, Jerome Friedman, Charles J Stone and Richard A Olshen. Classification and regression trees. CRC press, 1984. (Cited on page 13.)

[Breiman 1996] Leo Breiman. Bagging predictors. Machine learning, vol. 24, no. 2, pages 123–140, 1996. (Cited on pages 12 and 14.)

[Breiman 2001] Leo Breiman. Random forests. Machine learning, vol. 45, no. 1, pages 5–32, 2001. (Cited on page 12.)

[Cakmak 2019] Tülin Cakmak, Ahmet Tekin, Cagla Senel, Tugba Coban, Zeynep Eda Uran and Cemal Okan Sakar. Accurate Prediction of Advertisement Clicks based on Impression and Click-Through Rate using Extreme Gradient Boosting. In ICPRAM, pages 621–629, 2019. (Cited on page 11.)

[Chen 2016] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pages 785–794, 2016. (Cited on pages 15, 16 and 34.)

[Chiu 2002] Chaochang Chiu. A case-based customer classification approach for direct marketing. Expert Systems with Applications, vol. 22, no. 2, pages 163–168, 2002. (Cited on page 13.)

[ClearCode a] ClearCode. Demand Side. [Online; accessed 21-February-2020]. (Cited on page 7.)

[ClearCode b] ClearCode. DSP. [Online; accessed 4-June-2020]. (Cited on pages 7 and 8.)

[ClearCode c] ClearCode. Header Bidding. [Online; accessed 15-May-2020]. (Cited on page 5.)

[datacamp] datacamp. AdaBoost. [Online; accessed 6-March-2020]. (Cited on pages 12 and 13.)

[Dietterich 2000] Thomas G Dietterich. Ensemble methods in machine learning. In International workshop on multiple classifier systems, pages 1–15. Springer, 2000. (Cited on page 13.)

[D'Agaro 2018] Edo D'Agaro. Artificial intelligence used in genome analysis studies. The EuroBiotech Journal, vol. 2, no. 2, pages 78–88, 2018. (Cited on page 35.)

[Epom] Epom. DSP and Ad Server. [Online; accessed 15-May-2020]. (Cited on page 7.)

[Fawcett 2006] Tom Fawcett. An introduction to ROC analysis. Pattern recognition letters, vol. 27, no. 8, pages 861–874, 2006. (Cited on page 49.)

[Freund 1996] Yoav Freund and Robert E Schapire. Game theory, on-line prediction and boosting. In Proceedings of the ninth annual conference on Computational learning theory, pages 325–332, 1996. (Cited on page 14.)

[glassbox] glassbox. Confusion Matrix. [Online; accessed 1-June-2020]. (Cited on page 49.)

[groundai] groundai. Multi Layer Perceptron. [Online; accessed 4-August-2020]. (Cited on page 18.)

[Grove 1998] Adam J Grove and Dale Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In AAAI/IAAI, pages 692–699, 1998. (Cited on page 55.)

[Guo 2016] Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables. arXiv preprint arXiv:1604.06737, 2016. (Cited on page 37.)

[IAB a] IAB. IAB Internet Advertising Glossary. [Online; accessed 20-February-2020]. (Cited on page 6.)

[IAB b] IAB. IAB Internet Advertising Revenue Report. [Online; accessed 14-May-2020]. (Cited on page 1.)

[IAB c] IAB. VAST Template. [Online; accessed 23-February-2020]. (Cited on page 8.)

[jtsulliv] jtsulliv. Perceptron. [Online; accessed 3-August-2020]. (Cited on page 17.)

[LeBlanc 1996] Michael LeBlanc and Robert Tibshirani. Combining estimates in regression and classification. Journal of the American Statistical Association, vol. 91, no. 436, pages 1641–1650, 1996. (Cited on page 12.)

[McMahan 2013] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovinet al. Ad click prediction: a view from the trenches. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 1222–1230, 2013. (Cited on page 11.)

[Osisanwo 2017] FY Osisanwo, JET Akinsola, O Awodele, JO Hinmikaiye, O Olakanmi and J Akinjobi. Supervised machine learning algorithms: classification and comparison. International Journal of Computer Trends and Technology (IJCTT), vol. 48, no. 3, pages 128–138, 2017. (Cited on page 31.)

[Rodriguez 2006] Juan José Rodriguez, Ludmila I Kuncheva and Carlos J Alonso. Rotation forest: A new classifier ensemble method. IEEE transactions on pattern analysis and machine intelligence, vol. 28, no. 10, pages 1619–1630, 2006. (Cited on page 12.)

[Rosenblatt 1962] Frank Rosenblatt. Principles of neurodynamics: Perceptions and the theory of brain mechanisms. 1962. (Cited on page 16.)

[Rumelhart 1986] David E Rumelhart, Geoffrey E Hinton and Ronald J Williams. Learning representations by back-propagating errors. nature, vol. 323, no. 6088, pages 533–536, 1986. (Cited on page 18.)

[Smart] Smart. Optimizing network traffic in a massively parallel real-time bidding environment. [Online; accessed 4-June-2020]. (Cited on page 3.)

[Srivastava 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, vol. 15, no. 1, pages 1929–1958, 2014. (Cited on page 65.)

[Statista] Statista. Video Advertising. [Online; accessed 14-May-2020]. (Cited on page 2.)

[sumanthrb] sumanthrb. Single Layer Perceptron. [Online; accessed 6-August-2020]. (Cited on page 17.)

[towardsdatascience a] towardsdatascience. Ensemble Methods. [Online; accessed 1-August-2020]. (Cited on page 14.)

[TowardsDataScience b] TowardsDataScience. Medium. [Online; accessed 30-May-2020]. (Cited on page 36.)

[Wang 2015] Chong Wang, Achir Kalra, Cristian Borcea and Yi Chen. Viewability prediction for online display ads. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pages 413–422, 2015. (Cited on page 11.)

[Webb 2005] Geoffrey I Webb and Paul Conilione. Estimating bias and variance from data. Pre-publication manuscript (http://www. csse. monash. edu/webb/-Files/WebbConilione06. pdf), 2005. (Cited on page 13.)

[Wolpert 1992] David H Wolpert. Stacked generalization. Neural networks, vol. 5, no. 2, pages 241–259, 1992. (Cited on page 12.)

[Yan 2014] Ling Yan, Wu-Jun Li, Gui-Rong Xue and Dingyi Han. Coupled group lasso for web-scale ctr prediction in display advertising. In International Conference on Machine Learning, pages 802–810, 2014. (Cited on page 11.)

[Zhai 2016] Shuangfei Zhai, Keng-hao Chang, Ruofei Zhang and Zhongfei Mark Zhang. Deepintent: Learning attentions for online advertising with recurrent neural networks. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pages 1295–1304, 2016. (Cited on page 11.)

[Zhang 2005] Tong Zhang, Bin Yuet al. Boosting with early stopping: Convergence and consistency. The Annals of Statistics, vol. 33, no. 4, pages 1538–1579, 2005. (Cited on page 55.)

[Zhang 2015] Weinan Zhang, Ye Pan, Tianxiong Zhou and Jun Wang. An empirical study on display ad impression viewability measurements. arXiv preprint arXiv:1505.05788, 2015. (Cited on page 11.)

[Zhu 2002] H Zhu, PA Beling and GA Overstreet. A Bayesian framework for the combination of classifier outputs. Journal of the Operational Research Society, vol. 53, no. 7, pages 719–727, 2002. (Cited on page 12.)