Customer Lifetime Value Prediction for an Online Marketplace for Local Services Using Machine Learning

by

Mohammed Saif Ragab Kazamel

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Data Science

December 9, 2022

Customer Lifetime Value Prediction for an Online Marketplace for Local Services Using Machine Learning

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Mohammed Saif Ragab Kazamel

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Assoc. Prof. Mehmet Gönen (Advisor)

Prof. Fikri Karaesmen

Assoc. Prof. Mustafa Gökçe Baydoğan

Date: December 9, 2022

To those who deserved more than they got. To the many Einsteins who have lived and died in cotton fields and sweatshops.

ABSTRACT

Customer Lifetime Value Prediction for an Online Marketplace for Local Services Using Machine Learning

Mohammed Saif Ragab Kazamel

Master of Science in Data Science

December 9, 2022

In this age of direct and digital marketing, customer-oriented metrics, especially customer lifetime value (LTV), have become one of the most important business metrics for customer relationship management (CRM) and marketing strategies in general. The customer LTV, a quantification of the profit or revenue that a customer will bring to the firm over their entire relationship period, differs wildly for different customers, which makes predictions over the entire pool of the firm's customers quite challenging. The rapidly expanding field of machine learning is uniquely suitable for solving this problem. In this study, we develop a model for predicting customer LTV for a local services marketplace, namely Armut AŞ. The predicted LTV is then used for the optimization of the company's digital ads. The model was developed through experimenting with several machine learning algorithms including linear regression, decision tree, extreme gradient boosting (XGBoost), and artificial neural networks (ANNs), as well as experimenting with a combination of encoding techniques for the categorical features in the data set. The models were evaluated using 5-fold time-series cross-validation and yielded comparable results with normalized root mean square error (NRMSE) ranging from 1.740 to 1.776. The chosen model was XGBoost, and it was further optimized by segmenting the data based on its business model as well as expanding the training set time window to 12 months yielding the final model with an NRMSE of 1.725. This model was deployed and is currently operating at Armut AŞ.

ÖZETÇE

Yerel Hizmetler Pazar Yeri İçin Yapay Öğrenme Kullanarak Müşteri Yaşam Boyu Değeri Kestirilmesi

Mohammed Saif Ragab Kazamel

Veri Bilimleri, Yüksek Lisans

9 Aralık 2022

İçinde bulunduğumuz doğrudan ve dijital pazarlama çağında, müşteri odaklı metrikler, özellikle müşteri yaşam boyu değeri (YBD), müşteri ilişkileri yönetimi ve genel olarak pazarlama stratejileri için en önemli iş metriklerinden biri haline gelmiştir. Bir müşterinin tüm ilişki süresi boyunca firmaya getireceği kâr veya gelirin bir ölçüsü olan müşteri YBD'si, her bir müşteri için büyük farklılık gösterir ve bu da firmanın tüm müşteri havuzu üzerindeki YBD tahminlerini oldukça zorlaştırır. Hızla genişleyen yapay öğrenme alanı, bu sorunu çözmek için çok uygun bir araçtır. Bu çalışmada, yerel bir hizmet pazarı olan Armut AŞ'nin dijital reklamlarının optimizasyonunda kullanılan müşteri YBD'sini tahmin edecek bir model geliştirdik. Model, doğrusal regresyon, karar ağacı, aşırı gradyan artırma (XGBoost) ve yapay sinir ağları (YSA'lar) dahil olmak üzere çeşitli yapay öğrenme algoritmalarıyla denemeler yapılarak ve veri kümesindeki kategorik özellikler için kodlama tekniklerinin bir kombinasyonu ile deneyler yapılarak geliştirilmiştir. Modeller, 5 katlı zaman serisi çapraz doğrulama kullanılarak değerlendirilmiş olup, 1,740 ila 1,776 arasında değişen normalleştirilmiş ortalama karekök hatası (NOKH) ile karşılaştırılabilir sonuçlar vermiştir. Bu modeller arasından kullanılmak üzere XGBoost seçilmiştir. Bu model, verileri iş modeline göre bölümlere ayırarak ve eğitim seti zaman penceresini 12 aya genişleterek daha da optimize edilmiştir. Bu optimizasyon sonucunda 1,725 NOKH skoruna ulaşılmıştır. Elde edilen model devreye alınmıştır ve şu anda Armut AŞ'de çalışmaktadır.

ACKNOWLEDGMENTS

I would like to thank my advisor Assoc. Prof. Mehmet Gönen for his support, guidance, and understanding throughout my studies. It has been a tough, yet fruitful and enjoyable learning experience, and his support made it much more pleasant. Furthermore, I am very thankful to my thesis committee members Prof. Fikri Karaesmen and Assoc. Prof. Mustafa Gökçe Baydoğan for their valuable time and precious comments. I would also like to acknowledge Armut AŞ for providing the dataset used in this study. I wish to also thank all my friends especially Omar Gamal Abdelaty for their support and encouragement during my studies. My most sincere gratitude will always go to my family for their unbounded love and support. Last but not least, I am most grateful to my wife Şeyma Genç Kazamel who has been very supportive, caring, and encouraging throughout this journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AC	Acquisition Cost
ANN	Artificial Neural Network
ANOVA	Analysis of Variance
API	Application Programming Interface
AUC	Area Under the Curve
AUROC	Area Under the ROC curve
AWS	Amazon Web Services
BM	Business Model
CE	Customer Equity
CLV/CLTV	Customer Lifetime Value
CNBD	Condensed Negative Binomial Distribution
CNN	Convolutional Neural Network
CPA	Cost per Action
CPC	Cost per Click
CRM	Customer Relationship Management
DT	Decision Tree
ECS	Elastic Container Service
GBM	Gradient Boosting Machines
GCN	Graph Convolutional Network
GG	Gamma-Gamma model
GNN	Graph Neural Network
KDD	Knowledge Discovery and Data mining
KNN	K-Nearest Neighbors
LR	Linear Regression
LTV	Lifetime Value
MAE	Mean Absolute Error
MBG	Modified Beta-Geometric
MK-SVR	Multi Kernel Support Vector Regression
ML	Machine Learning
MLP	Multi-Layer Perceptron
NBD	Negative Binomial Distribution

NN	Neural Network
NRMSE	Normalized Root Mean Square Error
PFI	Permutation Feature Importance
REC	Retention Expansion Cost
ReLU	Rectified Linear Unit
RF	Random Forest
RFM	Recency, Frequency, Monetary value
RMSE	Root Mean Square Error
ROAS	Return on Ad Spend
ROC	Receiver Operating Characteristic
ROI	Return on Investment
SE	Squared Error
SEO	Search Engine Optimization
SHAP	Shapley Additive Explanations
SMOTE	Synthetic Minority Oversampling Technique
SQL	Structured Query Language
SSE	Sum of Squared Errors
SVM	Support Vector Machine
SVR	Support Vector Regression
tCPA	Target Cost per Action
tROAS	Target Return on Ad Spend
XGBoost	Extreme Gradient Boosting

Chapter 1:
INTRODUCTION

It might be surprising that contrary to popular perception, not all customers of a business should be valued. Indeed, that is because not all customers are equal. As customers are increasingly considered as the most important assets of any business, it thus follows that some of these assets could be negative assets, or at the very least less valuable than others. The concept of individual customer value is central to the increasingly important business strategies of customer relationship management (CRM). The central idea behind CRM is to adjust the allocated resources to customers based on their profitability to the firm, to attract high-value customers in customer acquisition, and to "fire" or limit some services to those customers who are dragging down the profitability of the firm. The idea of customer value is not only applied to marketing strategies. As the collective value of customers can be used as a proxy to determine the firm's total value and its future potential growth and hence, help with other business decisions and strategies (Gupta et al., 2006; Rust et al., 2004).

The metric used to measure the value of the customer over the entire period of their relationship with the firm is intuitively termed the customer lifetime value (CLV, CLTV, or Customer LTV). The increasing importance of this concept is not only due to its applicability in different important aspects of the business but is also due to the global trend in this age of information technology to make use of the readily available data collected by businesses about their transactions and interactions with the customers. These and other advantages discussed in later chapters made segmenting customers and predicting their LTV consistently and accurately throughout their entire relationship a high priority for many businesses (Chamberlain et al., 2017; Vanderveld et al., 2016; Zhao et al., 2022). Likewise, initial predicted customer LTV can be used for comparing the customer's expected value to acquisition cost, in the case of direct marketing or online marketing. This is one of the advantages of modern direct marketing, which puts a direct individual price on acquisition costs in contrast to more traditional marketing like TV ads.

However, this LTV prediction problem is no trivial task. Indeed, older customer segmentation based on customer purchase history, e.g., recency, frequency and monetary value (RFM) segmentation, only goes so far and probabilistic models only fit a specific

type of systems for which they are designed (Gupta et al., 2006). Estimating such information as LTV using big data is one of the main purposes of machine learning and artificial intelligence. Machine learning methods are, informally, algorithms that allow machines to produce accurate outputs based on their training or learning experience from previously collected data or information. For example, given enough information about the history of a specific customer and other similar customers to him/her a machine learning model, should be able to learn the pattern in his/her behavior and predict the customer's future behavior.

Contrary to other models, machine learning (ML) models do not require certain information to be present in the data before making a prediction, instead, they produce the best predictions they can given the data, regardless of whether or not the data are sufficient to produce accurate predictions. Thus, ML always gives an output that is as good as the input. In fact, it is possible to prove that some ML techniques like artificial neural networks (ANNs) are universal function approximators, meaning they can approximate any function, if one exists, from the input to the output (Alpaydın, 2020). This reduces the prediction problem, to a problem of collecting more high-quality data.

Because of its many advantages and great potential, machine learning has been widely used in many applications in the industry from aircraft design to manufacturing from banking to automatic translation from marketing and online advertisement to behavioral predictions. More specifically, it is particularly suitable for the customer LTV prediction problem, because of the ML model's ability to sift through massively big transactions and engagement data and find correlations between input features and the target customer LTV solving the problem of interest in this study.

The current problem uniqueness lies in the nature of Armut's business, which is a local services marketplace connecting contractors and freelancers providing all kinds of services with customers who need those services in a safe, reliable and convenient way in exchange for a small fee. Such fees are collected through biddings of the contractors or occasionally as a fraction of the payment by the customer. This means that the business profits are for the most part not directly tied to the delivery of the service itself, but rather to the competition in the market. Furthermore, the customer LTV prediction is made as soon as the customer makes a service request. This LTV prediction is supplied to the so-called value tracking of Google Ads service for online ads, which allows the Google optimizer algorithms to optimize the return on ad spend (ROAS) for search-based direct ads. These direct ads appear to customers already looking for this specific service and

hence are an invaluable source for customer acquisition. However, this early-on prediction means that the model has limited data available when making the predictions.

1.1 Our Contributions

In the current study, we compare multiple machine learning methods' performance on the problem of customer LTV prediction for search-based ads' value tracking. During data preprocessing, different categorical data encoding methods were compared including ordinal encoding, one-hot encoding, target encoding, and some combinations of them. Subsequently, the performance of the machine learning methods of linear regression (LR), decision tree (DT), XGBoost, and artificial neural networks (ANN) were compared using the different encoding methods listed before. The most suitable model was chosen and was further optimized by adjusting the training set time window and segmenting the customers based on the presence of previous service history and based on the business model of the requested service. Finally, the optimum model was compared to baseline predictions such as predicting zero, predicting mean, predicting service-specific mean, or the more advanced segmented moving average.

This study investigates the application of machine learning techniques to a customer LTV prediction that is complicated by the marketplace business model, which involves a complex behavior of multiple parties and is quite different from retail or freemium models used in the literature. In addition, the effect of customer segmentation and customer history on the LTV was also studied. Finally, the model's deployment, monitoring, and maintenance in the industry were also investigated.

Chapter 2:
BACKGROUND AND RELATED WORK

In this chapter, we provide a background of the different aspects of this project, which includes a quick introduction to customer relationship management (CRM), a discussion of customer lifetime value (CLTV), digital marketing, the business models, and marketing in Armut, the firm providing the data set. Subsequently, we move on to an introduction to important machine learning concepts and supervised learning techniques. Finally, we provide a review of the literature on previous applications of machine learning techniques in customer LTV prediction. This background discussion is in no way comprehensive, for each of those topics is a huge field on its own, instead, a focused approach was used based on what is relevant to this study and similar customer LTV prediction studies, whether that LTV is subsequently used for advertisement or other types of customer relationship management.

2.1 CLTV and Online Advertising

Customers are considered the most important asset for any business. However, as we mentioned, not all customers are equal; different customers may incur different costs and make different types and volumes of purchases leading to widely variable customer profitability and LTV. Accordingly, the amount of attention and resources a company extends to different customers should vary; this includes the customer acquisition cost, service and retention costs, and attention a customer receives in cross-selling and customer expansion efforts. Customer-specific decisions based on these differing LTVs represent a cornerstone in modern customer relationship management, which has become essential in the rapidly expanding type of customer-oriented businesses. Indeed, focusing too much on the wrong customers hurts the profitability of all businesses significantly. Moreover, CRM becomes absolutely central for some businesses such as companies with freemium products where a small fraction of high-value premium users, 10-20% of premium users, finance the product for most users and creates the difference between profitability and loss (Gupta et al., 2006; Sifa et al., 2018).

Once the customer LTV and related churn risk are known or predicted through a model, the customer relationship management involves many policies to increase the value of some customers, prevent the churn of others or upsell to others. This has been increasingly carried out through digital and direct marketing in recent years. Additionally,

customer acquisition cost can be compared to expected customer LTV for marketing budget spending optimization.

2.1.1 CLTV Definition, Importance, and Applications

CLTV has become an increasingly important metric in the past two decades as customer profitability analysis has been used as the basis for many marketing and managerial decisions. The customer profitability analysis is conceptually quite similar to a business's profitability analysis, except that instead of considering the entire pool of customers, the calculation is done for each customer individually. This provides a valuable diagnostic advantage that is not available from aggregate profit statements or total sales. It is sometimes advisable to 'fire' or restrict service to some customers who are a liability rather than an asset or create a tiered service system (Chang et al., 2012; Gupta et al., 2006) to allow the company to utilize its limited assets to take full advantage of the potential of high-value customers. For example, expending certain retention expenses can be beneficial with some customers, while low-value customers may only be profitable if their retention expenses are kept below a certain limit.

In addition, the LTV of a customer compared to his/her acquisition cost can quantify the return on investment in the marketing department which goes hand in hand with the increasingly popular trend to make the marketing department financially accountable, rather than being evaluated holistically using traditional metrics such as brand awareness or even sales (Gupta et al., 2006; Rust et al., 2004). Indeed, it has been shown that sales-driven decisions can hurt the overall profitability in the long run (Gupta et al., 2006). By treating marketing as an investment in an intangible asset, namely the relationship with the customers, rather than a cost, the business can optimize its marketing activities and spending to gain the highest return on that investment.

Moreover, the precise calculation or prediction of LTV is becoming increasingly feasible as businesses already collect massive amounts of data about all interactions with customers, which makes it quite easy to directly calculate LTV for previous customers and to train machine learning models for reliable predictions in the future (Chang et al., 2012; Gupta et al., 2006; Sifa et al., 2018). Thus, LTV allows the company to make good use of the data they already collect. LTV is also a powerful performance metric in other areas of the business such as product experimentation, customer service, evaluation of the platform user interfaces, and others (Vanderveld et al., 2016).

Customer Lifetime value is defined as the total present value of current and future profits from a customer over the entire relationship they have with the business. Chang et al. (2012) adopted this definition; "*the stream of expected future profits, net costs on a customer's transactions, discounted at some appropriate rate back to its current net present value.*" This value can be considered as (1) the marginal profits from the customer transactions (Chang, 2012; Gupta et al., 2006; Wang, 2008), (2) the revenues from the customer, especially used when costs are not allocated to certain customers, e.g., platforms maintenance (Sifa et al., 2018), or (3) the marginal profits less the marketing acquisition costs, (4) retention costs or (5) both costs (Chang, 2012; Gupta et al., 2006). Fixed costs may or may not be subtracted as well due to certain technical difficulties (Chang et al., 2012; Gupta et al., 2006). One of these difficulties with gross costs is that costs are tracked per activity in accounting e.g., TV ads and virtual platform maintenance, and assigning them to customers is particularly challenging.

In conclusion, no single approach to LTV has been agreed upon as being invariably superior (Chang et al., 2012; Gupta et al., 2006). Instead, different firms adjust the definition of LTV slightly to better suit their needs; a business with ambiguous retention costs may not include retention costs in LTV calculations and a business operating in a freemium setting may not include the bulk operating costs, which vary only slightly with the number of customers. Alternatively, a business with branches of widely differing fixed operating costs (rent, utilities, etc.) may include that in LTV and give a higher value to the customers in the branch with low overhead costs e.g., more discounts or promotions may be offered there. The LTV's general mathematical definition for a single customer can be written as

$$\mathrm{LTV} \equiv \sum_{t=0}^{T} \left(\sum_{j}^{N} \frac{(P_j - C_j)}{(1+i)^t} \right) - REC_t - \mathrm{AC}, \tag{1}$$

where t is the time period, T is the time horizon, N is the number of transactions in period t, P_j is the jth product's or service's price, C_j is the jth product's or service's direct cost and possibly fixed operating costs divided for a single customer, i is the discount rate, REC_t is the retention & expansion costs in period t such as marketing and promotions and AC is the acquisition cost (Chang et al., 2012; Gupta et al., 2006).

Some online businesses prefer to use revenue, i.e., discard costs from the equation, citing low or non-existing individual operating costs for customers (Drachen et al., 2018; Tekin et al., 2021; Zhao et al., 2022), while others use purchase value similar to revenue

because margins may fluctuate and do not reflect the customer behavior that needs to be elucidated and influenced by marketing efforts (Vanderveld et al., 2016). The same firm reportedly uses profit margin for customer LTV calculations for finance purposes but not marketing, thus having multiple LTV definitions based on application. They also have short-, medium- and long-term customer LTV models for differing time horizons.

For a customer with a constant profit margin m, constant retention probability r i.e., probability of them being an "alive" customer and that they make a purchase at time period t, and assuming infinite time horizon, and ignoring retention and expansion costs, the LTV formula reduces to

$$\text{LTV} = \sum_{t=0}^{\infty} \frac{(P - C)r^t}{(1+i)^t} = m \times \frac{r}{1+i-r}\,, \tag{2}$$

where m becomes multiplied by a constant factor that is commonly known as the margin multiple. Hence, LTV becomes a product of margin and a margin multiple. Acquisition cost is outside the sum and is subtracted once and hence, can be subtracted from this definition as well (Chang et al., 2012; Gupta et al., 2006). Finally, the related concept of customer equity (CE) is defined as *"the total of the discounted lifetime values summed over all of the firm's current and potential customers."* (Rust et al., 2004, p. 110).

To better understand customer classification from the LTV point of view, Figure 2.1 shows the main types of customers based on purchase frequency, monetary value of purchase, and churn rate relative to the average customer. These are the most important

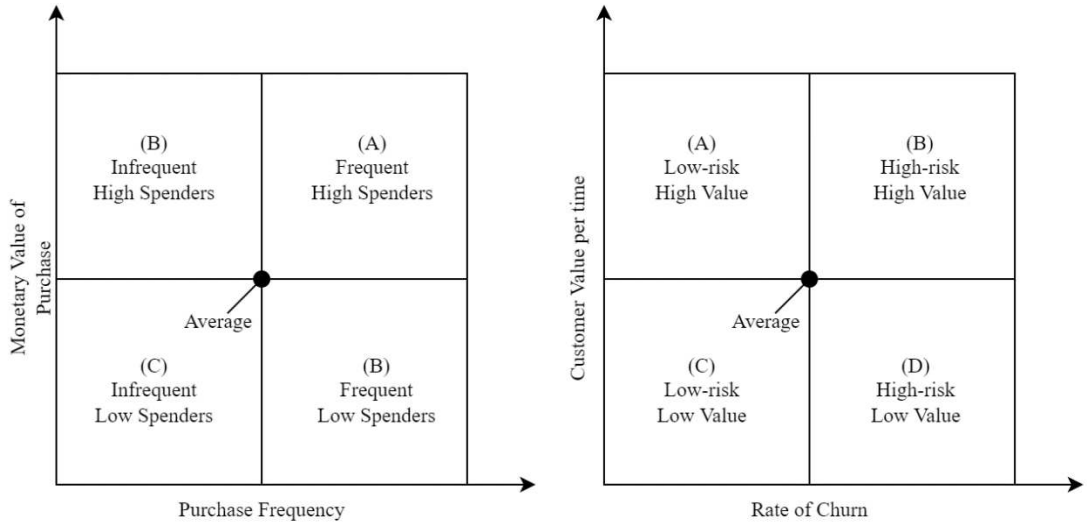


Figure 2.1 Customer classification based on RFM (a) with respect to the average based on the monetary value of purchases and frequency (b) Customer classification based on customers' total value per time and probability or rate of churn.

predictors of customer LTV as can be inferred directly from the definition and equation. In part (a), group A, frequent high spenders, are clearly the most important types of customers with the highest LTV and the firm's policies should attempt to expand and retain this group. Infrequent high spenders and frequent low spenders, both group (B), are average customers, to whom the firm should upsell more frequently or higher value products. Infrequent low spenders are low-value customers in whom the firm should invest minimally until they are turned into higher-value customers.

Similarly, Figure 2.1 (b) combines the total customer value per time, which includes both purchase monetary value and frequency from part (a), and churn rate. Here, group (A) of low-risk high-value customers are the loyal and highest value customers, i.e., frequent high spenders, who provide stable revenue to the firm. On the other hand, some of those frequent high spenders may be at risk of churn and urgently need to be retained and their churn needs to be prevented. On the contrary, low risk low value customers are loyal customers, to whom upselling is necessary, while high-risk low-value customers are fleeting customers, in whom the company should invest selectively similar to new customers' acquisition.

2.1.2 Customer LTV Prediction Modeling

There have been many efforts to create different types of models for LTV to varying degrees of success. Gupta et al. (2006) reviewed the main classes of models in detail. The most straightforward models are RFM models which classify customers based on recency, frequency, and monetary value of previous transactions. This is done by dividing the customer pool into n equal intervals for each of those metrics e.g., $10 \times 10 \times 10$, for a total of thousand cells, and giving each cell a score which is subsequently turned into an average LTV for the entire cell. RFM models usually show that recency is the strongest estimator of customer value, followed by frequency and transaction monetary value (Gupta et al., 2006).

Probability models assume that the customer activities can be modeled as a stochastic process with hidden characteristics and that transactions in the customer population follow a certain probability distribution of some kind. In non-contractual settings, the Pareto/NBD model is one of the first models for LTV and is considered a benchmark for more recent modified models, (Gupta et al., 2006). The model only requires each customer's recency and frequency. The Pareto/NBD model is usually augmented with a

monetary value model to directly produce the actual customer LTV. The principal assumption is that the customer behavior, which follows a certain distribution, changes from one customer to another but each customer's behavior does not change until they die/churn, i.e., stop being a customer. The returning or resurrection of customers can be modeled with the assumption that the customers are treated as a renewable resource. The assumptions (Gupta et al., 2006) are:

i. Customers continue to buy from the business till they "die"/churn.
ii. The number of transactions made by a customer follows a Poisson distribution.
iii. Heterogeneity/variation in the transaction rates of different customers and dropout ("death") rates of different customers both follow gamma distributions.
iv. The monetary value of transactions of a customer varies randomly around a mean value (normal distribution).
v. Transaction rates, dropout (death) rates, and monetary value distribution (mean & variance) are unique (independent) for each customer and are invariable with time.
vi. Customers lifetime periods have an exponential distribution.

Similar to probability models, there are econometric models that attempt to combine theoretical modeling with empirical knowledge using more general models including general hazards models for churn and even expansion and other models for customer acquisition. The models may also include several covariates that may vary across industries (Gupta et al., 2006). Applications of such models require experience in the specific market to gain insight into the effective covariates and develop a reliable model for the firm's specific position. One particularly attractive model is the one provided by Rust et al. (2004), which uses the Markov switching matrix to combine the modeling of retention and acquisition. Persistence models are, simply put, dynamic or continuously updating econometric models that utilize multivariate time-series analysis methods such as vector autoregressive (VAR) models to correlate some covariates, usually advertising campaigns, discounts, or product quality/customer service, with variables of interest such as LTV or CE as the covariates evolve in time (Gupta et al., 2006).

Finally, more recently with the rapid expansion of machine learning and data mining capabilities and instantaneous digitalization of most customer records as the transactions are taking place, highly predictive models are developed that are superior in accuracy, can extract information, and learn from big data using a large number of covariates and can incorporate theoretical models if need be.

2.1.3 Online Marketing

After analyzing previous LTV data or predicting LTV values for future customers using the models above, the firm turns them into actionable policies, and among the most important actionable areas based on LTV are marketing strategies.

Businesses that are customer-oriented use LTV to maximize their marketing expenditure on attracting high-value customers by optimizing return on investment (ROI) in marketing strategies or return on marketing as Rust et al. (2004) put it. It has been widely established that marketing as a whole, and especially advertising, has been evolving with communication technologies, and this digital age is no exception. Indeed, since 1994, online marketing has been building momentum as more and more people started using the world wide web and social media (Ha, 2008). Recently, direct digital marketing is the fastest growing form of marketing and is worth hundreds of billions of dollars worldwide. Online marketing budgets are increasing rapidly at annual rates around 10%; the rates ranged between 6-14% in the US and UK in 2017 and mobile platforms marketing alone grew by 45% (Kotler et al., 2019). Globally, digital ad spending represents more than 40% of total ad spending, and it is expanding faster than all other forms of marketing (Criteo, 2018).

Additionally, internet accessibility is continuously increasing. Today, 46% of the world population has access to the internet and the fraction of internet users is as high as 92% in developed countries. Some companies use digital marketing exclusively such as Amazon and many companies spend at least half their budgets on online marketing with all companies having some kind of online presence. That is because more than half of all households in the US shop regularly online and similar trends are increasingly observed worldwide (Kannan & Li, 2017; Kotler et al., 2019).

Direct marketing means engaging with individual customers by tailoring their experience for them specifically as a narrowly defined segment of customers or even to individual customers, not only based on their LTV but based on all data about them as a customer segment or individual (Kotler, 2019). While traditional marketing (TV ads, postal mail, telemarketing, etc.) could theoretically be direct, digital marketing is far more easily adapted to consumer preferences and purchase history. Digital marketing can be categorized as social media marketing, mobile platforms marketing, email marketing, and other online content marketing including websites, online videos, blogs, and online advertisements. Online advertisements are further divided into display ads, or paid

display ads, which are displayed on different websites and platforms, and search-based ads that appear to customers using a search engine with a specific keyword (Kannan & Li, 2017; Kotler et al., 2019).

Digital marketing, with its different categories, has many advantages for both customers and businesses. For customers, it is convenient, easy, private, and customized to their exact needs and preferences. For businesses, it has lower cost, is more efficient and quicker, and is easily more customizable to different consumers or customer segments preferences. Furthermore, it is greatly more flexible to create real-time marketing with personal engagements and links the firm's brand with big local or global events for customers (Kotler et al., 2019).

This personalization is possible because of the big user data gathered by internet giants like Google and Facebook which use these data to show ads to users who will be potentially interested in the advertiser's product. Additionally, search-based ads appear to prospective customers who are already looking for the corresponding specific service or product based on their search keywords and at the location of interest, and hence, are more likely to be interested and be acquired as new customers. Finally, online marketing usually allows the firm to automatically know which form of marketing led to the conversion, putting a direct acquisition cost for each customer (Google Ads Help, 2022; Kannan & Li, 2017).

2.1.4 Search-based Ads and Google Ads

With the established advantages of online marketing in mind, search-based ads are usually a significant part of the marketing strategy comprising 9% of world marketing spending (Criteo, 2018), double that if one considers search engine optimization (SEO). Search-based ads like other types of marketing help the business achieve their overall marketing campaign objectives, which may change depending on the type of business and the type of campaign. The desirable targeted action from the customer is termed conversion and is naturally defined differently by each campaign. The conversions may not be financial, such as newsletter subscription, phone app download, an account creation (signing up), or even simpler interactions on the website. This is termed micro-conversion, which is not the end goal of the campaign but rather a step towards that goal. Macro conversions, on the other hand, are the end-goal of the campaign such as product

purchase, paid software or magazine subscription, service request, donations, or other similar end-goal actions (Google Analytics Help, 2022; Lee et al., 2001).

E-commerce and online businesses usually have clicks that generate macro conversions online directly, while other businesses may have clicks that generate micro conversions which may be less valuable or generates leads that may or may not pan out and turn into a macro conversion eventually. Alternatively, a campaign may aim to promote micro conversions themselves as a part of raising brand awareness or for other reasons. For macro conversion generating ads, the marketing objective is usually to maximize the value of conversions per cost of ads or per the daily advertising budget. This requires conversion value tracking, which means the seller must report back the value of each transaction taking place. On the other hand, micro conversion generating ads may attempt to optimize bidding to promote the most promising leads or micro conversions. Here, the firm could aim to bid more on instances which are more likely to result in a conversion, while bidding less on instances with lower potential. This requires only conversion tracking i.e., the firm must keep track of which clicks turned into a conversion.

Since Google has over 90% of the search engine market (Statcounter Global Stats, 2022), Google Ads are the main platform that optimizes marketing spending followed by Microsoft Advertising (formerly Bing Ads), which serves both Bing, Yahoo, and AOL search engines exclusively (Microsoft Advertising Help Center, 2022). Google Ads, similar to other search-based marketing platforms, works by allowing campaigns to choose a set of keywords and if more than one firm has chosen the same keyword, a bidding system is in place to determine which ads are placed and their placing orders in the search engine results page. Bidding on keywords with no other bidders results in a relatively cheap cost per click. The bidding is done automatically, and one may let Google Ads or Bing Ads autobidder optimize the bidding process based on some criterion. Alternatively, sometimes the marketing team may prefer to set a manual maximum cost per click (CPC) for themselves and the bidding manager uses that hard limit in the bidding process. This is usually unadvisable because the search engine does not share its data or algorithm and manual optimization becomes extremely hard except in special cases.

The automated bidding system offers two main categories of optimization. Firstly, the optimizer may attempt to optimize the ad spending so that it achieves an average target cost of click per conversion as reported by conversion tracking or average revenue return on each click as reported by value tracking. Secondly, the optimizer may attempt

to maximize the conversions number or conversions total value by spending a daily campaign budget. For optimizing the value of each click based on a certain return over ad spending, a target ROAS algorithm is offered. Whereas for maximizing the total value from marketing by spending a specific budget a Maximize Conversion Value algorithm is offered. Similarly, for optimizing the conversion rate, a target Cost Per Action/Conversion algorithm is offered, whereas for maximizing conversions number, on a daily budget, a Maximize Conversion algorithm is offered. Marketing campaigns which aim for improving brand awareness and increasing traffic can use Maximize Clicks algorithm which attempts to spend a daily budget for traffic or clicks promotion or use a target Impression Share by attempting to achieve a certain appearance rate in searches regardless of cost but still within a set budget e.g. with 100% target, the automated bidder spends the daily budget to ensure that the ad will appear to 100% of all searches on that specific keyword; this is usually used with brand names (Google Ads Help, 2022; Kannan & Li, 2017). These different manual and automated bidding strategies are summarized in Table 2.1.

However, in order for the algorithm to optimize the value of certain ads, the marketer must provide a value for each transaction (value tracking) for the Google Ads algorithm to learn to either achieve a certain return on ad spend (ROAS) or even maximize the overall daily value by spending the entire daily budget. This value can be either a transactional value or, as we come back a full circle, a customer lifetime value.

Table 2.1: Summary of automated bidding strategies on Google Ads.

Manual Maximum Cost Per Click (CPC)	Sets a hard cap on the maximum cost per click that the bidder cannot exceed.
Target Cost Per Action (tCPA)	Optimizes the bidding so that the average cost per conversion is at the target value provided by the campaign manager; requires conversion tracking.
Maximize Conversions	Attempts to spend the entire daily budget maximizing the number of conversions.
Target Return on Ad Spend (tROAS)	Optimizes the bidding so that the average return/revenue/profit ratio to the cost of advertisement (Ad spending) is at the target ratio; requires conversion value tracking.

Maximize Conversions Value	Attempts to spend the entire daily budget maximizing the total value of all conversions.
Maximize Clicks	Attempts to spend the entire daily budget maximizing the number of clicks regardless of conversions number or value.
Target Impression Share	Attempts to appear at the target percentage of all searches by spending a maximum of the total daily budget.

Despite the advantages of the lifetime value, they may not always be suitable for every situation. Fruchter & Sigué (2009) found that loyal customers with longer relationships do not necessarily lead to improved profits, because some customers look for the most suitable product at every purchase. Indeed, long relationships become more profitable only if the customer-seller mutual trust is higher than the customer's tendency for opportunism. Otherwise, the seller should treat customers similarly whether new or repeat. Another case is when there are technical difficulties in predicting the customer LTV values because the business is new, or sufficient data for predictions is not available. However, given enough data about the customer in a setting satisfying the conditions above, predicting LTV using machine learning has shown great promise in this study for Armut A.Ş.

2.2 Armut's System and LTV in Armut

Armut is an online local services marketplace that matches customers with suitable service providers. In this section, we explain how Armut works and how LTV is defined and calculated at Armut so that we have the required context to fully understand the prediction problem.

2.2.1 Armut Business Models

Different services at Armut follow different processes from a service request to its fulfillment. These processes are what we refer to as business models (BMs). All service requests on Armut's platform start with the customer specifying the details of his service request, then the process diverges for services that use different business models. Armut has two main business models (Salah, 2019), which we explain below.

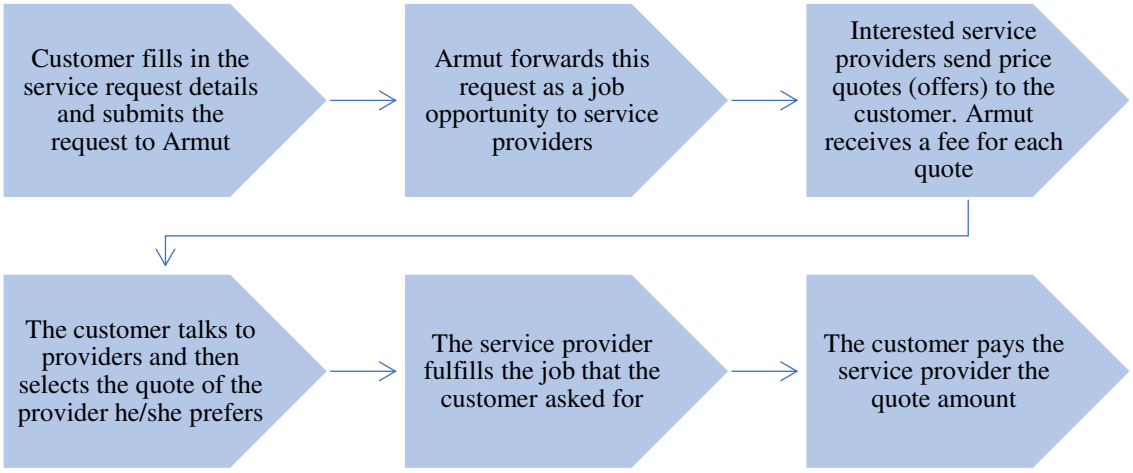


Figure 2.2: Business Model 1 (BM1) flow from service request to fulfillment.

Business Model 1 (Quote Model)

The first business model, shown in Figure 2.2, is called the quote model and it is used for most services at Armut. In this business model, a customer posts a detailed request for a specific service on Armut, which is forwarded to suitable providers who in turn send price quotes or offers to the customer. The customer then considers the offers and chooses one of the providers to carry out the job, after which the customer posts a review for the service provider on the platform. In this business model, Armut earns revenue from the fees the service providers pay to offer a quote, and service providers are paid for their services directly by the customers. Examples of services that follow this business model are house moving, plumbing, house painting, and private lessons.

After a customer receives quotes for his service request and contacts the quoters and gets the job done, he may or may not flag the job as done on Armut's platform and there is no penalty for not doing so, except that he will not be able to rate the provider if he does not flag it as done. Because of that, the fulfillment information for BM1 service requests in Armut's database is not very reliable. This point will be of importance when modeling LTV.

Business Model 2 (Reservation or Book-now Model)

The second business model, shown in Figure 2.3, is called the reservation or book-now model, where Armut works as a broker for the customer to acquire a specific service. In this business model, the customer does not choose the service provider who will perform his service. The customer makes a reservation for a service at a specific time, providing the necessary details of the request. Armut determines the price of the service based on these details and immediately requires full payment for the service up front.

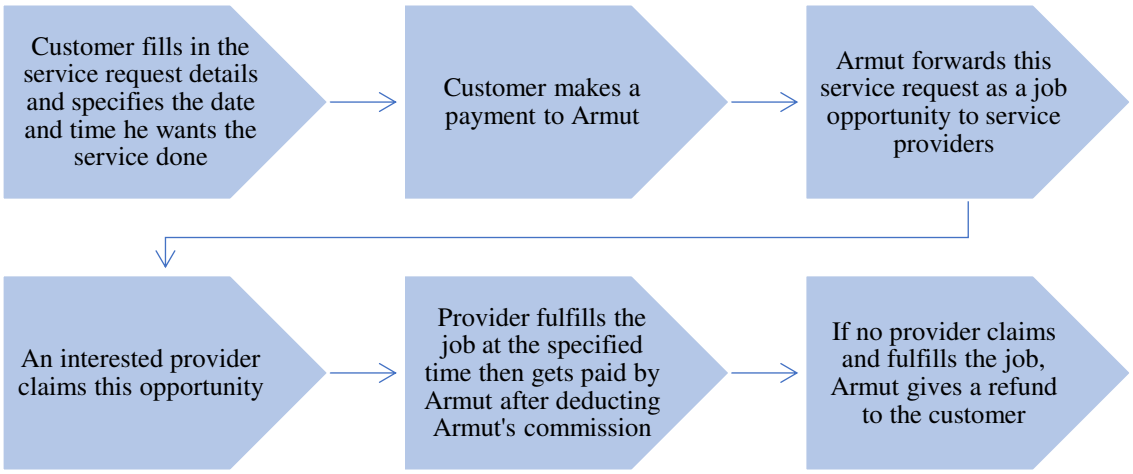


Figure 2.3: Business Model 2 (BM2) flow from service request to fulfillment.

Subsequently, Armut sends this request as a job opportunity at a lower price to the appropriate service providers keeping a small commission. An interested service provider claims the job, and fulfills it at the reservation time, after which payment is made to the service provider and the customer's review of the service provider is requested. If no service provider claims the job or if the service provider does not fulfill the job at the reservation time, Armut returns the full payment to the customer. Few services at Armut follow this business model, the most prominent of them is the home cleaning service. Manicure and pedicure service is another example of a BM2 service.

In this business model, the customers can also make a subscription for a service, which they receive periodically at fixed intervals. It is important to note that this business model is suitable for services that are relatively easy to price beforehand based on the details of the request. Also, we note that there are no quotes or quote-associated risks for the service provider that is no risk for quoting and gaining nothing if outbid by another provider. Armut earns revenue from the commission which is the difference between the customer's payment and that made to the provider. The differences between these business models will lead to some differences in the modeling process later on. The is also a difference between BM1 and BM2 regarding the fulfillment data of the service requests. In BM2 the customer pays the service price beforehand to Armut, so if the job is not fulfilled, without Armut's knowledge, the customer has to inform Armut in order to get a refund. Therefore, the service request fulfillment data is trustworthy for BM2 services.

2.2.2 Customer Statuses and LTV at Armut

As explained before (See section 2.1.1), customer retention and expansion (cross-selling) activities are different from acquisition ones. In Armut, retention and expansion activities are done separately through the mobile application and website notifications and methods other than paid advertisement, and hence, run minimal costs. Armut reserves the largest portion of its marketing budget for the acquisition of new customers and the "resurrection" of old inactive customers through search-based ads. Therefore, the customer LTV is limited to the revenue less the acquisition costs and is usually not discounted because of the short lifetime of customers, which extends to several weeks on average. However, the search engine's, i.e., Google's, ROAS optimizer value tracking requires the profit of the conversion to identify the return on that investment without subtracting AC. Hence, the LTV predicted is further reduced to the lifetime revenue of the customer.

When a customer makes a service request at Armut, he is classified into one of three categories as far as marketing is concerned. Acquired, Alive, or Resurrected. Acquired customers are new customers that have never used Armut before. Alive customers are customers who have used Armut recently, specifically, within the past 28 days. Resurrected customers are customers who have used Armut in the past but have churned, or alternatively, they are the customers who have used Armut in the past and then stopped using it for 28 days or more.

The lifetime of a customer starts when he makes a request as an acquired or a resurrected customer, and it does not end until the customer stops using Armut for 28 days or more (i.e., churns), and by using Armut here we mean making service requests. The customer's LTV is all the revenue that Armut received from the customer's service requests during that period.

In Figure 2.4, we give the example of two customers' service requests, we can investigate how these service requests translate to lifetimes and LTVs here. Customer 1 makes his first request as a new customer, then makes two more requests before churning, so his lifetime spans 35 days and his LTV is the sum of the revenues received from his three service requests. Customer 2 makes his first request as a new customer as well, then makes one more request and then churns. So, his lifetime spans 20 days and his LTV is the sum of the revenues gained from the two service requests. After churning, he comes

back to Armut as a resurrected customer and starts a new lifetime that spans 10 days and contains two service requests.

In summary, the customer LTV in Armut is the sum of the quote fees (for BM1) or commission (for BM2) of the current service request, plus the quote fees and commissions for all following service requests until the customer churns.

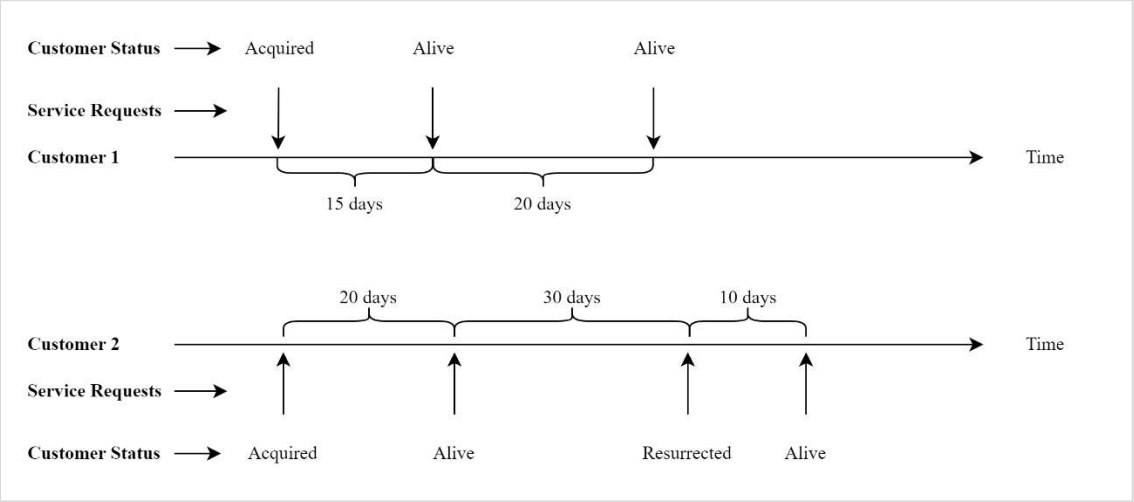


Figure 2.4 An example journey of two Armut customers and their status changes.

2.2.3 Search-based Ads in Armut

Armut places advertisements on different platforms and on TV, however, search-based Google Ads is the most important avenue for its marketing. Armut uses value tracking and provides LTV values for each customer that makes a service request after clicking a search-based ad. In Armut's case, the conversion/action is the service request, and the conversion value is the customer's LTV. Armut chooses the previously discussed automated bidding system of target return on ad spend ($LTV/Cost\ of\ Click$) (See section 2.1.4), which attempts to achieve the target return by bidding higher on ads that are likely to produce customers of higher value and lower for low-value customers, thus, keeping the actual ROAS as close as possible to the target ratio.

The LTV generating actions in Armut do not happen at the moment the service is requested, and more importantly, they are not performed by the customers who request the service; instead, they are performed by service providers rather than customers in BM1. Therefore, it is not possible to send the actual LTV at the time of the request to Google to perform the ROAS optimization accordingly. Instead, Armut chooses to predict the LTV of each service request at the moment it is created and sends that value to the value tracking system of Google Ads. This means that the predictive model used

by Armut has no access to any information that arrives after the service request is made. For BM2 service requests, the LTV prediction has to be made before the customer makes the payment. i.e., after step one and before step 2 in Figure 2.3.

LTV prediction at Armut was previously performed using a segment based moving average algorithm. This project aimed at and succeeded in developing a model that predicts LTV more accurately to enable google to optimize Armut's ads more effectively.

2.2.4 Segmented Moving Average Baseline

The LTV predictions were previously made using a segmented moving average-based algorithm. The algorithm splits the customers into segments using the details of their service requests and predicts the LTV of a new service request by a customer as the average LTV of the customers in the segment to which this job belongs. The segmentation is done based on customer type (Acquired, Resurrected), requested service, the service request details, its frequency (Only BM2), and province; and stores the segment's 3-week LTV averages. At prediction time if a segment contains less than eight jobs, the algorithm falls back to the average of the customer LTVs of the jobs at a more general segment, i.e., without considering the province and other dimensions. If that segment also does not contain enough instances, the algorithm returns the LTV average over a larger time window instead of 3 weeks. The algorithm is akin to a handmade regression tree trained on a rolling window of the data.

2.3 Overview of Machine Learning Methods

Because of the complexity of the customer LTV prediction problem and the presence of many features that are hard to use directly for predictions, it is a suitable problem for machine learning techniques, which utilizes the great computational power and data storage capacity of machines. Machine learning is a vast field, but we will touch on what we deem important or relevant for this discussion. That includes a focus on supervised learning techniques including, neural networks, random forests, gradient boosting methods, and a few others.

2.3.1 Supervised ML Algorithms

Here we discuss the most common single, as opposed to ensemble, supervised learning methods in machine learning including, k-nearest neighbor, support vector Regression (SVR), Decision Trees (DT), and artificial neural networks (ANNs).

K-nearest neighbor

This algorithm uses a non-parametric method which makes no assumptions about the probability distributions in the data set. Such a method is particularly useful when the data output density with respect to the inputs does not follow any typical distributions assumed in parametric methods. Another advantage is the method's simplicity and the simplicity of its underlying principle, namely that similar inputs should produce similar outputs. Given a hyperparameter $k \ll N$, where N is the data set size, the regressor estimates the predicted value to be an average of the values of the nearest surrounding k points, which are identified using a distance function, typically a Euclidean distance function. From this distance function, the k-nearest neighbors, say the nearest 3, 5, or 10 entries, are determined, the average of their target values is calculated and assigned to the predicted point. This is similar to a naive estimator which divides the space into bins of fixed width, which have certain target values. Instead, the bins have a fixed number of entries and a variable width (Alpaydın, 2020).

Support Vector Regression (SVR)

This algorithm is a kernel-based algorithm that is considered an extension to the Support Vector Machine (SVM) algorithm. It is also considered more complex and more powerful than many other algorithms. While the learner puts hyperplanes separating points belonging to different classes in SVM, the SVR learner attempts to find the best fit hyperplane that has the maximum number of points on the plane within a certain margin ϵ. A learner with a larger margin has less overfitting and the SVR algorithm acts to find an optimal model through optimization. At higher dimensions with higher number of features, the hyperplane gets more and more complex and the simpler linear SVR implementation, which uses linear Kernels, is usually faster with negligible performance difference. Additionally, the kernel function allows mapping nonlinear functions to higher order linear ones. Such advantages led to the widespread applications of SVR and SVM in the industry (Alpaydın, 2020; Tanrıver, 2021).

Decision Trees

The decision tree is a fairly simple yet efficient non-parametric method that is highly versatile in supervised learning, as a classifier and a regressor. The idea is relatively simple; starting from the entire data set the learner attempts to check for a certain criterion for one of the inputs or features dividing the set into two more homogenous groups. These two sets are further divided into two more sets each and so on until the sets are satisfactorily homogenous. Each splitting node is termed a *decision node*, while the ends of each branch are termed *leaf nodes*. The output assigned to all the points in the leaf node is the class of the majority of points in classification or an average of all the points in regression. After training, the algorithm provides a simple rule list of IF statements that is readily usable for predictions on new entries and is very easy to interpret even by any user and not necessarily a machine (Alpaydın, 2020; Kotsiantis, 2013). A simple example of a classification tree for dwarfism in humans is presented in Figure 2.5.

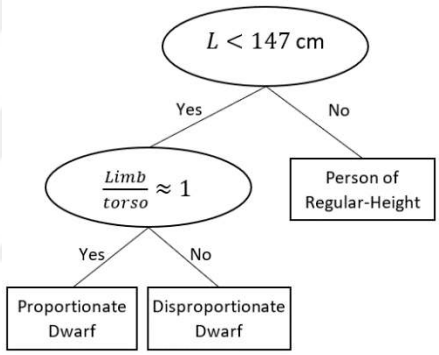


Figure 2.5: A simple example of a decision tree to classify dwarfism in humans.

The purity of the split at each decision node is determined by certain functions. The most common of those functions are statistical entropy, Gini index and misclassification error for classifiers, and sum of squared errors and sum of absolute errors for regressors. Among all features, all splits are weighed to determine the best split with the highest increase in purity at each decision node in a greedy fashion, that is, regardless of other nodes in the tree. This method leads to overfitting as the tree grows to perfect or near-perfect purity. The tree requires pruning either by putting a limit on the maximum number of subtrees which is termed *prepruning* or by pruning the subtrees that perform no better than their parents on the *validation* or *pruning set*, which is called *postpruning*. Sometimes, instead of using the most suitable feature at each split, a linear, or even non-linear, combination of the features may be used as a criterion. e.g., $\sum_i a_i x_i + a_0 > 0$, where x_i are the different features and a_i are the coefficients. Such trees are called

multivariate but are considerably more complex and less frequently used (Alpaydın, 2020; Kotsiantis, 2013).

Artificial Neural Networks (ANNs)

Artificial Neural Networks and deep learning are one of the main algorithms of machine learning that are widely used and still comprise a large part of the active research in the field of artificial intelligence (AI) nowadays. It is commonly applied to image recognition, neural machine translation and other pattern recognition problems; however, it is also applied to many other classification and regression problems.

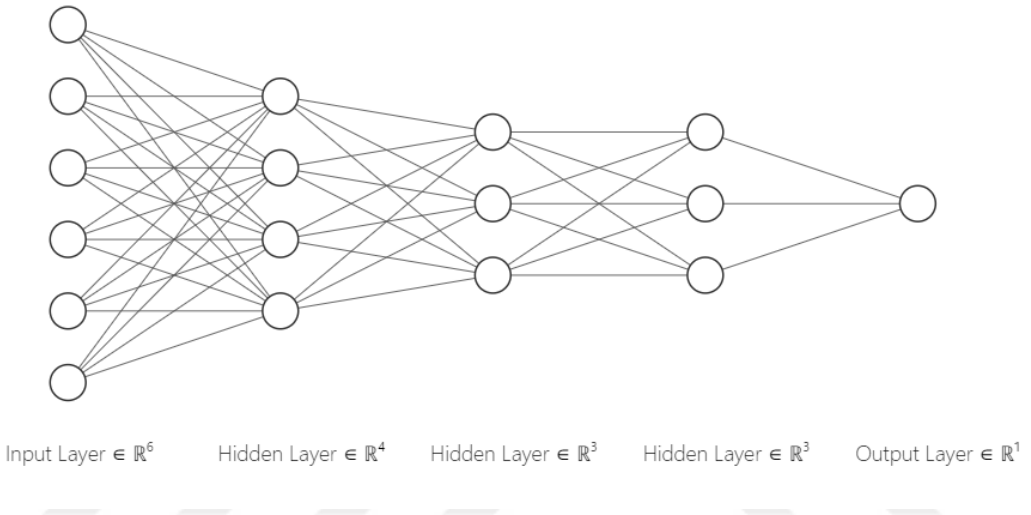


Figure 2.6 An example of a neural network structure. Generated using a free open-source software (Lenail, 2018).

A neural network is simply a network of perceptrons in multiple layers, which are all termed hidden layers except the first input layer and the last output layer (Figure 2.6). Each perceptron in a layer is an output of a function of a linear combination of every perceptron in the previous layer. The coefficients of these linear combinations are termed the connection or synaptic weights and are named after the synapses in the human brain. The functions in the hidden layers are termed activation functions and are used to introduce non-linearity in the learner and to provide "deeper" learning through more complex functions. This serves to produce a more abstract representation of the input with each layer so that the learner can produce more accurate outputs from any input. In regression problems, the output layer has no such function, and each output is a linear combination of the last hidden layer's perceptrons. In classification problems on the other hand, the output layer must use a probability function such as $softmax$ and $sigmoid$ functions. The structure of the hidden layers, their number, and the number of perceptrons

in each layer are adjusted according to the complexity and nature of the problem (Alpaydın, 2020; Goodfellow et al., 2016; Schmidhuber, 2015).

One of the main drawbacks of neural networks and deep learning, in general, is that these hidden layers and abstract representations are not interpretable or explainable to the users. Efforts to overcome this and create explainable artificial intelligence (XAI) are reviewed by Samek et al. (2021).

Given an input vector of size d, $x = (x_1, x_2, \cdots x_d)$, an output vector of size N, $y = (y_1, y_2, \cdots, y_N)$, a set of K hidden layers in the network, each of which has H_k perceptrons labeled z_{kh} and a set of weights connecting the $k-1$ layer to a z_{kh} labeled w_{khj}. The z_{kh} perceptron is evaluated as

$$z_{kh} = f\left(\sum_{j}^{H_{k-1}} w_{khj} z_{k-1,j}\right) = f(\vec{\mathrm{w}}_{kh} \cdot \vec{z}_{k-1}) \tag{3}$$

Note that a bias node, which allows shifting a perceptron value, is also added to each layer except for the output, $x_0 = z_{k,0} = +1$. Further note that the vector multiplication is done using the transpose as a matrix multiplication to imply a simple dot product of the two vectors giving a single scalar quantity as an argument to f, the activation function, which can be a saturating function like the sigmoid function or the tangent hyperbolic function (tanh), or a non-saturating function like the rectified linear unit function ($ReLU$),

$$ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & otherwise \end{cases} \tag{4}$$

As mentioned before, the output has no activation function and is given by the following equations for regression, binary classification and multiclass classification respectively

$$y = \sum_{j}^{H_K} w_{K+1,j} z_{K,j} = \vec{\mathrm{w}}_{K+1} \cdot \vec{z}_K \tag{5}$$

$$y = sigmoid\left(\sum_{j}^{H_K} w_{K+1,j} z_{K,j}\right) = \frac{1}{1 + \exp(\vec{\mathrm{w}}_{K+1} \cdot \vec{z}_K)} \tag{6}$$

$$y_i = \mathrm{softmax}\left(\sum_{j}^{H_K} w_{K+1,ij} z_{K,j}\right) = \frac{\exp(\vec{\mathrm{w}}_{K+1,i} \cdot \vec{z}_K)}{\sum_i^N \exp(\vec{\mathrm{w}}_{K+1,i} \cdot \vec{z}_K)} \tag{7}$$

The inputs can be z-normalized to the same scale with an average of 0 and a variance of 1 (See section 2.3.3). All the connection weights are initialized in a small interval

around zero for instance $[-0.01\ 0.01]$; however, the range should be decreased even more as the number of inputs increases and the structural complexity of the network increases. The weights are updated during training based on the training mode as either batch, mini-batch or online. In batch mode, a pass over all the patterns in the training set is termed an epoch and the weights are updated after each epoch. In mini-batch mode, the update is done after passing over a small subset of points (32 or 64 points, for instance). In online mode, the update is done after each point. However, the mini-batch multi-pass mode is quite common because of its faster convergence and the mini-batches mode minimizes the effects of outliers, unlike online training.

The following mathematical discussion of ANNs will follow Alpaydın's (2020) notation, for more detailed discussions, the reader should check out other references, e.g., Deep Learning (Goodfellow et al., 2016). As with other algorithms in machine learning, the weights are updated based on an error function fitting the type of the problem. For regression problems, the error function is usually the sum of squared errors function, whereas for two class and multiclass discrimination problems, sigmoid and softmax functions are used and the error is a cross entropy function. For a mini-batch size of B, the errors for regression, binary classification and multiclass classification are given by

$$E = \frac{1}{2}\sum_{b}^{B}\left(y_{true}^{b} - y_{p}^{b}\right)^{2} \tag{8}$$

$$E = -\sum_{b}^{B}\left[y_{true}^{b}\ln\left(y_{p}^{b}\right) + \left(1 - y_{true}^{b}\right)\ln\left(1 - y_{p}^{b}\right)\right] \tag{9}$$

$$E = -\sum_{b}^{B}\sum_{i}^{N}\left[y_{i,true}^{b}\ln\left(y_{i,p}^{b}\right)\right], \tag{10}$$

where the $y_{i,true}^{b}$ is the true value for the ith output or category, in case of multiple outputs on the same network, in the bth element of the batch, and $y_{i,p}^{b}$ is the predicted value for that output. The b superscript is **not** a power, but rather the index of the sample in the batch, which is to be removed in the online (one by one) mode of training.

Next, using gradient descent, the weight updates in the last hidden layer are calculated from the corresponding error function. For regression, the derivative is straightforward, and the update magnitude is given by:

$$\Delta w_{K+1,h} = -\eta\frac{\partial E}{\partial w_{K+1,h}} = \eta\sum_{b}^{B}\left(y_{true}^{b} - y_{p}^{b}\right)z_{Kh}^{b}, \tag{11}$$

where η is the learning rate which will be discussed in the next section. Note that this is for the weight connections between the last hidden layer and the output. For layers before that, one must carry out backpropagation calculations, which is named so because the error propagates from the last hidden layer all the way to the first layer. So, for the next to last hidden layer, the update magnitude becomes

$$\Delta w_{Khj} = -\eta \frac{\partial E}{\partial w_{Khj}} = -\eta \sum_{b}^{B} \frac{\partial E}{\partial y_p^b} \frac{\partial y_p^b}{\partial z_{Kh}^b} \frac{\partial z_{Kh}^b}{\partial w_{Khj}} \tag{12}$$

$$\Delta w_{Khj} = -\eta \sum_{b}^{B} \underbrace{-\left(y_{true}^b - y_p^b\right)}_{\partial E/\partial y_p^b} \underbrace{\left(w_{K+1,h}\right)}_{\partial y_p^b/\partial z_{Kh}^b} \underbrace{\left(z_{K-1,j}^b f'^{\left(\vec{w}_{Kh} \cdot \vec{z}_{K-1}^b\right)}\right)}_{\partial z_{Kh}^b/\partial w_{Khj}}, \tag{13}$$

where f' is the derivative of the activation function and each term in square brackets is equivalent to each of the partial derivatives in the chain rule as shown. This is expanded further into more layers.

Alternatively, a recursive algorithm of errors derivatives and update magnitudes is much more convenient. For the last three layers weights in a network,

$$e_{K+1}^b = y_{true}^b - y_p^b \tag{14}$$

$$\Delta w_{K+1,h} = \eta \sum_{b}^{B} e_{K+1}^b z_{Kh}^b \tag{15}$$

$$e_{Kh}^b = \sum_{i}^{N} e_{K+1}^b w_{K+1,h} \tag{16}$$

$$\Delta w_{Khj} = \eta \sum_{b}^{B} e_{Kh}^b z_{K-1,j}^b f'\left(\vec{w}_{Kh} \cdot \vec{z}_{K-1,}^b\right) \tag{17}$$

$$e_{K-1,h}^b = \sum_{j}^{H_K} e_{Kj}^b w_{Kjh} \tag{18}$$

$$\Delta w_{K-1,hj} = \eta \sum_{b}^{B} e_{K-1,h}^b z_{K-2,j}^b f'\left(\vec{w}_{K-1,h} \cdot \vec{z}_{K-2}^b\right) \tag{19}$$

Thus, for the kth layer, we can update the coefficients using

$$e_{kh}^b = \sum_{j}^{H_{k+1}} e_{k+1,j}^b w_{k+1,jh} \tag{20}$$

$$\Delta w_{khj} = \eta \sum_{b}^{B} e_{kh}^b z_{k-1,j}^b f'\left(\vec{w}_{kh} \cdot \vec{z}_{k-1}^b\right) \tag{21}$$

It is important to note that the actual calculations are done in terms of the Jacobian matrices J_k of vectors $\vec{z}_k^b$ with respect to matrices W_k for computational efficiency.

Deep Learning Optimizer & Regularization

In all these calculations, the learning rate (or factor), which has typical values of 0.01 to 0.001, is multiplied with all updates to limit and decrease the size of the update step and allow steady and smooth convergence. The error function derivative on its own may lead to overshooting and oscillations of the model and may lead to divergence.

However, the learning rate is sometimes not enough to prevent oscillations. To allow only persistent trends of the error function, the updates are assigned a momentum that builds up with each iteration and the effect of individual iterations is reduced (Alpaydın, 2020; Goodfellow et al., 2016). The simplest implementation for momentum is, for an iteration t:

$$s_i^t = \alpha s_i^{t-1} + (1-\alpha)\frac{\partial E^t}{\partial w_i} \tag{22}$$

$$\Delta w_i = -\eta s_i^t, \tag{23}$$

where s_i^t is the running average for weight update Δw_i at iteration t and α is sometimes termed the forgetting hyperparameter, which ranges from 0 to 1 but has a typical value of 0.9. The main disadvantage of momentum has to do with the required extra memory to store the momentum, but it is usually worth it. In addition, improved convergence can be achieved by implementing an adaptive learning factor that decreases the learning rate even further as the training progresses.

$$r_i^t = \rho r_i^{t-1} + (1-\rho)\left(\frac{\partial E^t}{\partial w_i}\right)^2, \tag{24}$$

where r_i^t is the accumulated past gradient for each weight w_i in the network and is an adaptive hyperparameter that is usually taken as 0.999. The implementation of adaptive learning alone in weight updates is as follows:

$$\Delta w_i = -\frac{\eta}{\sqrt{r_i^t}} \cdot \frac{\partial E^t}{\partial w_i} \tag{25}$$

Both momentum and adaptive learning factor are combined in the Adam algorithm:

$$\tilde{s}_i^t = \frac{s_i^t}{1-(\alpha)^t} \quad \text{(a)}$$

$$\tilde{r}_i^t = \frac{r_i^t}{1-(\rho)^t} \quad \text{(b)} \tag{26}$$

$$\Delta w_i = -\eta \frac{\tilde{s}_i^t}{\sqrt{\tilde{r}_i^t}}, \tag{27}$$

where $\tilde{s}_i^t$ & $\tilde{r}_i^t$ are bias-corrected running average and accumulated past gradient respectively which corrects those terms since they are initialized at zero and are biased towards zero in the early iterations. The corrections tend to 1 after many iterations since each of the hyperparameters raised to the 't'th power tends towards zero and no further correction is necessary.

Finally, the neural network model, like all models, needs to be regularized to ensure that the network's complexity matches the problem and is not overfitted to the training set. This can be achieved by using a suitably matched network architecture to the problem. However, finetuning the network architecture is more challenging; regularization implementation in a relatively large model that goes on to learn what is necessary without overfitting is more reliable. This regularization can be hardwired in the model through direct hints in the error function, or a weight decay term in the error function. The following is a simple implementation of weight decay:

$$E' = E + \frac{\lambda}{2} \sum_i {w_i}^2 \tag{28}$$

$$\Delta w_i = -\eta \frac{\partial E^t}{\partial w_i} - \lambda' w_i \tag{29}$$

Alternatively, dropout is a process similar to introducing data noise in other algorithms, which increases the robustness of the network as a whole as it helps distribute the decision-making to whole layers instead of a few distinctive patterns which may or may not be missing in real model applications. The dropout probability p is a hyperparameter with which any of the inputs or the hidden layers perceptrons can be dropped out completely by setting them to zero for 1 iteration, and the more inputs and hidden perceptrons there are, the higher p can be made. Each layer's activation function is divided by $1 - p$ to scale the remaining perceptrons to the same magnitude. This serves to distribute the weights over a larger number of units instead of depending too much on a few inputs or perceptrons and thus creates a more general and robust model (Alpaydın, 2020; Goodfellow et al., 2016).

2.3.2 Ensemble Supervised ML Algorithms

Despite the diversity of available algorithms and approaches in machine learning, there is no free lunch. There is no single algorithm that is going to outperform all others

on all problems, but rather each model will have its pitfalls. Hence, ensemble learning attempts to generate several base learners that complement one another to improve the accuracy and generalization of the combined model. This is a two-fold problem as we need to generate complementary models and then combine them effectively. However, the accuracy of individual models is secondary to them being complementary. For instance, a classifier with 70% accuracy combined with a second classifier specializing in the remaining misclassified 30% is much better than two overlapping classifiers with 90% accuracy. Hence, the base learners are optimized to be diverse and of reasonable accuracy rather than having optimized overall accuracy. However, this depends on the scheme of the combination of learners.

The diversity of the learners can be achieved in different ways. The base learners may have completely different models using different methods, for example, a parametric model can be combined with a non-parametric one to account for entries not following the predicted distribution from the parametric model. Similarly, one may use the same model but with different hyperparameters such as the number of neighbors in a k-nearest neighbors method, or the maximum tree depth in the decision trees method. Alternatively, the same learner may be trained on randomly different parts of the training set or specifically on misclassified entries by the previous learner, thus each learner would train on different fractions of the training set making slightly different predictions. Sometimes the models could be trained on different input representations of the same data such as lip reading from image processing by one model of an input video while another model analyzes the sound clip of the same video to improve speech recognition. Sometimes, this is called multi-view learning (Alpaydın, 2020).

The way the different models are combined is also varied. A multi-expert combination method uses the different models in parallel and concludes the overall output with some kind of average. This average is calculated over the output of all learners in the global approach or with the help of a gating model a specialized fraction of the learners on local parts of the input space in the local approach. The average can be done by simple voting or weighted stacking. On the other hand, a multi-stage combination method trains or tests the different learners sequentially; a learner is trained on the ones misclassified by the previous learner or is consulted when the previous learners are not confident. Meta-learning methods use another machine learning algorithm to combine the output of the learners to produce the final output (Alpaydın, 2020, Sagi & Rokach, 2017).

Random Forests (RF)

One of the most common combination methods using decision trees is random forests. As the name implies, the trees are created randomly by learning over a random subset of the training data set with replacement. Similarly, the features evaluated at each split node are randomized in what is known as *feature bagging*. This improves the diversity of trees to avoid the exclusive dependence on strong predictor features. Since the learning process is random, the final output is given by the median or arithmetic average of all trees in regression and simple voting in classification (Parmar et al., 2019).

Gradient Boosting and eXtreme Gradient Boost (XGBoost)

Instead of depending on randomness to foster diversity, boosting algorithms actively attempt to generate complementary learners sequentially to avoid the mistakes of their predecessors. The concept of boosting can be achieved by multiple algorithms. One of the earliest and most important of such algorithms is *Adaptive Boosting* algorithm or *Adaboost* for short.

Adaboost algorithm trains many weak learners sequentially on the same training set. However, the probability of each data point to be drawn to teach a learner increases each time it is misclassified. All the entries start with an equal probability of being drawn, $P_{i,1} = 1/N$. Given a weak learner with an error rate $(\epsilon < 0.5)$, the algorithm defines a scaling parameter of the n-th learner $\beta_n = \epsilon_n/(1-\epsilon_n)$ such that the probability of a data point number i which has been misclassified by the $n-1$ learner being drawn again to teach the n-th learner is given recursively by $P_{i,n} = \beta_{n-1}P_{i,n-1}$, while the probability for a correctly classified point j remains the same $P_{jn} = P_{j,n-1}$. The probability is normalized afterwards to sum up to one. If the error rate is more than half, the algorithm stops training more learners and the final output is given by weighted voting based on the accuracy of each learner as given by $w_i = -\ln(\beta_i)$ (Alpaydın, 2020). There are several other variations of the method optimized for several more specific sets of problems.

Gradient tree boosting and Extreme Gradient Boosting (XGBoost) are slightly different boosting methods that utilize decision trees specifically. We will focus on XGBoost here to avoid redundancy since it is a variant of the original method but more popular and superior than the original. XGBoost gives high-performance results and is at the cutting edge of machine learning algorithms due, in part, to its applicability to a variety of problems. This is clearly exemplified in its use by many of the winning teams in machine learning competitions such as Kaggle competitions and the KDD Cup and

others, while other methods lag farther behind. One of the main advantages of the method is its extremely high scalability which improves its running speed by more than an order of magnitude on both single machines with limited memory and clusters. The other advantages and upgrades done to the method include the unique tree-building algorithm and intuitive handling of sparse data and the use of a theory-based weighted quantile sketch algorithm. In addition, XGBoost utilizes out-of-core computations and cache-aware access along with parallel learning to reduce the computations speed and allow the processing of millions of entries on average desktops (Chen & Guestrin, 2016).

The discussion of XGBoost detailed here is based on the method's authors' publication (Chen & Guestrin, 2016) and the used library documentation (Cho & Chen, 2021). In line with its versatility, XGBoost utilizes the same method in regression as well as classification, where the output is the continuous probability. Due to this similarity, the understanding of regression should be sufficient to gain an understanding of the method since the differences in classifications are minor. As is typical in gradient tree boosting algorithms, the learning trees are initiated from an initial average guess and then minor improvements are made with each split. XGBoost uses a regularized objective loss function to determine these cumulative corrections in a series of trees until the algorithm exits and the final output is given as the sum of these improvements and the initial base value. Given n examples, and m input features for each example $\{\vec{x}_i\}$ and corresponding n outputs $\{y_{i,true}\}$, the XGBoost model sums N trees outputs and an optional initial base b to provide predicted values $\{y_{i,p}\}$

$$y_{i,p} = \Phi(x_i) = b + \sum_{k}^{N} f_k(\vec{x}_i), \quad f_k \in F, \tag{30}$$

where $F = \{f(\vec{x}) = w_{q(\vec{x})}\}(q: R^m \to T, w \in R^T)$ is the space of regression trees. Given a specific tree structure with T leaves, the q function maps the input features to the index of the leaves in the tree, where $q \in [1, T]$, w is the vector of improvement values or scores in the tree and b is the optional initial base value of the predictions, which could be the average, median or mode value for the training set or the midpoint of the output range. Each f_k corresponds to a different regression tree generated during the training of the model.

The regularized objective loss function to be minimized is given by:

$$L = \sum_{i}^{n} l\left(y_{i,true}, y_{i,p}\right) + \sum_{k} \Omega(f_k), \tag{31}$$

where $l(y_{i,true}, y_{i,p})$ is a suitable differentiable convex loss function that estimates the error in predictions over all entries. Similar to the ones used in neural networks equations (9-11), the commonly used loss function for regression is a scaled squared error:

$$l\left(y_{i,true}, y_{i,p}\right) = \frac{1}{2}\left(y_{i,true} - y_{i,p}\right)^2, \tag{32}$$

where the $1/2$ coefficient is added to simplify the derivative for the optimization process. Other loss functions such as the absolute error or unscaled squared error could also be used. Almost the largest difference between classification and regression arises from the different loss functions used for classification. In classification, cross-entropy is used and is given by:

$$l = y_{i,true} \ln\left(y_{i,p}\right) + \left(1 - y_{i,true}\right) \ln\left(1 - y_{i,p}\right), \tag{33}$$

where $y_{i,p}$ is the predicted probability while $y_{i,true}$ is the observed probability.

Next, the regularization term in the objective function of XGBoost, equation (31), is given by:

$$\Omega(f_k) = \gamma T_k + \frac{1}{2}\lambda \|\vec{w}_k\|^2, \tag{34}$$

where T_k is the number of leaves in the tree and γ and λ are regularization parameters. The first term penalizes the number of leaves and results in smoother results for the entire tree, while the second term acts like ridge regression to penalize overfitting and reduce variance error in the model. If the regularization parameters are set to zero, the objective function becomes the sum of the loss function over all points, and it reverts to Gradient Tree Boosting.

The tree ensemble is trained recursively with each tree generated after the other. When generating the m-th tree, the algorithm minimizes the following function which is defined recursively:

$$L_k = \sum_{i}^{n} \left[l\left(y_{i,true}, y_{i,p}^{k-1} + f_k(\vec{x}_i)\right)\right] + \Omega(f_k), \tag{35}$$

where $f_0(x_i)$, is initialized either randomly or based on the mean value or the midpoint of the range interval. Equation (35) shows that the algorithm is greedy as it acts to minimize L_k at each step regardless of the rest of the model. To simplify the model, the

loss function is approximated using the Taylor series truncated after the second derivative term.

$$L_k \cong \sum_i^n \left[l\left(y_{i,true}, y_{i,p}^{k-1}\right) + g_i f_k(\vec{x}_i) + \frac{1}{2} h_i f_k^2(\vec{x}_i) \right] + \Omega(f_k)$$
$$g_i = \frac{\partial l\left(y_{i,true}, y_{i,p}^{k-1}\right)}{\partial y_{i,p}^{k-1}}; \quad h_i = \frac{\partial^2 l\left(y_{i,true}, y_{i,p}^{k-1}\right)}{\left(\partial y_{i,p}^{k-1}\right)^2}, \tag{36}$$

where g_i and h_i are the gradient and second order gradient of the loss function with respect to prediction.

After removing the constant terms from equation (36), since we are only interested in the derivative and expanding the regularization term equation (34), the function becomes

$$\tilde{L}_k = \sum_i^n \left[g_i f_k(\vec{x}_i) + \frac{1}{2} h_i f_k^2(\vec{x}_i) \right] + \gamma T_k + \frac{1}{2} \lambda \sum_j^T w_j^2 \tag{37}$$

Given I_j is the instance set of points in leaf with index j.

$$\tilde{L}_k = \sum_j^T \left[\left(\sum\nolimits_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\lambda + \sum\nolimits_{i \in I_j} h_i \right) w_j^2 \right] + \gamma T_k \tag{38}$$

After unconstrained minimization of this function giving a fixed tree structure, it gives the unique minimum

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\lambda' + \sum_{i \in I_j} h_i} \tag{39}$$

Calculating the corresponding optimal value gives

$$\tilde{L}_k = -\frac{1}{2} \sum_j^{T_k} \left[\frac{\left(\sum_{i \in I_j} g_i \right)^2}{\lambda' + \sum_{i \in I_j} h_i} \right] + \gamma T_k \tag{40}$$

The optimal loss function is usually used as a score to measure the improvement or reduction in loss function by each split in the tree. This is used similar to impurity scores for regular decision trees or other algorithms. Since the tree generation is greedy, each node split increases the number of leaves by one. As such, given instance sets $I_P = I_L \cup I_R$ for the parent, left and right leaves respectively. The reduction of loss function or gain is calculated as follows:

$$gain = \frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g_i \right)^2}{\lambda' + \sum_{i \in I_L} h_i} + \frac{\left(\sum_{i \in I_R} g_i \right)^2}{\lambda' + \sum_{i \in I_R} h_i} - \frac{\left(\sum_{i \in I_P} g_i \right)^2}{\lambda' + \sum_{i \in I_P} h_i} \right] - \gamma$$

A positive gain would indicate an improvement by the splitting and the split is carried out. Otherwise, the parent leaf is left without splitting. To simplify the gain formula, the gamma regularization parameter can be redefined as $\frac{1}{2}\gamma_{new}$ and since, the objective is to check the sign of the gain, rather than the magnitude the $1/2$ coefficient is dropped entirely such that the gain equation is redefined as:

$$gain = \frac{\left(\sum_{i\in I_L} g_i\right)^2}{\lambda' + \sum_{i\in I_L} h_i} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\lambda' + \sum_{i\in I_R} h_i} - \frac{\left(\sum_{i\in I_P} g_i\right)^2}{\lambda' + \sum_{i\in I_P} h_i} - \gamma_{new} \tag{41}$$

Equation (41) and the γ_{new} parameter are reportedly more often used in practice because of convenience. Similar to other methods, XGBoost utilizes a small learning rate to further decrease overfitting and avoid unstable training or quick sub-optimal convergence. This improves diversity as it gives a chance for later trees to improve the model incrementally resulting in a diverse well-rounded model. Furthermore, the use and setting of the column (feature) subsampling technique is a useful hyperparameter in XGBoost. The idea is to simply subsample, that is, use a subset of the input features in each tree or node so that different trees are using different features, which improves the diversity and performance of the model (Chen & Guestrin, 2016; Cho & Chen, 2021).

Since the exact greedy algorithm evaluates the split of each feature at each possible splitting point, it is computationally expensive and inefficient when the entire set is too large to fit in the memory. XGBoost allows to use an approximately greedy algorithm that proposes candidate splitting points for each feature at different percentiles, either globally at the beginning of the tree construction or locally at each node. The percentiles are not evenly distributed but rather form a weighted quantile sketch. What that means is that each entry has a corresponding weight, and the different quantiles have similar weights rather than the same number of points. This method allows for small quantiles with smaller numbers of entries at points of a steep gradient. At the same time, it still takes into account the number of entries in the bins. This distributed weighted quantile sketch algorithm in XGBoost is, in fact, a novel one with a theoretically guaranteed efficiency. The use of the approximate greedy algorithm is reserved for large data sets where the exact algorithm would be inefficient (Chen & Guestrin, 2016).

Another simple yet powerful novelty in the XGBoost method is the sparsity-aware split finding. Sparse input means that the input values for a certain feature or features are missing or unknown for some entries in the data set. XGBoost, unlike most other gradient boosting algorithms, handles sparsity in inputs by defining a default direction at each split

for missing values in the input. The default direction at each split is learned from the data set by evaluating the gain equation (41) in assigning them to the direction with maximum gain. This handling of sparse data is not only versatile and useful in data manipulation but is also much faster than naive algorithms (Chen & Guestrin, 2016).

2.3.3 Preprocessing of Data

Feature engineering remains a key step in machine learning to realize the full potential of the different models. This includes categorical feature encoding especially, in features with high cardinality. The encoding methods can be divided into target-agnostic methods, and target statistics methods. The simplest form of encoding may be ordinal encoding which encodes the categories as an integer array and is particularly useful in ordinal categorical features, e.g., satisfaction level (Very satisfied (4), satisfied (3), unsatisfied (2), very unsatisfied (1)). However, it may also be used for nominal data encoding, however, it performs poorly with such nominal data (Géron, 2019).

One-hot encoding, sometimes called one-out-of-N encoding, is widely used encoding for categorical features of relatively low cardinality. This encoding method is done for a feature of N cardinality by creating an $N \times N$ matrix of binary attributes for each category so that only one category is 1 (hot), while others are 0 (cold) e.g., for $\{\text{Ankara, Istanbul, Izmir}\}$ set, the following matrix is encoded $\{[1,0,0];[0,1,0];\ [0,0,1]\}$. These attribute variables for each category are sometimes termed dummy variables. These attributes create vectors that are equidistant and orthogonal to one another. However, dirty data sets with different morphological representations of the same category will have different encoding and cannot be grouped together. Moreover, categories that are somewhat related to one another cannot be reflected in this encoding because all categories are equidistant. Furthermore, the matrix itself is rigid and it is impossible to add a new category that is not encoded initially. Finally, even if these problems are tackled, the inherent size of the feature matrix increases directly with the cardinality of the categorical feature and one-hot encoding becomes cumbersome and impractical at high cardinality (Cerda et al., 2018; Géron, 2019).

Target-based encoding methods attempt to encode the categories based on their effect or association with the target variable; this can be done using the impact, mean, or likelihood of the target variable. The simplest of target encoding for regression problems

is to encode a certain feature category as the mean of the target values for all observations in that category. In other words, if the category is used to predict the target value as the mean of observations of this category in the data set, this prediction is used to encode this category.

$$x_l = \frac{\sum_i^{N_l} y_i}{N_l}, \tag{42}$$

where i spans the observations that belong only to the category l, N_l are the number of all observations of l category in the training set, y_i are target variable values for each observation i in the training set (Cerda et al., 2018; Pargent et al., 2022).

Scaling of features can also be a useful pretreatment technique for all types of inputs, after encoding that is. Weight decay in neural networks regularization is an example of a situation that requires scaling since the different weights need to have the same scale to be penalized similarly. However, in most instances scaling is not usually required; scaling is done because it helps spread crowded points making the difference between them clearer. Scaling can be done by min-max scaling or by z-normalization also known as standardization. The min-max scaling can be done by subtracting the minimum value min from every feature value and dividing it by $(max - min)$. It could be subsequently multiplied by a new scale if the [0,1] interval is not desired, e.g., 100-scale; the scaling can be done as:

$$x_{i,scaled} = \frac{x_i - \min(\vec{x})}{\max(\vec{x}) - \min(\vec{x})} \times s_{new}, \tag{43}$$

where x_i, $x_{i,scaled}$ are the original and scaled feature value respectively and s_{new} is the new desired scale, which is conventionally left having a value of one (Geron, 2019). Alternatively, z-normalization results in features having a mean value of zero and unit variance. This is done by subtracting the mean from every feature value and dividing by the standard deviation.

$$x_{i,scaled} = \frac{x_i - \mu}{\sigma}, \tag{44}$$

where μ is the mean of the feature values and σ is the standard deviation of its values. This however does not put a limit on the maximum value of the feature which can be problematic for any models tuned for $|x_i| < 1$, but is excellent for dealing with outliers (Alpaydın, 2020; Geron, 2019).

2.4 Related Works

The customer LTV prediction problem, like similar business problems, is not identical in all settings. Indeed, as explained before (See section 2.1.1), the exact definition of the customer LTV varies slightly in different contexts and for different purposes, e.g., marketing as opposed to finance. The business model and industry seem to be the most important factors affecting the LTV calculation and modeling used. Among the different business types that have different approaches to the LTV prediction problem are, contractual and non-contractual businesses, online freemium businesses, which in turn may or may not be contractual, online retail businesses, online service businesses, B2B businesses, and offline businesses.

This varying nature of the problem must be taken into account when reviewing the literature, comparing different results, or even attempting to adapt a specific strategy or model. Here we classify the studies reviewed as freemium non-contractual, online retail, and others since these two classes are heavily studied and ML models for LTV predictions are heavily applied in these two types of business. The literature survey was carried out using the keywords “Customer Lifetime Value prediction”, “LTV prediction”, and “CLTV prediction” with and without the keyword “Machine learning” on the Web of Science, Scopus, and Google Scholar databases. The results were examined to ensure that they are customer LTV prediction studies that used machine learning and they were subsequently used as a seed using their citations to find more articles so that the survey would be as comprehensive as possible.

2.4.1 Freemium Non-contractual Businesses

Starting with freemium non-contractual businesses, which mostly involve free-to-play (F2P) games, one notices the prevailing characteristic of this model is that a small fraction of the users are paying premium customers who bring the entire business revenue. Among them, high-paying customers, e.g., the top 20%, finance most other customers. This makes identifying those customers through LTV predictions essential. This also means that the customer data is highly imbalanced, Sifa et al. (2018) used synthetic minority oversampling (SMOTE) to rectify the data imbalance in a large free-to-play game. A large error is observed because of the non-zero predictions for non-premium users with an overall NRMSE -RMSE in units of the average target value- of 21.8 for a multilayer perceptron (MLP) without SMOTE. This error is irrelevant to the

practical model performance since the target customer segment of premium and high-value users showed a much lower NRMSE of 2.90 and 1.48 respectively. By considering LTV ranked set, the top 25% of the set showed over 80% recall (hit rate). The false positive premium users are meant to be used for targeted acquisition to convert them into premium users. The most significant features include total purchase amount (M) and number of purchases (F) followed by in-game currency features with average importance from total playtime.

Chen et al. (2018) simulated the LTV values of a F2P mobile game using deep and convolution neural networks and compared them to probability models such as Pareto/NBD and MGB/CNBD-K. The neural network models performed considerably better than the parametric models with only 1.05 NRMSE from the convolution neural network followed closely by the MLP model. On the other hand, parametric probability models had NRMSE larger than 1.7. Furthermore, for the priority top spenders, the percent error is 15.6% while probability models had more than double that error for top spenders.

Similarly, Drachen et al. (2018), carried out a two-fold study on a mobile F2P game. They studied the effect of socialization behavior on the LTV of players and classified players as social and otherwise. They found no correlation between social behavior and the LTV value of the player. The LTV prediction problem was divided into two steps: predicting players' classes as premium or free players, and the regression analysis of the LTV of premium players (using either the best model for step 1 or simpler heuristics). Random forests proved most superior followed by XGBoost with NRMSE of 0.938 and 1.062, respectively. In classification, the random forest has an area under the ROC curve of 0.749 while XGBoost showed a slightly higher area of 0.785. Comparing this study to others proved difficult because the two steps were not combined into a single metric to determine the overall accuracy of regression of LTV on premium users including those misclassified as free users.

Tekin et al. (2021) used Fuzzy C Means Clustering to cluster players into groups and apply common tree-based non-parametric regressors such as XGboost, Catboost, and LightGBM (with different hyperparameters) to each cluster. The fuzzy nature of the clustering technique means that the objects or users can belong to multiple clusters. These intermediate clusters are treated as different clusters. The algorithm then chooses the best regressor for this cluster. This improved ensemble technique allows to avoid the problems of each regressor. Fuzzy clustering naturally produced significant improvements on its

component algorithms, XGboost, Catboost, and LightGBM, with an RMSE value of 4.18. It was followed by Catboost and LightGBM with RMSE values of 5.87 and 7.7, respectively. XGBoost unexpectedly performed the worst with an RMSE of 11.04 and was only optimum for a single cluster out of nine with a negligible margin over CatBoost.

Most recently, Zhao et al. (2022) developed a novel approach for LTV predictions using a unique algorithm, a variant of neural networks, utilizing individual and social behavior data inside the modeled F2P game to predict churn and payments made by players and combine them to predict the LTV value of individual players. Their algorithm, termed PerCLTV, produced superior results in a large study involving most traditional machine learning techniques like RF, XGBoost, neural networks: MLP, CNN, GNN, GCN, and ConvTrans, and others. The algorithm decreased the RMSE of payments regression by around 5-15% and improved the AUC for the churn prediction by 1-4% on different games data sets. When it comes to features, they argue that RFM indicators are less informative in the gaming setting because consumption of in-game purchases is not the customer goal but rather the gameplay itself which makes feature engineering quite important as opposed to highly expensive hand-crafted features.

2.4.2 Online Retail Businesses & E-commerce

Chen & Fan (2013) used multi-kernel support vector regression (MK-SVR) to model customer LTV dynamically by implementing the expected promotional response of customers in addition to the customer purchase behavior. This allows for direct machine learning-based promotional optimization in the same step to maximize potential customer LTV. In addition, the application of multi-step longitudinal data to study the temporal patterns is also notable. The MK-SVR performed best on the fictitious AdventureWorks data set, a manufacturer database with many data sets including sales and customers. However, it was second best after linear regression on the FoodMart data set, a proprietary data set of a retailer, with only minor improvements over the LR model; MAE of MK-SVR was 0.0664 compared to 0.0585 for LR. The promotional optimization was done only on the AW data set with MK-SVR performing best closely followed by LR. Overall, the use of MK-SVR produced good results that are comparable to linear regression, which begs the question regarding the trade-off between model complexity and performance improvement or gain.

A study of an LTV prediction for marketing purposes in a large online retailer, Groupon, is provided by Vanderveld et al. (2016). They explained the deployed LTV prediction approach used by Groupon and several other e-commerce companies for customer relationship management purposes. Most notably, they use a two-stage system of frequency classification, the first is a heuristic classification: "unactivated", new, one-time, sporadic, and power users. This is an easily interpretable classification as compared to clustering techniques used by others. The classification is followed by a customer LTV regression using a random forest model for each class of customers. In addition, certain warning-based updates are coded into the system to trigger updates or LTV recalculation of the customer. Furthermore, the LTV predictions are done for different time horizons of 3 to 12 months to better understand the customer potential. This creates a highly dynamic system that responds to customer behavior continuously. The system has a spearman correlation coefficient of 0.72 and NRMSE of 1.8 (medium period). However, the short time window predictions have a slightly higher error with a Spearman coefficient of 0.53 and NRMSE of 6.1 due to the inherent difficulty and noise associated with short-term purchase predictions.

Chamberlain et al. (2017), use a slightly different approach to the LTV regression problem that involves more heavy feature engineering before feeding features to their deployed RF regressor. Firstly, the system continuously, i.e., on a daily basis, updates the feature set using an automatic feature generator and customer embedding generator, followed by a 1-year data preprocessing step. The previous 1-year data, which is labeled, is fed to the random forest algorithm which ranks the customers according to their LTV providing their percentiles. This calibration step is followed by mapping the percentiles of customers to monetary value using a decision tree which is claimed to produce more accurate and robust results than direct regression. The RF Spearman correlation factor is 0.56 and 0.46 including and excluding zero-LTV customers respectively and a churn prediction AUROC curve of 0.798.

The feature engineering is done using neural networks; however, they predicted that a neural network that surpasses the RF model without overfitting would be much more computationally expensive to train on a daily basis. RF model is the chosen model for deployment for that retailer firm. Unfortunately, a comparison with direct regression over LTV data with and without feature engineering is not presented to better understand the extent and type of effects on the LTV prediction model performance. The periods of feature engineering and training are consecutive and disjoint to avoid information

leakage. The yearly patterns of data are supposed to capture the seasonality while being recent enough to be reliable for training. Finally, the model predictions are used as an input to other business systems that provide customer-tailored engagement, retention, and upselling strategies.

Bernat (2019), compared the commonly used probability model Pareto/NBD after modification with and without the gamma-gamma submodel, proportional hazard model and a non-parametric duration model, and gradient tree boosting machine learning model for predictions of LTV for an online Dutch fashion retailer. The Pareto/NBD+GG (gamma-gamma submodel) probabilistic model showed superior results to the machine learning gradient tree boosting algorithm with a slight decrease in the RMSE which is less than 1%. This is partly because the XGBoost training set included no inactive customers and created a model that struggles to make predictions for them once deployed. This could be improved by suitably augmenting the training data set. Furthermore, a comparison between using the squared errors (SE) function and fair loss function during training showed that the SE model produced superior results overall even though XGBoost trained using a fair loss function showed the lowest overall MAE. The fair loss function is a proxy to the absolute error function with continuous second derivative, which is necessary for XGBoost operation as explained before.

Another big online retailer study was carried out by Win & Bo (2020). They applied fine a tuned random forest learner and an AdaBoost learner on the public 4-year data set to classify the customers based on their LTV instead of directly predicting the numeric value. Similar to the ranking done by Chamberlain et al. (2017), the class of customer or his percentile is the focus of the prediction problem rather than the exact numeric value. The random forest showed slightly improved results showing a total accuracy of around 84.3%. Classification of the high-value customers correctly showed an F1 score of 86%. Revenue, shipping cost, frequency, profit, and recency are the ordered highest impact features used. Note that the first two features are the most important in defining the profit margin and hence the monetary value. However, using them separately showed greater impact which goes to emphasize the importance of feature engineering. Overall, the important features as in most retail settings are RFM (recency, frequency, monetary value). The order of importance varies, but these features and their derivatives summarize a wealth of information about customer purchase behavior.

Bauer & Jannach (2021) uses a novel deep learning architecture using recurrent neural networks (RNN) on a pair of data sets, one public, and the other proprietary. The

RNN model uses sequence-to-sequence learning by considering the LTV problem as a time-series forecasting problem and then aggregating the values of each time slot to produce the entire LTV of the customer. The method works by simply predicting each term in the time horizon sum in equation (1). This is combined with a GBM model trained on the same data without temporal evolution. The combination is done by GBM stacking. The hybrid machine learning algorithm showed only slight improvements over RNN (GRU) less than a 1% decrease in RMSE, followed closely by pure GBM models with a 2-3% decrease in RMSE on both sets. The GBM performance was better than RF-with embeddings and probability models Pareto/NBD and others. The industry proprietary data set showed lower RMSE values but similar trends.

2.4.3 Contractual and Other Businesses

Ekinci (2011) compared the performance of linear regression to a 4-layer MLP model for customer LTV prediction for a financial services firm and claimed that the LR model showed slightly better results based on the value of R^2 of 0.441 compared to 0.425 of the MLP. This claim is not entirely reliable because the R^2 metric does not reflect a complete picture of the performance of regressors. The MLP model hyperparameters were not clearly stated; however, optimization of the network may have offered superior results. Furthermore, better optimizers have been developed since 2011 such as Adam optimizers which could skew the comparison results in favor of ANN even further. The biggest advantage of LR is its interpretability which was used heavily in that study to discuss the different feature effects on the customer LTV which included, monetary value and profit margins. This is considerably more difficult to infer from the MLP hidden layers weights.

Tsai et al. (2013) created a binary classification hybrid model for a stainless-steel pipe manufacturer classifying customers as high-value (top 20%) or low-value customers using hybrid classification + classification or clustering + classification schemes, where the first step acts as a preprocessing filtering step. The main idea of this hybrid machine learning technique is to remove unrepresentative data from the data set that could not be well categorized or data that formed a cluster which has no clear class. Such data are removed as unrepresentative which can increase model robustness. The hybrid double decision trees classifier performed quite well with 99.73% accuracy as compared to the baseline single DT, which had an accuracy of 96.74%. Such high accuracy from single

and double decision trees could be due to the inherent oversimplicity of the classification problem; that is the data contains enough information that allows accurate predictions. The fact that the company presumably operates as a B2B may also suggest consistent customer profiles, which are easier to predict.

From B2B businesses to contractual businesses, there are significant differences. In contractual settings, churn prediction represents the most important aspect of LTV. Prasasti et al. (2014) carried out a contractual 1-year based LTV prediction study for a cloud-based software company with a yearly subscription. Hence, the churn prediction for the following year is a binary classification problem which was carried out using an SVM and C4.5 decision tree classifiers with mixed results from the two classifiers over different product classes with ~78% accuracy and ~84.5% F1 score. Because the risk is inserted into the LTV prediction formula, the following year's contribution is less than half of the current year as churn risks appear to be on average larger than 0.5.

Wang et al. (2008) also carried out LTV predictions for a telecommunication firm in a contractual setting using MLP. They reported an SSE of 0.0798 with a 10-fold cross validation, for which a rotating 1205 records were sampled for each validation set. However, classification metrics were used in the analysis of a regression problem after being poorly redefined and were subsequently miscalculated. These miscalculations and the SSE reporting made comparison to more reliable studies challenging. Further, there may have been cherry picking of data in cross validation. This study is cited here for the sake of completeness.

In Table 2.2, the discussed works are summarized in reverse chronological order so that the most recent relevant works are presented first. Note that the RFM indicated in the prediction models is the RFM model while the RFM in the important features column refers to the features themselves, i.e., recency, monetary value, and frequency or related features such as revenue or total profit or number of items per order and others. Finally, note that older studies or ones that used purely probabilistic or econometric methods without machine learning were not included in this review since such models do not apply well to the problem of interest in this study. For instance, see Abe (2009) and Glady et al. (2009).

Table 2.2: Summary of CLTV prediction studies that involve machine learning predictors; evaluation metrics values are reported for "chosen methods" only. If multiple values exist for one metric, they are for multiple data sets. Data set order is the same for each metric.

Author	Prediction Models	Chosen/ Optimal Method(s)	Evaluation Metric(s)	Metric Values	Important Features	Firm Industry
Zhao et al. (2022)	RFM, Pareto/NBD, Cox proportional hazard model (PHM), Random Forest (RF), XGBoost, multilayer perceptron (MLP), Convolutional Neural Network (CNN), Temporal Convolutional Networks (TCN), Transformer, Convolutional Transformer (ConvTrans), Graph Neural Networks (GNN), Graph Convolutional Network (GCN) and PerCLTV (their own model) w & w/o multi-task learner (MTL) & others	PerCLTV	RMSE	91	—	F2P Game
				26		
			AUC (churn)	**93.8**		
				96.8		
		PerCLTV w/o MTL	RMSE	**91**		
				28		
			AUC (churn)	92.1		
				95.6		
Tekin et al. (2021)	XGBoost, CatBoost, Light GradBoost Fuzzy clustering	Fuzzy clustering	RMSE	4.18	—	F2P Game

Author	Prediction Models	Chosen/ Optimal Method(s)	Evaluation Metric(s)	Metric Values	Important Features	Firm Industry
Bauer & Jannach (2021)	NBD, Markov chains, RF, Recurrent Neural Network (RNN), Gradient Boost Machine (GBM), hybrid (RNN+GBM)	Hybrid	RMSE	69.3	RFM, brand purchased	Retailer
				14.0		
			MAE	31.9		
				3.9		
Win & Bo (2020)	RF, Adaboost	RF	Accuracy	84.3%	Revenue, Shipping cost, Frequency	Retailer
			F1 score	86%		
Bernat (2019)	Pareto/NBD, Cox proportional hazard function (PHM), Gradient tree boosting (GBM)	Pareto/NBD+ GG	RMSE	**191.6**	RFM, items returned	Retailer
			MAE	**76.3**		
		GBM	RMSE	193.1		
			MAE	80.6		
Sien Chen (2018)	XGBoost	XGBoost	RMSE	13	Advance booking time	Airliners
			R^2	0.394		
Drachen et al. (2018)	RF, XGBoost (Class &Regression), AdaBoost & C5.0 trees (classification only)	RF	AUC	74.9%	Current absence time, total days played,	F2P Game
			AUPR	**15.4%**		
			NRMSE	**0.938**		
			R^2	**0.097**		

Author	Prediction Models	Chosen/ Optimal Method(s)	Evaluation Metric(s)	Metric Values	Important Features	Firm Industry
		XGBoost	AUC	**78.5%**	Avg moves per round	
			AUPR	12.1%		
			NRMSE	1.062		
			R^2	0.096		
Chen et al. (2018)	Pareto/NBD, deep neural network (MLP), Convolution neural network (CNN)	CNN	NRMSE	**1.05**	—	F2P Game
			%E-top spenders	**15.64%**		
		MLP	NRMSE	1.12		
			%E-top spenders	15.76%		
Sifa et al. (2018)	RF, Linear Reg (LR), Decision Tree (DT), MLP (all w & w/o SMOTE)	MLP	NRMSE-Total	21.8	RFM-related, Game currency, total playtime	F2P Game
			NRMSE-Premium	2.9		
			NRMSE-High Value	1.48		
			For top 25%, Recall	81.1%		
Chamberlain et al. (2017)	RF, (neural network for features engineering)	RF	Spearman Corr.	0.56	RFM, web/app session logs (online engagement)	Retailer
			Spearman Corr. (non-zero users)	0.46		
			AUC (churn)	79.8%		

Author	Prediction Models	Chosen/ Optimal Method(s)	Evaluation Metric(s)	Metric Values	Important Features	Firm Industry
Vanderveld et al. (2016)	RF (short S and long L time horizons)	RF	Pearson Corr.-S	0.45	online engagement	Retailer
			Spearman Corr.-S	0.53		
			NRMSE-S	6.08		
			Pearson Corr.-L	0.73		
			Spearman Corr.-L	0.77		
			NRMSE-L	1.67		
Prasasti et al. (2014)	support vector machine (SVM), C4.5 DT	SVM	Accuracy	72.7-82.9%	Total renewal count, pricing, optional service use	Contractual Cloud-based software firm
			F1 Score	79.4-87.5%		
		C4.5 DT	Accuracy	72.1-82.9%		
			F1 Score	78.7-87.0%		
Tsai et al. (2013)	DT, Logistic regression (LR), MLP classifiers, k-means, and SOM clustering algorithms	DT+DT hybrid model	Accuracy	99.73%	RFM	SS-pipes manufacturer
Chen & Fan (2013)	multikernel support vector regressor (MK-SVR), SVR, LR, MLP, radial	MK-SVR	MAE	0.0664	RFM, demographics	Retail
				0.0993		

Author	Prediction Models	Chosen/ Optimal Method(s)	Evaluation Metric(s)	Metric Values	Important Features	Firm Industry
	basis forward neural network (RBFNN), DT	LR	MAE	**0.0585** 0.1426		
Yeliz Ekinci (2011)	MLP, LR	LR	R^2	0.441	Monetary value, profit margin	Banking
Wang et al. (2008)	MLP	MLP	SSE (1205 points)	0.0798	—	Telecommunication

Chapter 3:
COMPUTATIONAL EXPERIMENTS

In this chapter, we explain the development and evaluation process of our machine learning model as well as the later processes of deployment, monitoring, and retraining. We first explain the data set at hand and the validation scheme in detail. After that, we detail the model development process, which we performed in several steps, and in each step, we evaluated different alternatives, selected the optimal alternative, and used it in the following steps. The model development steps include selecting the machine learning algorithm and feature encoding, selecting the time window of the training set, deciding on whether to break the data set into multiple parts and train a model for each part, and selecting the optimal hyperparameters for the model. After that, we examine the performance of the model more closely, including its performance on different sections of the data and the features importance in the model. After that, we explain the deployment, feature calculation in the live system, monitoring, and retraining processes. We then examine the live or post-release model performance.

3.1 Preliminaries

3.1.1 Prediction Problem Specifications

Our problem is to predict the LTV that will be generated by a customer at the time the customer requests a service from Armut's platform. We predict the LTV for two types of customers, acquired and resurrected. If the customer requests a business model 1 (BM1) service, we have to make the prediction at the time he submits his service request to Armut. If the customer requests a business model 2 (BM2) service, we have to make the prediction after the customer specifies the details of his request and before he makes the payment. This is an important detail because, at the time of prediction for BM2 requests, we do not know if the customer will follow through with the request and complete the payment or not. For more information on Armut's business models, see section 2.2.1.

As we mentioned in section 2.2 the LTV of a customer in Armut keeps accumulating until he churns, with churning defined as not requesting services for 28 consecutive days. This means that to observe the final value of our target variable (LTV) for some of our

records, we have to potentially wait for years until the customer stops requesting services. On the other hand, most customers' lifetimes end within a month. Because we need to train our model on recent data, we decided to use one-month LTV as a proxy for LTV for customers who have not yet churned at the time of prediction. This allows us to train the model on data up to one month before real-time for the cost of underestimating the LTV of a minority of customers.

3.1.2 Data Set

Our data set contains features representing the information available at the time customers made their service request, i.e., started their lifetime, and the corresponding LTV they generated over their lifetime until they churned. The data set contains millions of records spanning the years 2020 and 2021. Approximately 50% of the customers in the data set are acquired and 50% are resurrected and around 92% of the service requests are BM1 requests and 8% are BM2. Around a quarter of the target LTV values are zero, meaning that we do not suffer from the high imbalance problem seen in some other industries like freemium games (Sifa et al., 2018).

Table 3.1: Summary of input features, including the feature name, type, number of categories for categorical features, and a short description.

Field	**Type**	**Number of Categories**	**Description**
Call Preference	Categorical	3	Whether the service provider can call the customer after sending an offer
Estimated Price	Numerical	-	A moving-average prediction of the price of the job requested (the amount that the customer will pay the service provider)
Supply-Demand Ratio	Numerical	-	The ratio of price quotes to service requests in the customer's province and the customer's service in the past four weeks
Fulfillment Rate	Numerical	-	Number of fulfilled requests / number of all requests, in the customer's

			province and the customer's service in the past four weeks
Customer Status	Categorical	2	Whether the customer is new or resurrected
Job Description Word Count	Numerical	-	Number of words in the job description
Job Start Timeframe	Categorical	4	Whether the job is wanted soon or further in the future
Requested Service	Categorical	>1000	ID of the requested service
Business Model	Categorical	2	Business model of the requested service
Province	Categorical	82	ID of the province in which the customer is making the request
Subprovince	Categorical	>100	ID of the subprovince in which the customer is making the request
BM2 Subscription Frequency	Categorical	6	If the requested service is a BM 2 service, is the service for only one time or every week, 2 weeks, or 3 weeks…
RFM Recency	Numerical	-	For resurrected customers, Number of days since the customer's last service request if he requested a service in the past year
RFM Frequency	Numerical	-	For resurrected customers, Number of customer's service requests within the past 12 months
RFM Monetary	Numerical	-	For resurrected customers, Sum of past year's LTV gained from the customer's service requests

Our input features are summarized in Table 3.1. They can be categorized into four groups, namely, features related to the customer's current service request, features related

to the customer and his preferences, features related to the customer's transaction history if the customer is resurrected, and finally features we call market status features.

The service request features contain the requested service, estimated price, job start timeframe, job description word count, business model, and BM2 service frequency. The requested service is the ID of the service requested. Armut offers services that range from home remodeling to gardening to private lessons and many in between. The estimated price is a moving average based estimate of the quote value that the customer will receive from the service providers, this is an important feature since the quote fee that Armut charges the service providers is correlated with the estimated price. The job start timeframe explains the urgency of the job. It can be one of four values, namely, within 3 weeks, within 2 months, within 6 months, and just looking for prices. The job description word count is the number of words that the customer wrote in the free text area when asked to describe the requested service. Business model is whether the service belongs to business model 1 or 2. BM2 service frequency is the subscription frequency if the customer is subscribing to a BM2 service, i.e., whether he wants the service every week, 2 weeks, or 3 weeks and so on.

The customer and preference features are call preference, province, subprovince, and customer status. The call preference feature can have one of three values, quotes can call me, quoters can call me without seeing my number, and quoters cannot call me. Province and subprovince specify the area where the customer wants the service. Customer status is whether the customer is acquired or resurrected at the time of requesting the service.

The customer history features are recency, frequency, and monetary value. They are calculated from the past 12 months' transactions of the customer before the current service request. Recency is the number of days passed since the last service request. Frequency is the number of service requests the customer made over the past year. Monetary is the sum of the customer's past year's LTV.

The market status features are the fulfillment rate and the supply-demand ratio. They describe the state of the market in the service the customer requested and the location he requested it for. Fulfillment rate is the ratio of fulfilled requests to all requests of the customer's service in the customer's province in the past four weeks. Supply-demand ratio is the ratio of quotes/claims to service requests of the customer's service in the customer's province in the past four weeks.

Table 3.2: Correlation between input features and the target variable (LTV).

Field	Type	Pearson r (Numerical) / Correlation Ratio (Categorical)
Requested Service	Categorical	0.4
Subprovince	Categorical	0.19
Province	Categorical	0.16
Job Start Timeframe	Categorical	0.12
Customer Status	Categorical	0.1
Call Preference	Categorical	0.06
Business Model	Categorical	0.05
BM2 Subscription Frequency	Categorical	0.05
Supply-Demand Ratio	Numerical	0.26
Fulfillment Rate	Numerical	0.22
RFM Monetary	Numerical	0.16
RFM Frequency	Numerical	0.12
Job Description Word Count	Numerical	0.11
RFM Recency	Numerical	-0.09
Estimated Price	Numerical	0

We calculated the correlation between input features and our target variable to gain some insight into our data before modeling. Using Python's Sweetviz library, we calculated Pearson correlation for numerical features and correlation ratio (η), the square root of the effect size (η^2) in ANOVA, for categorical features (Tabachnick et al., 2019).

We see in Table 3.2 the LTV is highly correlated with the requested service and to a lesser extent with the customer's location (province and subprovince). That is because in some services and locations Armut has numerous registered service providers who can fulfill the customer's request and satisfy the customer, earning money for themselves and generating revenue for Armut in the process. In other areas and services, there are few or no service providers so the customer may not have his request fulfilled.

We also notice that the subscription frequency for business model 2 services is not highly correlated with LTV. That is because as we mentioned in section 3.1.1, the prediction is made before the payment for business model 2 and the training data contains a lot of examples where the customer selected a recurring subscription but then did not complete the payment process and the request's LTV became zero.

If we look at the numerical variables, we see that the LTV is correlated with the market status features, namely, the supply-demand ratio and the fulfillment rate. That is because the more quotes the service providers make in an area and a service, the more likely the current customer is to receive quotes when he requests the same service in the same area.

We can see on the other hand that the estimated price of a service request does not correlate with the LTV generated from that request. That is because the estimated price of a service request (a proxy for how big or small that service request is) does not directly correlate with whether this request will get quotes and get fulfilled or not, but rather it has to do with whether there are suitable service providers for that request in the customer's area. Although the feature is not correlated with LTV but that does not mean it is not useful as it may give helpful information to the predictive models when combined with other features.

Table 3.3: Datapoint concentration across different dimensions.

Dimension	**Gini Index**
Requested Service	0.885
Province	0.875

We would also like to touch on the fact that the service requests are not equally distributed between different services and locations. In Table 3.3 we calculated the Gini index (Sen & Foster, 1997) for the distribution of service requests in our data set among different services and provinces. Gini index is a measure of inequality where the value of zero means all units are distributed equally and the value of one means that all units belong to one entity and other entities do not have any units, in other words, extreme inequality. We can see from the high Gini values that the majority of service requests are requests of few services and are in few provinces, with the rest of the services and provinces having fewer datapoints/service requests.

3.1.3 Validation Strategy

The aim of model validation is to give us an estimate of the model's performance on never-before-seen real-world data, and for the validation process to give us the most accurate performance estimation, the relationship between the training and validation data should be as close as possible to the relationship between the training and test data (i.e., live data after deployment). In our case, the model will be trained on a training set,

then deployed and used to predict LTV for live data that forms at a later point in time than the data in the training set. We need to preserve this temporal relationship when validating our model, for the validation to give us a good estimate of real-world performance. Therefore, we did not use k-fold cross-validation instead, we used time-series cross-validation (Hyndman & Athanasopoulos, 2021).

Our implementation of time-series cross-validation is as shown in Figure 3.1. We use a fixed training set time window, and we leave one month between the training and validation set because this will be the case after deployment. We do not use the final month's data because customer LTV, our target variable, accumulates during that period.

To validate a model, i.e., calculate the performance metric for that model, we train a separate instance of the model with the same hyperparameters on the training set of each fold, then make a prediction for the respective validation set, then calculate the performance metric, NRMSE in our case, of each fold separately. Then take the average of the performance on the folds. The fact that NRMSE is normalized and scale-free makes it useful when comparing results across different problems. It is also useful for concealing the scale of sensitive data.

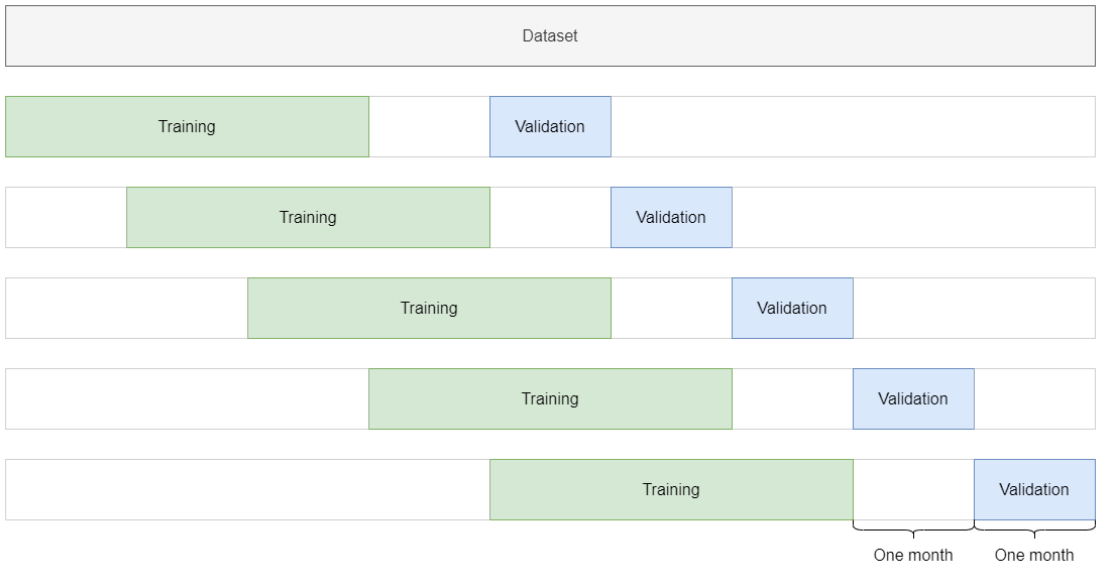


Figure 3.1 Our implementation of time-series cross-validation. A rolling training set with a fixed time window, followed by a one-month break for LTV accumulation, then a one-month validation set.

We calculate NRMSE as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}} \quad (45)$$

$$NRMSE = RMSE/\bar{y}, \quad (46)$$

where $NRMSE$ is the normalized root mean square error of a fold, N is the number of data points in the fold, y_i is the true target value of point i, $\hat{y}_i$ is the model prediction for point i, $\bar{y}$ is the average of the target values in the fold.

3.1.4 Development Environment

All the work presented here was carried out in a Python environment with Python version 3.9. The Python packages used and their versions are shown in Table 3.4.

Table 3.4: Versions of Python libraries used in this study.

Python Package	**Version**
scikit-learn	1.0
SciPy	1.7
XGBoost	1.5
PyTorch	1.11
Category Encoders	2.4
Plotly	5.7
Flask	2.1
Sweetviz	4.1

3.2 Model Development

Different ML algorithms' performance vary on different problem data sets, depending on the nature of the relationship between the input features and the target variable. For example, if the relationship is mostly linear, a linear model will produce good results, otherwise, a non-linear model will be needed to effectively model the data. Similarly, if the relationship between input features and the output is clear, simple models can produce good results, otherwise, more complex models will be needed for effectively modeling the data.

To find the ML algorithm most suitable for our problem's data, we tested different algorithms that have different levels of complexity and different learning techniques. We

also presented the data to the models in different ways (encodings) to get the most out of each model. We optimized the most important hyperparameters for the models on each feature encoding case. We did not try using the pareto/NBD model because it is not applicable for half of our customers since they are new and do not have a purchase history.

In this section, we chose the time window of the training set in each validation fold to be three months. Here we will first explain the encoding methods we used and then move on to explain the machine learning algorithms we used and their results.

3.2.1 Feature Encoding Descriptions

Table 3.5 contains the summary of the encoding cases, further implementation details are stated below. The low cardinality mentioned in the header of the table means categorical variables that contain less than ten categories.

Table 3.5: Summary of used encoding cases.

Encoding Case Name	**Numerical Feature Processing**	**Low Cardinality Categorical Feature Processing**	**High Cardinality Categorical Feature Processing**
Ordinal Encoding	None	Ordinal Encoding	Ordinal Encoding
One-hot Encoding	None	One-hot Encoding	One-hot Encoding
Target Encoding	None	Target Encoding	Target Encoding
One-hot & Target Encoding	None	One-hot Encoding	Target Encoding
One-hot & Ordinal Encoding	None	One-hot Encoding	Ordinal Encoding

- Ordinal Encoding: When encoding a categorical feature, we pool all categories that have 30 instances or less into an “others” category. Categories not encountered in the training set are also mapped to this others category. We created our own wrapper over scikit-learn’s OrdinalEncoder object to map the others category to the value 0 and map the actual categories starting from 1.
- One-hot Encoding: Similarly, to ordinal encoding, we pooled the categories that have 30 or less instances into an others category. In one-hot encoding, this has the additional benefit of reducing the size of the encoded matrix, as out of memory

problems are common with one-hot encoded data (See section 2.3.5). Scikit-learn's OneHotEncoder object was used for encoding. The data is converted to SciPy's sparse csr_matrix before feeding it to Linear Regression, Decision Tree, and XGBoost models. The data was not converted to sparse format for the neural network.

- Target Encoding: TargetEncoder object from Category Encoders package was used for target encoding. The parameter min_samples_leaf was set to 30 so categories that have less than 30 instances in the training set get assigned the entire data set's target mean instead of their own target mean.
- One-hot & Target Encoding: In some cases, we used one-hot encoding for categorical variables with less than ten categories and target encoding for variables with more than ten categories.
- One-hot & Ordinal Encoding: We used one-hot encoding for categorical variables with less than ten categories and ordinal encoding for variables with more than ten categories. This encoding method was made for the neural network where ordinally encoded variables were later converted to embeddings.

3.2.2 Machine Learning Algorithms

Here we present the ML algorithms we used in our experiments and their results on each encoding case. We wanted to add KNN model to our experiments, however we found that its inference/prediction speed is prohibitively slow, so we did not include it in our study. That is also possibly the reason why it is also not used in the literature.

For each ML algorithm and feature encoding case, we reported the average NRMSE of the five validation folds, as well as NRMSE range / average. This second metric is observed as an indicator of the model's instability or sensitivity to the data.

Linear Regression

Linear regression is a simple, interpretable, and fast-to-train machine learning model that performs well on linearly correlated data and serves as a good baseline for ML problems. We trained an Ordinary Least Squares (OLS) multiple linear regression model using the LinearRegression object from scikit-learn library. We did not process/scale the numerical features before feeding them into the model because the model is indifferent to feature scale, however, we tried three different encoding methods for categorical features. They are summarized in Table 3.6.

Table 3.6: Linear regression model performance with different encodings.

Model	Categorical Encoding	5-Fold NRMSE Average	NRMSE Range / Average
Linear Regression	One-hot Encoding	**1.77600**	0.19702
Linear Regression	Target Encoding	1.78145	0.20333
Linear Regression	One-hot & Target Encoding	1.78677	0.19963

In Table 3.6, we see that the linear regression model gave the best performance with the one-hot encoded features. This is possibly because one-hot encoding gives the model the freedom to weigh each category in the way that minimizes error, while target encoding specifies the exact values for each category that will be multiplied by the model's parameter.

Decision Tree

Decision tree models are simple yet effective non-linear ML models that are also useful for data exploration. We trained a decision tree model using the DecisionTreeRegressor object from scikit-learn library. We did not process/scale the numerical features before feeding them into the model because the model is indifferent to the feature scale, however, we tried four different encoding methods for categorical features. They are summarized in Table 3.7.

Table 3.7: Decision tree model performance with different encodings.

Model	Categorical Encoding	5-Fold NRMSE Average	NRMSE Range / Average
Decision Tree	Ordinal Encoding	1.82417	0.16336
Decision Tree	One-hot Encoding	1.83370	0.16905
Decision Tree	Target Encoding	**1.75639**	0.19280
Decision Tree	One-hot & Target Encoding	1.75674	0.19154

We optimized the min_samples_split hyperparameter for each of the encodings. The parameter sets the minimum number of samples in a node to be considered for splitting. If a node has less than that number of samples, it is not split any further. This hyperparameter can be set to a specific number or it can be set to a specific ratio of the training set size. Tested different ratios and found that 0.003 gives the optimal result for

all four encoding cases. The rest of the hyperparameters were left with their default values. Namely, criterion parameter is set to squared_error and max_depth is set to None.

In Table 3.7, we see that the model gave its best performance with the target encoded features. This is possibly because target encoding orders the categories of each variable in a way that minimizes or rather reduces the number of splits required to minimize the prediction error. That is because the encoded values of the categories are directly correlated with the target variable.

XGBoost

XGBoost is a powerful and efficient ML algorithm that frequently produces state-of-the-are results on tabular data (Grinsztajn et al., 2022). We trained an XGBoost model using the XGBoost Learning API, as opposed to XGBoost Scikit-Learn API. We did not process/scale the numerical features before feeding them into the model because the model is indifferent to the feature scale, however, we tried four different encoding methods for categorical features. They are summarized in Table 3.8.

Table 3.8: XGBoost model performance with different encodings.

Model	**Categorical Encoding**	**5-Fold NRMSE Average**	**NRMSE Range / Average**
XGBoost	Ordinal Encoding	1.76875	0.19332
XGBoost	One-hot Encoding	1.76221	0.18932
XGBoost	Target Encoding	**1.74603**	0.20665
XGBoost	One-hot & Target Encoding	1.75104	0.19777

We optimized the num_boost_round hyperparameter for each of the encodings. The hyperparameter sets the number of boosting iterations during training. We found the optimal values shown in Table 3.9 below.

Table 3.9: Optimal number of boosting rounds for each XGBoost model.

Feature Encoding	**Optimal num_boost_round**
Ordinal	118
One-hot Encoding	120
Target Encoding	21
One-hot & Target Encoding	11

Other hyperparameters were left at their default values. Namely, objective is set to reg:squarederror with eval_metric as rmse, learning_rate is set to 0.3, max_depth is set to 6, subsample and colsample_by* parameters are set to 1, lambda is 1 and alpha is 0.

As with decision trees, we see that the XGBoost model worked best with the target encoded variables. This may indicate that tree-based models generally work better with target encoding than with other types of categorical encoding.

Neural Network

Neural networks are powerful algorithms that can effectively approximate any kind of relationship between the input and target variables (Goodfellow et al., 2016). Our trained networks used the following hyperparameters. Adam optimizer, 0.001 learning rate, 256 batch size, MSE loss, 100 max epochs. We tried three different encoding methods with the neural network, the results of each of them are shown in Table 3.10.

Table 3.10: Neural network model performance with different encodings.

Model	**Categorical Encoding**	**5-Fold NRMSE Average**	**NRMSE Range / Average**
Neural Network	Target Encoding	1.74477	0.18362
Neural Network	One-hot Encoding	**1.73983**	0.18965
Neural Network	One-hot & Ordinal Encoding (Embedding)	1.74480	0.20904

For one-hot and target encoded cases, we used neural networks with the four hidden layers. The number of nodes in the first layer of those networks is calculated as

$$n_k = \max\left(\left\lfloor\frac{n_{input}}{2}\right\rfloor, 50\right), \tag{47}$$

where $k = 1$, and the number of nodes in subsequent hidden layers can be calculated as

$$n_k = \left\lfloor\frac{n_{k-1}}{2}\right\rfloor, \tag{48}$$

where $k \in \{2, 3, 4\}$. Each hidden layer uses ReLU as the activation function.

For the one-hot and ordinal encoded case, we created a neural network that has two components, the embedding layers and the feed-forward network. There are three "parallel" embedding layers for the three high cardinality categorical variables, namely, Service ID, Province, Subprovince. In the forward propagation step, each of those variables is converted to an embedding vector. The vectors are then concatenated with each other and with the rest of the features and fed to a feedforward network (Figure 3.2). The length of the embedding vector e is for the ith categorical variable is calculated as,

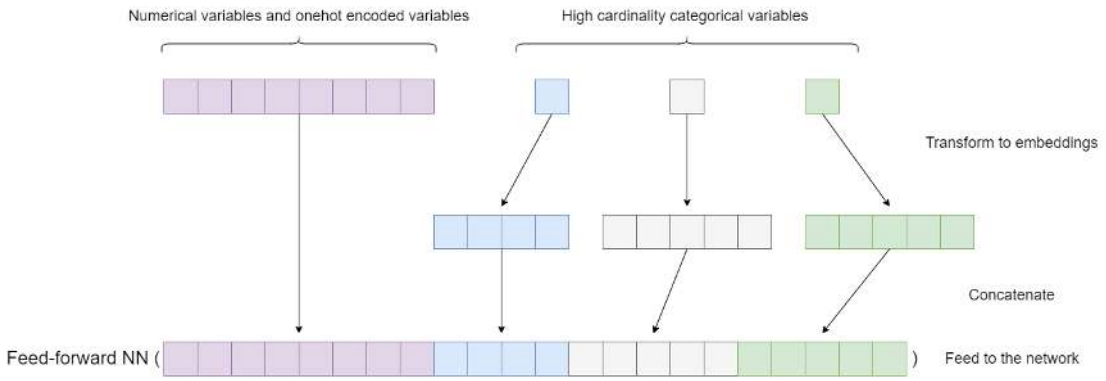


Figure 3.2 The architecture of the neural network with embedding layers.

$$e_i = \min\left(\left\lfloor \frac{c_i}{2} \right\rfloor, 50\right), \tag{49}$$

where c_i is the number of categories of the ith categorical variable.

The Feed-forward NN has the same structure as the one detailed above in the other encoding cases. For each of the three neural networks, the number of training epochs was optimized. The optimal number of epochs is shown in Table 3.11 below.

Table 3.11: Optimal number of epochs for each neural network.

Feature Encoding	**Optimal number of epochs**
One-hot Encoding	26
Target Encoding	29
One-hot & Ordinal Encoding	16

We tried scaling numerical features, but it did not make a difference in the prediction performance. The reported results are without numerical scaling. Although the embedding network took the most of effort to implement, it did not provide better results. Although the network with one-hot inputs gave the best performance, it had the disadvantage of being the largest and most memory requiring network.

Comparing the Different Models

Table 3.12: Results of encoding and ML algorithm experiments.

Model	Categorical Encoding	5-Fold NRMSE Average	NRMSE Range / Average
Linear Regression	One-hot Encoding	1.77600	0.19702
Decision Tree	Target Encoding	1.75639	0.19280
XGBoost	Target Encoding	1.74603	0.20665
Neural Network	One-hot Encoding	**1.73983**	0.18965

As we can see in Table 3.12, although more complex models like the neural network and XGBoost gave better results than simple models like decision trees on our data, the difference was small, suggesting that the amount of information in our data that could be utilized by a more complex model is limited. Simpler models have the advantage of being easily interpretable and explainable, so depending on the business case, they may be preferred to more complex models, even if the complex models have a better performance.

While neural networks gave the best results on our data, we decided to continue with XGBoost because neural networks are more difficult to maintain and debug when used in production, need experienced engineers for maintenance and modification, are slower to train, and require more powerful hardware. Also, because the gains of the neural network over the XGBoost model are not sufficiently high to justify its burden.

3.2.3 Training Set Time Window Optimization

After we decided to continue with the XGBoost model with the target encoded features, we tested expanding the time window of the training set as we did the previous experiments with training sets spanning 3 months each. We tested changing this time window to span 12 months for the data to contain instances from all seasons. The results are shown in Table 3.13.

Table 3.13: Results of training set time window expansion experiment.

Model	Feature Encoding	Training set time window	5-Fold NRMSE Average	NRMSE Range / Average
XGBoost	Target Encoding	3 months	1.74603	0.20665
XGBoost	Target Encoding	6 months	1.75114	0.19931
XGBoost	Target Encoding	12 months	**1.73397**	0.19906

In the table we see that the model trained on 12 months data gave the best performance. We attribute that to the seasonality factor. A lot of services have a seasonal trend, for example, the many people request the boiler repair service at the start of the winter. A lot of students request private lesson service before exams, and similar trends exist for other services. If the model is making predictions for the service requests in a specific month this year, it would help if the model saw training examples from the same month last year. Because they tend to be similar because of seasonality.

We also see that the model trained on 6-month data performed worse the 3-month model. That is because the 6-month model contains more outdated data points compared to the 3-month model. At the same time the data is from a different season compared to the data the model is trying to predict. The 12-month model also contains outdated data, but the fact that it contains similar data to the ones we are trying to predict counterbalances that disadvantage. In the future works section, we will discuss methods that can decrease the effect of outdated data.

3.2.4 Data Set and Model Partitioning

After finding the optimal training set time window for our model, we looked for other ways to further increase the performance of our model. Since we have groups of data with different characteristics in our data set, we thought it might be beneficial to train separate models for these groups. That way, models can use group specific features more effectively. This method is heavily utilized in the literature (Tekin et al., 2021; Tsai et al., 2013; Vanderveld et al., 2016).

We have two dimensions which split the data into distinctive groups. The first is the business model dimension, where BM1 service fulfillment and revenue generation processes are different from BM2. We also have a BM2 specific feature which is the subscription frequency, where the subscription option is only available for BM2 services.

The other dimension that divides our data is the customer status where acquired customers are never-before-seen customers whereas resurrected customers are customer who have a history with Armut. We also have features related to customer history in the case of resurrected customers.

Table 3.14: Results of the model partitioning experiment.

Model	**Feature Encoding**	**Training set time window**	**Partition**	**5-Fold NRMSE Average**	**NRMSE Range / Average**
XGBoost	Target Encoding	12 months	No partition	1.73397	0.19906
XGBoost	Target Encoding	12 months	Business model	**1.72468**	0.19113
XGBoost	Target Encoding	12 months	Customer status	1.73192	0.18873

We created the partitioned models by splitting the training set the different partitions, training a model separately on each partition, and optimizing the number of boosting rounds for each model independently. To calculate the validation performance, for each validation fold, we split the validation data into the different partitions, predicted the target value of each partition with the corresponding model, combined the partitions again to get the full fold, and finally calculated NRMSE for the fold.

In Table 3.14 we can see that splitting the data across the customer status dimension does not significantly improve the model's performance. That is possibly because in Armut's case, acquired and resurrected customers are not different enough from each other that they would benefit from having separate models.

Conversely, we see that splitting the data across the business model dimension increased the performance. That is because the BM1 and BM2 revenues are affected by different factors and splitting the models gave each model the chance to focus on the factors and features that affect the target value the most for each case.

We conclude that having separate models for different data sections with different patterns increases the overall performance because each model can focus on learning one pattern. However, the question remains open as to whether the performance would have increased if the model was a neural network instead of XGBoost. Since the neural networks are universal approximators, they are expected to fit any kind of data, at least

in theory, but whether practical limitations such as the number of training examples and the size of the network limit its learning ability so that having separate models would improve performance is still not known, in the context of our study.

3.3 Model Prediction and Performance Analysis

After developing our model, we here compare its performance with several baselines to gain insight into the magnitude of improvement it brings. We also examine the distribution of the predictions with respect to the actual values to see the distribution of the model's errors. We also examine model's performance of different sections of the data and reason about the performance differences between these sections. We finally look into the feature importance for BM1 and BM2 models and examine which features are important for each model.

3.3.1 Baseline Comparison

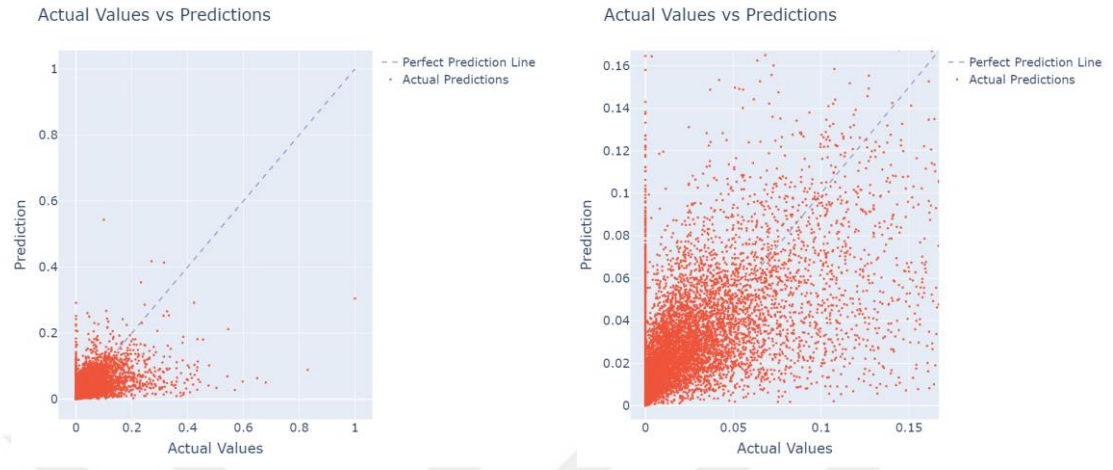
We calculated the performance of several baselines in the same way we calculate the performance of our ML model, namely, by calculating NRMSE of the predictions of those baselines on each of our five validation folds, then taking the average of the NRMSEs. We looked at four baselines, predicting the LTV of all customers to be zero, predicting the LTV to be the mean LTV value of the training set, predicting the LTV to be the mean LTV of the requested service in the training set, and finally the moving average model which was used in production until we replaced it with the ML model.

Table 3.15: Final model performance compared to baselines.

Model/Baseline	**5-Fold NRMSE Average**	**XGBoost improvement over the baseline**
XGBoost	1.72468	-
Moving Average Model	1.82781	5.6%
Predicting Service Based Means	1.85438	7%
Predicting the Mean	1.95505	11.8%
Predicting Zeros	2.19659	21.5%

If we look at Table 3.15, the first thing we notice is that the difference between the worst baseline and our best model is relatively small. We think that is because the features we have about the customers and the requested service provide limited information

regarding the amount of LTV the customer will generate. We also see that the improvement over the moving average model is 5.6%, that improvement was sufficient for the company to decide to take the model to production.



(a) (b)

Figure 3.3: Actual LTV values versus model predictions plot. (b) is a close-up version of (a).

3.3.2 Examining Predictions Compared to Actual Values

Figure 3.3 shows the model's predictions plotted against the actual target values. A sample of 10,000 points from the validation folds was used to construct this plot. The values in the plot were normalized by the maximum LTV value within the 10,000 points.

In part (a) of Figure 3.3, we notice that all high LTV examples, starting from 0.4 of the maximum value, are underestimated. That is possibly because these outlier values are grouped with other lower value points in the leaf node, so the final prediction is pulled down by those other points. We also notice that there are a lot of zero-LTV customers predicted to have LTV. This happens because the input features do not contain enough information to differentiate between those points and the actual high LTV customers.

A lot of those zero-LTV points are BM2 service requests. Since the prediction is made before the customer makes the payment, the service request specifications can be similar to a high LTV request, but the customer then does not complete the payment and the LTV of the request/customer stays zero. In part (b) of Figure 3.3, we see that the predictions scatter around the perfect prediction line with no sign of large bias.

For the instances where the actual LTV is zero and the model predicted a high LTV and vice versa, we can examine them and try to engineer features that differentiate them from the actual high LTV instances, since the cost of these errors is particularly high. We

can also find out which features contribute to the model's errors in those instances using local model interpretation methods like SHAP (Lundberg & Lee, 2017) since these methods explain the effect of each feature on the model's prediction. This would be beneficial when explaining the reasons behind the model's mistakes to different stakeholders.

3.3.3 Performance on Different Sections of the Data Set

In this section, we split the validation folds into multiple sections and calculate the NRMSE of each section independently, with the aim of examining which parts of the data does the model predict well and which parts are harder to predict. Further effort could go into extracting or engineering features that make it easier for the model to predict the sections that are hard to predict.

Table 3.16: Final model performance on different sections of the data.

Data Section	**5-Fold NRMSE Average**
Business Model 1	1.64647
Business Model 2	2.55442
Acquired Customers	1.70165
Resurrected Customers	1.69696
Big Provinces	1.64513
Small Provinces	1.85354
Big Services	1.53293
Small Services	1.79643

In Table 3.16, we see that the performance of the BM1 model is better than the BM2 model. That is because we predict BM2 LTV before the customer makes the payment and it is unknown if he will complete the service request or not. We also see that the performance on resurrected and acquired customers is almost the same and that leads us to the conclusion that the customer history at Armut does not provide a lot of extra information about his future LTV. We can also see that the performance in big provinces, i.e. provinces that have a lot of service requests, is better than the performance in small ones, and the same trend is true for services. This is because in big provinces or services, the model has a lot of past examples similar to the current one, and by learning from them the model can more accurately determine the LTV of the current example.

3.3.4 Permutation Feature Importance (PFI)

Permutation feature importance (Breiman, 2001; Molnar, 2022) is a method to measure how informative each feature is to the model regarding the target variable. We calculated the PFI scores for BM1 and BM2 separately as follows:

- Compute the reference NRMSE score s of our XGBoost model m on the first validation fold data D
- For each feature j (column of D):
  - For each repetition k in 1, …, 30:
    - Randomly shuffle column j of dataset D to generate a corrupted version of the data named $\widetilde{D}_{k,j}$
    - Compute the score $s_{k,j}$ of model m on the corrupted data $\widetilde{D}_{k,j}$
  - Compute importance i_j for feature f_j defined as:

$$i_j = s - \frac{1}{30}\sum_{k=1}^{30} s_{kj} \tag{50}$$

All categorical variables fed to the models when calculating PFI are target encoded. The feature importance i for each feature j for BM1 and BM2 models is shown in Figure 3.4 and Figure 3.5 below.

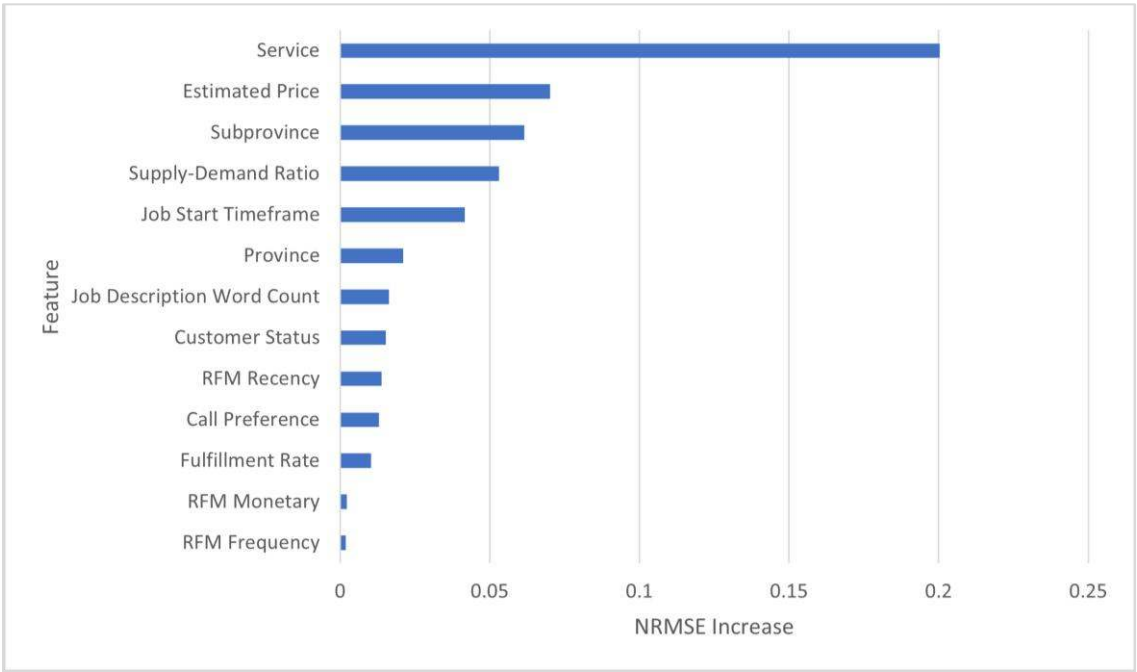


Figure 3.4: Permutation feature importance for the final BM1 model.

In Figure 3.4, we see that for BM1 requests, the service requested is the most informative feature. That is plausible because there are many services that can be

requested through Armut and some of them have a lot of service providers and a high probability of being fulfilled and some others have a low quote and fulfillment probability. This gives the model a lot of information about future LTV that can be earned from the service. The same logic applies to location features, namely, province and subprovince. Since they have correlated information, it appears that the model utilized the subprovince feature more than the more general province feature. We can also see that the RFM features are not very informative for our model. This leads us to the conclusion that the past customer behavior does not affect the customer's future potential LTV, at least when requesting BM1 services. We also notice that the estimated price of the service request is the second most important feature although it does not correlate with LTV as we saw in section 3.1.2. That is because it provides important information when combined with the quote or fulfillment probability of the service, since Armut's revenue is correlated with the price of the service.

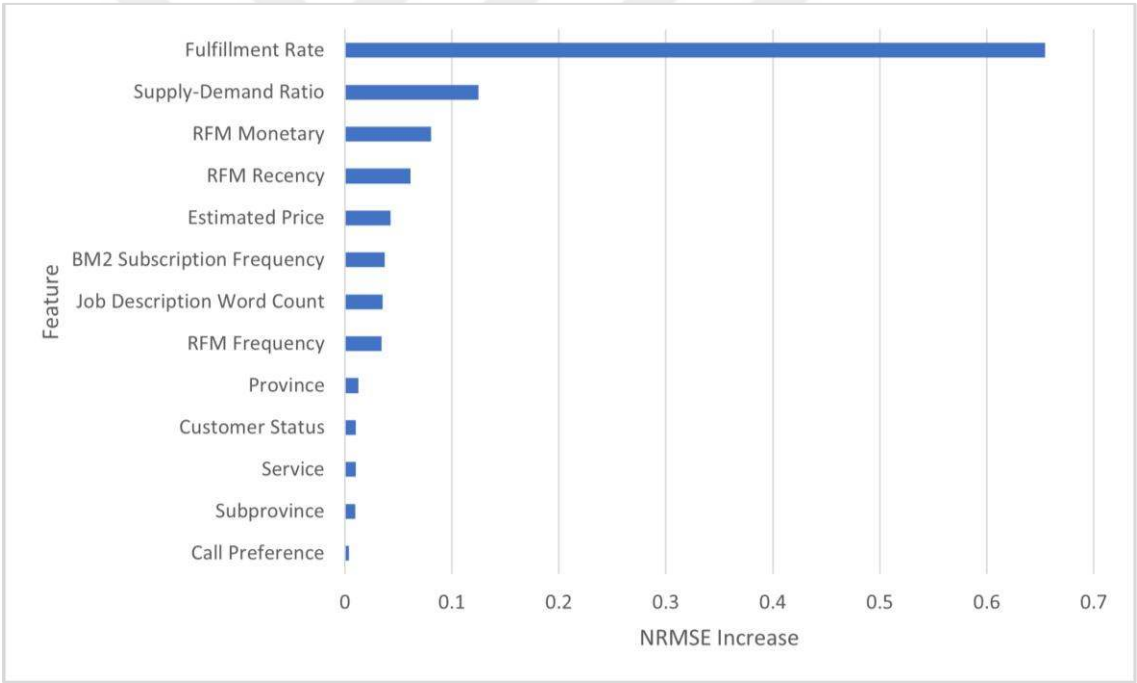


Figure 3.5: Permutation feature importance for the final BM2 model.

When we look at the feature importance of the BM2 model (Figure 3.5), the first thing we notice is that its important features are vastly different from the BM1 model. We also notice that the fulfillment rate of past services in the same area holds a disproportionate amount of importance. This is reasonable because the service fulfillment data is a lot more accurate for BM2 (See section 2.2). We also notice customer history features, i.e. RFM, are more important for BM2 than for BM1. That means that customers who request BM2 services have some consistent properties that are helpful when

predicting LTV. In contrast to BM1, we see that the requested service is not an important feature for BM2 model. That can be attributed to the fewness of BM2 services (See section 2.2) and that most relevant information are already provided by the more important features.

3.4 Inference Phase

This section contains information about the inference phase, referred to as inference time in (Lakshmanan et al., 2020), by which we mean the phase when an ML model is "deployed into production" and receives inputs from a live system rather than a still set of data. This phase takes place in the industry after training the model and obtaining satisfying results on validation data. It contains deploying the model, verifying its performance on live data, monitoring its performance, and retraining it.

3.4.1 Deployment

Since the performance improvement of our model over the previous one was satisfactory, we decided to take the model to production. In general, the performance of the model on live data is not guaranteed to be similar to the performance on the validation data due to many mistakes that can be made in the training phase. Therefore, ML models are generally shadow-deployed, meaning that they are deployed and make predictions on live data, but their predictions are not used in production. Their predictions, however, are monitored and the performance on live data is verified to be the same as the performance on the validation data. We carried out this process at Armut and the model performance was the same on live data. More specifically, the improvement over the segmented moving average baseline ranged from 5% to 7% on live data depending on the time window, while it was 5.6% in our validation data. Therefore, we started using our model in production. Live system performance being close to validation performance is a sign that there is no data-leakage in the training process and that the features are calculated in the same manner in the training and inference phases. In the rest of this section, we explain the important details of the deployment process.

While developing the model, the historical data used for training and validation was queried from our data warehouse and was queried in bulk. In production, we cannot use the data warehouse because the data warehouse is geared for analytical queries and is not suitable for operational workloads. In addition, real-time data are not stored in the

warehouse. We therefore rewrote the queries to pull the features for a specific service request, rather than for a bulk of requests, and to run on our live database, rather than the data warehouse.

We built a web API to serve the predictions of the model. The popular Python package Flask was used for this task and the API was hosted on an AWS ECS instance. The API we created has a prediction endpoint which receives a request for LTV prediction and returns the appropriate prediction. The endpoint receives the service request ID as a parameter in the prediction request. The endpoint then queries the database for the required features, preprocesses those features and feeds them into the model and returns the model prediction to the request sender.

Some of the features we require for LTV prediction are time consuming to compute in real-time, namely, the supply-demand ratio and the fulfillment rate. Calculating these features in real time increases the response time of the endpoint to unacceptable levels. The solution to this problem is to precalculate these features and store them in the database so that the model can read these precalculated features directly from the database. The database that contains the precalculated features is usually called a Feature Store (Lakshmanan et al., 2020). For this project, we store the supply-demand ratio and the fulfillment rate features in the feature store and load the precalculated values directly when we are making a prediction. The precalculated values are updated (recalculated) every day.

3.4.2 Monitoring and Live Performance

We needed to monitor the performance of the model on the live data after shadow-deploying it to decide whether to use it in production or not. Monitoring the model performance keeps being an essential requirement after starting to use it in production as well since problems with input data can occur and that would deteriorate the model's predictions.

We monitor our model's performance first by recording all its predictions in our data warehouse. Since the predictions and the target values are now both in the data warehouse, we use SQL to calculate NRMSE values for daily predictions. We also record the predictions of our segmented moving average baseline on the same data and calculate its daily NRMSE as well. We plot both daily performance lines using our data visualization tool and the result is a plot like in Figure 3.6.

We see that every day the model has a different performance value, that is because some days contain records that are harder to predict than others. However, we also see that the performance improvement over the baseline is in the same range throughout the days in Figure 3.6. We set up alarms to notify us when our model's performance improvement over the baseline drops below 3%, so that we can investigate the reasons.

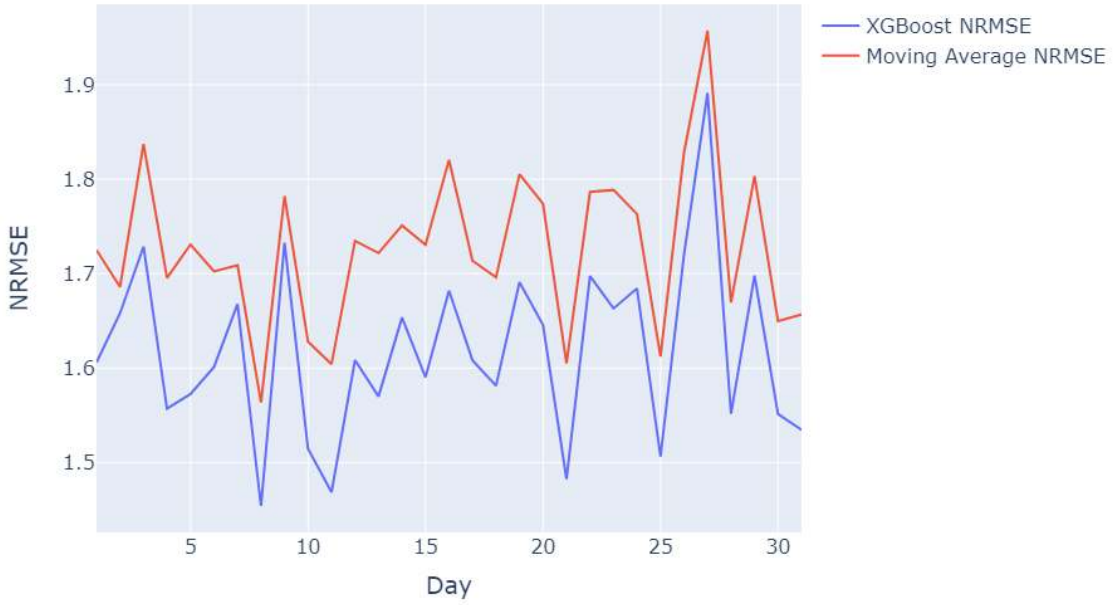


Figure 3.6: Daily model performance versus the baseline on the live system.

3.4.3 Retraining

We set up a scheduled task to retrain the model every week on a rolling window of one year training data. We use a validation set of one month time window and compare the performance of the current model and the newly trained model on it. If the performance of the new model is better than the previous model on the validation set, we take it to production, otherwise, we do not. This retraining process ensures that the model used in production is always up to date and familiar with new data.

Chapter 4:
CONCLUSION

In this thesis, we outlined how we used machine learning to solve the customer lifetime value prediction problem in Armut, one of the leading local services marketplaces globally. This thesis presents a unique case in the CLV prediction literature since it is the first study to discuss CLV prediction in the service marketplace industry, which has different challenges and data arrangement compared to other industries.

To find the ML algorithm most suitable for our problem's data, we utilized four different ML models with varying complexity and learning mechanisms, namely, linear regression, decision tree, XGBoost, and neural network. We did not include the KNN algorithm because it was prohibitively slow in this problem.

For our model validation to give generalizable results, the validation setup had to be as close as possible to the model's setup in the inference phase, i.e., after deployment. Specifically, because our problem's data contain an important temporal/seasonal pattern, it was important not to use k-fold cross-validation, as it evaluates models trained on future data using past data. That is why we used time-series cross-validation and we believe it contributed to our model's success after being deployed, since the model showed the same level of performance, as measured by NRMSE, on the validation set as well as on live data.

We experimented with different encoding techniques for the data before feeding it to each ML model we tried. The optimal encoding method for each model appears to have a small yet significant positive effect on the overall model performance with no single encoding method being generally superior with every ML algorithm. Using different encoding methods on different categorical features of the data produced inferior results across the board compared to using a single encoding method for all categorical features.

Even though our data is tabular, the neural network model performed best, followed closely by XGBoost, and then decision tree and linear models. This proves that the performance of tree-based models is not always superior to NNs when learning from tabular data. However, the minimal performance improvement was not enough to offset the upsides and practicality of tree-based models like XGBoost in our case. These include the short training time, lower computational requirement in training and inference, and

ease of maintenance and debugging. The usage of the simpler models, linear regression and decision trees, is recommended when the interpretability of the results is necessary.

Furthermore, the small differences in the performance between complex and simple models in our case suggest that the amount of complex information in our data that could be utilized by a highly complex model is limited. That should be taken into consideration before utilizing increasingly complex models.

Optimizing the training set time window showed some improvements with the 12-month time window, utilizing the seasonal patterns in the data set. Furthermore, developing multiple models for different segments of the data that are identified by inspection produced mixed results. These segments include, whether the customer is resurrected or new, which appears to have no significant effect on the model's performance. These resurrected customers are, therefore, identical to new customers which suggests that the past customer behavior is not informative of future customer LTV in the local services industry. On the other hand, developing a separate model for each of Armut's business models proved advantageous suggesting that this type of segmentation is quite beneficial when feasible.

The consistent significant superiority of the model performance by nearly 6% over the previously deployed segmented moving average model led to its adaptation in production. Here, we recommend to always shadow deploy the model first and monitor its online performance then debug or modify it as necessary before using its predictions in the live system. Even after deployment, it is imperative to continuously monitor the model's performance compared to benchmark or baseline methods. Depending on the implemented time window, daily or weekly retraining of the model is beneficial to keep the model up to date.

4.1 Future Work

There are many possible directions to consider in future research for LTV prediction problems and for this problem in particular. The LTV values of old records in the training data set, where prices are outdated, can be inflated to reflect the present value of the LTV during training. This requires automatic importing of inflation data from an online source or manual inflation data entry complicating the model further. Whether such complications are outweighed by the benefits to the model performance needs to be properly investigated.

As observed in Figure 3.3, the model predicted a high LTV for some zero LTV customers and vice versa, we can examine these instances and try to engineer features that differentiate them from the actual high LTV customers, since the cost of these errors is particularly high. We can also find out which features contribute to the model's errors in those instances using local model interpretation methods, since these methods explain the effect of each feature on the model's prediction.

Since segmenting the data and training a separate model on each segment proved useful, a more systematic way to segment the data or search for beneficial segmentations can be utilized.

BIBLIOGRAPHY

Abe, M. (2009). “Counting Your Customers” One by One: A Hierarchical Bayes Extension to the Pareto/NBD Model. *Marketing Science*, *28*(3), 541–553. https://doi.org/10.1287/mksc.1090.0502

Alpaydin, E. (2020). *Introduction to machine learning* (Fourth edition). The MIT Press. https://mitpress.mit.edu/9780262043793/introduction-to-machine-learning/

Bauer, J., & Jannach, D. (2021). Improved Customer Lifetime Value Prediction With Sequence-To-Sequence Learning and Feature-Based Models. *ACM Transactions on Knowledge Discovery from Data*, *15*(5), 1–37. https://doi.org/10.1145/3441444

Bernat, J. R. (2019). *Modelling Customer Lifetime Value in a Continuous, Non-Contractual Time Setting* [Econometrics & Management Science Master Thesis, Erasmus University Rotterdam]. https://thesis.eur.nl/pub/45923

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Cerda, P., Varoquaux, G., & Kégl, B. (2018). Similarity encoding for learning with dirty categorical variables. *Machine Learning*, *107*(8–10), 1477–1494. https://doi.org/10.1007/s10994-018-5724-2

Chamberlain, B. P., Cardoso, Â., Liu, C. H. B., Pagliari, R., & Deisenroth, M. P. (2017). Customer Lifetime Value Prediction Using Embeddings. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1753–1762. https://doi.org/10.1145/3097983.3098123

Chang, W., Chang, C., & Li, Q. (2012). Customer Lifetime Value: A Review. *Social Behavior and Personality: An International Journal*, *40*(7), 1057–1064. https://doi.org/10.2224/sbp.2012.40.7.1057

Chen, P. P., Guitart, A., del Rio, A. F., & Perianez, A. (2018). Customer Lifetime Value in Video Games Using Deep Learning and Parametric Models. *2018 IEEE International Conference on Big Data (Big Data)*, 2134–2140. https://doi.org/10.1109/BigData.2018.8622151

Chen, S. (2018). Estimating Customer Lifetime Value Using Machine Learning Techniques. In C. Thomas (Ed.), *Data Mining*. InTech. https://doi.org/10.5772/intechopen.76990

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

Chen, Z.-Y., & Fan, Z.-P. (2013). Dynamic customer lifetime value prediction using longitudinal data: An improved multiple kernel SVR approach. *Knowledge-Based Systems*, *43*, 123–134. https://doi.org/10.1016/j.knosys.2013.01.022

Cho, P., & Chen, T. (2021). *XGBoost Tutorials* [Documentation]. XGBoost Documentation. https://xgboost.readthedocs.io/en/stable/tutorials/index.html

Criteo. (2018). *State of Ad Tech 2019, Global* (p. 29). Criteo. https://www.criteo.com/wp-content/uploads/2018/12/StateOfAdTechReport_Global.pdf

Drachen, A., Pastor, M., Liu, A., Fontaine, D. J., Chang, Y., Runge, J., Sifa, R., & Klabjan, D. (2018). To be or not to be...social: incorporating simple social features in mobile game customer lifetime value predictions. *Proceedings of the Australasian Computer Science Week Multiconference*, 1–10. https://doi.org/10.1145/3167918.3167925

Ekinci, Y. (2011). *Analysis of Customer Lifetime Value: Development of Alternative Models* [Ph.D. Thesis, Istanbul Technical University]. https://tez.yok.gov.tr/UlusalTezMerkezi/giris.jsp

Fisher, R. A. (1950). *Statistical methods for research workers* (11th ed.). Oliver and Boyd.

Fruchter, G. E., & Sigué, S. P. (2009). Social Relationship and Transactional Marketing Policies—Maximizing Customer Lifetime Value. *Journal of Optimization Theory and Applications*, *142*(3), 469–492. https://doi.org/10.1007/s10957-009-9536-1

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems* (Second edition). O'Reilly Media, Inc.

Glady, N., Baesens, B., & Croux, C. (2009). A modified Pareto/NBD approach for predicting customer lifetime value. *Expert Systems with Applications*, *36*(2), 2062–2071. https://doi.org/10.1016/j.eswa.2007.12.049

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press.

Google Ads Help Team. (2022). *About Automated Bidding* [Documentation & Technical Support]. Google Ads Help. https://support.google.com/google-ads/answer/2979071?hl=en

Google Analytics Support Team. (2022). *Macro / Micro conversions & Funnels* [Documentation & Technical Support]. Google Analytics Help. https://support.google.com/analytics/answer/11053642?hl=en

Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). *Why do tree-based models still outperform deep learning on tabular data?* (arXiv:2207.08815). arXiv. http://arxiv.org/abs/2207.08815

Gupta, S., Hanssens, D., Hardie, B., Kahn, W., Kumar, V., Lin, N., Ravishanker, N., & Sriram, S. (2006). Modeling Customer Lifetime Value. *Journal of Service Research*, *9*(2), 139–155. https://doi.org/10.1177/1094670506293810

Ha, L. (2008). Online Advertising Research in Advertising Journals: A Review. *Journal of Current Issues & Research in Advertising*, *30*(1), 31–48. https://doi.org/10.1080/10641734.2008.10505236

Hyndman, R. J., & Athanasopoulos, G. (2021). *Forecasting: principles and practice* (3rd edition). OTexts.

Kannan, P. K., & Li, H. "Alice." (2017). Digital marketing: A framework, review and research agenda. *International Journal of Research in Marketing*, *34*(1), 22–45. https://doi.org/10.1016/j.ijresmar.2016.11.006

Kotler, P., Armstrong, G., & Harris, L. C. (2019). *Principles of marketing* (Eighth European Edition). Pearson.

Kotsiantis, S. B. (2013). Decision trees: a recent overview. *Artificial Intelligence Review*, *39*(4), 261–283. https://doi.org/10.1007/s10462-011-9272-4

Lakshmanan, V., Robinson, S., & Munn, M. (2020). *Machine learning design patterns: solutions to common challenges in data preparation, model building, and MLOps* (First edition). O'Reilly Media.

Lee, J., Podlaseck, M., Schonberg, E., & Hoch, R. (2001). Visualization and Analysis of Clickstream Data of Online Stores for Understanding Web Merchandising. *Data Mining and Knowledge Discovery*, *5*(1/2), 59–84. https://doi.org/10.1023/A:1009843912662

Lenail, A. (2018). *Publication-ready NN-architecture schematics Generator*. Alex Lenail. https://alexlenail.me/NN-SVG

Lundberg, S., & Lee, S.-I. (2017). *A Unified Approach to Interpreting Model Predictions* (arXiv:1705.07874). arXiv. http://arxiv.org/abs/1705.07874

Microsoft Ads. (2022). *About Ad distribution* [Documentation & Technical Support]. Microsoft Advertising Help Center. https://help.ads.microsoft.com/#apex/ads/en/50871/0

Molnar, C. (2022). *Interpretable machine learning: a guide for making black box models explainable* (Second edition). Christoph Molnar.

Pargent, F., Pfisterer, F., Thomas, J., & Bischl, B. (2022). Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Computational Statistics*. https://doi.org/10.1007/s00180-022-01207-6

Parmar, A., Katariya, R., & Patel, V. (2019). A Review on Random Forest: An Ensemble Classifier. In J. Hemanth, X. Fernando, P. Lafata, & Z. Baig (Eds.), *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018* (Vol. 26, pp. 758–763). Springer International Publishing. https://doi.org/10.1007/978-3-030-03146-6_86

Prasasti, N., Okada, M., Kanamori, K., & Ohwada, H. (2014). Customer Lifetime Value and Defection Possibility Prediction Model Using Machine Learning: An Application to a Cloud-Based Software Company. In N. T. Nguyen, B. Attachoo, B. Trawiński, & K. Somboonviwat (Eds.), *Intelligent Information and Database Systems* (Vol. 8398, pp. 62–71). Springer International Publishing. https://doi.org/10.1007/978-3-319-05458-2_7

Rust, R. T., Lemon, K. N., & Zeithaml, V. A. (2004). Return on Marketing: Using Customer Equity to Focus Marketing Strategy. *Journal of Marketing*, *68*(1), 109–127. https://doi.org/10.1509/jmkg.68.1.109.24030

Sagi, O., & Rokach, L. (2018). Ensemble learning: A survey. *WIREs Data Mining and Knowledge Discovery*, *8*(4). https://doi.org/10.1002/widm.1249

Salah, M. (2019, June 3). Armut — How The Marketplace Really Works? [Blog]. *Armut Labs*. https://labs.armut.com/104a07161442

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Sen, A., & Foster, J. E. (1997). *On economic inequality* (Enl. ed). Clarendon Press ; Oxford University Press.

Sifa, R., Runge, J., Bauckhage, C., & Klapper, D. (2018). *Customer Lifetime Value Prediction in Non-Contractual Freemium Settings: Chasing High-Value Users Using Deep Neural Networks and SMOTE*. Hawaii International Conference on System Sciences. https://doi.org/10.24251/HICSS.2018.115

Statcounter. (2022). *Search Engine Market Share Worldwide* [Internet Trends Statistics]. Statcounter Global Stats. https://gs.statcounter.com/search-engine-market-share

Tabachnick, B. G., Fidell, L. S., & Ullman, J. B. (2019). *Using multivariate statistics* (Seventh edition). Pearson.

Tanrıver, G. (2021). *Automated Detection of Oral Lesions Using Deep Learning for Early Diagnosis of Oral Cancer* [Master of Science Thesis, Koç University]. https://tez.yok.gov.tr/UlusalTezMerkezi/giris.jsp

Tekin, A. T., Kaya, T., & Cebi, F. (2021). Customer lifetime value prediction for gaming industry: fuzzy clustering based approach. *Journal of Intelligent & Fuzzy Systems*, 1–10. https://doi.org/10.3233/JIFS-219177

Tsai, C., Hu, Y., Hung, C., & Hsu, Y. (2013). A comparative study of hybrid machine learning techniques for customer lifetime value prediction. *Kybernetes*, *42*(3), 357–370. https://doi.org/10.1108/03684921311323626

Vanderveld, A., Pandey, A., Han, A., & Parekh, R. (2016). An Engagement-Based Customer Lifetime Value System for E-commerce. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 293–302. https://doi.org/10.1145/2939672.2939693

Wang, Y., Sanguansintukul, S., & Lursinsap, C. (2008). The customer lifetime value prediction in mobile telecommunications. *2008 4th IEEE International Conference on Management of Innovation and Technology*. https://doi.org/10.1109/ICMIT.2008.4654427

Win, T. T., & Bo, K. S. (2020). Predicting Customer Class using Customer Lifetime Value with Random Forest Algorithm. *2020 International Conference on Advanced Information Technologies (ICAIT)*, 236–241. https://doi.org/10.1109/ICAIT51105.2020.9261792

Zhao, S., Wu, R., Tao, J., Qu, M., Zhao, M., Fan, C., & Zhao, H. (2022). perCLTV: A General System for Personalized Customer Lifetime Value Prediction in Online Games. *ACM Transactions on Information Systems*, 3530012. https://doi.org/10.1145/3530012