1 A Quadratic Mean Based Supervised Learning Model for Managing Data Skewness

This chapter reviews a novel approach for addressing data skewness in supervised learning models, presented in the paper "A Quadratic Mean Based Supervised Learning Model for Managing Data Skewness." The authors propose a framework called QMLearn, which introduces a new way of calculating empirical risk using the quadratic mean, aimed at improving model robustness on imbalanced datasets.

1.1 Introduction to the Problem of Data Skewness

Data skewness, or imbalance, is a prevalent issue in supervised learning where the distribution of the dependent variable is uneven. This imbalance often causes traditional models to become biased towards the majority class, resulting in poor performance on the minority class. This paper identifies the limitations of traditional empirical risk minimization methods and introduces QMLearn to better handle skewed data.

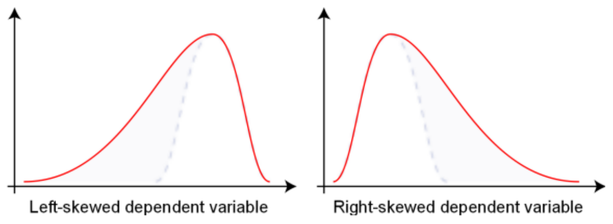


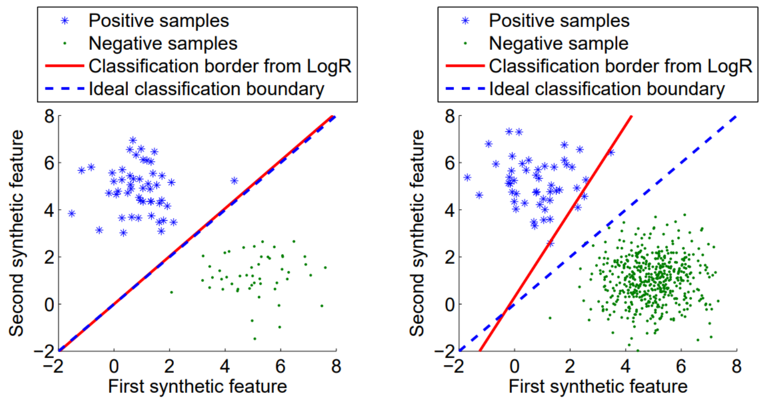
Figure 1: Data skewness.

1.2 Limitations of Traditional Learning Models

Traditional learning models, such as logistic regression and SVMs, typically minimize an empirical risk function defined as the arithmetic mean of the loss across all training examples:

$$R_{\text{emp}}(w) = \frac{1}{n}\sum_{i=1}^{n} l(x_i, y_i, w) \tag{1}$$

where n is the total number of training instances, x_i represents the feature vector, y_i the true label, and w the model parameters. This method often results in models that are biased towards the majority class, as illustrated in Figure 2.



(a) Class ratio = 1:1 (b) Class ratio = 1:10

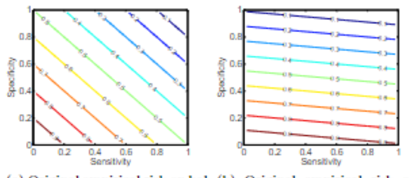
Figure 2: Classification boundaries using traditional logistic regression on balanced (left) and imbalanced (right) datasets.

1.3 Introduction to the QMLearn Framework

The QMLearn framework modifies the traditional empirical risk by using the quadratic mean rather than the arithmetic mean. This redefinition is designed to address the imbalance by equalizing the influence of both classes on the model's training process. The quadratic mean-based empirical risk function is defined as:

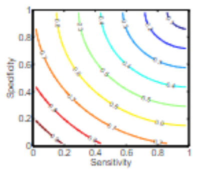
$$R_{\text{emp}}^{Q}(w) = \sqrt{\frac{\left(\frac{\sum_{i=1}^{n_1} l(x_i,y_i,w)}{n_1}\right)^2 + \left(\frac{\sum_{i=n_1+1}^{n} l(x_i,y_i,w)}{n_2}\right)^2}{2}} \tag{2}$$

where n_1 and n_2 are the number of instances in each class. This approach balances the error contributions from both classes, making the model more robust against skewed distributions.

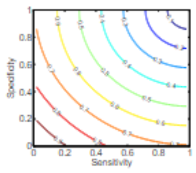


(a) Original empirical risk on balanced data (b) Original empirical risk on imbalanced data (Pos:Neg = 1:10)

Figure 3: empirical risk on balanced (left) and imbalanced (right) data.



(a) QMLearn-based empirical risk on balanced data



(b) QMLearn-based empirical risk on imbalanced data (Pos:Neg = 1:10)

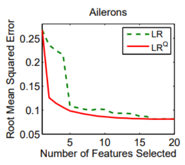
Figure 4: QMLearn-based empirical risk on balanced (left) and imbalanced (right) data.

1.4 Convex Optimization for Efficient Learning

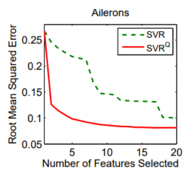
The QMLearn method is formulated as a convex optimization problem, maintaining the convexity of the empirical risk function through the use of the quadratic mean. This allows for efficient computation and ensures that the solution is optimal, even for large-scale datasets.

1.5 Experimental Validation

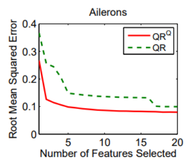
Extensive experiments were conducted to validate the effectiveness of QMLearn compared to traditional models like logistic regression, SVMs, and quantile regression. The results, depicted in Figure 5, show that QMLearn consistently outperforms traditional methods, particularly on imbalanced datasets.



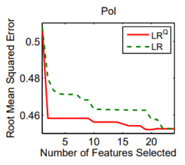
(a) Improvements on ordinal linear regression in data set "Ailerons"



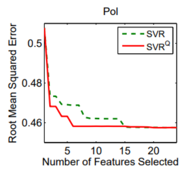
(b) Improvements on SVR in data set "Ailerons"



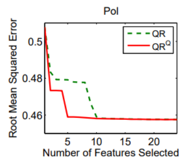
(c) Improvements on quantile regression in data set "Ailerons"



(d) Improvements on ordinal linear regression in data set "Pol"



(e) Improvements on SVR in data set "Pol"



(f) Improvements on quantile regression in data set "Pol"

Figure 5: Performance improvements of QMLearn-based models in regression tasks on datasets 'Ailerons' and 'Pol'.

1.6 Using the QMLearn Package

To implement the QMLearn framework in your own work, you can install the QMLearn package by following the instructions available at qmlearn.rutgers.edu/source/install.html. For more details on installation and usage, please refer to the official QMLearn documentation.

1.7 Conclusion

The QMLearn framework offers a robust solution for managing data skewness in supervised learning models. By redefining the empirical risk function using the quadratic mean, QMLearn ensures balanced error consideration across classes, improving model performance on imbalanced datasets. Future research could explore the application of QMLearn to other types of models and further investigate its theoretical underpinnings.

2 Machine Learning for Resource Estimation in Highly Skewed Gold Deposits

This chapter explores the methodologies proposed in the paper "A Novel Approach for Resource Estimation of Highly Skewed Gold Using Machine Learning Algorithms." The authors introduce a machine learning-based framework to address the challenges posed by highly skewed distributions of gold grades in vein deposits. Traditional geostatistical methods, such as ordinary kriging and indicator kriging, often struggle to model these distributions accurately due to their reliance on assumptions of normality and spatial continuity. The proposed machine learning approaches offer a more flexible and robust alternative for estimating resources in such complex geological settings.

2.1 Understanding Skewed Vein Deposits

Vein deposits often exhibit highly skewed distributions of gold grades, primarily due to the nugget effect, which causes significant variability in gold concentration. In these deposits, the majority of samples have low gold grades, while a few samples exhibit very high grades. This creates a long-tailed, or right-skewed, distribution, which complicates the accurate estimation of gold resources. The challenge lies in developing models that can accurately predict both the frequent low-grade occurrences and the rare high-grade values without bias.

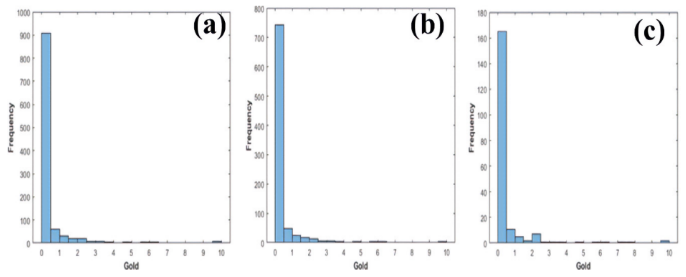


Figure 6: Histogram of gold values for (a) entire dataset, (b) training dataset, and (c) testing dataset. The histograms illustrate the right-skewed nature of the gold distribution, with a high frequency of low-grade values and a few high-grade values.

2.2 Data Normalization Techniques

To effectively handle the skewed distribution of gold grades, the authors applied two normalization techniques: logarithmic normalization and z-score normalization. These techniques are crucial for transforming the data into a form that is more suitable for machine learning models, particularly when dealing with highly skewed data.

2.2.1 Logarithmic Normalization

Logarithmic normalization is a transformation that reduces the impact of outliers by compressing the range of values. This technique is particularly effective for right-skewed data, as it mitigates the effect of extreme high-grade values, thereby creating a more balanced distribution. By applying a logarithmic transformation, the differences between high-grade and low-grade values are reduced, making the data more symmetric and better suited for regression modeling. This transformation is defined as:

$$x' = \log(x+1)$$

where x is the original data point, and x' is the transformed data. The addition of 1 ensures that zero values are handled appropriately, as the logarithm of zero is undefined.

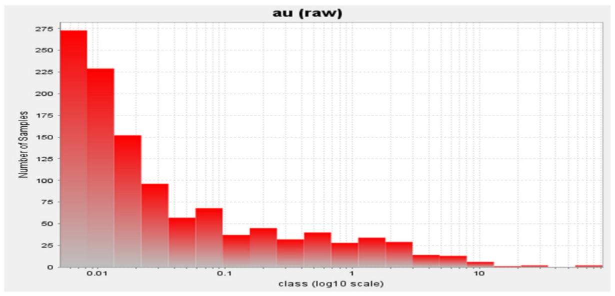


Figure 7: Logarithmic histogram of gold distribution, showing how logarithmic normalization reduces skewness by transforming the original right-skewed data into a more balanced distribution.

2.2.2 Z-Score Normalization

Z-score normalization standardizes the data by subtracting the mean and dividing by the standard deviation. This technique centers the data around zero with a unit variance, stabilizing the variance across different scales of data. Z-score normalization is effective when the data distribution is approximately normal. However, when applied to skewed data, it can still be useful by scaling the data in a way that allows machine learning models to better capture the underlying patterns. The z-score normalization formula is:

$$z = \frac{x-\mu}{\sigma}$$

where x is the data point, μ is the mean of the dataset, and σ is the standard deviation.

2.3 Data Segmentation Using Marine Predators Algorithm (MPA)

To ensure that both high-grade and low-grade gold samples are adequately represented in the training and testing datasets, the authors employed the Marine Predators Algorithm (MPA) for data segmentation. MPA is an optimization algorithm inspired by the foraging strategies of marine predators. It is used to create balanced datasets by effectively capturing the distribution of gold grades across different segments. This segmentation approach prevents the model from being biased towards either end of the spectrum, enhancing its accuracy and generalization capabilities.

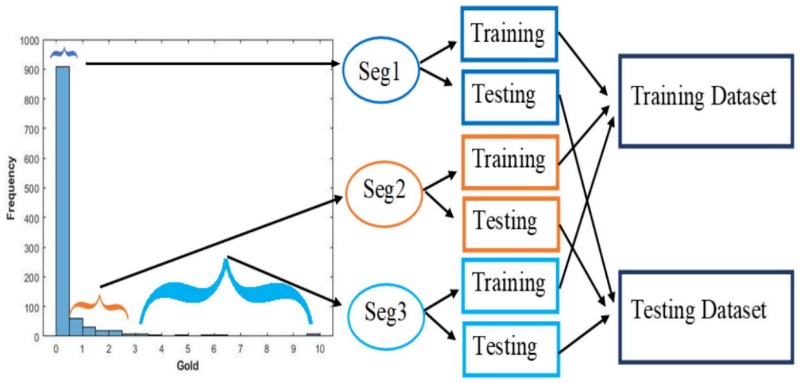


Figure 8: Data segmentation of gold values for training and testing based on the histogram plot. The Marine Predators Algorithm (MPA) was used to create balanced training and testing datasets, effectively capturing the distribution of gold grades across different segments.

2.4 Machine Learning Models for Estimation

The study explored several machine learning algorithms, including Gaussian Process Regression (GPR), Support Vector Regression (SVR), Decision Tree Ensemble (DTE), Fully Connected Neural Network (FCNN), and K-Nearest Neighbors (K-NN). Each model has unique strengths and weaknesses in handling skewed data distributions, making them suitable for different aspects of the resource estimation task.

2.4.1 Gaussian Process Regression (GPR)

GPR is a non-parametric, probabilistic model that provides a flexible framework for regression tasks. It models the distribution of the target variable and its uncertainty, making it well-suited for handling the variability in gold grades. GPR can capture complex, non-linear relationships in the data, which is particularly important for accurately predicting rare, high-grade values.

2.4.2 Support Vector Regression (SVR)

SVR uses kernel functions to handle non-linear relationships in the data by maximizing the margin of error while minimizing model complexity. This approach is effective in managing skewness and outliers, as it focuses on a subset of critical points (support vectors) that define the model, rather than all the data points.

2.4.3 Decision Tree Ensemble (DTE)

Ensemble methods like Random Forests or Gradient Boosting Machines aggregate multiple decision trees to improve prediction accuracy and robustness. These models can handle varying distributions within the data by creating diverse decision paths, making them suitable for estimating resources in skewed deposits.

2.4.4 Fully Connected Neural Network (FCNN)

Neural networks can learn complex patterns and relationships in the data through multiple layers of interconnected neurons. FCNNs are particularly effective for capturing non-linear dependencies and can adapt to skewed distributions by learning directly from the data without assuming any prior distribution.

2.4.5 K-Nearest Neighbors (K-NN)

K-NN is a non-parametric method that predicts output values based on the average output of the k-nearest training samples. It is capable of handling skewed data by considering local structures rather than global distribution assumptions, making it a useful tool for resource estimation in complex geological settings.

2.5 Performance Evaluation

The authors evaluated the performance of these machine learning models using both normalized datasets (z-score and logarithmic) to handle the skewness effectively. The results demonstrated that machine learning models could predict gold grades accurately, even in the presence of highly skewed data.

2.5.1 Performance after Logarithmic Normalization

After applying logarithmic normalization, the machine learning models showed high accuracy in capturing the variability of gold grades, effectively handling the skewed data. This approach allowed the models to generalize well across different gold grades, as shown in the predicted versus actual plots.

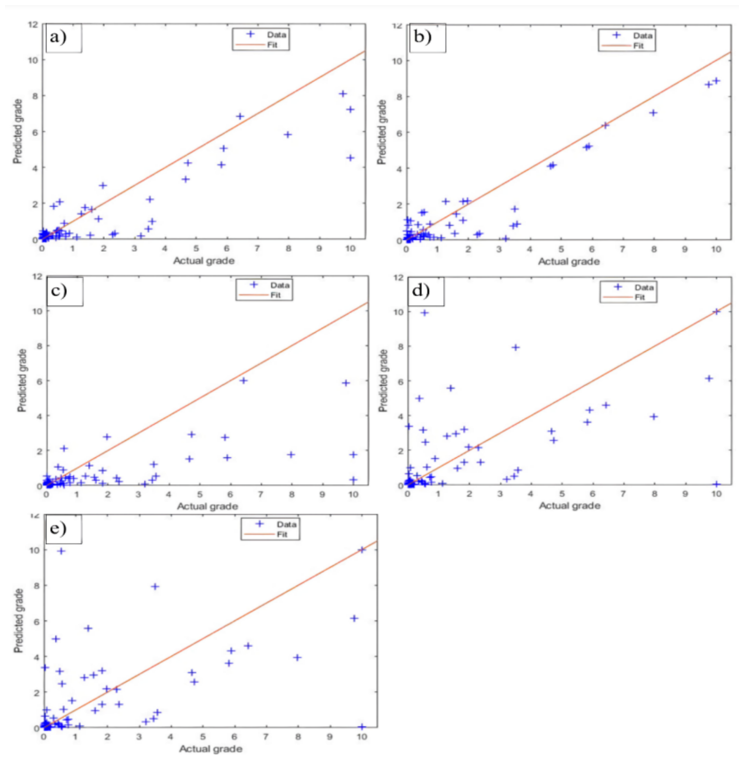


Figure 9: Predicted vs. actual gold grades after logarithmic normalization for (a) Gaussian Process Regression (GPR), (b) Support Vector Regression (SVR), (c) Decision Tree Ensemble (DTE). The models show good agreement with the actual values, indicating successful handling of the skewed data.

2.5.2 Performance after Z-Score Normalization

Similarly, z-score normalization also provided effective results, particularly for models that assume normal distribution of data. The prediction accuracy remained high across different gold grades, showcasing the robustness of machine learning models in resource estimation tasks.

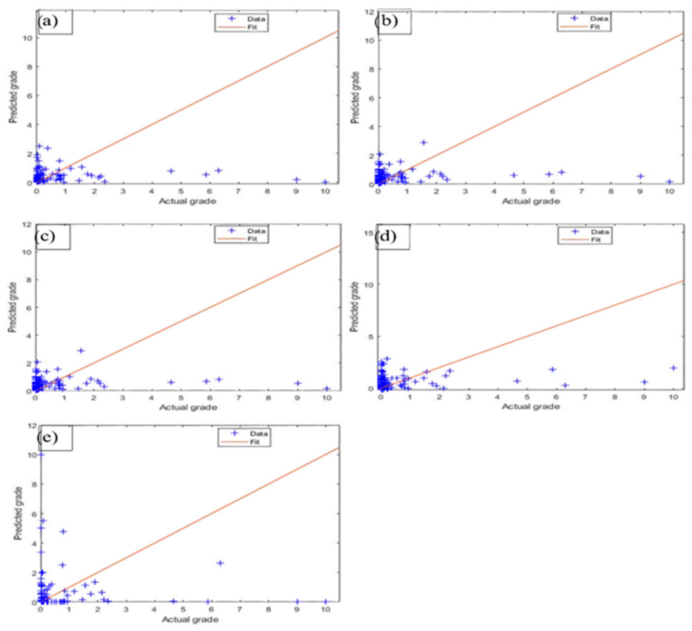


Figure 10: Predicted vs. actual gold grades after z-score normalization for (a) Gaussian Process Regression (GPR), (b) Support Vector Regression (SVR), (c) Decision Tree Ensemble (DTE). The plots demonstrate the models' capability to predict gold grades accurately, maintaining consistency across different normalization techniques.

2.6 Conclusion

The paper presents a robust methodology for estimating gold resources in highly skewed vein deposits using machine learning algorithms. By employing normalization techniques and advanced data segmentation strategies, the study highlights the potential of machine learning to outperform traditional geostatistical methods in complex geological settings. The findings suggest that machine learning models, particularly when optimized with appropriate preprocessing and data handling techniques, can provide accurate and reliable resource estimates even in the presence of significant data skewness. Future research may focus on refining these models further and exploring their application to other types of mineral deposits.

3 Distributional Robustness Loss for Long-Tail Learning

This chapter summarizes the methodologies proposed in the paper ”Distributional Robustness Loss for Long-Tail Learning” by Dvir Samuel and Gal Chechik. The paper addresses the challenge of learning from unbalanced datasets with long-tailed distributions, where a few classes have a large number of samples (head classes), while many classes have very few samples (tail classes). The authors introduce a novel loss function based on Distributionally Robust Optimization (DRO) to improve the learning of representations for both head and tail classes in deep learning models.

3.1 Challenges of Long-Tail Learning

In real-world datasets, long-tailed distributions are common, where the frequency of classes follows a steep drop-off from the head to the tail. This imbalance causes deep models to be biased towards head classes, leading to poor recognition performance for tail classes. Traditional approaches to handling unbalanced data, such as data resampling, loss reweighting, and classifier adjustments, primarily focus on balancing the classifier's output. However, these methods do not adequately address the underlying representation learned by the model, which remains biased towards the head classes.

3.2 Distributionally Robust Optimization (DRO)

To mitigate the biases in representation learning, the authors propose using Distributionally Robust Optimization (DRO). DRO aims to minimize the worst-case loss within an uncertainty set of possible distributions, thereby enhancing the model's robustness to shifts in the data distribution. The DRO framework is particularly suitable for long-tail learning, as it allows the model to better generalize to tail classes, which are underrepresented in the training data.

The DRO problem is formulated as follows:

$$\text{DRO} : \min_{f} \sup_{Q \in \mathcal{U}} E_{(x,y)\sim Q}[l(f(x), y)] \tag{3}$$

Here, f represents the model, $\mathcal{U}$ is an uncertainty set around the empirical training distribution $\hat{P}$, and $l(f(x), y)$ is the loss function. The goal is to minimize the worst-case expected loss over all distributions Q within the uncertainty set $\mathcal{U}$.

3.3 Distributional Robustness Loss (DRO-LT)

The authors introduce a novel loss function, called DRO-LT Loss, designed to improve representation learning under long-tailed data distributions. This loss function extends standard contrastive losses, which pull samples closer to the centroid of their own class and push away samples from other classes. DRO-LT Loss, however, accounts for the uncertainty in estimating class centroids, particularly for tail classes with fewer samples.

The DRO-LT loss function is defined as:

$$L_{\text{Robust}} = -\sum_{c \in C} w(c) \sum_{z \in S_c} \log \frac{e^{-d(\hat{\mu}_c, z) - 2\epsilon_c}}{\sum_{z' \in Z} e^{-d(\hat{\mu}_c, z') - 2\epsilon_c \delta(z', c)}} \tag{4}$$

where:

- $\hat{\mu}_c$ is the empirical centroid of class c. - $d(\hat{\mu}_c, z)$ is the distance between the feature representation z and the centroid $\hat{\mu}_c$. - ϵ_c is the robustness margin for class c, which accounts for the uncertainty in the centroid estimation. - $\delta(z', c) = 1$ if z' is of class c, and 0 otherwise. - $w(c)$ are class weights that ensure balanced contributions from all classes.

3.4 Optimization Strategy

To compute the DRO-LT loss, an initial feature representation of the data is required for estimating class centroids. The training process involves two stages:

1. **Initial Training**: The model is first trained using standard cross-entropy loss to learn initial feature representations and centroids. 2. **Robust Training**: The DRO-LT loss is then introduced by combining it with the standard cross-entropy loss, allowing the model to learn robust representations that generalize well to tail classes.

The combined loss function used for training is:

$$L = \lambda L_{\text{CE}} + (1 - \lambda) L_{\text{Robust}} \tag{5}$$

where L_{CE} is the standard cross-entropy loss, and λ is a trade-off parameter that balances the two loss components.

3.5 Empirical Results

The authors evaluated their approach on several long-tailed visual recognition benchmarks, including CIFAR100-LT, ImageNet-LT, and iNaturalist. The results demonstrate that the DRO-LT loss consistently outperforms state-of-the-art methods in long-tail learning. It improves recognition accuracy for tail classes while maintaining high accuracy for head classes, showcasing the effectiveness of the proposed method in learning balanced representations.

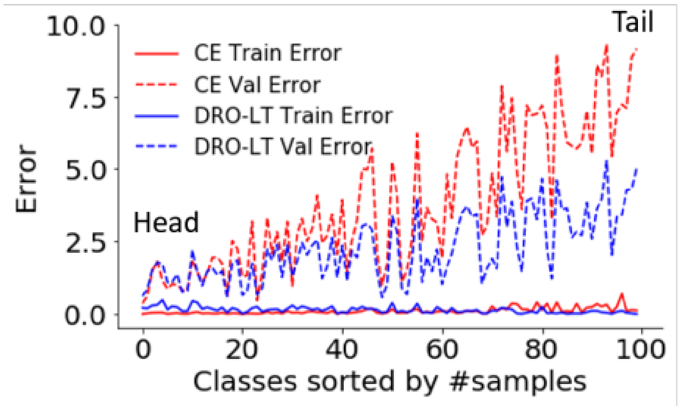


Figure 11: Performance comparison of DRO-LT with state-of-the-art methods on long-tailed benchmarks. The DRO-LT method achieves superior accuracy for both head and tail classes.

3.6 Conclusion

The paper introduces a novel approach for long-tail learning by focusing on distributionally robust optimization. The DRO-LT loss effectively addresses the challenges of unbalanced data by improving the learned representations for both head and tail classes. This robust representation learning method sets new state-of-the-art results on multiple benchmarks, indicating its potential for broader applications in unbalanced data scenarios.

4 A Deep Probabilistic Model for Customer Lifetime Value Prediction

This chapter summarizes the key ideas and solutions proposed in the paper "A Deep Probabilistic Model for Customer Lifetime Value Prediction" by Xiaojing Wang, Tianqi Liu, and Jingang Miao. The paper addresses the challenges of predicting Customer Lifetime Value (LTV) when dealing with highly skewed and zero-inflated data distributions. The authors propose a novel probabilistic approach to improve the accuracy and robustness of LTV predictions.

4.1 Challenges in Customer Lifetime Value Prediction

Predicting Customer Lifetime Value (LTV) is essential for businesses to understand the potential future revenue from their customers. However, LTV prediction presents significant challenges due to the nature of the data:

- **Skewed Distribution**: LTV data often exhibits a heavy-tailed, right-skewed distribution where most customers contribute little to no revenue, while a few high-value customers contribute disproportionately.
- **Zero-Inflation**: A large portion of customers may have zero lifetime value, representing one-time purchasers who do not return. This results in a high proportion of zeroes in the dataset, complicating the modeling process.

Traditional regression models, such as those using Mean Squared Error (MSE) loss, struggle with these challenges as they are sensitive to outliers and assume a normal distribution of errors.

4.2 Zero-Inflated Lognormal (ZILN) Distribution

To address the skewed and zero-inflated nature of LTV data, the authors propose modeling the distribution of LTV using a **Zero-Inflated Lognormal (ZILN) distribution**. This approach captures both the probability of a customer having zero LTV and the variability among returning customers who have a non-zero LTV.

The ZILN distribution combines a *point mass at zero* (to handle zero-inflation) with a *lognormal distribution* (to handle the positive skewness of non-zero LTV values). This model is particularly suitable for LTV prediction as it provides a flexible framework that can capture the distinct characteristics of the data.

4.3 ZILN Distribution Loss Function

The authors derive a loss function based on the negative log-likelihood of a ZILN-distributed random variable. This loss function effectively models the two main components of LTV:

- **The probability of zero LTV** (for customers who do not make any further purchases).
- **The lognormal distribution of non-zero LTVs** (for customers who do make additional purchases).

The ZILN loss function is defined as:

$$L_{\text{ZILN}}(x; p, \mu, \sigma) = -1_{\{x=0\}} \log(1-p) - 1_{\{x>0\}} \left(\log p - L_{\text{Lognormal}}(x; \mu, \sigma)\right) \quad (6)$$

where:

- x represents the observed LTV.
- p is the probability of the LTV being non-zero.
- μ and σ are the mean and standard deviation of the lognormal distribution for non-zero LTVs.
- $L_{\text{Lognormal}}(x; \mu, \sigma)$ is the negative log-likelihood of the lognormal distribution.

This loss function allows the model to learn from both zero and non-zero LTVs, accurately capturing the distribution of LTV data.

4.4 Handling Skewness and Zero-Inflation

The ZILN loss provides several key advantages for handling the skewed and zero-inflated nature of LTV data:

- **Capturing Zero-Inflation**: By modeling the probability of zero LTV directly, the ZILN loss effectively handles the high proportion of zeroes in the dataset, providing a more accurate representation of one-time purchasers.
- **Modeling Heavy Tails**: The lognormal component of the ZILN distribution captures the right-skewed, heavy-tailed nature of non-zero LTV values, allowing the model to account for the variability among returning customers.
- **Flexibility and Robustness**: The ZILN loss function provides a flexible framework that can be applied to both linear and non-linear models, enhancing robustness and improving generalization performance in the presence of skewed data.

4.5 Solution and Benefits of the ZILN Approach

The authors propose using the ZILN loss within a deep learning framework to leverage its flexibility and modeling power. This approach offers several benefits:

- **Unified Modeling**: The ZILN loss allows the model to perform both classification (predicting whether a customer will return) and regression (predicting the LTV of returning customers) in a unified manner.
- **Improved Prediction Accuracy**: By accurately modeling the zero-inflated and skewed nature of LTV data, the ZILN loss improves prediction accuracy, particularly for tail customers who contribute disproportionately to revenue.
- **Scalability and Adaptability**: The probabilistic nature of the ZILN model makes it scalable to large datasets and adaptable to various domains where similar data characteristics are present.

4.6 Conclusion

The paper introduces a deep probabilistic approach for predicting customer lifetime value using a novel Zero-Inflated Lognormal (ZILN) loss function. By effectively modeling the zero-inflated and heavy-tailed nature of LTV data, the ZILN approach provides a robust solution for businesses to predict future customer value accurately. This methodology addresses the limitations of traditional regression models, offering improved accuracy and adaptability for customer-centric strategies.

5 A Novel Approach to Predicting Customer Lifetime Value in B2B SaaS Companies

This chapter explores the methodologies proposed in the report "A Novel Approach to Predicting Customer Lifetime Value in B2B SaaS Companies" by Stephan Curiskis, Xiaojing Dong, Fan Jiang, and Mark Scarr. The authors present a flexible machine learning framework designed to predict Customer Lifetime Value (CLV) in the context of Business-to-Business (B2B) Software-as-a-Service (SaaS) companies. The proposed approach addresses several challenges unique to the B2B SaaS environment, including customer heterogeneity, multiple product offerings, and constrained temporal data.

5.1 Challenges in CLV Prediction for B2B SaaS

Predicting Customer Lifetime Value (CLV) is critical for B2B SaaS companies due to the longer sales cycles, higher customer acquisition costs, and diverse customer needs. The heterogeneity of customer behaviors and the variety of products offered add complexity to the prediction task. Furthermore, limited temporal data makes it challenging to forecast long-term customer value accurately.

5.2 Hierarchical Ensembled CLV Model

To address these challenges, the authors propose a hierarchical ensembled CLV model that leverages a combination of supervised learning techniques. The model is designed to improve prediction accuracy by integrating multiple layers of information and capturing the complex relationships between customer attributes and their future value.

5.2.1 Problem Framing

The CLV prediction problem is framed as a lump sum estimation across multiple products, allowing the model to utilize various supervised learning techniques and enhance feature richness. This approach is particularly effective in handling the constrained temporal data often found in B2B SaaS settings.

5.2.2 Hierarchical T-Period Model

The hierarchical model involves a two-stage process to predict CLV:

1. **T' Period Model**: The first stage involves training a model using historical data from n periods to predict short-term customer value (T' period). This stage focuses on capturing immediate trends and behaviors based on the available data.

2. **T Period Model**: The second stage maps T' period predictions to T period predictions using another model that incorporates slowly changing features, such as firmographics. This stage extends the short-term predictions to long-term forecasts, leveraging more stable customer characteristics.

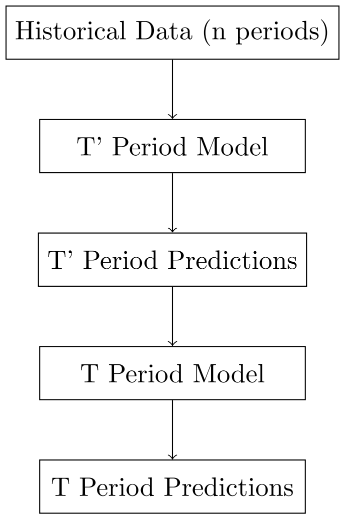


Figure 12: Hierarchical T-Period Model for CLV Prediction

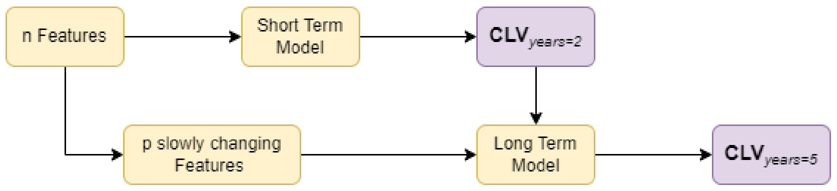


Figure 13: Diagram

5.3 Ensembled Customer Segment Model

Recognizing the diversity of CLV drivers across different customer segments, the authors adopt an ensembled approach. The dataset is segmented based on key features identified through error diagnostics, and different types of prediction models are applied to these segments. This segmentation strategy allows the model to capture unique patterns in customer behavior and product usage, leading to more accurate and tailored CLV predictions.

5.4 Conclusion

The hierarchical ensembled CLV model proposed in this report offers a robust solution for predicting customer lifetime value in B2B SaaS settings. By integrating multiple supervised learning techniques and leveraging a hierarchical approach, the model effectively addresses key challenges such as data constraints, customer heterogeneity, and multiple product offerings. This framework is adaptable to other contexts with similar challenges, providing valuable insights for marketing, customer retention, and resource allocation strategies.

article graphicx amsmath float

6 Implementation of Proposed Models

In this chapter, we discuss the implementation of the methodologies and models proposed in the reviewed papers. Instead of using neural networks (NN) as many of the original papers did, we chose to implement the models using XGBoost, a powerful gradient boosting framework. This decision was motivated by two key reasons that align with our project's requirements and constraints.

6.1 Rationale for Using XGBoost Over Neural Networks

Our choice to implement the models using XGBoost rather than neural networks was driven by practical considerations related to both the capabilities of existing models at BNP Paribas and the nature of our data.

6.1.1 Compatibility with BNP Paribas's AGBoost Model

Firstly, the models currently in development at BNP Paribas, specifically AGBoost, are variations built on top of XGBoost. AGBoost, while robust, currently lacks the flexibility to define custom loss functions—a feature that is heavily relied upon in many of the reviewed papers to handle specific challenges like data skewness and distributional robustness. However, AGBoost developers have indicated that they are working on incorporating this functionality in future updates.

Given this context, reimplementing the models using XGBoost makes it easier to transfer these implementations to AGBoost once custom loss function capabilities are added. By aligning our work with the existing infrastructure at BNP Paribas, we ensure a smoother integration and deployment process, reducing the overhead associated with adapting neural network-based models to the AGBoost framework.

6.1.2 Performance on Small Datasets

The second reason for choosing XGBoost over neural networks is related to the size of our datasets. Neural networks are known to perform well with large datasets due to their ability to capture complex patterns and interactions. However, they are also notorious for underperforming when applied to smaller datasets, where classical machine learning models like XGBoost often excel.

XGBoost is particularly well-suited for datasets with a limited number of observations, as it builds an ensemble of decision trees that can effectively handle both the variance and bias inherent in small datasets. By using XGBoost, we leverage its strengths in handling small data volumes, ensuring more robust and reliable model performance compared to what might be achieved with a neural network on the same data.

6.2 Implementation Strategy

To implement the models proposed in the reviewed papers, we adapted the methodologies to fit within the XGBoost framework. This involved translating the theoretical foundations and specific loss functions described in the papers into XGBoost-compatible implementations, focusing on capturing the essence of each approach while maintaining flexibility for future adaptations.

7 Implementation of the Quadratic Mean Loss

In this section, we explain the implementation of the quadratic mean loss function in the context of XGBoost, specifically tailored for regression tasks. The quadratic mean loss function is designed to manage skewed data by balancing the error contributions from different segments of the data distribution.

7.1 Quadratic Mean Loss Function

The original form of the quadratic mean loss, $R^Q_{\text{emp}}(w)$, is defined as:

$$R^Q_{\text{emp}}(w) = \sqrt{\frac{\left(\frac{\sum_{i=1}^{n_1} l(x_i,y_i,w)}{n_1}\right)^2 + \left(\frac{\sum_{i=n_1+1}^{n} l(x_i,y_i,w)}{n_2}\right)^2}{2}} \tag{7}$$

To simplify this for our specific implementation, we replace the general loss $l(x_i, y_i, w)$ with the Mean Squared Error (MSE) and set $n_1 = n_2 = \frac{n}{2}$, where n is the total number of observations. This gives us the modified quadratic mean loss function:

$$R^Q_{\text{emp}}(w) = \sqrt{\frac{\left(\frac{2}{n}\sum_{i=1}^{\frac{n}{2}}(\hat{y}_i - y_i)^2\right)^2 + \left(\frac{2}{n}\sum_{i=\frac{n}{2}+1}^{n}(\hat{y}_i - y_i)^2\right)^2}{2}} \tag{8}$$

7.2 Using A and B for Simplification

To further simplify the notation, let:

$$A = \frac{2}{n}\sum_{i=1}^{\frac{n}{2}}(\hat{y}_i - y_i)^2, \quad B = \frac{2}{n}\sum_{i=\frac{n}{2}+1}^{n}(\hat{y}_i - y_i)^2$$

With these definitions, the quadratic mean loss function can be rewritten as:

$$R^Q_{\text{emp}}(w) = \sqrt{\frac{A^2 + B^2}{2}} \tag{9}$$

7.3 Gradient and Hessian Calculation

To implement this loss function in XGBoost, we need to compute its gradient and Hessian with respect to the predicted values $\hat{y}_i$. The gradient and Hessian are derived as follows:

7.3.1 Gradient Calculation

The gradient of $R^Q_{\text{emp}}(w)$ with respect to $\hat{y}_i$ is given by:

$$\text{grad}_i = \frac{\partial R^Q_{\text{emp}}(w)}{\partial \hat{y}_i} = \frac{1}{2} \cdot (R^Q_{\text{emp}}(w))^{-0.5} \cdot C_i \cdot C'_i \tag{10}$$

where: - $C_i = A$ if $i \leq \frac{n}{2}$, otherwise $C_i = B$ - $C'_i = \frac{4(\hat{y}_i - y_i)}{n}$

7.3.2 Hessian Calculation

The Hessian of $R^Q_{\text{emp}}(w)$ with respect to $\hat{y}_i$ is:

$$\text{hess}_i = \left(-\frac{1}{4} \cdot (R^Q_{\text{emp}}(w))^{-1.5} \cdot C_i \cdot C'_i\right) + \frac{1}{2} \cdot (R^Q_{\text{emp}}(w))^{-0.5} \cdot \left((C'_i)^2 + \frac{4C_i}{n}\right) \tag{11}$$

By implementing these formulas for the gradient and Hessian, we can define a custom loss function in XGBoost that leverages the quadratic mean loss to better handle skewed datasets in regression tasks.

8 Implementation of the Deep Probabilistic Model for Customer Lifetime Value Prediction

In this section, we discuss the implementation of the deep probabilistic model for customer lifetime value (CLV) prediction as proposed in the paper "A Deep Probabilistic Model for Customer Lifetime Value Prediction." The primary challenge in implementing this model in XGBoost lies in the parametric nature of the loss function used in the original neural network (NN) implementation. Due to this complexity, we opted to implement the model using neural networks and evaluate its performance on our data. Additionally, we explored a variant of the model's idea by adapting XGBoost with a log-transformation-based loss function.

8.1 Challenges with Parametric Loss Functions in XGBoost

The loss function described in the paper is parametric, meaning it involves estimating parameters that define the distribution of the target variable, specifically the Zero-Inflated Lognormal (ZILN) distribution. In the context of neural networks, this loss function is easier to implement as the network can simultaneously learn the parameters and the mappings. However, implementing such a parametric loss function in XGBoost is more complicated due to the framework's design, which does not naturally accommodate parameter estimation as part of its objective function optimization.

8.2 Neural Network Implementation

Given the challenges associated with implementing a parametric loss function in XGBoost, we chose to implement the original model using a neural network. This approach allowed us to replicate the probabilistic loss function as proposed in the paper and test it on our dataset to evaluate its effectiveness in predicting customer lifetime value.

8.3 Adapting a Log-Transformation-Based Loss Function in XGBoost

To align more closely with the XGBoost framework and leverage its strengths, we explored an alternative approach inspired by the original paper. This approach involves using a log-transformation-based loss function, specifically:

$$L_{\log} = (\log(y) - \log(\hat{y}))^2$$

This log loss function captures the essence of a log transformation without explicitly transforming the target variable y. The advantage of using this loss function is that it enables the model to benefit from the log transformation's properties—such as compressing the range of target values and reducing the impact of outliers—while remaining within the flexible and efficient XGBoost framework.

8.4 Gradient and Hessian Calculation for Log Loss Function

To implement this log loss function in XGBoost, we need to calculate the gradient and Hessian:

8.4.1 Gradient Calculation

The gradient of the log loss function $L_{\log}$ with respect to $\hat{y}$ is:

$$\text{grad}_i = \frac{\partial L_{\log}}{\partial \hat{y}_i} = -\frac{2(\log(y_i) - \log(\hat{y}_i))}{\hat{y}_i} \tag{12}$$

8.4.2 Hessian Calculation

The Hessian of the log loss function $L_{\log}$ with respect to $\hat{y}$ is:

$$\text{hess}_i = \frac{\partial^2 L_{\log}}{\partial \hat{y}_i^2} = \frac{2(\log(y_i) - \log(\hat{y}_i))}{\hat{y}_i^2} + \frac{2}{\hat{y}_i^2} \tag{13}$$

8.5 Implementation in XGBoost

The Python implementation of the log loss function in XGBoost involves defining a custom objective function that returns these gradients and Hessians:

```
import xgboost as xgb
import numpy as np

def log_loss(y_true, y_pred):
    # Avoid division by zero and log(0)
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, None)

    # Compute gradient and Hessian
    grad = -2 * (np.log(y_true) - np.log(y_pred)) /
        y_pred
    hess = 2 * ((np.log(y_true) - np.log(y_pred)) / (
        y_pred**2) + 1 / y_pred)

    return grad, hess

# Example usage with XGBoost
params = {'objective': log_loss, 'max_depth': 5, 'eta'
    : 0.1}
```

```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

model = xgb.train(params, dtrain, num_boost_round=100,
     evals=[(dtest, 'test')])
```

By implementing this log-transformation-based loss function, we leverage XGBoost's efficient gradient boosting while capturing the benefits of log transformation. This approach allows us to approximate the idea presented in the original paper, adapting it to fit the strengths of the XGBoost framework.

9 Exploratory Implementations

In addition to the methodologies derived from the reviewed papers, we conducted several exploratory implementations to evaluate alternative approaches for customer lifetime value (CLV) prediction. These methods were not directly inspired by any specific paper but were tested to explore their potential effectiveness in handling skewed and heavy-tailed data distributions. The following sections detail the three additional implementations: quantile regression, gamma distribution regression, and Tweedie distribution regression, all using XGBoost.

9.1 Quantile Regression with XGBoost

Quantile regression is a statistical technique used to estimate the conditional quantiles of a response variable. Unlike ordinary least squares regression, which estimates the mean of the dependent variable, quantile regression estimates specific quantiles (e.g., the median or the 90th percentile). This makes quantile regression particularly useful in situations where the data is skewed or has outliers, as it provides a more comprehensive view of the possible outcomes.

9.1.1 Quantile Loss Function

For quantile regression, the loss function is defined as the quantile loss, which is given by:

$$L_{\text{quantile}}(\hat{y}, y) = \sum_{i=1}^{n} \rho_\tau(y_i - \hat{y}_i)$$

where ρ_τ is the quantile function defined as:

$$\rho_\tau(u) = \begin{cases} \tau u & \text{if } u \geq 0 \\ (\tau - 1)u & \text{if } u < 0 \end{cases}$$

Here, τ is the desired quantile (e.g., $\tau = 0.5$ for the median). The loss function focuses on minimizing the distance between the predicted and actual quantiles, providing a robust prediction that is less sensitive to outliers.

9.1.2 Implementation in XGBoost

To implement quantile regression in XGBoost, we need to define a custom objective function that calculates the gradient and Hessian for the quantile loss:

```
import numpy as np

def quantile_loss(y_true, y_pred, quantile=0.5):
    error = y_true - y_pred
    grad = np.where(error > 0, -quantile, 1 - quantile
        )
    hess = np.ones_like(y_true)

    return grad, hess
```

This custom loss function can be used in XGBoost to perform quantile regression by specifying it as the objective function.

9.2 Gamma Distribution Regression with XGBoost

Gamma regression is used to model continuous, positive-valued target variables that are skewed to the right. It assumes that the response variable follows a gamma distribution, which is often suitable for modeling financial data, such as CLV, where values are positive and heavily skewed.

9.2.1 Gamma Loss Function

In gamma regression, the loss function is derived from the gamma distribution's log-likelihood:

$$L_{\text{gamma}}(\hat{y}, y) = \sum_{i=1}^{n} \left(y_i \frac{\log(y_i) - \log(\hat{y}_i)}{\hat{y}_i} - \log(y_i) \right)$$

The goal is to minimize the difference between the observed and predicted values while accounting for the variance structure inherent in gamma-distributed data.

9.2.2 Implementation in XGBoost

To implement gamma regression in XGBoost, we define a custom objective function that calculates the gradient and Hessian for the gamma loss:

```
def gamma_loss(y_true, y_pred):
    grad = (y_pred - y_true) / y_pred
    hess = y_true / (y_pred ** 2)

    return grad, hess
```

This custom loss function allows XGBoost to optimize the model parameters for gamma-distributed target variables, making it suitable for skewed financial data like CLV.

9.3 Tweedie Distribution Regression with XGBoost

The Tweedie distribution is a family of distributions that includes several common distributions (normal, Poisson, gamma) as special cases. Tweedie regression is useful for modeling data that are mixtures of zeros and positive continuous values, such as insurance claims or customer spend data. It is characterized by a power variance function, making it suitable for a wide range of data types.

9.3.1 Tweedie Loss Function

The Tweedie loss function is derived from the Tweedie distribution's log-likelihood:

$$L_{\text{tweedie}}(\hat{y}, y) = \sum_{i=1}^{n} \left(\frac{y_i \hat{y}_i^{1-p}}{1-p} - \frac{\hat{y}_i^{2-p}}{2-p} \right)$$

where p is the Tweedie variance power parameter. The parameter p determines the specific form of the distribution: - $p = 0$: Normal distribution. - $0 < p < 1$: No distribution. - $p = 1$: Poisson distribution. - $1 < p < 2$: Compound Poisson-gamma distribution (useful for CLV). - $p = 2$: Gamma distribution. - $p > 2$: Inverse Gaussian distribution.

9.3.2 Implementation in XGBoost

To implement Tweedie regression in XGBoost, we define a custom objective function that calculates the gradient and Hessian for the Tweedie loss:

```
def tweedie_loss(y_true, y_pred, variance_power=1.5):
    grad = -y_true * np.power(y_pred, -variance_power)
        + np.power(y_pred, 1 - variance_power)
    hess = y_true * variance_power * np.power(y_pred,
        -variance_power - 1) + (1 - variance_power) *
        np.power(y_pred, -variance_power)

    return grad, hess
```

This custom loss function can be used in XGBoost to perform Tweedie regression, allowing for modeling of mixed discrete-continuous data distributions.

9.4 Conclusion

These exploratory implementations—quantile regression, gamma distribution regression, and Tweedie distribution regression—provide alternative methods for modeling skewed and heavy-tailed data using XGBoost. By testing these methods, we aim to identify the most effective approach for predicting customer lifetime value, given the specific characteristics of our dataset.

10 Results and Discussion

This section presents the results of the various models and implementations tested for customer lifetime value (CLV) prediction and classification. We evaluate the performance of each model using the coefficient of determination (R^2) for regression tasks and the F1 score for classification tasks. These results specifically pertain to professional clients; the results for enterprise clients will be discussed later.

10.1 Regression Results

We tested several models and approaches for predicting CLV, including standard AGBoost, quadratic mean regression, quantile regression, a neural network with ZILN loss, Tweedie regression, and a custom log loss function in XGBoost. The R^2 values for each model are as follows:

- **Standard AGBoost**: $R^2 = 12$
- **Standard AGBoost with Optimized Parameters using Genetic Algorithm**: $R^2 = 14$
- **Quadratic Mean Regression**: $R^2 = 18$
- **Segmented Quantile Regression**: $R^2 = 61$
- **Neural Network with ZILN Loss**: $R^2 = 22$
- **XGBoost Tweedie Regression**: $R^2 = 16$
- **Segmented Prediction in Two Phases**: $R^2 = 17$
- **XGBoost with Log Loss Function**: $R^2 = 12$

These results indicate that segmented quantile regression achieved the highest R^2 value, significantly outperforming the other models. The neural network with ZILN loss also showed strong performance, while the standard AGBoost model, even with optimized parameters, performed less effectively.

10.2 Classification Results

For the classification tasks, we evaluated the models based on the F1 score, which considers both precision and recall. Different sampling and weighting strategies were applied to improve model performance on the skewed data:

- **Regular Model**: F1 Score = 13
- **Using SMOTE (Synthetic Minority Over-sampling Technique)**: F1 Score = 25
- **Random Under-Sampling**: F1 Score = 22

- **Class Weight Adjustment**: F1 Score = 24
- **Threshold Moving**: F1 Score = 17.5

Applying SMOTE resulted in the highest F1 score, suggesting that oversampling the minority class effectively improves model performance in the context of imbalanced datasets. Class weight adjustment and random under-sampling also showed improvements over the baseline model.

10.3 Genetic Algorithm for Parameter Optimization

A genetic algorithm (GA) is an optimization technique inspired by the process of natural selection. It is used to find approximate solutions to optimization and search problems by evolving a population of candidate solutions towards better solutions over time.

10.3.1 How Genetic Algorithms Work

Genetic algorithms work by mimicking the biological processes of selection, crossover, and mutation:

- **Initialization**: A population of candidate solutions (individuals) is generated randomly.
- **Selection**: Individuals are selected from the current population based on their fitness scores, which evaluate how well they solve the problem at hand.
- **Crossover**: Pairs of selected individuals are combined to produce offspring, mixing their features to create new solutions.
- **Mutation**: Some offspring undergo random changes to introduce diversity and explore new areas of the solution space.
- **Iteration**: The new generation of solutions replaces the old one, and the process repeats until a stopping criterion is met (e.g., a maximum number of generations or a satisfactory fitness level).

10.3.2 Using Genetic Algorithms for Parameter Optimization

In the context of machine learning, genetic algorithms can be used to optimize hyperparameters of models, such as learning rates, tree depths, and regularization parameters in XGBoost or AGBoost. By evolving a population of hyperparameter sets, the algorithm identifies combinations that lead to the best model performance.

For this study, we used a genetic algorithm to optimize the parameters of the AGBoost model, resulting in an improved R^2 from 12 to 14. This demonstrates the potential of genetic algorithms to enhance model performance through effective hyperparameter tuning.

10.4 Conclusion

The results demonstrate the varying effectiveness of different models and techniques in predicting CLV and classifying customers. Segmented quantile regression and the neural network with ZILN loss provided the best results for regression, while SMOTE significantly improved classification performance. The use of genetic algorithms for parameter optimization showed promise in enhancing model accuracy, suggesting that further exploration of optimization techniques could yield additional gains.