CONVOLVE 3.0

Optimizing Customer Engagement with Personalized Time Slots

Team Ping Pong Pajeets

Ankit Raj

Kishlay Raj

Divyam Kulshrestha

Table of Contents

PROBLEM STATEMENT

- **Problem Overview :**
  - Banks have limited opportunities to send marketing emails due to regulatory and cost constraints.
  - To maximize impact, emails must be sent when customers are most likely to engage.
  - Traditional bulk email campaigns fail because they don’t consider individual customer behavior, leading to low engagement rates.

- **Main Objective :**
  - Build a robust ML model that can rank 28 time slots for each customer from best to worst based on engagement probability.
  - The goal is personalization at scale, ensuring each customer receives emails at their optimal time without overwhelming them.
  - Ensure scalability so that the solution can work efficiently for a large customer base and potentially adapt to real-world deployment.

◆ Key Challenges identified :

- Large-Scale Data Processing : 6 months of communication history across 2 lakh+ customers resulted in millions of rows of data.
- Unlike standard classification tasks, where we predict a single outcome, this is a ranking problem. Traditional ML models struggle with ranking tasks, requiring specialized approaches.
- Sparse & Imbalanced Data : Many emails were sent, but only a fraction were opened, leading to an imbalance in engagement labels.

◆ Impact of the Proposed Solution :

- Increased Engagement & Customer Satisfaction : Personalized email timings ensure non-intrusive communication, leading to to higher open rates and better customer interaction.
- Optimized Marketing Efficiency & ROI : We ensure maximum impact with fewer emails, leading to higher conversion rates and better ROI.
- Regulatory Compliance & Scalability : Since banks have strict limits on communication frequency, sending emails at optimal times ensures maximum impact within the regulatory limits. Our model can also be extended to SMS and push notifications

THE DATASET

Communications History

87,97,911

Rows

8

Columns

`Offer_id, product_category, open_timestamp, send_timestamp`

Main Features

Customer Demographics (CDNA)

12,85,402

Rows

303

Columns

`v43(marital status), v102(employment status), batch_date`

Main Features

Dataset Challenges & Complexity:

- Large-Scale Data Processing: Millions of rows, making it memory-intensive to process, requiring feature selection.
- Sparse Data & Missing Engagement Information: Many customers received emails but never opened them, leading to a high number of null values in Open_timestamp.
- Challenges in Merging Datasets: CDNA was recorded weekly, while emails were event-based, requiring precise alignment.

This large dataset made data preprocessing extremely crucial.

DATA PRE-PROCESSING

Memory Limitations

PROBLEMS

- Memory Constraints : Kaggle's RAM limitations are insufficient for handling the dataset, resulting in processing challenges.
- System Instability : Frequent crashes occur due to excessive memory usage, disrupting workflow and analysis.

PROCESS

- Tried to split the CSV into 10 parts to clean and remove unnecessary columns separately. However, during merging, we faced **memory issues.**
- What Worked? : First, we clean the data and then proceed with the rest of the tasks.
  However, if RAM usage starts to exceed its limit, we process a small portion at a time. As soon as we notice RAM is getting close to the limit, we save the progress as a CSV file. We then start a new notebook and continue from that saved point.

Data Cleaning

PROBLEMS

- Most rows were empty or had excessive missing values, with the same data represented in various formats (e.g., "IN", "India", "india")
- Duplicate or irrelevant columns and columns with only one unique value or highly skewed data.

APPROACH

- We unified similar categories (e.g., "IN" ,"India", "india"), manually cleaned noisy columns, and combined redundant information to simplify the dataset.
- Removed:
  - Columns with only one unique value.
  - Columns where more than 80% of entries were NaN.
  - Highly skewed columns (dominant single value across entries).
  - Repeated columns containing duplicate information.

Data Cleaning

!

!

Dimensionality Reduction Methods

Methods	How It Works	Why We Rejected It
LDA (Linear Discriminant Analysis)	Maximizes class separability using labeled data	Needs labeled data; our problem is unsupervised.
t-SNE (t-Distributed Stochastic Neighbor Embedding)	Focuses on preserving local structure (good for visualization)	Computationally expensive, not suitable for feature selection.
UMAP (Uniform Manifold Approximation and Projection)	Similar to t-SNE, balances global & local structure	Stochastic, not deterministic, results may vary.
Autoencoder(Neural Networks)	Learns compressed representations using deep learning	To obtain useful results, a lot of training would be needed.

PCA (Principal Component Analysis)

PROBLEMS

- The original dataset contained an extensive set of customer demographic features, leading to high dimensionality that increased the risk of overfitting and slowed down computations.

APPROACH

- Why PCA?
  - Efficient: Works well with large datasets.
  - Prevents Overfitting: Reduces noise and collinearity.
  - Computationally Cheaper: Faster than deep-learning-based techniques.
- We used PCA to compress our over-300 demographic features into a more manageable and informative representation.
- PCA reduces high-dimensional data into a smaller set of uncorrelated features, lowering computational costs and overfitting risk while preserving the key patterns for improved model performance and interpretability.

Our final dataset comprised 69 features, ready for merging.

DATA MERGING

Data Merging Challenges:
Merging was difficult because the communication history (event-level) and customer CDNA (periodically updated) had mismatched row counts and granularity.

Converting Timestamps to Time Slots:
We created a function to map both open and send timestamps into one of 28 weekly time slots based on time of day and day of the week.

Merging Datasets:
We merged the datasets using Customer Code and aligned the batch month with the send timestamp month, creating a unified dataset of interaction history and customer-level features.

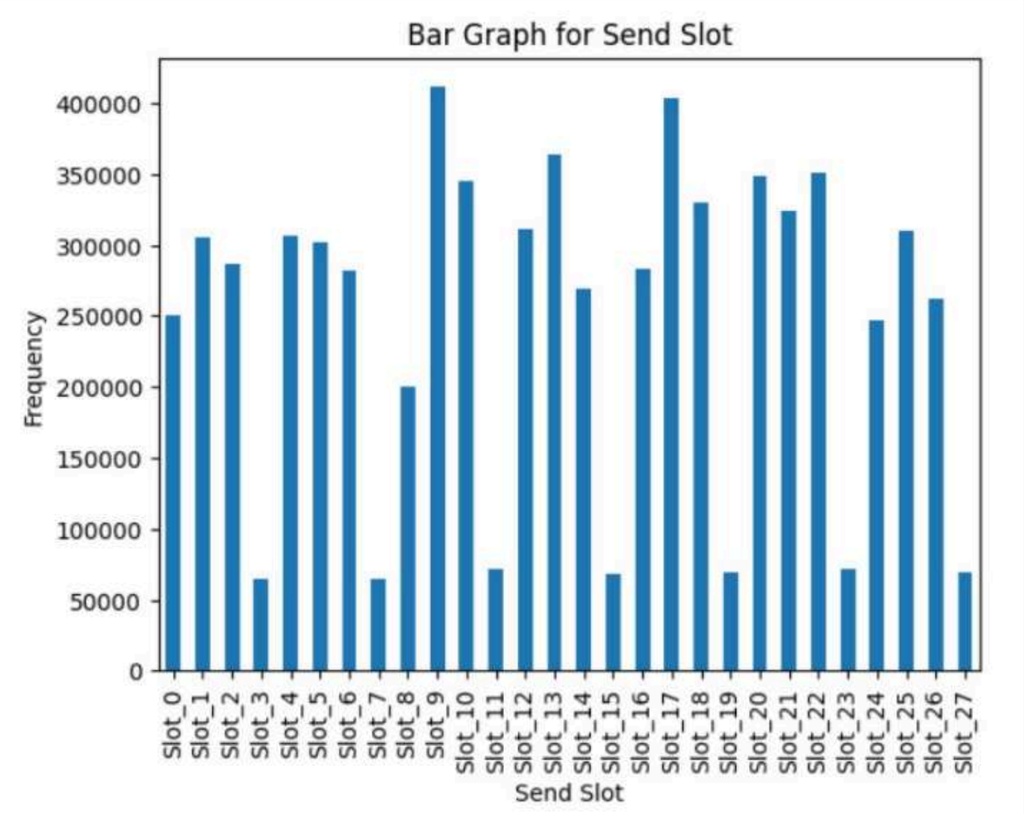
Converting Timestamps to Time Slots:

We created a function to map both open and send timestamps into one of 28 weekly time slots based on time of day and day of the week.

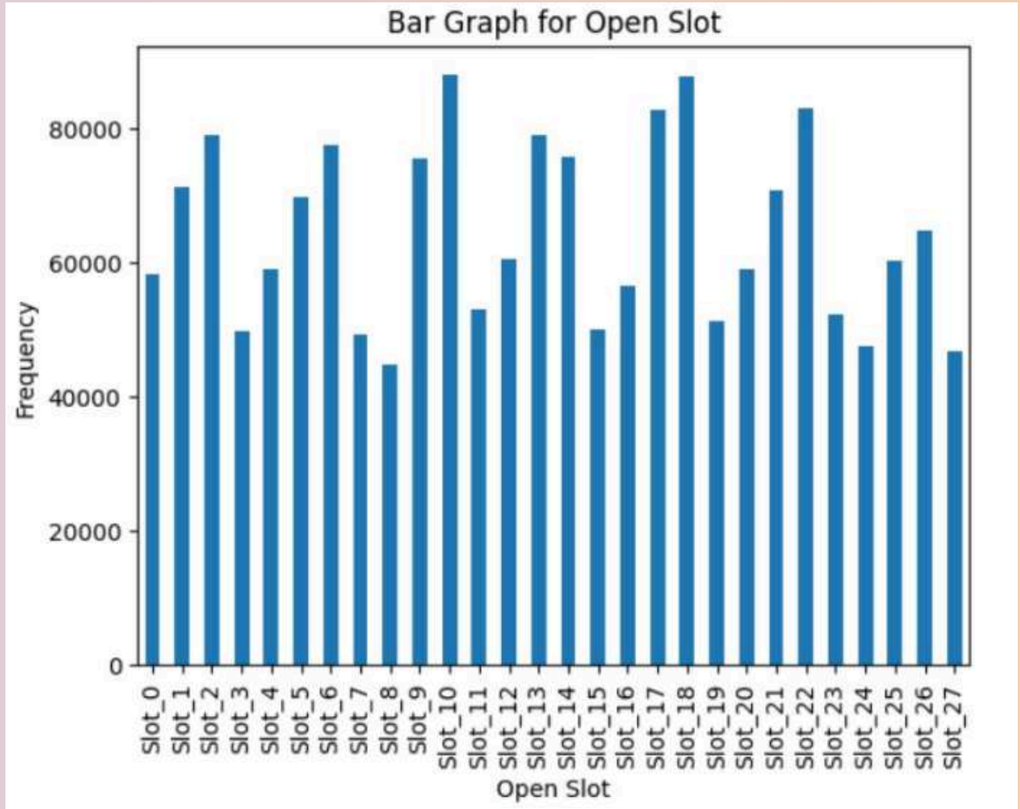
```
def calculate_slots(timestamps):
    """
    Vectorized function to calculate slots (1-28) for given timestamps.
    """
    day_of_week = timestamps.dt.weekday
    hour = timestamps.dt.hour
    slot_within_day = np.select(
        condlist=[
            (hour >= 9) & (hour < 12),
            (hour >= 12) & (hour < 15),
            (hour >= 15) & (hour < 18),
            (hour >= 18) & (hour < 21),
        ],
        choicelist=[1, 2, 3, 4],
        default=np.nan
    )
    overall_slot = (day_of_week * 4 + slot_within_day).astype('Int64')
    return overall_slot
```

DATA INSIGHTS

SLOT DISTRIBUTION ANALYSIS



This bar graph shows the distribution of emails across four daily slots, with fewer emails sent in the last slot, indicating reduced activity. Most emails were sent during the middle slots, reflecting peak engagement. Additionally, the mid-week saw higher email volumes, highlighting a clear usage pattern.



This graph shows email opening patterns across four daily slots, with fewer openings in the last slot of each day. Similar to the send slot graph, peak activity is seen in mid slots, aligning with mid-day and mid-week engagement trends.

Send_Open_Insights

Difference between Sending and Opening time:

Why It Matters?

- Helps optimize future send times for better engagement.
- Identifies gaps between delivery and actual interaction.
- Improves A/B testing and personalization strategies.

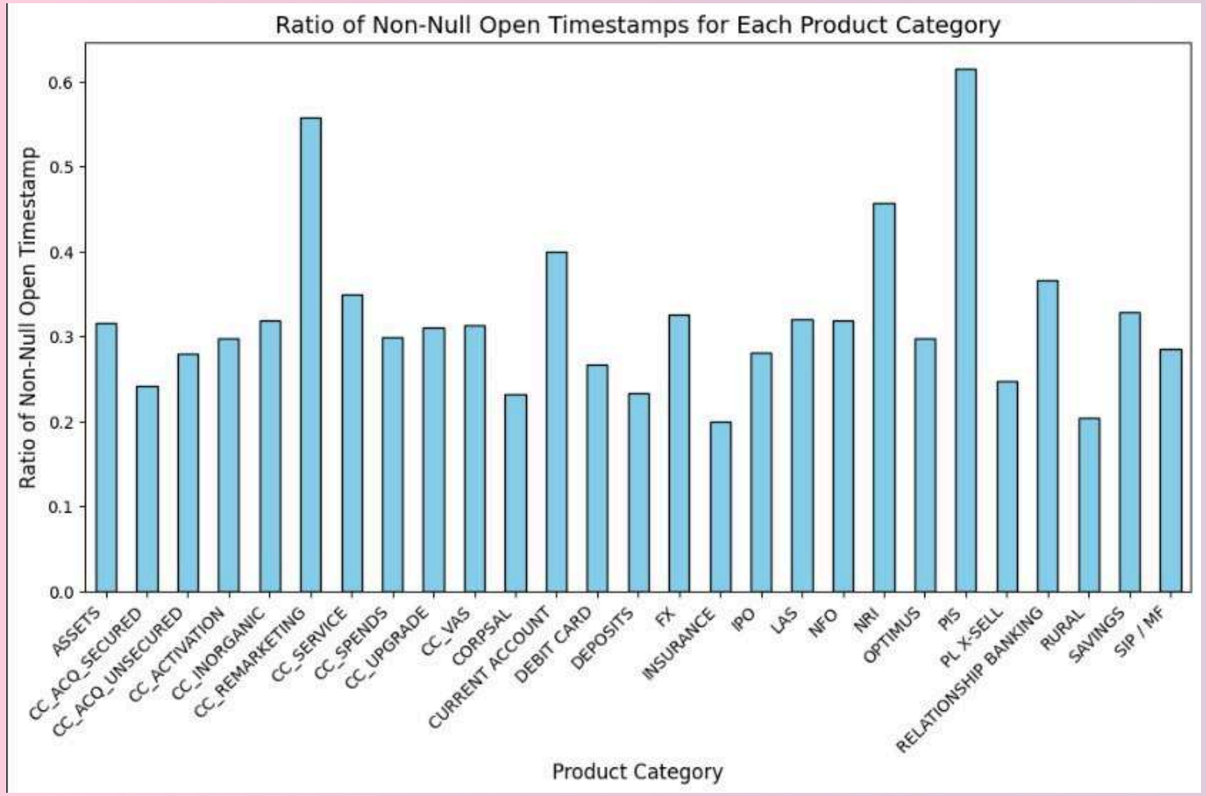
Before Removing Outliers:

- Mean Difference: 12 hrs 10 min 4 sec
- Median Difference: 3 hrs 2 min 23 sec
- Max Difference: 100 hrs 0 min 0 sec

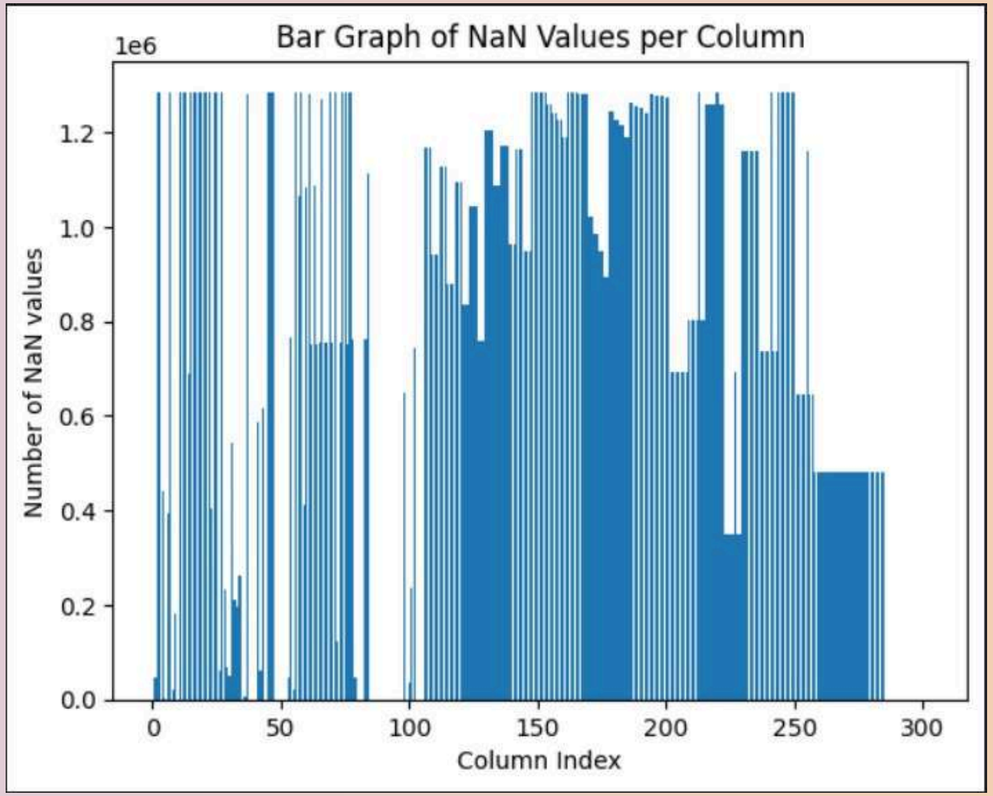
After Removing Outliers (Assuming > 100 hrs as Outliers)

- Mean Difference: 24 hrs 22 min 30 sec
- Median Difference: 3 hrs 51 min 26 sec
- Max Difference: 473 hrs 51 min 58 sec

Some More Insights



The chart shows PIS (portfolio investment scheme) and Credit Card Remarketing as high performers, with steady engagement in categories like CURRENT_ACCOUNT. However, low ratios in INSURANCE and RURAL categories highlight areas needing improved strategies.



This bar graph represents the distribution of NaN values across columns in our dataset. It highlights the sparsity of the dataset, with some columns being almost entirely empty, suggesting a need for careful preprocessing .

PATH TO THE FINAL MODEL

Model Tried : **Random Forest**

Why not Random Forest?

1. **Computationally Expensive:**
   - Training multiple independent decision trees requires high memory and processing power, making it inefficient for very large datasets and making predictions slow.
2. **Inefficient for Sparse & Imbalanced Data:**
   - It does not adapt well to sparse datasets where most entries are missing or where class distribution is highly imbalanced.
3. **Difficult to Tune for Large Datasets:**
   - Hyperparameter tuning for number of trees, depth, and splits becomes slow and resource-intensive as dataset size grows.

Conclusion:

Random Forest was rejected because it is slow, inefficient for large datasets, hard to tune, and struggles with sparse, high-dimensional data.

Model Tried : **Neural Networks**

Why Not Neural Networks?

1. Computationally Expensive:
   - Training deep learning models requires high processing power (GPUs/TPUs), making them inefficient for very large datasets.
2. Difficult to Interpret:
   - Unlike traditional models, neural networks act as a black box, making it hard to understand feature importance.
3. Slow Training & Hyperparameter Tuning:
   - Finding the right architecture, number of layers, learning rate, and activation functions takes significant time and resources.

Conclusion:

- Neural networks were rejected because they are computationally expensive, require large labeled data, have a high risk of overfitting, lack interpretability, and are slow tune.

Model Implemented : **XGBoost**

Why XGBoost?

1. **Fast & Scalable**: Handles large datasets efficiently, unlike Random Forest (slow).
2. **Regularization (L1 & L2)**: Prevents overfitting, Neural Networks need manual tuning.
3. **Handles Missing Data**: Learns best splits automatically, unlike Random Forest & Neural Networks (require imputation).
4. **Gradient Boosting Advantage**: Sequential learning improves predictions, while Random Forest trains trees independently.
5. **Feature Importance & Interpretability**: Unlike Neural Networks (black box) and Random Forest (no fine control), XGBoost provides clear feature rankings.
6. **Memory Efficient**: Uses parallel processing, unlike Random Forest (memory-heavy) and Neural Networks (high RAM usage).

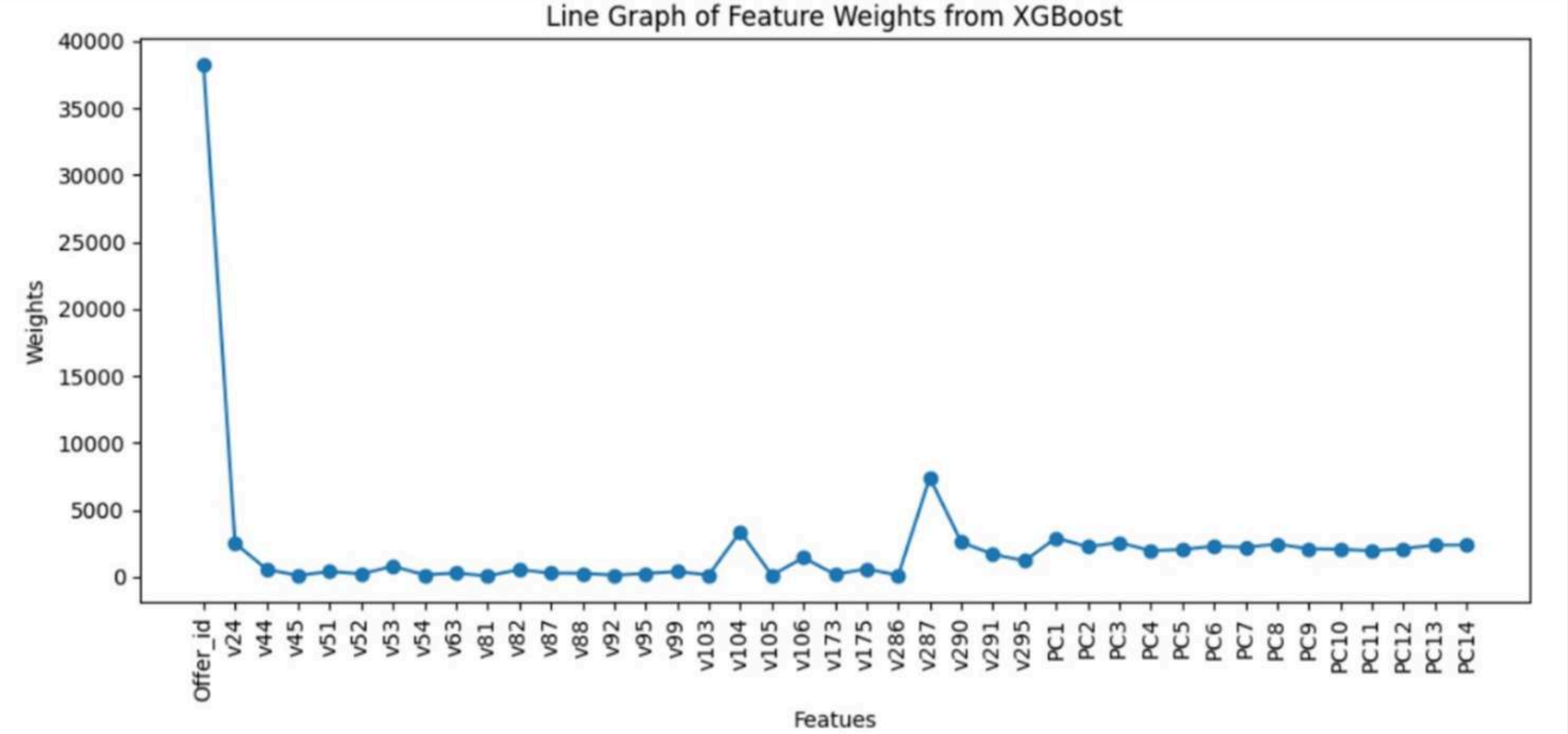
Conclusion: XGBoost is faster, scalable, less memory-intensive, and interpretable, making it the best choice.

- Implementation & Model Training

We preprocessed and merged datasets, transforming timestamps into 28 time slots for better temporal analysis. Using XGBoost (multi:softprob), we trained a model with max_depth=6, learning_rate=0.1, and n_estimators=100, optimizing for multi-class classification with merror as the evaluation metric.

- Optimization & Outcomes

Hyperparameters were fine-tuned using cross-validation and Mean Average Precision, ensuring optimal performance. Feature importance analysis revealed key patterns, allowing us to rank time slots effectively. This approach improved model accuracy and provided actionable insights for better customer engagement.



- We increased the regularization parameters, specifically alpha (L1 regularization) and lambda (L2 regularization), to reduce the dominance of Offer_id.
- This adjustment encouraged the model to distribute feature importance more evenly, allowing it to rely on other predictors.
- As a result, the model demonstrated better generalization and improved performance on unseen data.

Advanced Modeling & Future Enhancements

NEXT-GENERATION MODELING & FUTURE ROADMAP

- **Leveraging Time Series Data**

With detailed time series data like transaction logs and login patterns, models like LSTM and GRU can capture behavioral trends, enabling dynamic predictions that adapt over time.

- **Hybrid Modeling Approach**

We plan to combine XGBoost for static features with an RNN for sequential data. This hybrid approach leverages the strengths of both tree-based models and deep learning to improve time-dependent predictions.

- **Enhanced Temporal Feature Engineering**

Techniques like lag features, moving averages, and trend analysis will help the model detect short-term changes and long-term trends, refining predictions for optimal customer engagement.

DEPLOYMENT & SCALABILITY

- **A/B Testing for Model Validation:**

• Future deployments will include A/B testing to compare the new model's performance against current systems in a controlled environment.
• This method will allow us to validate improvements in real-time engagement metrics and make data-driven decisions before a full rollout.

- **Scalable Real-Time Architecture :**

We plan to deploy our model using a microservices & a cloud-based architecture that supports real-time data ingestion and prediction.

- **Extending to Additional Banking Services :**

Beyond email communications, the model can be extended to optimize customer interactions across various channels, such as mobile notifications, SMS, and online banking alerts.

THANK YOU!