Mobile Ad Click-Through Rate Prediction

Final Project

Absera Temesgen Yihunie

NEW YORK UNIVERSITY

ECON-UB 232 Data Bootcamp, F24

JF Koehler

Dec 14, 2024

Introduction

Overview of the data, predictive task, and summary findings

The Avazu Click-Through Rate (CTR) dataset, sourced from Kaggle, is a dataset for studying user interaction with mobile advertisements. For my final project, I aim to develop a predictive model that accurately determines whether a user will click on a mobile ad based on the features provided in the dataset. The target variable is binary—indicating whether an ad was clicked (1) or not clicked (0)—and the features include categorical attributes such as device type, ad placement information, site or app category, and the timestamp of the ad impression.

Dataset Background

The dataset was originally created for a Kaggle competition aimed at improving ad click prediction systems, which are crucial in online advertising for "measuring performance and optimizing strategies like sponsored search and real-time bidding" (Kaggle page). The dataset includes 11 days' worth of ad impression data, challenging participants to develop models that outperform standard classification algorithms. Submissions were evaluated using Logarithmic Loss.

Predicting click-through rates is an exciting challenge that I find particularly interesting. Through this project, I hope to explore patterns in user interactions and develop a model that can guide smarter advertising strategies.

Dataset Description

The dataset is a CSV file with 1 million rows and 24 columns, each representing details about ad impressions. While the data is well-organized, I will need to handle categorical columns by encoding to prepare it for modeling.

Data Description

Data columns and description

The dataset consists of a mix of many categorical features inclduing the target variable **click** (indicating whether an ad was clicked or not) and other variables like **hour** and **C1 to C21** that provide additional context about the ad impression. Other features include **site_id**, **app_id**, **device_model**, and others, representing specific details about the ad, site, app, and device used. The data is well-structured, with no missing values, but the categorical features will need encoding before being used in a predictive model.

Description of features

Feature	Description
ID	Unique ID for each ad
Click	True or False if an ad was clicked
Hour	Time stamp of when the ad was displayed
Device ID	The interacting device unique ID
Device IP	IP address of the device
Device model	Model of the device: iphone, samsung etc..
Connection type	Type of internet connection wifie, cellular etc..
Device type	Type of the device
Site ID	The ID of the website visited
App ID	The ID of the app used by user
Site category	Category of the site
App category	Category of the app
Site domain	Domain of the site interacted with
App domain	Domain of the app
Banner position	Position of the ad displayed on the screen
C1, C14-C21	Anonymized features

Table: List of feature of the dataset and their short descriptions

Below is more information about each feature.

```
 #   Column            Non-Null Count    Dtype
---  ------            --------------    -----
 0   id                1000000 non-null  float64
 1   click             1000000 non-null  int64
 2   hour              1000000 non-null  int64
 3   C1                1000000 non-null  int64
 4   banner_pos        1000000 non-null  int64
 5   site_id           1000000 non-null  object
 6   site_domain       1000000 non-null  object
 7   site_category     1000000 non-null  object
 8   app_id            1000000 non-null  object
 9   app_domain        1000000 non-null  object
 10  app_category      1000000 non-null  object
 11  device_id         1000000 non-null  object
 12  device_ip         1000000 non-null  object
 13  device_model      1000000 non-null  object
 14  device_type       1000000 non-null  int64
 15  device_conn_type  1000000 non-null  int64
 16  C14               1000000 non-null  int64
 17  C15               1000000 non-null  int64
 18  C16               1000000 non-null  int64
 19  C17               1000000 non-null  int64
 20  C18               1000000 non-null  int64
 21  C19               1000000 non-null  int64
 22  C20               1000000 non-null  int64
 23  C21               1000000 non-null  int64
 24  weekday           1000000 non-null  int64
```

The dataset contains features with varying numbers of unique values. For example, **id** has 1,000,000 unique values, which makes it unsuitable for predicting click behavior, as it does not provide any meaningful differentiation between the classes. Similarly, features like **device_id** and **device_ip** have large numbers of unique values, making them less useful for prediction. These features are likely to cause overfitting due to their high cardinality, so they can be removed or transformed into more useful representations. In contrast, features with fewer unique values, such as **device_type** (5 unique values) and **weekday** (7 unique values), may provide valuable insights for prediction and can be used directly in the model after encoding.

Here is the count of unique values for each feature.

```
Feature             unique count
-----                --------
id                   1000000
```

```
click                  2
hour                  24
C1                     7
banner_pos             7
site_id             2651
site_domain         2871
site_category         22
app_id              3157
app_domain           198
app_category          26
device_id         150501
device_ip         555248
device_model        5168
device_type            5
device_conn_type       4
C14                 2243
C15                    8
C16                    9
C17                  420
C18                    4
C19                   66
C20                  163
C21                   60
weekday                7
```

Summary of table:

- Low-cardinality features (e.g., **banner_pos**, **device_type**): Likely to be directly useful for modeling.
- High-cardinality features (e.g., **device_id**, **site_id**): Require dimensionality reduction (e.g., target encoding) to avoid inflating model complexity.
- Time-related features (hour, weekday): Important for identifying temporal trends and cyclic patterns in user behavior.
- Anonymized features (**C1–C21**): Need to explore each one independently to determine their impact on click.

Let's begin the analysis with the **site_category** feature. On the left, we observe the distribution of the unique site categories, where it is shown that a few categories dominate the data. Specifically, four categories have a much higher representation than the others. On the right, we see the average click rate for each site category. It's clear that some categories show significantly higher click rates, which indicates that they play a role in predicting whether an ad will be clicked or not. These categories will be valuable features in building a predictive model, since they provide important information into the click rate.

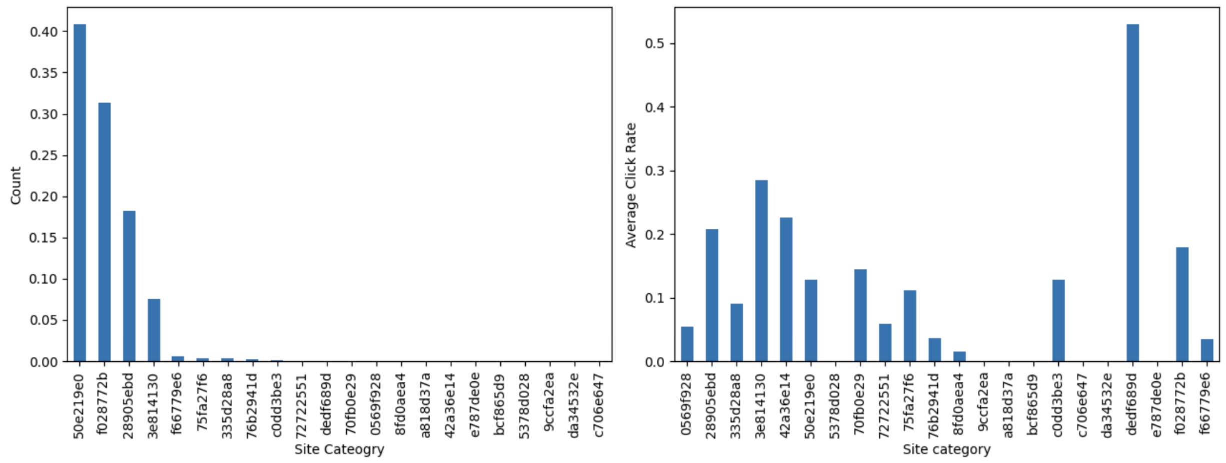

Figure: *Left Plot:* Class Counts by Site Category. *Right Plot*: Average Click Rate by Site category.

Similarly, the **app_category** feature shows a similar trend to the site category. The distribution of app categories reveals that a few categories dominate the dataset, just like with site categories. I have done the same analysis for the rest of the categorical features and observed the same pattern except for device_ip which has too many unique values and probably will not give too much insight in the prediction.

Now, let's take a look at **banner_position**. This feature is largely dominated by just two positions, but what stands out is that some positions have a click-through rate as high as 30%. This indicates that banner position may play a significant role in determining whether an ad is clicked. Given this information, it’s good idea to keep banner position in the training features list.

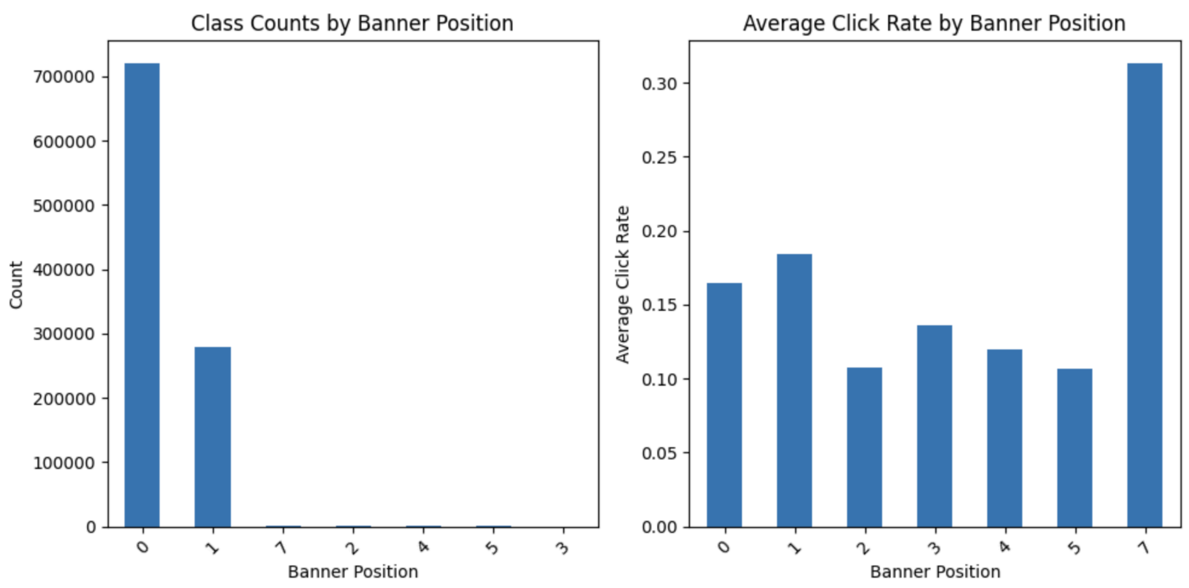

Figure: Banner Position Analysis. ***Left Plot:*** Class Counts by Banner Position — The majority of the data falls under banner position 0, followed by position 1. ***Right Plot***: Average Click Rate by Banner Position — While banner position 0 and 1 dominate in counts, the highest click rate is observed at position 7.

In the figure below, on the left, we can see the click count throughout the day, which shows how click behavior varies at different hours. The data reveals clear patterns, with certain hours seeing significantly higher click counts. On the right, we observe the click distribution across the week, where certain days show higher activity. These temporal features—hour of the day and day of the week—could be crucial differentiators and will be usefed in the training.

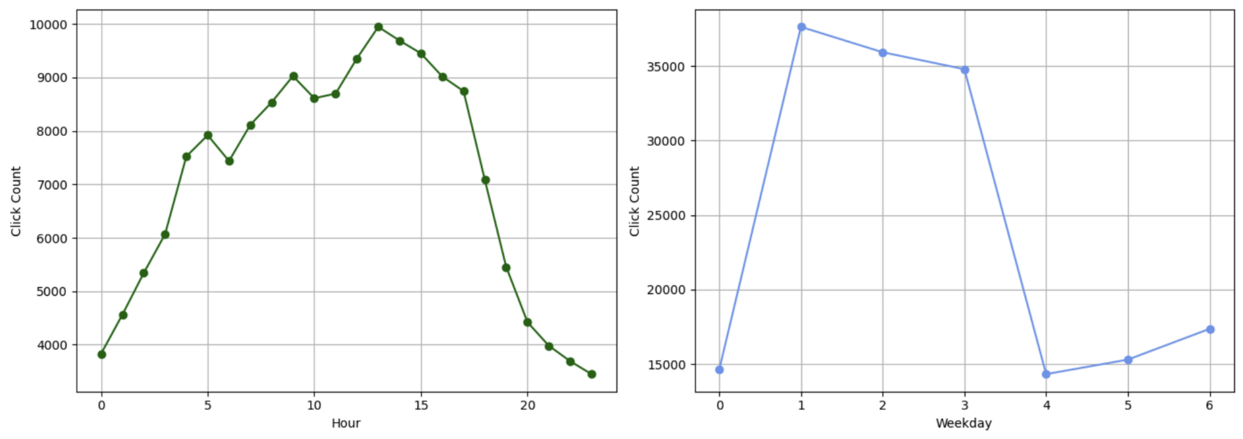

Figure: Click Count Analysis Across Hours and Weekdays

Exploring the anonymized features from **C1 to C20**, we notice that the proportion of clicks (orange bars) to the total count (blue bars) is not evenly distributed across the top values. This indicates that certain feature values have a stronger influence on the likelihood of a click. Additionally, there's an observable class imbalance, where the number of clicks is significantly lower compared to the total count in some categories. A particular example is **C20**, where the value "-1" has a high count.

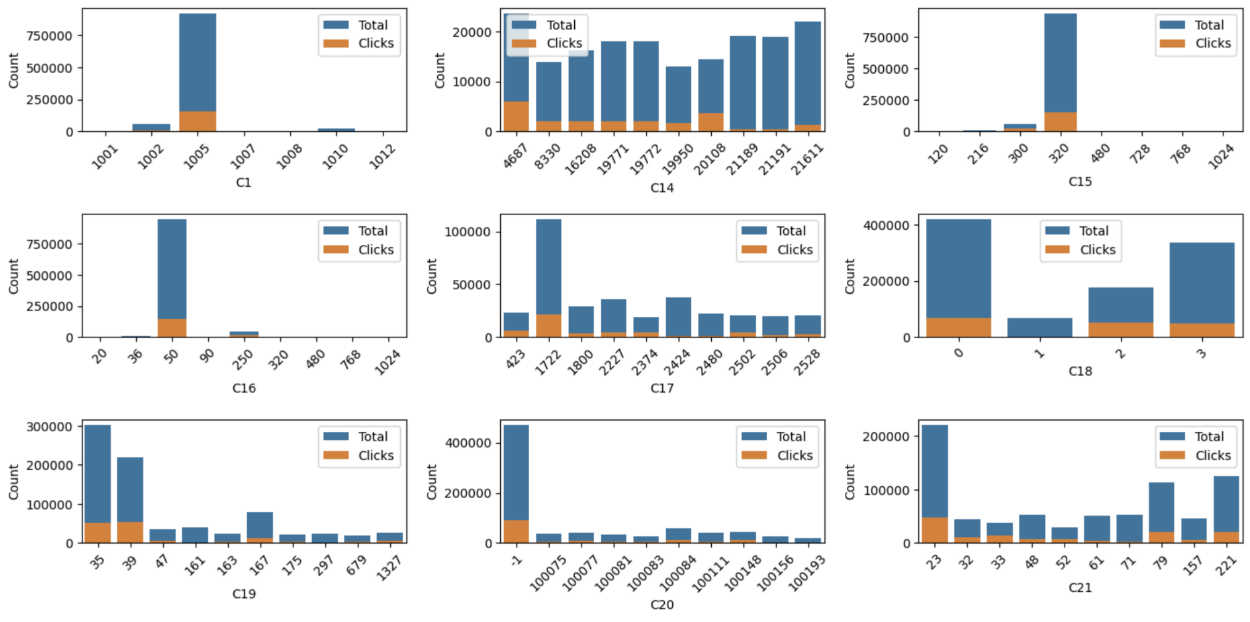

Figure: Distribution of Clicks and Total Samples for Categorical Features in the anonymized features

Summary of anonymized features:

- C1, C15, C16, and C20 are highly imbalanced features and have a single dominant category that accounts for the vast majority of the data. These features may be less predictive due to lack of variety.
- C14, C17, C19, and C21 are moderately balanced features and show more diversity and may offer good information for the CTR prediction model.
- C18 is a low-dimensional feature which has a small number of categories (0–3) and could represent ordinal or binary user behavior.

More exploratory data analysis results for the rest of the features can be found in the code notebook.

The correlation heatmap below highlights several important relationships between features. We observe a strong correlation between **C14** and **C17**, **C1** and **device_type**, one of these features could be removed to reduce model complexity. On the other hand, there is a negative correlation between **C21** and **C18**, but it is still beneficial to keep both features, as they may provide insights later. Another notable correlation is between **banner_position** and **device_type**, which is expected, as mobile devices typically have fewer available banner positions, and the layout could vary depending on the screen type.

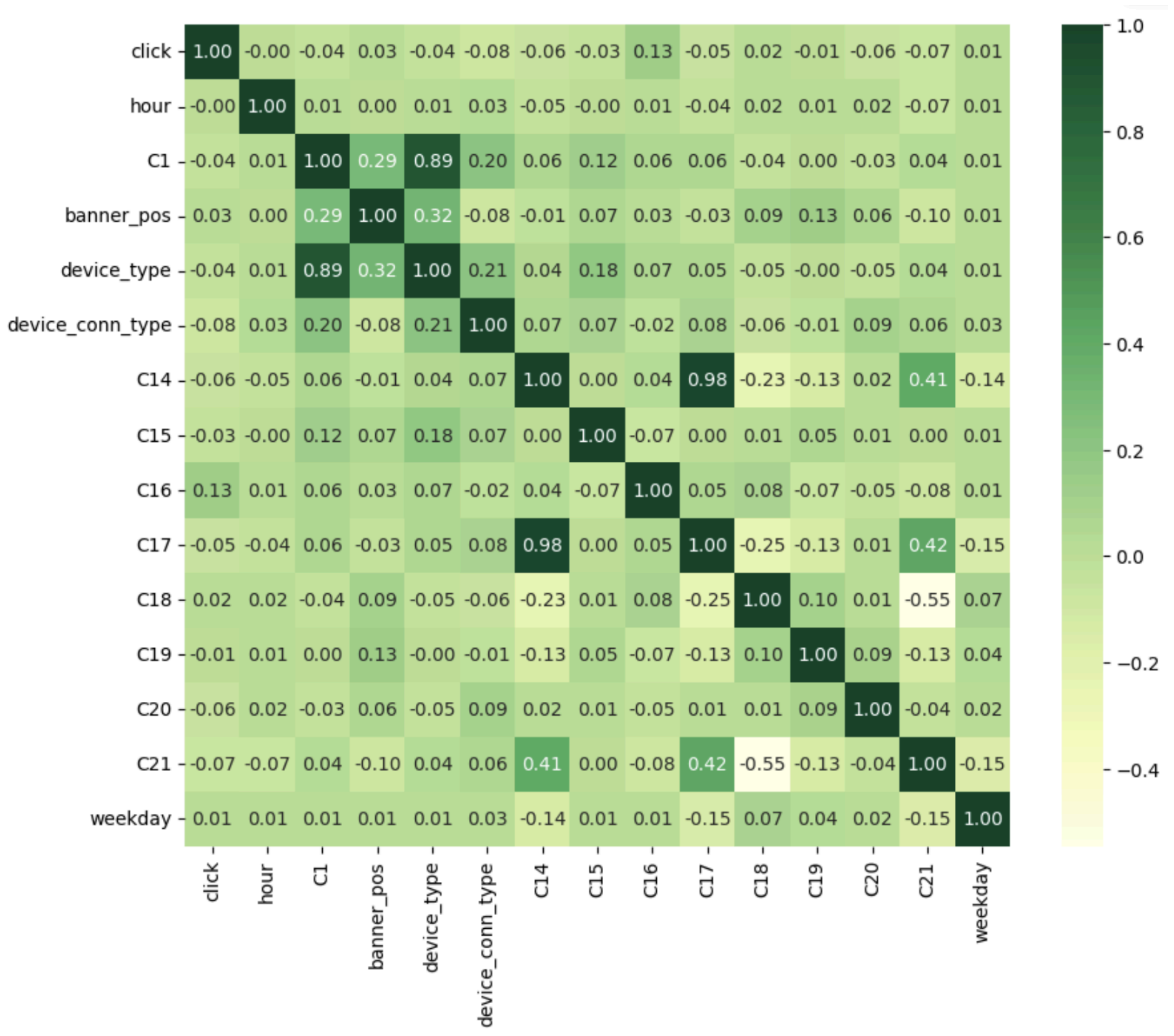

Figure: Correlation heatmap of features showing correlation values between 0 and 1.

Feature to drop	Reason

ID	Has too many unique values
C1	Too correlated with device type
C14	Too correlated with C17
Device IP	Has to many unique values
Device ID	Has too many unique values

Table: List of features I dropped from the input data

The next question is how to encode all of these categorical variable. Given that many features have a high number of unique values, using label encoding or one-hot-encoding directly could significantly increase the dimensionality of the dataset. This could lead to a more complex model with higher computational costs. Instead, a better approach would be to use **target encoding**, which encodes categorical features based on the mean of the target variable (click rate, in this case) in that category. This method helps to preserve the information in the categorical features without drastically increasing the dimensionality, making it more efficient for model training and improving performance.

Models and Methods

Overview of models and implementation

As baseline models, I will train Logistic Regression, Random Forest, and XGBoost. XGBoost builds decision trees sequentially, unlike Random Forest, which constructs trees in parallel. XGBoost has been shown to perform better with unbalanced datasets like this one.

The initial results from the baseline models, however, are not very promising. They show the class imbalance in the dataset, with the models struggling to accurately predict the minority class (clicks) compared to the majority class (non-clicks).

Initial model performance of my Logistics Regression model

```
Accuracy:  0.8287

Recall:  0.0037947873154086016
Precision:  0.24618320610687022
F1 Score:  0.0074743612028506864

Confusion matrix:
 [[165611    395]
 [ 33865    129]]

Log Loss: 0.4444
```

The Logistic Regression model achieves an accuracy of 82.8%, but this metric alone doesn't provide the full picture. Most of the accurate predictions are for the negative class (non-clicks), which skews the result. When we examine the confusion matrix, we observe that the true positives (clicks) are significantly lower than the false positives, indicating that the model is not effectively identifying the clicks as it should. This is a common issue with imbalanced datasets, where the model tends to favor the majority class. However, since this is a baseline model, we will explore methods like balancing the dataset or adjusting the model to improve its performance in predicting clicks.

XGBoost

```
Accuracy:  0.831535

Recall:  0.03609460493028181
Precision:  0.5699024616813748
F1 Score:  0.06788945140675574

Confusion matrix:
 [[165080    926]
 [ 32767   1227]]

Log Loss: 0.4189
```

The XGBoost model outperforms the Logistic Regression model across all evaluation metrics, with a notably lower loss. This demonstrates its ability to capture more complex patterns and handle the imbalanced nature of the dataset more effectively.

The next iterations of model training will use the data after feature engineering. In terms of feature engineering, below techniques were applied:

- Three features were dropped due to high correlations with others, which could introduce multicollinearity and affect the model's performance.
- Two features were removed because they had too many unique values, making them uninformative for the model, similar to the id column.
- Target encoding was applied to convert categorical columns into numerical format, preserving the relationship between categories and the target variable without unnecessarily increasing the dimensionality.
- StandardScaler was used to standardize the features, ensuring that all features are on a similar scale by removing the mean and scaling to unit variance. This helps to improve the performance of models.

After these engineering steps, the XGBoost model showed improved performance, which shows the effectiveness of the engineering.

```
Accuracy:  0.833925

Recall:  0.06386421133141143
Precision:  0.6093179904574797
F1 Score:  0.11561093804084459

Confusion matrix:
 [[164614   1392]
 [ 31823   2171]]

Log Loss: 0.4035
```

The improvements in model performance are evident, with the log loss decreasing further, indicating a better fit of the model to the data. Additionally, we see an increase in both recall and precision, suggesting that the model is now more effectively identifying positive cases (clicks). The true positive count has risen in the confusion matrix, further showing the fact that the model is performing better in identifying clicks, which is the goal for this task.

Now that the feature engineering has been shown to improve the model, the next step is to optimize the performance even further by performing hyperparameter tuning for the XGBoost model. This will help find the best set of hyperparameters to enhance the model's accuracy and generalization, ensuring it performs well on unseen data.

```
Best parameters: {'learning_rate': 0.2, 'max_depth': 6, 'n_estimators':
100, 'objective': 'binary:logistic'}

Accuracy:  0.834535

Recall:  0.08180855445078543
Precision:  0.5966530787384682
F1 Score:  0.1438882421420256

Confusion matrix:
 [[164126   1880]
 [ 31213   2781]]

Log Loss: 0.3988
```

This is the best performance that was achieved in this dataset. The feature importance values indicate how much each feature contributes to predicting whether an ad is clicked. For example, **site_id** (0.5229) is the most important feature, suggesting that the site on which the ad is displayed has a strong influence on the click-through rate. **app_id** (0.1911) also plays a significant role, reflecting that the app hosting the ad can affect user interaction. **site_domain** (0.0343), **device_model** (0.0269), and **device_type** (0.0262) have smaller but still notable contributions, indicating that the specific site domain and the user's device characteristics, like model and type, also influence the likelihood of an ad being clicked, though to a lesser extent.

Feature Importance for XGBoost

---	------
site_id	0.5228648
app_id	0.19111544
site_domain	0.03429635
device_model	0.026887788
device_type	0.026226418
C21	0.02579968
banner_pos	0.024287205
app_category	0.021355659
C15	0.018695753
C18	0.016962927
C17	0.014976882

```
C19         0.012632364
device_conn_type 0.012051126
app_domain   0.01193616
site_category 0.011275358
hour        0.010210388
C16         0.00987415
weekday     0.008551598
```

Conclusion and Next Steps

Summary of models and next steps for further analysis

- Logistic Regression: A baseline model with an accuracy of 82%, but it struggled with the imbalanced nature of the dataset. The confusion matrix showed that the model predominantly predicted the negative class, leading to low true positives for clicks.
- Random Forest: Performed decently but not as well as XGBoost in handling the imbalanced data and takes far more time to train.
- XGBoost: Outperformed other models in all evaluation metrics, including a lower log loss and higher recall and precision. The feature engineering steps, such as feature removal and target encoding, contributed significantly to the improved performance.

Next Steps for Further Analysis

1. **Address Class Imbalance**: The most important action is handling the class imbalance. This could be achieved by sampling techniques like SMOTE (Synthetic Minority Over-sampling Technique) which can be used to generate synthetic samples for the minority class (clicks), balancing the dataset and potentially improving model performance further.
2. **Neural Networks**: Given the complex relationships between features, exploring neural networks might help capture non-linear patterns in the data, offering another layer of performance improvement.
3. **Feature engineering:** instead of dropping some features just because they have high cardinality, it could improve the model if I combine and merge. For example: device id and device ip
4. **Model Blending**: Combine models (Logistic Regression, Random Forest, XGBoost) using **stacking** or **averaging** to improve predictions.

5. **Principal Component Analysis (PCA)**: Use **PCA** for **dimensionality reduction** to reduce the set of features that bring the most variation into the dataset to improve efficiency and reduce overfitting.