Applied ML Final Project Report - Team 21

Jace Yang, Kevia Qu, Sarosh Sopariwalla, Yu-Chieh Chen, Yunzhe Zhang

I. Background & Context

The main goal of this project is to predict whether or not an app will be downloaded after its ad is clicked. Since ad network companies are paid for every click garnered by their advertisement, there has been a rise in "fraudulent clicks": fake ad clicks that are made only to increase the profits of ad companies. In fact, some fraudsters will go as far as using many smartphones or creating bots to generate as many ads clicks as possible, all without leading to any download of the app.

TalkingData, a big data company, currently tracks the journey of each user's clicks; if they find an IP address with many clicks but very few installs, they will blacklist these IP addresses and devices as fraudulent. While this method has worked so far, it would be much more efficient if we could predict the likelihood of a user downloading an app after clicking on the ad.

II. Exploratory Data Analysis (Dataset Introduction)

The dataset was obtained from the TalkingData AdTracking Fraud Detection Challenge hosted by Kaggle, and it covers 180+ million clicks in China over 4 days, totaling 11 GB of data. Despite the quantity of data, each sample is only described by 8 categorical variables (shown in the left-most column of **Figure 1**), and only 0.24% of the total dataset is of clicks that led to app downloads (shown in **External Figure 1** from EDA deliverable).

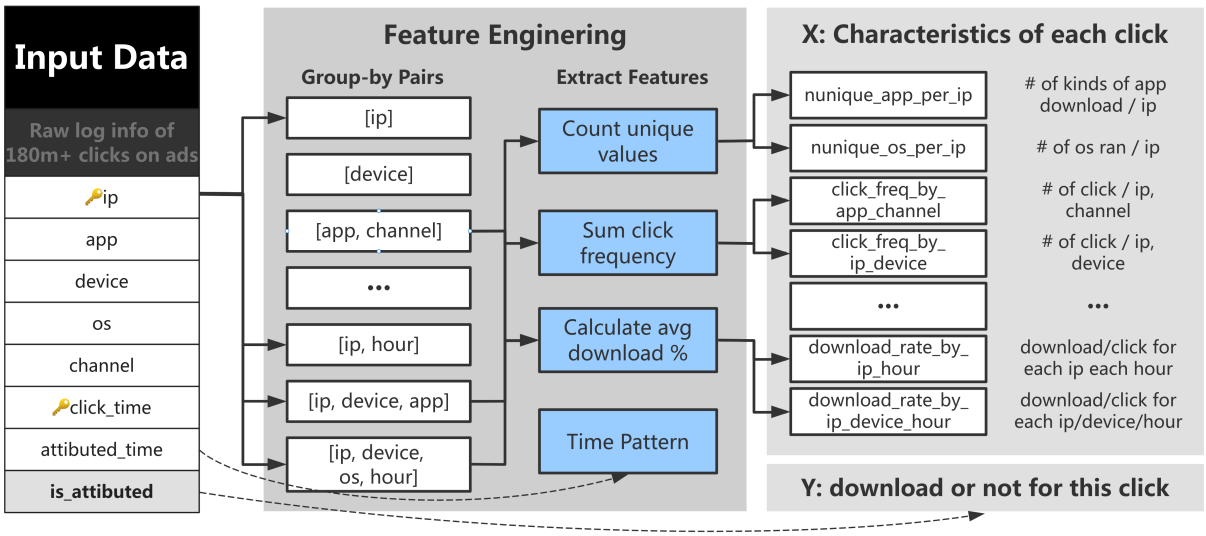


Figure 1: *ETL data flow of how additional features were engineered from the 8 variables of the dataset ("Input Data")*

To handle these three challenges (memory, lack of features, and data imbalance), several techniques were employed. First the existing features were recast to downsize the dataset which reduced the data from 11GB to 5GB. Due to the lack of descriptive variables in this dataset, new features were generated using the original 7 features. Since five were discrete and two were datetime objects, different combinations of these features were grouped together to generate numerical features (process described in **Figure 1**). The new generated features provide additional descriptive information related to the target `is_attributed`. **External Figure 2** shows the distribution of `is_attributed` in created fields and these features can clearly show the distinctions between the two labels.

Initial exploratory analysis of the data prior to modelling revealed several insights into why the classes were so imbalanced: people rarely download apps after clicking the ads and "click-bots" generate millions of clicks but only few downloads (**External Figure 3**).

III. Building a Binary Classifier

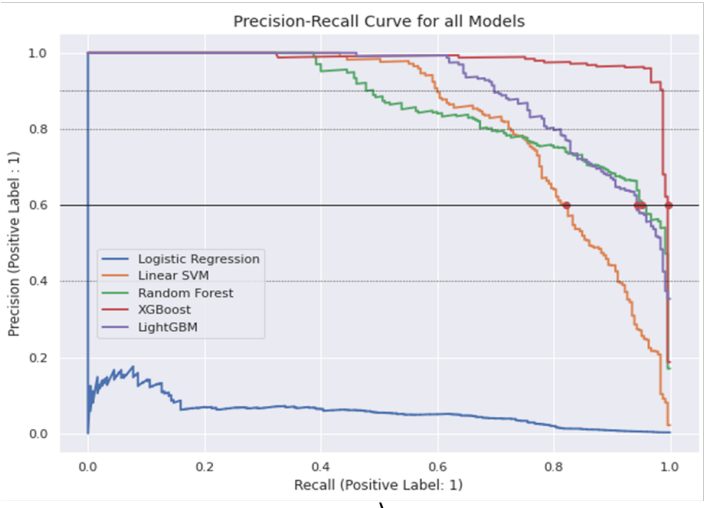
Sampling

To handle training on such a large imbalanced dataset, we chose to first take a stratified split of the data and take a small fraction of 500,000 points, split again into an 80/20 dev/test ratio. The development data was then oversampled using SMOTE to increase the number of samples with class

label ‘1’ and this data was used for model training and hyperparameter tuning, using cross validation.

Dummy Classifiers

To define baseline performance, we began with two dummy classifiers. The first classifier always predicts the majority class. Since our data is so imbalanced, this allows us to achieve an accuracy of about 99.8%. However, the recall and precision are both zero. We also consider an informed dummy classifier, where we make predictions based on the “download rate by IP” feature. If “download rate by IP” is greater than 80%, we predict there will be a download; otherwise, we predict no download. This informed dummy classifier achieves a recall of 10% and a precision of 100% on the test set. With these baselines in mind, our goal is to now create a model that can outperform the dummy classifier.



a)

Model	Recall when Precision is at a Threshold of:				AP	F-1 Score
	0.4	0.6	0.8	0.9		
Logistic Regression	NaN	NaN	NaN	NaN	0.024	0.058
Linear SVM	0.910	0.820	0.722	0.600	0.748	0.837
Random Forest	0.996	0.951	0.683	0.480	0.778	0.871
XGBoost	0.996	0.996	0.988	0.988	0.953	0.982
LightGBM	0.992	0.943	0.800	0.698	0.799	0.910

b)

Figure 2: a) *Precision-Recall Curves for all models tested. Red dots represent recall when precision is at a threshold of 0.6* ***b)*** *Summary performance metrics of plotted models.*

Logistic Regression

Since we are working with a binary classification problem, it makes sense to start with a simple linear model such as Logistic Regression. However, by doing so, we implicitly assume that the log odds of y are linearly related to X. Using grid search and cross validation, we find the best parameters for our Logistic Regression model are $C = 2.78$ with a $L1$ penalty norm (C denotes the inverse of regularization strength, so smaller values specify stronger regularization.) The Precision-Recall (PR) Curve for this model (**Figure 2a**) in blue, indicates that our model is fundamentally flawed. Typically, the PR Curve would be concave down -indicating relatively high precision and recall scores. We achieved a recall of 84% with a precision of 1.2% without manipulating the prediction threshold. Clearly, adjusting the threshold would not significantly increase the performance, so we move on to a new model.

Support Vector Machines

Support Vector Machines (SVMs) tend to be a useful linear model to work with, since it can also work for non-linearly separable data using the “kernel trick”. However, since our dataset is so large, applying the kernel trick led to an extremely long model training time (well over 12 hours). Therefore, we used a Linear SVM. Another benefit of using SVMs is that its dual formulation works in the feature space rather than the sample space; this drastically reduces the training time compared to the primal formulation. After hyperparameter tuning on the regularization parameter C, calibrating the model, and adjusting the threshold, the SVM model achieves a recall of 80.4% with a precision of 62.9% (**Figure 2a**) clearly outperforming Logistic Regression. A small decrease in recall allowed for a massive increase in precision. For the remainder of our models, we will deem 60% precision as the minimum precision needed for a “good” model and then attempt to maximize the recall. From the PR Curve (orange), we see two key takeaways: first, the SVM makes much better predictions than the Logistic Regression model. Second, when we threshold our model such that we achieve 60% precision, we are able to achieve a recall of 82% (represented by the red point).

Random Forest

Now that we have explored some simple linear models, it makes sense to move on and see if we can use ensemble methods to increase model performance. Similar to the Support Vector Machine, we

built a Random Forest that went through hyperparameter tuning and calibrated cross-validation to get the final precision-recall curve seen (**Figure 2a**, green). In particular, we tuned the number of trees, max depth of the trees, as well as the minimum number of samples needed to split a leaf. The black line in the figure denotes where precision is 60%; thus, if we define the threshold properly, we can achieve a recall of 95% with 60% precision. This is a massive improvement in the recall score we saw with the SVM; in particular, at the 60% precision level, recall increased from 82% to 95%. We will now explore boosting ensemble methods to see if we can make our model even better.

XGBoost

Next, we tried the commonly-used boosting model, XGBoost. We tuned its most important hyperparameters: learning rate, max depth, and number of base estimators. Scoring the model with the most optimally found hyperparameters on the test set yielded a recall of 99.6% at the precision threshold of 60%; a significant improvement over the other tested models. From the PR Curve (**Figure 2a**, red), we saw that XGBoost also yielded the best recall with the least tradeoff in precision; notably achieving a recall 98.8% even at a precision threshold of 90% (**Figure 2b**).

As the best model compared to other trials, we trained XGBoost with its best hyperparameters and made predictions on the unseen data on Kaggle test set. Examining the feature importance of this final deployed model (**Figure 3**), we found the important engineered features aligned with our findings in the EDAV portion. "download_rate_by_ip" and "download_rate_by_ip_device_app" can quickly identify click-fraudsters who have a historically low download rate in their ip address and devices used. Similarly, "click_freq_by_ip_device_os_hour" identifies single users' clicks in each hour of the day, which can distinguish bots from normal users. Finally, the features related to device type are considered unimportant by the models, since the same types of devices are used by normal users and fraudsters.

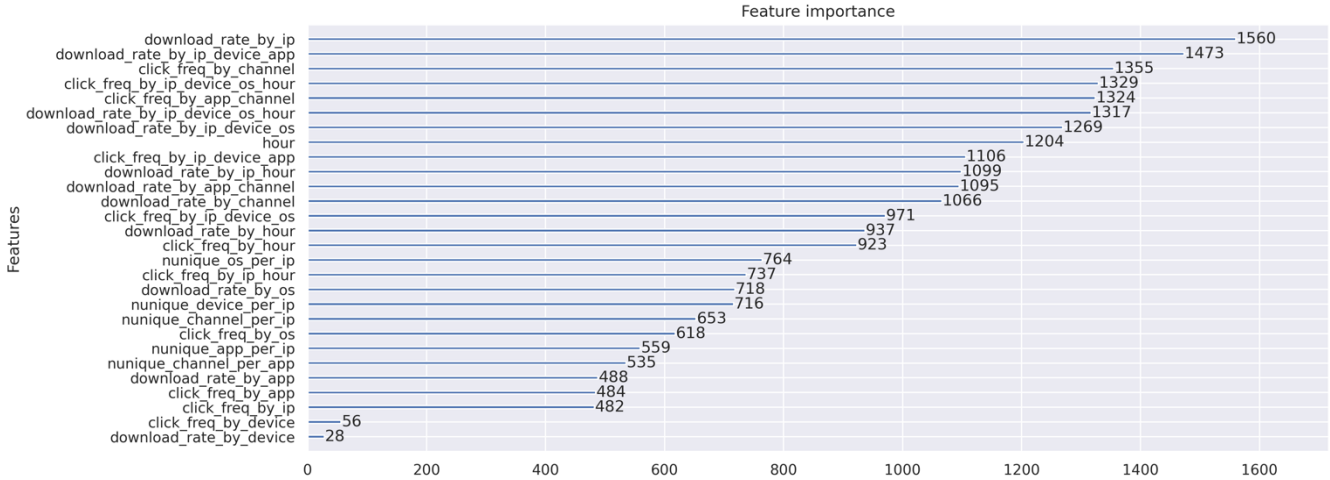


Figure 3: *Features from final trained XGBoost model, ranked by highest feature importance*

LightGBM

We also attempted to use LightGBM as another type of boosted tree-based model, which performs leaf-wise splits instead of level-wise splits more common in other boosting methods. This approach allows for more loss reduction and faster training times, which is well suited for large datasets. As with the other models, training was performed on a SMOTE-sampled dataset and hyperparameter tuning was performed on the parameters of number of estimators, number of leaves, and max depth. While this model trained in a fraction of the time of XGBoost, the best model performed worse, only achieving a recall of 94% at a precision threshold of 60%. (**Figure 2a**, purple).

IV. Summary of Results

Our initial goal was to create a model that performs better than our "informed" dummy classifier which achieved a recall of 10% with a precision of 100%. Of the five models we trained, four of them clearly outperformed the dummy classifier. Logistic regression performed poorly due to high bias: in particular, the assumption that the log odds of y are linearly related to X was incorrect. The remainder of the models all performed significantly better than the informed dummy classifier; clearly, XGBoost performed the best as seen by the recall, average precision and F1 scores in the table (**Figure 2b**). However, XGBoost does have significantly longer training times; thus, if resources are an issue to consider, the LightGBM model would be the next best model for deployment. A submission of our XGBoost model on Kaggle with completely unseen data yielded a score of 0.74.