Team 2: Patrick Wang, Angela Guevarra, Lydia Zhang, Kyu Kim

ISOM 674: Machine Learning

Final Project Report

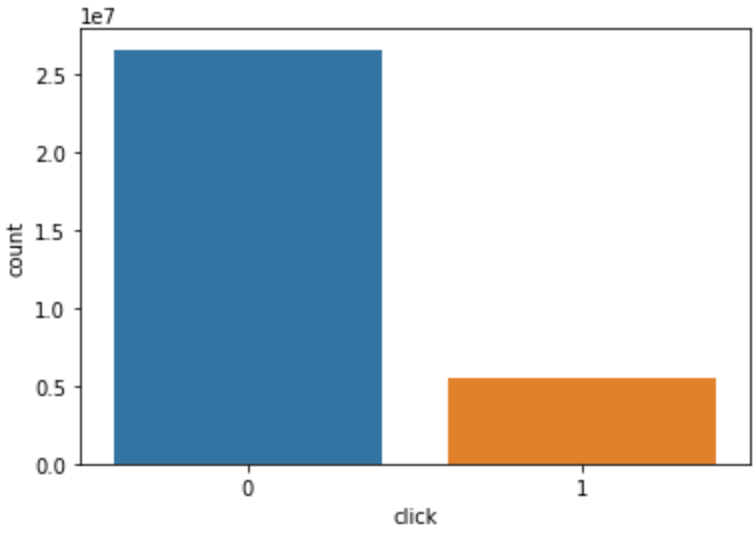
# Predicting Click Through Rate

## Introduction

With the flourish of  e-commerce over the past decade, there has been a huge volume of advertisements being published on the Internet in hopes of reaching their consumers. These marketing campaigns are a huge cost to the company to roll out and therefore it is important to target the right consumer with the right advertisement that appeals to their interest. Click-through rate (CTR) analysis is an essential task for advertisers of online websites and applications. Having an accurate and fast CTR prediction can help online websites and applications improve the user experience, increase the revenue from advertisements, and identify the most attractive information on a platform. In CTR prediction, even a small improvement can potentially lead to a large increase in revenue. Thus, our project involves helping predict clicks for these on-line advertisements by creating a classification model with data over 9 days (from October 21, 2014 to October 29, 2014) that has a record of over 30 million advertisements that were served to various consumers. From our multiple time-consuming model runtimes, the change of log loss metric is not very significant. However, an absolute improvement of 1‰ in log loss (or AUC) is considered as practically significant in real CTR prediction problems, as predicted by Google and Microsoft.

## Data Understanding

The original training data set has 31,991,090 rows and 24 variables. The target variable for analysis was “click”, which is a binary variable indicating whether a customer clicked on an ad.  As seen from the histogram below, the dataset is imbalanced with around 17% click rate for online ads and 83% for no-click.




There are 23 predictor variables in the model, out of which 22 are categorical and one variable is continuous. The variables contained information encoded as identifiers about the site where the ad was hosted such as domain, id, and category; the application being used, its domain and category; and descriptive elements about the device such as IP address, model, and type. Since the values for target variables are available, supervised learning algorithms were used. Additionally, when looking at the rest of the dataset, there are no missing values and the variables are either of numeric or character type.

**Data Preparation**

We used StratifiedShuffleSplit to take a sample from the original dataset to keep the ratio of target variable. The target variable for analysis was “click”, which is a binary variable showing whether the advertisement was clicked.

For feature engineering, we adapted different approaches for each model. The details will be explained accordingly in the model section. The reason why we did so is because sophisticated models such as Neural Network do not need as much feature engineering, as its algorithm structure determined that it will naturally consider the interaction of regressors, give less weight to unimportant variables, etc.

**Modeling Process:**

CTR prediction often involves a huge dataset and most of the features are categorical with multiple values. Normally, we use one-hot encoding to deal with categorical variables. However, the large-scaled datasets

with categorical features will lead to a large sparse feature space that is difficult for building the model and making the prediction.

Therefore, normal models are probably less feasible to deal with this kind of datasets. Based on our analysis and multiple kinds of models we tried from logistic regression to Neural Network. Neural Network models have shown better performance. But even for neural network models, CTR prediction problems are typically harder to compute given its sparse feature space. With the sparse features, and multiple values for each feature, the high-dimensional feature space from one-hot encoding will lead to the curve of dimensionality.

To solve this problem, we applied more general models as well as some neural network models that are developed for the CTR prediction problems, such as DeepFM, ONN, FiBiNET, DCN, FGCNN, and so on. We will walk you through the different models we tried for our project and conclude with our final model.

**Model: Logistic Regression**

Logistic regression is a classic model and has been applied to click rate problems widely. For our Logistic regression model, we optimized it based on feature engineering and hyperparameter tuning through random search. Our feature engineering included operations on 'hour', such as generating the day of week (Wednesday, Saturday, etc), day of the month (21st, 23rd, and more), hour of the day, and more. For categorical variables, we applied one-hot-encoding. We also considered balancing the ratio of 0 and 1 for 'click', but noticed that the models trained by balanced datasets perform much worse on both balanced and unbalanced data, with on average a 0.22 increase in log loss. Therefore, we did not include this in the final logistic regression model. For hyperparameter tuning, we used the python function RandomizedSearchCV to explore the best combination of solver, penalty, and regularization strength. Unfortunately the randomized search did not work on large datasets, but we found that the best parameters so far are using penalty l1 (lasso), regularization strength of 1, and liblinear as solver. Its log loss is 0.42.

**Model: Random Forest**

Random Forest is a classification algorithm made up of several decision trees. Because it already consists of many individual decision trees, we opted for a Random Forest model over a singular Decision Tree as it may also help avoid overfitting that may occur with building only one singular decision tree. After exploring the dataset and realizing that most of our variables were categorical, using Random Forest was a solid choice as it can handle binary as well as categorical features. Additionally, it handles unbalanced data well and due to the

unbalanced values of click and no-click mentioned earlier, this is a great feature! It also works great with high dimensionality (which is needed for this project).

We decided to build this model on Python using the RandomForestClassifier package from sklearn. On top of the feature engineering mentioned in one of the previous sections, the procedure for this model involved transforming the dependent variable, "click", into a factor. We then ran a RandomizedSearchCV to tune the hyperparameters that will yield the best hyperparameter values for the best model. In this project, we mainly focused on tuning max_depth and n_estimators as we felt these were the most important to spend computing power on. Afterwards, we used the log loss to determine it's score which ended up to be at 0.401.

```
0.4010763021776031
```

**Model: CatBoost**

CatBoost is a machine learning algorithm that uses gradient boosting on decision trees. (CatBoost.ai, 2021) It is a gradient boosting framework which attempts to solve for categorical features using a permutation driven alternative compared to the classical algorithm (Cornell, 2021).

For this project, one of the biggest reasons to pursue CatBoost was because after exploring the dataset, we found out that most of the columns were categorical and thought that maybe running a gradient boost framework that is focused on categorical variables would work better.

After exploring the catboost.ai webpage and learning how to utilize this machine learning algorithm, there was some preprocessing and data cleaning that we had to do. Other than the typical creating the training and validation dataset, we also had to get rid of the first column of the dataset provided by Professor Easton as it wasn't categorical and caused errors when running CatBoost.

After that, we also had to create certain parameters that were needed for CatBoost. The first thing we created was "cat_features", which needed to be a "one-dimensional array of categorical columns indices" (CatBoost.ai, 2021). For this specific project we set the cat_features as values from 0 to 20. Other than the "cat_features", there were no other parameters that we had to set for this project.

For this project, we chose a few attributes that we thought would be helpful in performing the best or attaining the lowest log-loss function. The attributes that we have used for our specific CatBoost algorithm are listed below:

- *Random_seed: This is the random seed used for training*

- *Custom_loss*: We specify the model to calculate the Log Loss function to find the best performing iteration.
- *Iterations*: The value set by us determines the maximum number of trees that can be built when solving machine learning problems.
- *Learning_rate*: The value set for this attribute is for the learning_rate of the machine learning algorithm and is used for reducing the gradient step.
- *Task_type*: Task type specifies the model to use a certain processing unit within the computer.
- *Boosting_type*: This allows us to choose what boosting scheme we want to use for the model. The "Plain" version we used is the classic gradient boosting scheme.
- *Depth*: This value determines the depth of the tree.
- *L2_leaf_reg*: The value for this attribute can be any positive value and it is the coefficient of the L2 regularization term of the cost function.
- *Random_strength*: The value here specifies the amount of randomness to use for scoring splits when the tree structure is selected. This parameter is useful when trying to avoid overfitting of the model.
- *Od_type*: This is the overfitting detector that can be activated if used in the attributes. Similar to that of the "early_stopping_rounds" attribute in lightGBM or xgboost, the one in CatBoost is defined as "Iter".
- *Od_wait*: The value set here determines the number of rounds after the best iteration to wait before stopping.

After running the model, we got the final model to provide a log loss value of 0.39014.

```
290:    learn: 0.3897083    test: 0.3901815 best: 0.3901548 (280)   total: 1m 3s    remaining: 45.5s
300:    learn: 0.3895536    test: 0.3901526 best: 0.3901526 (300)   total: 1m 5s    remaining: 43.4s
310:    learn: 0.3894484    test: 0.3901453 best: 0.3901448 (309)   total: 1m 7s    remaining: 41.3s
320:    learn: 0.3893491    test: 0.3901662 best: 0.3901448 (309)   total: 1m 10s   remaining: 39.1s
330:    learn: 0.3892001    test: 0.3901823 best: 0.3901448 (309)   total: 1m 12s   remaining: 36.8s
340:    learn: 0.3890643    test: 0.3902122 best: 0.3901448 (309)   total: 1m 14s   remaining: 34.7s
350:    learn: 0.3889541    test: 0.3902128 best: 0.3901448 (309)   total: 1m 16s   remaining: 32.4s
bestTest = 0.3901448372
bestIteration = 309
Shrink model to first 310 iterations.
```

**Final Model: DCN**

After testing the models, we chose the Deep & Cross Network (DCN) model as our final algorithm based on the considerations of the log loss score, computational time, computational cost, and the difficulty to implement. DCN model enables Web-scale automatic feature learning with both sparse and dense inputs. DCN

efficiently captures effective feature interactions of bounded degrees, learns highly nonlinear interactions, requires no manual feature engineering or exhaustive searching, and has low computational cost. DCN solves the problem with an embedding and stacking layer, followed by a cross network and a deep network in parallel. Figure 1 shows the network representation. [1]

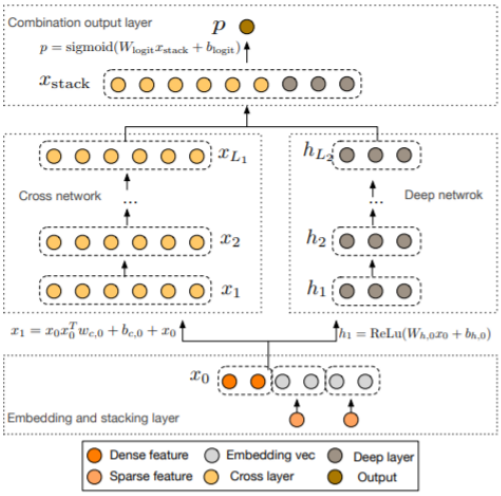



**Figure 1: The Deep & Cross Network**

We implemented the model on PyTorch and used the fuxictr package to run different models and make predictions. For the data preparation process, we removed the id column and transformed the hour column to the hour, the weekday and is_weekend. Since the DCN model can learn nonlinear interactions, we didn't apply data engineering for this model.

We did hyper-parameter tuning using fuxictr package and used the parameters on our final model. Some of the parameters are listed below.

- *'batch_size'*: 10000, the number of samples that will be propagated through the network.
- *'epochs'*: 100, the number of passes of the entire training dataset (most of the models don't need 100 epochs because of early stopping).
- *'learning_rate'*: 0.001, the starting learning rate.
- *'task'*: 'binary_classification'

[1] Ruoxi Wang, Bin Fu, Gang Fu, Mingliang Wang, 2017, Deep & Cross Network for Ad Click Predictions

- *'loss':* 'binary_crossentropy', binary cross entropy as the loss function.
- *'metrics'*: ['log loss', 'AUC'], metric to evaluate the model, AUC and logloss.
- *'min_categr_count'*: 2, dealing with infrequent values with counts less than 2.
- *'dnn_activations'*: "relu", activation function for the dnn layer.
- *"dnn_hidden_units"*: [2000, 2000, 2000], 'stacked_dnn_hidden_units': [500,500,500], 'parallel_dnn_hidden_units': [500,500,500], units for different layers.
- *'embedding_dim':* 16, 'embedding_droput': 0, 'embedding_regularizer': 1e-8, embedding method parameters for dealing with the categorical features.
- *'optimizer'*: 'adam', Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.
- *'gpu': 0*, using gpu to run neural network models will be faster.

```
2021-12-18 22:43:01,002 P19596 INFO ***** validation/test results *****
2021-12-18 22:43:28,750 P19596 INFO [Metrics] logloss: 0.377845 - AUC: 0.783271
{'logloss': 0.37784506805299123, 'AUC': 0.7832710203051496}
```

**Appendix**

Project-Code1-Logit-Team2.py: a python file of Logistic Regression

Project-Code2-RandomForest-Team2.py: a python file of Random Forest model

Project-Code3-Catboost-Team2.py: a python file for Catboost model

Project-Code4-DCN-Team2.py: a python file for DCN model

**Bibliography:**

https://catboost.ai/en/docs/
https://arxiv.org/archive/cs.LG
https://stackoverflow.com/questions/45533159/how-to-work-with-the-catboost-overfitting-detector