ISOM 674 Final Project

Team 8

Kenny Lee, Rain Tian, Lacey Lu, Cecilia Wang

EDA and Feature Engineering

The training dataset consists of 31,991,090 rows of data with no missing entries. There are 24 columns of data which are all categorical. One of the columns, 'hour', is a time stamp in the format year, month, day, hour. All the data is from October of 2014, with the only differences being the day and the hour. We split 'hour' column into 2 additional columns, 'day' and 'hour', and deleted the original 'hour' column. The other columns were kept the same as categories. Within each column, we looked at the number of unique values for each column, detailed below.

id	31991090
click	2
hour	216
C1	7
banner_pos	7
site_id	4581
site_domain	7341
site_category	26
app_id	8088
app_domain	526
app_category	36
device_id	2296165
device_ip	5762925
device_model	8058
device_type	5
device_conn_type	4
C14	2465
C15	8
C16	9
C17	407
C18	4
C19	66
C20	171
C21	55

Before splitting the 'hour' column

From the table, we can see that some columns such as device_id and device_ip have more than a million unique values. This poses an issue for encoding the variables so we performed

cardinality reduction on the columns with many unique values. We then had to decide what columns to reduce. We found that for most columns, a few values constituted a majority of the observations. We decided to perform cardinality reduction on columns with over 100 unique values. The 11 columns were: site_id, site_domain, app_id, app_domain, device_id, device_ip, device_model, C14, C17, and C20. We then faced an issue of identifying a good cutoff value (i.e. keep the top 1%, .1%, .01%, .001%). For cutoff values, we tested different cutoff points and found that the optimal value was keeping the top .000001 of the columns and assigned the value 'other' to any row that had a value outside the cutoff. After the cardinality reduction, there was significantly less cardinality as shown below.

id	15000000
click	2
C1	7
banner_pos	7
site_id	2341
site_domain	2293
site_category	25
app_id	2377
app_domain	155
app_category	33
device_id	11444
device_ip	108358
device_model	4708
device_type	5
device_conn_type	4
C14	2046
C15	8
C16	9
C17	395
C18	4
C19	66
C20	163
C21	55
hr	24
day	9

Training set unique values

After cardinality reduction we still had to change the columns in order to use them in our model. Some of the techniques we considered were one-hot encoding, frequency encoding, and using the default encoding method. On the Catboost documentation page, we read that it specifically states not to use one hot encoding. We tested all 3 options, and decided it made the most sense to go with using the default encoding method specified in Catboost. Catboost has its own way of handling categorical features, using target-based statistics for each categorical variable. We tried a few runs of frequency encoding, and the results were poor, so we decided not to pursue it further.

For training our models, we randomly sampled 15 million rows of data using the random sample package built into Pandas. We then further split the 15 million in an 80-20 split (12 million, 3 million). The reason for the 80-20 split is for the evaluation metric in Catboost. In Catboost, we used the 80% for training and the 20% was used as an evaluation set, in which Catboost evaluates the log loss of the 20 at every iteration. Although this is an unbiased, out of sample dataset, it's relatively small at only 3 million rows and was only used as a reference. In our actual validation set, we used the rest of the 16,991,090 rows to gauge the performance as a true out of sample test. Additionally, by sampling 15 million rows, we could use the rest of the 16,991,090 rows as a validation set which is comparable to the actual test set of ~13 million.

Training set size	Eval test size (small validation set within Catboost)	Validation set size
12,000,000	3,000,000	16,991,090

Modeling

In deciding what model to use, we considered two different paths, tree-based algorithms and neural nets. We ultimately decided to use Catboost for a few different reasons.

First, Catboost was made for categorical datasets, and has a built-in function that automatically handles categorical variables. Any type of prior encoding is strongly discouraged on their document website, and the names of columns that are categorical must be specified before being fed into the model.

Second, Catboost works very well out of the box, without any parameter tuning. For example, the learning rate is automatically determined by Catboost based on the dataset properties and the number of iterations. We achieved very good log loss scores without any feature engineering or tuning compared to other models.

Third, the objective of our project is a predictive binary classification task, using structured data. Neural nets work well for predictions such as natural language processing or image recognition, but we believed a tree-based algorithm would work best for us.

Once we cleaned and prepared the data, we set up the classifier. We set the loss function and evaluation metric as log loss, since the evaluation metric is log loss. We then set the number of iterations for the model as 500 and utilized Catboost's built-in overfitting

detector, which would stop the model if the log loss did not improve within 10 iterations. We then fit the training data and specified the evaluation data as a metric.

Due to the computation time, we used a couple tricks recommended on the Catboost documentation website to speed up the process. First, we utilized the higher memory bandwidth of the GPU of the computer to run our model. Additionally, we converted our dataset into NumPy arrays before we put them in our model to utilize their special data structures for faster computations. These two tricks sped up the computation time by at least 5x by our estimations.

Model Tuning

Ideally, we would have liked to tune our model with at least a 5-fold cv. However, this was not possible due to the limitations of our computers. Hyperparameter tuning using Bayesian Optimization on ~2,000,000 sets of data with a 5-fold cv took several hours and was only utilizing ~6.5% of the total training data. Instead, we decided to train the model using 12 million rows and skipped cross validation. We justified this decision because the validation set was sufficiently large at over 16 million rows and was comparable to the size of the test data prediction. We arrived at 12 million for the training data, because that is about the maximum amount of data our computers can handle in Catboost with RAM being the limiting factor.

Of all the parameters to tune, we decided to only focus on depth, learning rate, and number of iterations. The other default parameters seemed reasonable and we left them as is. For depth, we decided to test out different ranges from 6-10 as per their documentation website. Because of the unclear improvement in log loss and the added training time for more depth, we decided to keep the default value of 6. We initially also tried to change around the learning rate through hand tuning. Catboost automatically chooses a learning rate based on the dataset and number of iterations, but we often found that there was no overfitting of the training set as the last iteration was the best score. Therefore, we manually increased the learning rate by small increments, to try and get the training set to converge faster. We still had the overfitting detector in place to stop the model in case there were signs of overfitting. Interestingly, hand tuning the learning rate and the corresponding number of iterations did not improve our models consistently and took a significantly longer amount of time to train as opposed to using the learning rate set by Catboost. We also decided to tune the iterations along with the learning rate, as the learning rate is inversely proportional to the number of iterations. Again however, there was not consistent improvements in our tests when changing the learning rate and the iterations. Additionally, the increased number of iterations significantly increased our training time, so we decided to limit the number of iterations to 500.

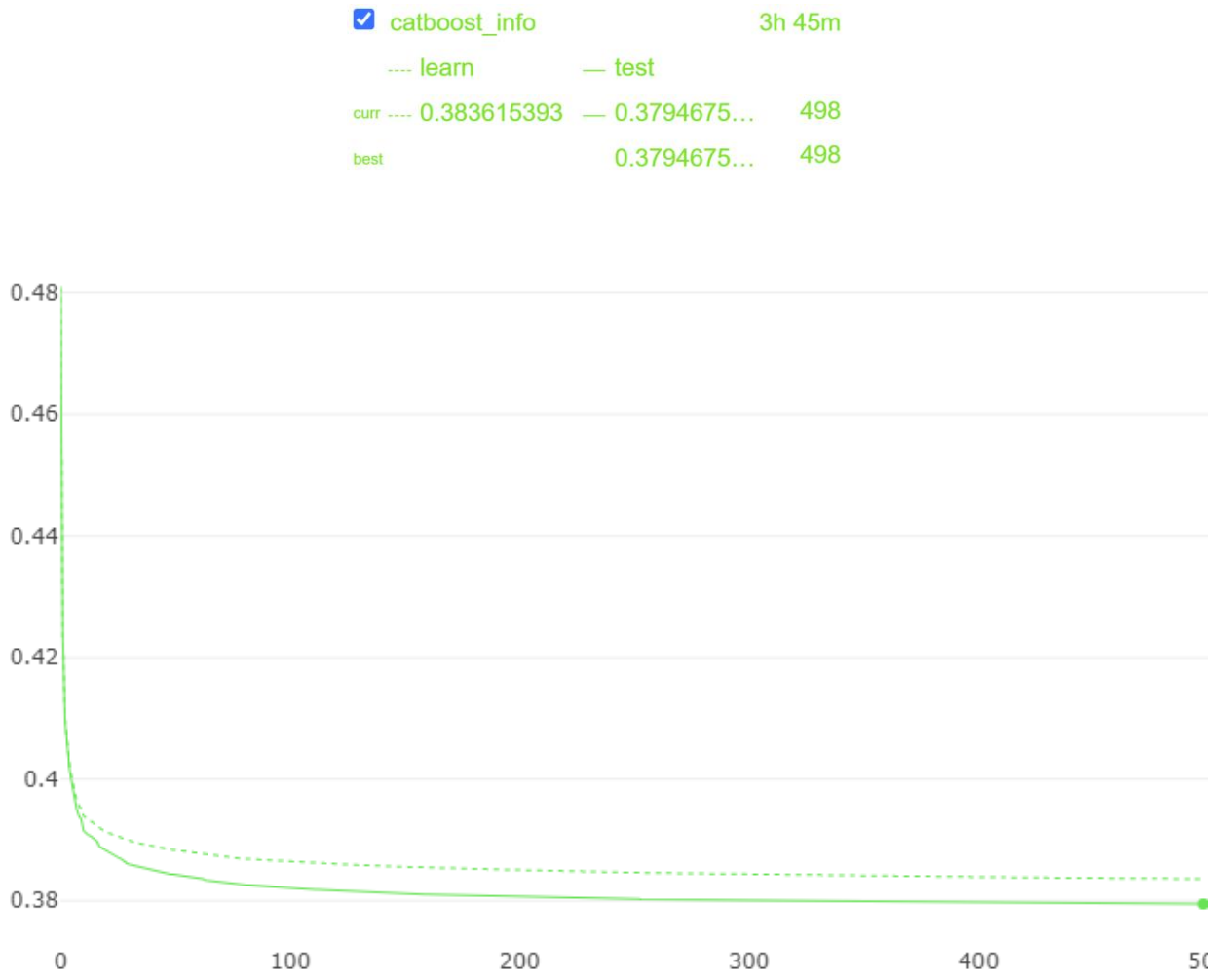
We found that there are diminishing returns with the default 1000 iterations and there was a minimal difference in log loss scores. The Catboost model would frequently not improve in 7 or 8 iterations at a time and the improvements were extremely modest.

Evaluation

We achieved the best validation score of .3799 on the validation data when using the Catboost model. Even after changing the random state for the training data split and the Catboost model, we were able to consistently get a log loss on the validation data within .0001 in our 3 trials. The consistent scores across the different samples suggest a low variance for this model. The overfitting detector we specified in the model was never used although we did observe the model would sometimes not improve within 5 or so iterations. An interesting observation about our model is that it seems to underfit the data. Across all 3 trials, the validation log loss was consistently slightly lower than the training, although we would have to do more research for a definitive answer why. Each trial took about 4-5 hours with random kernel crashes interrupting some trials.

Trial	Training size	Training log loss	Validation size	Validation log loss
1	15,000,000	.3833	16,991,090	.3799
2	15,000,000	.3834	16,991,090	.3800
3	15,000,000	.3836	16,991,090	.3799

Below you can see a fitting graph of one of our trials, with number of iterations on the x axis and the logloss on the y axis. The solid green line is the log loss of the test set and the dotted line is the training set.



Conclusion and areas for further improvements

At every step of this project there are areas for further improvement and exploration. Some important questions are: What model should be used? What columns should be kept/removed? What are good cutoff values for cardinality reduction? What is the best way to encode the categorical variables? For each model, how should the parameters be tuned? We quickly realized there are too many combinations of possibilities to pragmatically explore and modeling decisions were made based on a combination of heuristics and research findings.

Given more time and computing power, we would like to explore different models (neural nets, SVM, etc.), use cross validation to better understand the model's generalizability, and use Bayesian Optimization for model hyperparameter tuning. In respect to trying different models, we believe Catboost is the best tree-based algorithm for this project and therefore would like to explore outside of tree-based algorithms. We would also like to tune our models.

We previously used Bayesian Optimization in prior projects with great success, but were unable to do so in this project due to the computational expense. Tuning our models would likely give better results since we can select the best parameters for this dataset.

The bottleneck of this project is the lack of computing power. Models would often take several hours to run and kernels would crash if there was not enough RAM. Additionally, the CPU would start to overheat, causing the processor to be throttled and cause models to take longer to run. There is a physical limit to the amount of heat small laptops can dissipate, and in the future, powerful AWS instances should instead be used to train the models quickly and efficiently.

Code Files

We only have 1 code file in python where all the lines are commented on. This file contains all the steps in our project.