Online Advertisement Click Prediction

I. Summary of Results

Our approach to this classification was to build models from simpler to more advanced and increase the sample size for training data and perform feature transformation as we proceeded. For benchmark, we built three base models: random forest, logistic regression and KNN. These three models were trained using 100k observations each and among them Logistic Regression performed the best (had the lowest log loss). Some more feature transformation and selection later, Logistic Regression's log loss hovered around 0.41. In order to improve this performance, we built advanced models such as XGBoost, Light GBM and Cat Boosting.

Before building these models, additional feature engineering was done by label encoding some categorical variables and creating count variables. Feature selection was also performed for each of these models using 200k observations and finally out-of-sample performance was recorded after training the models on 3 million observations. The validation sample to gauge out of sample performance consisted of 1 million observations. Finally, Light GBM and Cat Boosting performed the best among all the models, after which we decided to use an ensemble method to smooth out the variances in predictions. For this ensemble method, a weighted average between Cat Boosting (95%) and LightGBM (5%) was used. The final predictions were then made after training these models on 5 million observations and taking a weighted average for predictions.

Table 1: Best Log Loss for Out-of-Sample Performance

	Logistic Regression	**XGBoost**	**LightGBM**	**Cat Boosting**
Log Loss	0.413	0.399	0.399	0.388

II. Data Understanding

The original training dataset contained over 30 million rows consisting of 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 (created from a random sample of 100K observations), the dataset is imbalanced with around 17% click rate for online ads. 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.

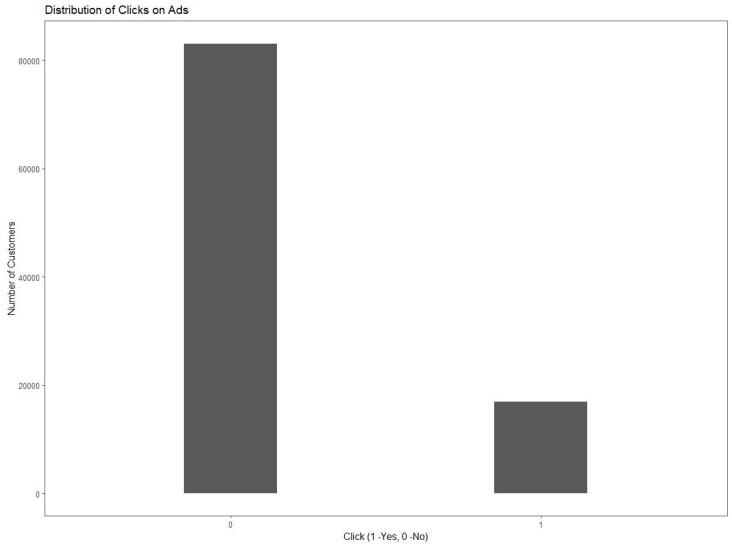

Figure 1: Distribution of Click on Ads

III. Feature Transformation for Base Models

***Encoding*:** Since most of the variables were categorical, the variables were converted into factors using as.factor function. After converting them into factors, the variables with less than 21 categories were encoded into (n-1) binary variables using one_hot function from mltools package. The original column with categories were dropped during the one hot encoding. Similarly, for variables with over 21 categories, 20 most frequent categories were obtained and encoded into binary variables.

***Feature Transformation*:** The hour variable in the dataset consisted of date and hour the ad was displayed. In its original form, the variable did not provide much information except for the date and time combination for each ad. To analyze the effect of days and times on ad clicks, the variable was converted into time bins and days of the week. For days, the dates were extracted from the variable and assigned to the day of the week. For instance, October 21, 2014 was a Tuesday, so it would have 1 for "Tuesday" variable. Similarly, for hours, a variable was created called hour to indicate which hour of the day the ad was placed. Additionally, this newly created hour variables was binned into four variables: morning (5-11am), day (11am-5pm), evening (5-8pm) and night (8pm-12am). These time bins would capture influence of times of day on ad clicks. After encoding and feature transformation, there were 138 features.

***Standardization*:** For the KNN model, the features in the dataset were standardized by subtracting the mean and dividing by the standard deviation of the training data. The standardization procedure was similar to the one we have learned in class.

IV. Base Models and Evaluation

***Sampling*:** The large dataset of 30 million rows created challenges with computation and memory. Hence, the models were initially trained with a smaller sample of 100K observations. This ensured that computations were relatively faster.

Modeling: Initially, base classification models were built for predicting click using logistic regression, random forest, and knn. The idea was to choose potentially best performing among these models to use for feature selection, cross-validation, and final model selection. The data was initially divided into training (70%) and validation (30%) sets, with 70K observations in the training sets and 30K observations in the validation set. The validation data set was used to test the out-of-sample performance of the models. The initial model was run with 138 variables most of which were binary variables to predict click.

Logistic Regression: Stepwise logistic regression was run using all variables in the dataset using glm function with family as binomial with link as logit. Forward and backward selection was allowed in the step function meaning variables could be added or removed based on their effect on the AIC. The model was trained using the training data and predictions were made for the validation data, after which the ROC was plotted, log loss and AUC were calculated.

Random Forest: Random forest model was built using the training data by assigning 500 as the number of trees and the default value for metry. The predictions were then made using the validation data after which the performance of all the three models were compared.

KNN: Before building the KNN model, the variables in the training and validation datasets were standardized by subtracting the mean of training data and dividing by the standard deviation of the training data. Following this, a knn function was built by varying the number of neighbors from 5 to 100 with an interval of 10. The predictions for each of these functions were made and log loss, as well as AUC, calculated. The value for the best k based on the lowest log loss and its corresponding value for log loss was obtained.

Evaluation: The log loss and AUC for the best model is given in the table below. Among knn, random forest and logistic regression models, logistic regression has the lowest log loss value and is the best performing model. The low performance for KNN can be attributed to the curse of dimensionality where observations are far away from each other in the feature space, which makes predictions less accurate. After this step, further feature transformation and selection was done to tune the logistic regression model and its log loss.

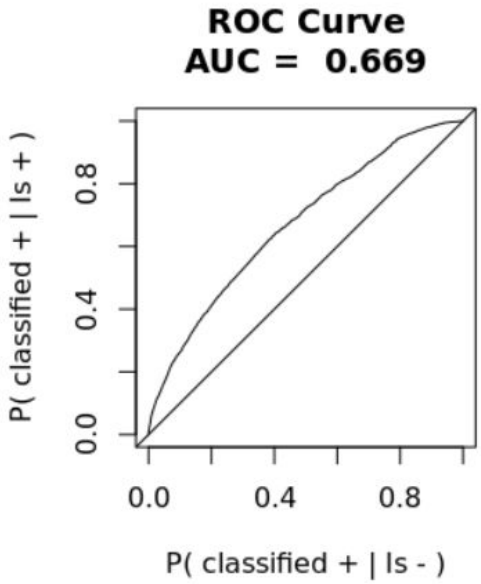

Figure 2: ROC for Logistic Regression

Table 2: Log Loss for Out-of-Sample Prediction - Best Initial Model was Logistic Regression

	Log Loss	**AUC**
Logistic Regression	0.419	0.669

V. Feature Transformation to enhance Logistic Regression Model

After selecting Logistic Regression as the best performing model, it was decided that feature selection would be used. Hence, more features were created in this step that could be removed during the feature selection phase. In this step, we changed the way variables with over 21 categories were encoded by only focusing on the most frequent categories and also added interaction variables. After this step, there were 424 features that were used in the feature selection before training the Logistic Regression model.

Encoding*:* For variables with more than 21 categories, instead of using correlation as above for filtering out the least relevant categories, top 20 most frequent categories were obtained and only these most frequent categories were encoded as binary variables.

Interactions: Four new interaction terms were created to improve the model performance:

1) "site_id" was pasted to "app_id" to create a new variable "app_site", which records a situation in which certain application is utilized to show the ad on the certain web site. For example, a certain record with app_id = ecad2386 and site_id = 1fbe01fe has an app_site = ecad23861fbe01fe.

2) "site_device" was created by pasting "device_model" to "site_domain" to present if users prefer to use certain devices to visit certain web sites. This new variable makes sense since not all web sites are suitable to be visited using a single type of device. A good example is that few

websites were not designed compatible with mobile devices and would be difficult to read unless using a web browser on computers.
3) “week_app”, a combination of the manually engineered variable “weekday” (Monday through Friday) and “app_id”, to identify the situation if few apps are more likely to be visited on weekdays.
4) “week_site” is a combination of the variable “weekday” and “site_id”, sharing the same idea with “week_app” but focusing more on the click of web sites. The four new categorical variables generated were encoded like the ones with over 21 categories and the top 20 most frequent categories were obtained.

VI. Feature Selection to enhance Logistic Regression

The analysis from this section on (apart from feature transformation) were performed in Python because of computational efficiency and compatibility with advanced models. After performing feature transformations mentioned in the section above, feature selection was performed before building the final logistic regression model.

***Feature Selection*:** Before building the models on 424 features, feature selection was performed to eliminate irrelevant features and prevent overfitting. Forward sequential feature selection was performed using feature selection from Python’s mlxtend package with scoring criteria as negative log loss.Feature selection was done using cross-validation of 5 folds to select the best features. Feature selection retained 101 features, which were used for modeling. Logistic regression was built similar to the previous part using 100k randomly sampled observations using only the features after feature selection. In this step, another cross-validation was performed to train the model using 5 folds.

Results: Despite feature selection and using cross-validation, the performance of Logistic Regression had a limit of around 0.413. At this point, we hypothesized that using advanced models based on gradient boosting might enhance the out of sample performance of these models. In order to test this hypothesis, the same 100k observations and 424 features were used to train two advanced models: XGBoost and LightGBM. The comparison process and results are explained in the following section.

VII. Comparing Logistic Regression to Advanced Models

Advanced Models: Before establishing logistic regression as the model used for our prediction, its performance was compared with two more advanced gradient boosting techniques: XGBoost and LightGBM. The term advanced comes from the idea of boosting behind these techniques. Boosting is similar to bagging, but in a more intelligent way, by giving more weight to hard-to-classify observations. This is important in this dataset because this is an imbalanced dataset and click rates are low. XGBoosting and LightGBM are all upgraded versions of Gradient Boosting Modeling while Cat Boosting (used later) skips the step to label encoding or one-hot encoding. Cross-validation was applied when training models with k = 5 and an

out-of-sample performance was evaluated, similar to Logistic Regression, using a validation set. Hyperparameter tuning was then performed to these advanced models to gauge their best performance and prevent overfitting.

***Feature Selection*:** Before building the models on 424 features, feature selection was performed to eliminate irrelevant features and prevent overfitting. Similar to the feature selection for Logistic Regression, forward sequential feature selection was performed in Python for both the advanced models using 5 folds cross validation.

XGBoost: XGBoost stands for "Extreme Gradient Boosting", an updated version of Gradient Boosting. Gradient Boosting trains many models sequentially by generating new models with gradually minimizing loss function (the log loss in this case) of the whole system using Gradient Descent method. The learning procedure consecutively fits new models to provide a more accurate estimate of the response variable. Compared with Gradient Boosting, XGBoost allows to customize loss function and regularized boosting with cross validation. It also applies parallel computing which saves time. XGBoost model was built on the same sample of 100k observations that Logistic Regression was built using the following parameters: colsample_bytree=0.6, gamma=0.1, learning_rate=0.05, max_depth=7, n_estimators=200 and random state was assigned as 42 to be able to replicate the results of the model. This model was further improved using hyperparameter tuning as described below.

LightGBM: LightGBM (Light Gradient Boosting Machine) combines both the advantages from XGBoost and Gradient Boosting and steps further. It is even more efficient than XGBoost with more time and more memories saved. To be more specific, instead of using a pre-stored decision tree algorithm such as XGBoost, LightGBM applied leaf-wise growing strategy with constraints on depth to improve the efficiency of building trees while preventing overfitting. LightGBM built using the following parameters: num_leaves = 31, max_depth = -1, learning_rate = 0.1, n_estimators = 100 and random state = 42 to keep consistent with previous models. Similar to XGBoost, further improvements were made using hyperparameter tuning.

Hyperparameter Tuning: Grid search was used to tune the hyperparameters in each of the advanced models using 5 fold cross validation to obtain the lowest log loss. For XGBoost, the learning rate and maximum depth parameters were tuned. The best log loss for out of sample for the two advanced models along with Logistic Regression is given below. As seen from the table, the advanced models performed better than Logistic Regression. Hence, the comparison from here on is focused on the advanced boosting models.

Table 3: Best Log Loss for Out-of-Sample Performance

	Logistic Regression	**XGBoost**	**LightGBM**
Log Loss	0.4127	0.4114	0.4108

VIII. Feature Transformation for Advanced Models

After establishing that advanced models had potential for better performance than logistic regression, additional feature engineering/ transformation was done before finally training the advanced models. These transformations included creating new interaction terms, creating count variables, transforming the count variables by taking a log and label encoding categorical variables instead of one hot encoding them as binary variables. This is because the advanced models used are good with handling categorical data. After label encoding the variables there were a total of 55 features.

Interaction Variables: In addition to the interaction variables created in the previous feature transformation section, some more were created. The first one created was to capture the interaction of site id and device id, another was created to capture the interaction between apps and devices and finally the last one was created to capture the interaction between a device and site id. The interaction variables created in the previous step was also retained.

Count Variables: Count variables were created to count instances of particular combinations in the dataset. For instance, a variable count_site_app would count the number of unique combinations of site ip and app ip. These count variables were also transformed by taking a log of their values. Among other feature engineering and transformation, creating count variables seems to have particularly helped the out of sample performance.

Label Encoding: The categorical variables were label encoded after converting into factor variables. Label encoding means converting the labels or categories in a categorical variable into numeric form so it can be well understood by some models. This was done instead of hot encoding variables into binary outcomes because both LightGBM and Cat Boosting work very well with categorical data and do not require data pre-processing.

IX. Advanced Modeling with Feature Selection and Hyperparameter Tuning

***Feature Selection*:** Before building the models, feature selection was performed as before to eliminate redundant features and prevent overfitting. Forward sequential feature selection was used with 5 folds cross validation in Python. The feature selection was performed using 200k observations and different features were selected as important for each of the three models.

Sample Size: Compared to the previous steps, a larger sample size of 3 million training observations and 1 million validation observations were used in this step. The validation set of one million was used to estimate an unbiased out of sample performance as in the previous steps. In addition to training XGBoost and LightGBM model, an additional advanced model was trained in this step: Cat Boost.

Cat Boost: Cat Boosting stands for Category and Boosting. It utilizes the same boosting strategy and improves its processing method to categorical variables. Usually, categorically variables are preprocessed with label encoding or one-hot encoding, Cat Boost allows these variables to be processed during the modeling process instead of being specifically preprocessed and the models generated are more robust. Cat Boost was built using the following parameters: custom_loss =

log_loss (customized), logging_level = 'Silent', iterations = 100 and random state = 42 to keep consistent with previous models.
XGBoost: XGBoost model was built using the same parameters as before: colsample_bytree=0.6, gamma=0.1, learning_rate=0.05, max_depth=7, n_estimators=200 and random state was assigned as 42 to be able to replicate the results of the model. This model was further improved using hyperparameter tuning as described below.
LightGBM: LightGBM was built using similar parameters as before. Like XGBoost, further improvements were made using hyperparameter tuning.

Hyperparameter Tuning: Hyperparameters in the models were tuned using grid search similar to the previous step with 5 fold cross validation with the objective to reduce the log loss. For LightGBM, learning rate and number of leaves were tuned during grid search.

Evaluation: Finally, the performance of the three advanced models were estimated based on the log loss in the validation data. As seen from Table 4 below, Cat Boosting has the best performance with lowest log loss followed by LightGBM and XGBoost. Since multiple models have good performance, we looked into ensemble methods to further smooth out our predictions and evaluate the log loss.

Table 4: Best Log Loss for Out-of-Sample Performance

	XGBoost	LightGBM	Cat Boosting
Log Loss	0.399	0.399	0.388

X. Final Predictions and Conclusion

Ensemble Methods: Unlike basic machine learning models, ensemble methods allow us to build a multitude of base models such as decision tree or logistic regression using different subsets of the data and splitting criterion. We can then combine the performance of all model iterations to get the generalized performance of the model. Additionally, we can further build on the performance of the ensemble method through boosting. Boosting is a technique used to strengthen weak algorithms like small decision trees by placing a weight on previously misclassified instances, iterating, and adding more weight to recurrent misclassified observations with the end goal of improving the overall model performance. By implementing a combination of models (bagging) and improving optimization through boosting, ensemble methods can lead to better out-sample performance for machine learning models.
Analyzing the benefits of the ensemble method, we wanted to use it to smooth out our predictions. Since Cat Boosting had the best performance in out-of-sample data followed by LightGBM, we used an ensemble method by taking a weighted average of predictions between

cat boosting (weighted 95%) and light GBM (weighted 5%). The ensemble method's out-of-sample performance as seen in Table 5 was very similar to Cat Boosting model's. This was expected since Cat Boosting predictions have a higher weight in the ensemble method's predictions. These models were then trained on 5 million observations and tested on different data sets. The out-of-sample performance was stable for the ensemble method of around 0.39. The final predictions for this project were then made after training these models on 5 million observations, and ensembling the predictions based on the weighted average criteria mentioned.

Table 5: Best Log Loss for Out-of-Sample Performance

	LightGBM	**Cat Boosting**	**Ensemble Method**
Log Loss	0.399	0.388	0.388

Conclusion: The project in addition to applying the tools and concepts we had learned throughout the semester, provided insights on working with big data and various models. First takeaway while working with such a big dataset was that it is much easier to work with a smaller sample to increase computational efficiency and reduce time spent on models. It provides the agility to enhance and tune the models as well. After sampling 100k observations, we compared different samples of 100k observations and the distribution of variables were very similar. This provided us with confidence to build the initial models using the relatively smaller sample of 100k. Similarly, it was also surprising that while working with binary variables, sometimes simpler models such as logistic models can be very powerful. Finally, for every model it is important to understand what type of variables it performs best with. For instance, binary encoded variables worked best with logistic regression; however, categorical variables worked best with advanced models.

Appendix: Code Files and Explanations

R_InitialModels.R: This file shows initial feature transformation and initial models (logistic regression, knn and random forest) being built. The models were built using 139 predictor variables and their performance based on log loss is evaluated. The feature transformations in this file includes encoding categorical variables with less than 21 categories into binary variables. For features with over 21 categories, only categories that have higher absolute correlation than 0.05 with predictor variable is retained. For device id and ip, most frequent categories were retained. Finally, hour variable was converted into time bins and weekdays, after which the data was split into training and test, and modeling performed.

FeatureTransformation.R: This file shows the final feature transformations in R after which the training and validation data were written to csv and imported in Python for further modeling. Changes were made to the previous file in how variables with over 21 categories are encoded. Instead of using correlation, 20 most frequent categories were obtained and encoded into binary variables. Similarly, interaction features were also created.

Logitic_Boosting.ipynb: In this file, feature selection for logistic regression was performed and its performance was compared to that of XGBoost and LightGBM. Additionally, feature selection and parameter tuning was also performed for the boosting models.

LabelEncoding: This file is an addition to feature transformation done in the FeatureTransformation file to include count variables and categorical ones that advanced models handle well. In this file, some count variables have been created. For instance, the frequency of using a particular site on a particular day is counted in a variable called count_app_site. Similarly, some variables have been label encoded instead of one hot encoding. This is done because advanced boosting models work well with categorical variables.

AdvancedModels.ipynb: This file shows comparison of performance between the advanced models: XGBoost, LightGBM and Cat Boosting, in addition to feature selection and hyperparameter tuning for each of the models.