Adtracking Fraud Detection using Boosting Algorithms and Neural Networks

Catherine Lee
clee200@ucsc.edu
University of California, Santa Cruz
Santa Cruz, California

Kyle O'Brien
kdobrien@ucsc.edu
University of California, Santa Cruz
Santa Cruz, California

Amit Saxena
asaxena@ucsc.edu
University of California, Santa Cruz
Santa Cruz, California

ABSTRACT

This document details the motivations, methodologies, and results of our team's effort to develop a machine learning model to predict with a mobile advertisement click is fraudulent or not. Each of the following sections details our approach and outcomes - the results of which allowed us to achieve 83% accuracy using XGBoost.

1 MOTIVATION AND OBJECTIVE

Fraudulent mobile ad clicks have become an increasingly difficult problem for advertisers as the world experiences a proliferation of smart mobile devices. Advertisers pay affiliate website when their users click on one of their mobile ads.

This compensation model has incentivized bad actors to generate fraudulent clicks on mobile ads in the hope of conning advertisers into paying them a commission. Advertisers must compensate affiliates even when these clicks do not result in a download of the advertised application.

In response to this problem, we've developed a machine learning model that can predict whether a user will download an app after clicking a mobile advertisement. By training our model on a dataset of over 100,000 mobile advertisement clicks spanning three days, we were able to achieve 83% accuracy using XGBoost.

2 DEVELOPMENT ENVIRONMENT

Our team elected to spend the upfront time investing in shared developer tooling and processes with the aim of avoiding the issue and blocking issues around dependency requirements. Python is especially difficult to work within teams natively due to a wise distribution of versions of the ore language and popular libraries. Additionally, each one of our team members working on different operating systems. This presented an initial challenge of assuring our development environments were consistent and that we wouldn't run into the issue of "well it works on my machine". The two primary tools we've employed to address this is Pipenv and Docker.

Pipenv is a new Python environment manager where developers can explicitly define the Python version and libraries in a Pipfile local to that project. This is especially convenient since we can then not have to rely on global package installations to satisfy dependency requirements for our project.

Docker is an application level virtualization tool. The primary draw of spending the time configuring Docker was that it went beyond Pipenv and allowed us to run our project inside an entirely virtualized managed Ubuntu environment. Thus even though all of our team members work on different operating systems, our project behaves exactly the same since it's ran inside a container.

While the investment in these developer productivity tools came at the cost of initial development towards our model, this will prevent many hours of bugs and dependency configuration in the future.

3 DATASET

3.1 Features

Our dataset includes training data csv file (train.csv) with these columns/features:

- Click_id: click id (input to our model)
- ip: ip address of click.
- app: app id for marketing.
- device: device type id of user mobile phone (e.g., iphone 6 plus, iphone 7, huawei mate 7, etc.)
- os: os version id of user mobile phone
- channel: channel id of mobile ad publisher
- click_time: timestamp of click (UTC)
- attributed_time: if user download the app for after clicking an ad, this is the time of the app download
- is_attributed: the target that is to be predicted, indicating the app was downloaded (output to our model)

We are also given a testing data csv file (test.csv) with the same columns above but without the is_attributed column (the column we are trying to predict). The is_attributed column has either of 2 values: ”1” stands for a positively attributed click (not fraudulent) and ”0” stands for a non-attributed (is fraudulent) click

The training data set contained 20 million clicks which was too much for any of the group members to load in memory at once. Thus, we initially used the sample training set (train_sample.csv) provided by Kaggle of 100,000 rows to explore and train our first models.

We are also given a testing data csv file (test.csv) with the same columns above but without the is_attributed column (the column we are trying to predict).

3.2 Exploratory Data Analysis

When exploring the sample training set of 100,000 clicks, we first noticed a huge imbalance in the number of positive attributed clicks

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

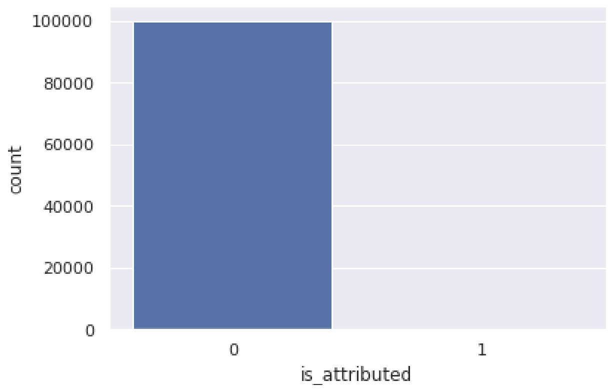


Figure 1: Before Downsizing

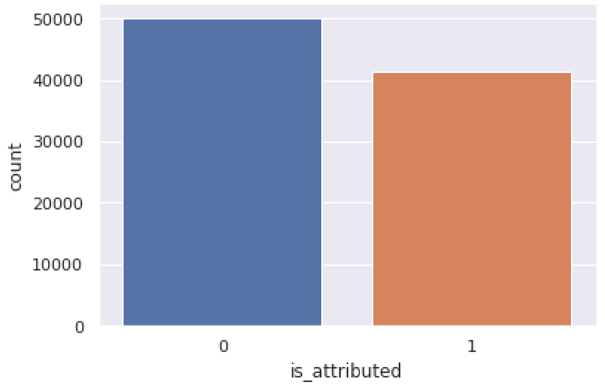


Figure 2: After Downsizing

and non-attributed clicks. There were only 227 attributed clicks out of the 100,000 total clicks. Thus, there were only 227 non-null values for attributed_time because this feature only applies for clicks that were attributed in the first place. Thus, we decided to build a script that would randomly select 100,000 clicks from the full training set of 200 million clicks such that there was a roughly even distribution of ”1” and ”0” for the is_attributed column.

We then wanted to see correlations between click counts and click conversion rate (defined as the percentage of the current click count that was not fraudulent) against the features ip, app, os, device, channel, and time.

Interestingly we found that certain IP id's had thousands of clicks associated with them. That lead us to believe that the IP feature correlated to network IPs rather than device level IPs. Furthermore, we were able to see patterns in click counts over time, where certain times in day experienced dips in click frequency, but we were not able to see patterns in click conversion rate over time, app, os, device, or channel.

We were also curious about differences in our training and test data. Since time data was a feature (click_time), the immediate question was whether the training and test data were obtained at overlapping or discrete time intervals. Upon exploration, the training data was obtained for 4 days between 2017-11-06 and 2017-11-09, while the test data was obtained on 2017-11-10. This confirmed our belief that click_time values should be converted to values relative to the start of the day in order for this column to be valuable in a model.

3.3 Feature Engineering

The click_time feature was given to us as a string in the ”yyyy-mm-dd hh:mm:ss”. So in order to use this feature we wanted to engineer it into a qualitative value that our models could use to extrapolate patterns. So we used click_time to create the columns ”click_time_secs” and ”click_time_mins” that were each the number of seconds and minutes since midnight for each time value. We hoped that our models would be able to infer patterns between time of day and fraudulent clicks with these engineered columns.

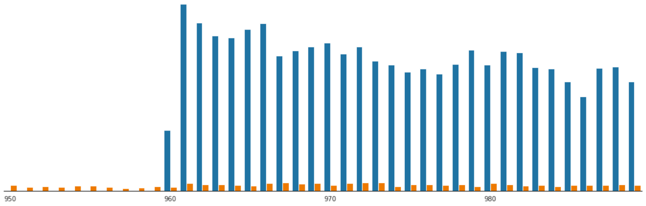


Figure 3: Count of is_attributed vs Click Time in minutes (Orange being class of 1. Blue being class of 0).

Furthermore, to train our data on an equal amount of fraudulent and non-fraudulent data, we created a script to randomly select 100,000 clicks from the training set such that the 2 values for is_attributed were in roughly an even distribution. The consequences of doing this was that conversation rate ratios according to click_time features would not resemble the original ratios in the training set, thus possible making the click_time features less reliable to train against.

Also, we decided to remove the attribute_time from all our engineered training sets because this column was not present as a feature in the test set (this was intentionally done by the Kaggle problem because a non-null value would mean a click was attributed, thereby making the problem trivial).

4 MODELS AND ALGORITHMS

Our dataset contains a considerable amount of labeled data, so we wanted an algorithm that is supervised and intended for classification, not regression. There is no one model that works best for every problem, so we tried a myriad of binary classification algorithms ranging from basic techniques such as Logistic Regression to Ensemble Learning to neural networks.

We first split our training dataset with 100,000 data points into a training and validation set. We trained and fitted our models on the training set and then did an initial evaluation of the model on the validation set.

4.1 General

We used Logistic Regression, Naive Bayes, Perceptron, Decision Tree and Support Vector Machine with out of the box hyperparameters.

4.2 Ensemble Learning

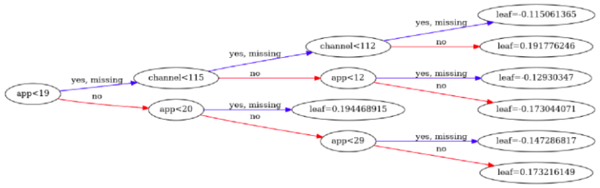


Figure 4: XGBoost

Ensemble learning is the technique of combining weak classifiers and taking the majority outcome (the way the majority outcome is determined is dependent on the type of ensemble) out of all the weak classifiers.

The boosting algorithms we used were Gradient Boosting Trees, XGBoost, Adaboost, and Random Forest.

4.3 Neural Network (Multilayer Perceptron)

We trained a three-layer sequential neural network model in Keras. We found that the ideal hyperparameters were 40 neurons in the first layer, 10 neurons in the second layer, and 1 neuron in the last layer with an input dimension of 4 and a sigmoid activation. We used the Adam optimizer which is similar to Stochastic Gradient Descent. To calculate our loss, we used binary cross entropy. We used 33% of our training data for our validation set and ran it for 300 epochs with a batch size of 32.

5 RESULTS AND ANALYSIS

In a binary classification problem, our goal is to get greater than 50% accuracy to justify that our algorithm works better than blanket guessing.

Initially, all of our algorithms had a 99% accuracy on the validation set when making predictions on the imbalanced dataset. When we looked closely, the true positive (with positive being a class of "1") rate was close to zero percent because our model was incentivized to predict a class of 0 for everything, since it had been exposed to a deluge of training data points where the outcome was overwhelmingly "0".

It was easy to miss the detail that our true positive rate was very low because we were only looking at the mean accuracy. The mean accuracy will still output 99% even if we predict all the outliers incorrectly, because we have 227 data points for outcome of 1 and greater than 99,000 for outcome of 0. We decided that using mean accuracy alone to measure the goodness of our model was not reliable, because our data set has an imbalance of outcomes.

To combat this issue, we included more robust and thorough evaluation metrics to ensure that that our model was recognizing patterns, rather than just guessing âĂIJ0âĂİ. We looked at other metrics in conjunction with the mean accuracy to determine the best model for this problem such as Precision, Recall, and F1 Score. Precision is the the number of true positives over all actual results. Recall is the number of true positives over predicted results. Using recall ensures that we catch our true positives. Although both metrics are important, we are prioritizing our recall, since our problem calls for us to detect outliers (class of "1"). F1 Score is the mean of both precision and recall. Since F1 Score takes into account both metrics, we will be looking at F1 Score to evaluate our models in addition to the mean accuracy.

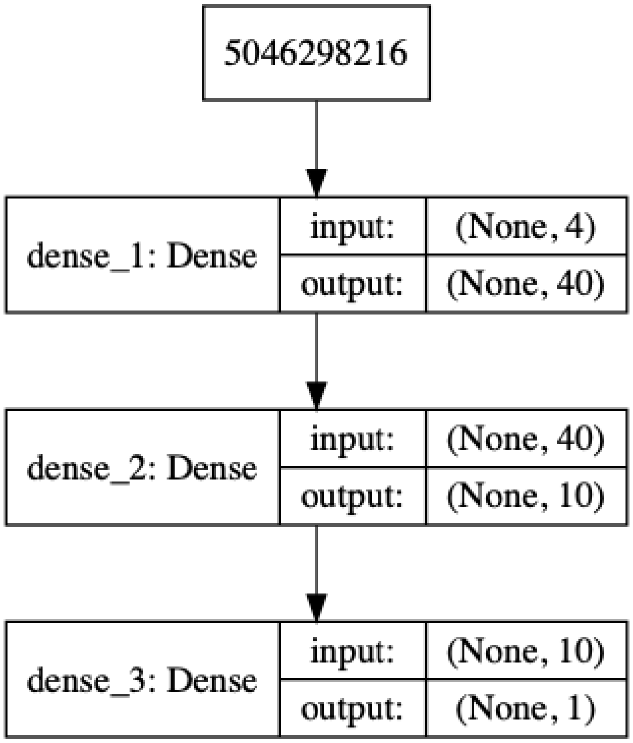


Figure 5: Layers in Neural Network

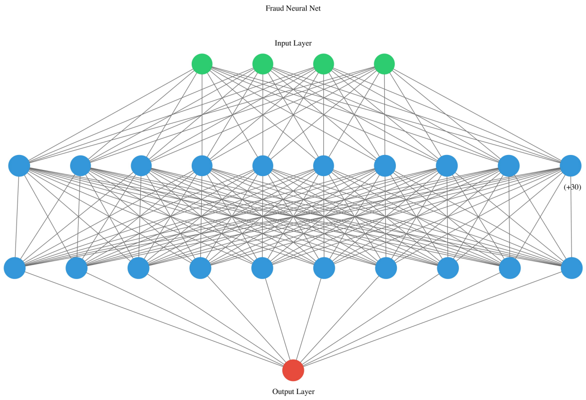


Figure 6: Neurons

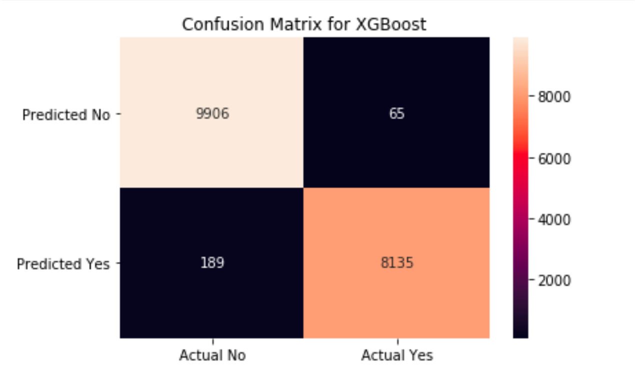
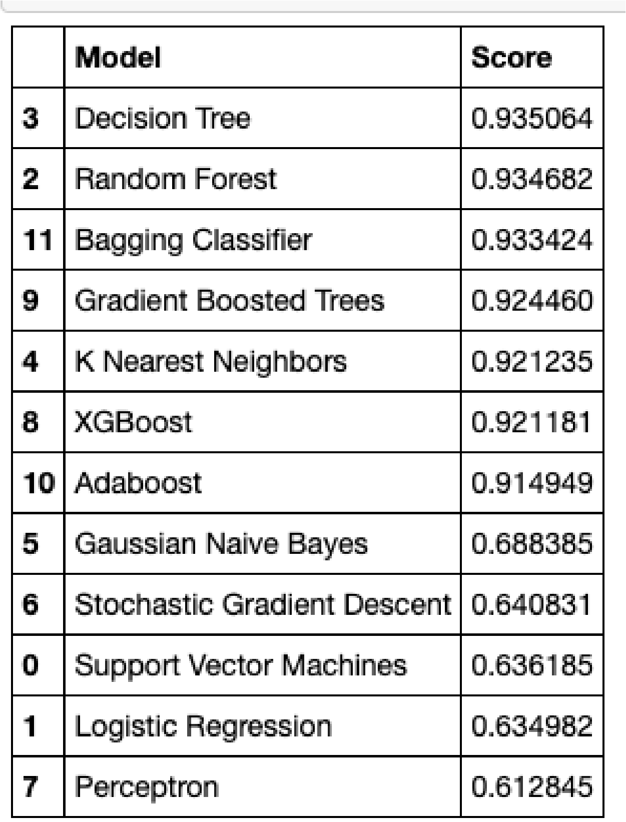


Figure 7: Confusion Matrix for XGBoost



	Model	Score
3	Decision Tree	0.935064
2	Random Forest	0.934682
11	Bagging Classifier	0.933424
9	Gradient Boosted Trees	0.924460
4	K Nearest Neighbors	0.921235
8	XGBoost	0.921181
10	Adaboost	0.914949
5	Gaussian Naive Bayes	0.688385
6	Stochastic Gradient Descent	0.640831
0	Support Vector Machines	0.636185
1	Logistic Regression	0.634982
7	Perceptron	0.612845

Figure 8: Mean Accuracy for validation set

I hypothesized that Logistic Regression would be the model that gave us the highest accuracy, because it has a sigmoid activation function that is ideal for binary classification and can fit itself to outliers, but it was actually one of lowest scoring models both in terms of mean accuracy and f1 score.

Logistic Regression, Gaussian Naive Bayes, Stochastic Gradient Descent, Perceptron, and Support Vector Machines all scored below 70% mean accuracy.

We had more success with ensemble learning algorithms. Gradient Boosted Trees, Random Forest, XGBoost, and Decision Tree had over 90% mean accuracy and f1 score on the validation set.

We submitted the top performing algorithms on the validation set to Kaggle to see our test accuracy. The test set on Kaggle has a 50/50 split of classes of 1 and 0. The algorithms that we submitted (Decision Tree, KNN, XGBoost, and Random Forest) all had scored around 80% accuracy, with XGBoost scoring the highest with 83%. These algorithms were the same ones that scored over 90% on the validation set. This discrepancy was very strange, but we suspect that this is because the testing set was time stamped 3 days before our training dataset.

In general, ensemble methods performed much better than others and out of the ensemble methods, boosting techniques were the most successful. We believe this is because boosting is effective at fraud detection cases due to it combatting imbalanced datasets by having many classifiers make a prediction and then taking a weighted average out of all the weak classifiers, creating a stronger confidence in our final prediction.

6 CONTRIBUTION

6.1 Kyle O'Brien

- Development Environment: Drawing from past experiences of teams working on machine learning models professionally, Kyle prevented our team from running into dependency issues by using Docker and Pipenv to develop a virtual development environment where all dependencies were clearly specified and automatically installed. This saved us hours of each team members having to sync the versions of Python and the supporting packages on our local machine in order to have the model run the same across machines.
- Training Data: Kyle allowed our team to address the issue of class imbalance in our training data by developing a Python script that went through the 183 million CSV records in our training data and returned a new equalized CSV file of 100 thousand records. This step was essential in allowing our model to train correctly.
- Pitch/Presentation Strategy: Drawing from experience pitching projects at hackathons and public speaking skills, Kyle determined our pitching strategy and coached team members in how to pitch to professors and industry professionals under pressure. This allowed our team to deliver compelling, informative, and engaging presentations at the final poster presentation event.

6.2 Catherine Lee

- Models and Classification Metrics: Researched other methodologies (credit card fraud detection) and implemented all binary classification algorithms for our project such as Decision Tree, Voting Classifier, Bagging Classifier, Random Forest, Gradient Tree Boosting, K Nearest Neighbors, XGBoost, Adaboost, Gaussian Naive Bayes, Stochastic Gradient descent, logistic regression, perceptron, SVM, and neural network architecture (and hyperparameters). She displayed all classification metrics such as mean accuracy, precision, recall, and f1 score for each algorithm.

- Train/Validation/Test: Submitted models to Kaggle after training and validating to see our test accuracy.
- Exploratory Data Analysis: Interpreted data and recognized patterns in our data (such as the 99% to 1% class imbalance) with techniques such as a correlation matrix, click_time vs. time series graph, and a visualization of positive to negative classes. She suggested and delegated tasks to the team such as visualizing XGBoost with the graphviz library, feature engineering of click_time feature, and creating confusion matrices based on analyses of the data.
- Paper: Read research papers and documentation in order to write these three sections: Models and Algorithms, Results and Analysis, Future Work

6.3 Amit Saxena

- Visualizations: Worked on a variety of visualizations for our different models, Created confusion matrices for decision trees and random forest model, Created an ROC-AUC graph for Random Forest model, Debugged and fixed library issues regarding 3rd party libraries used for complex visualizations (i.e. xgboost and graphviz)
- Feature Engineering: Converted the default feature ”click_time” from the format ”yyyy-mm-dd hh:mm:ss” to new columns ”click_time_secs” and ”click_time_mins” which respectively represent the amount of seconds and minutes of that click time since the beginning of the day.
- Development Environment: Setup the Docker environment to use volumes so container data persisted to host machine data Troubleshooted Docker and pipenv errors to keep everyone's development consistent and easily workable

7 ACKNOWLEDGMENTS

Thanks to Narges Norouzi, Rafael Espericueta, and Marcelo SieroâĂŹs instruction and guidance for a great quarter!

8 FUTURE WORK

There are some complex models that have been used for other types of fraud detection, but not specifically for adtracking fraud detection. Autoencoder models have been used in credit card fraud detection cases, so this is something we can try to apply to adtracking fraud detection. We decided to not try it in our project initially because it only obtained a 70% accuracy detecting credit card fraud [1]. Another approach we can try is generative adversarial models, which have been used to detect telecom fraud to produce a probability for how fraudulent a data point is [2].

9 REFERENCES

References

[1] https://medium.com/@palashshinde6/credit-card-fraud-detection-using-deep-learning-models-a8a918e29d4e

[2] Zheng, Zhou, Sheng, Xue, Chen, 2018. Y.-J. Zheng, X.-H. Zhou, W.-G. Sheng, Y. Xue, S.-Y. Chen. Generative adversarial network based telecom fraud detection at the receiving bank. Neural Netw (2018). Google Scholar

[3] Rushin, G., Stancil, C., Sun, M., Adams, S., Beling, P. (2017, April). Horse race analysis in credit card fraudâĂŤdeep learning, logistic regression, and Gradient Boosted Tree. In Systems and Information Engineering Design Symposium (SIEDS), 2017 (pp. 117-121). IEEE.