**Predicting AD click fraud**

**ML predictive analysis using Azure ML and Databricks Spark ML**

Hai Anh Le

Neha Gupta

Maria Boldina

MSIS graduate students

California State University, Los Angeles

**Abstract:** This project will illustrate the usage of Azure ML and Databricks on machine learning analysis. We have used data from a Kaggle competition named “TalkingData AdTracking Fraud Detection Challenge”. Our goal is to use two algorithms in Azure ML and Databricks to predict who will download an application after clicking an ad. If the person clicked on an ad without downloading, we will categorize them as fraud soley for the purpose of this project.

URL: <https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data>

Training data size: 7GB ● Format: .csv

Python implemmented data: 1GB (15%) ● Format: .csv

Technical specifications:

* Azure ML: Free workspace, 10GB storage, single node, South Central US Region
* Databricks: subscription version, cluster 4.0 (Apache Spark 2.3.0, Scala 2.11), 2 Spark workers, 16GB storage and 4 cores, Python 2.7

**1. Introduction**

TalkingData is the largest independent big data service company in China. Their network covers 70% of the mobile services nationwide with 3 billion ad clicks per day. Amongst those clicks, 90% are potentially fraudulent. Click fraud is happening at an overwhelming volume leading to misusage of data and wasting money. Hence, Kaggle (a platform for predictive modelling and analytics competitions from the U.S.) has partnered up with TalkingData to help resolve this issue. Our team has come across the competition and decided to participate in this process and make it our own term project. Below is the process how we integrated this project to predict ad click frauds:

* Master training data: 7GB with 200 million ad clicks within a 4-day period.
* Filtering data via Python using Linux terminal: new data is 1GB.
* In the dataset, we have 8 columns. Feature columns include: IP, app, device, OS, channel, click\_time, attributed\_time. Label column: is\_attributed.
* Building two algorithms on Azure ML: Two-Class Decision Forrest and Two-Class Decision Jungle.
* Building two algorithms on Databricks: Decision Tree Classifier and Random Forest Classifier
* Visualization and comparison between 4 models.

2. Data processing and constraints

**2.1 Constraints**

* In our original dataset, there is only 0.19% of the ad clicks out of 200 millions that downloaded the application and 99.81% did not download.
* We categorize downloaded as positive (1) and not downloaded as negative (0).

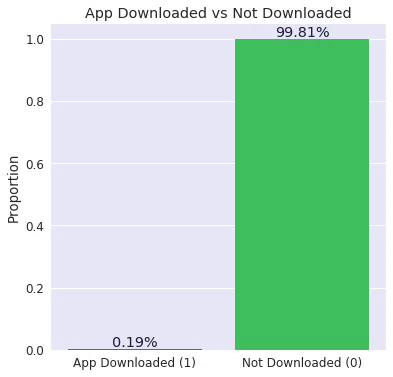


Figure 1. Actual data comparision

* Althrough after applying Python to reduce the datasize, we had to partition and sample to even minimize it more (only used 8% of 1GB).
* We used two data balancing methods: SMOTE and Stratified.

**2.2 Processing imblanced data**

* SMOTE:

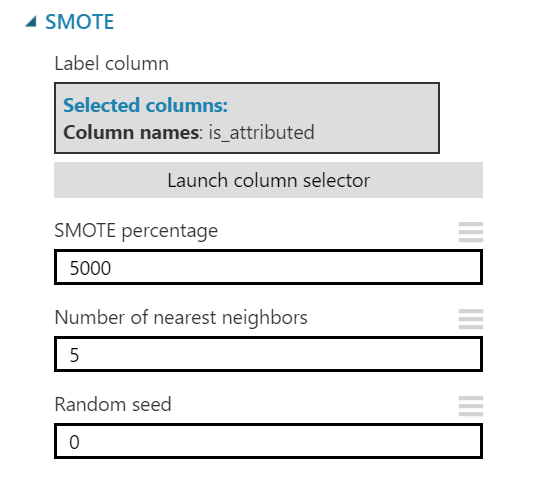


Figure 2. SMOTE setting

SMOTE stands for Synthetic Minority Over Sampling Technique which takes a subset of data from the minority class and creates new synthetic similar instances. It helps balancing data & avoid overfitting. In our case, SMOTE increased percent of minority class (1) from 0.19% to 11%.

* Stratified split:

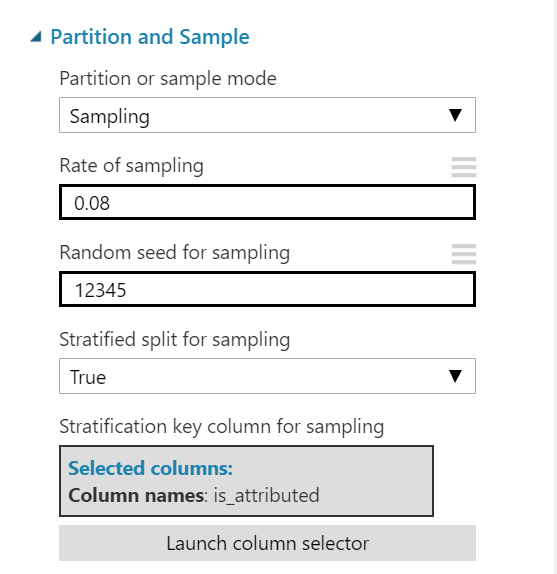


Figure 3. Stratified split

Stratified Split ensures that the output dataset contains a representative sample of the values in the selected column. In our case, it ensures when we random sample, the results are not all 0’s. We used 8% of the 1GB for the speed purpose.

**3. Azure ML**

We have selected to use Two-Class Classification for our prediction purposes. Particularly, it is binary classification. It is the task of classifying the elements of a given set into two groups. As mentioned in the introduction, we have 7 feature columns which contain information and data that would determine our prediction and 1 label column which contains the actual values (1 and 0). Our goal is to create a scored label column that contains predicted values (new 1 and 0) to compare with the actual values.

Within Classification, we chose two decision tree methods: Decision Jungle and Decision Forest. Decision trees often perform well on imbalanced datasets because their hierarchical structure allows them to learn signals from both classes.

Before going into details about the two class decision models, we would like to discuss about setting performance metrics.

Table 1. Class distribution

A screenshot of a cell phone

Description generated with very high confidence

In the classification model, there are four main measures: accuracy, precision, recall and F1 score. Both precision and recall work well if there’s an uneven class distribution as is often case. They both focus on the performance of positives rather than negatives, which is why it’s important to correctly assign the “positive” predicate to the value of most interest. Particularly for our project, **Precision** is the key measure with False Positives (FP) indicate that the model falsely predicted an application was downloaded while it was not actually downloaded. Focusing on improving False Positives will help save money for advertising.

**3.1 Two-Class Decision Jungle**

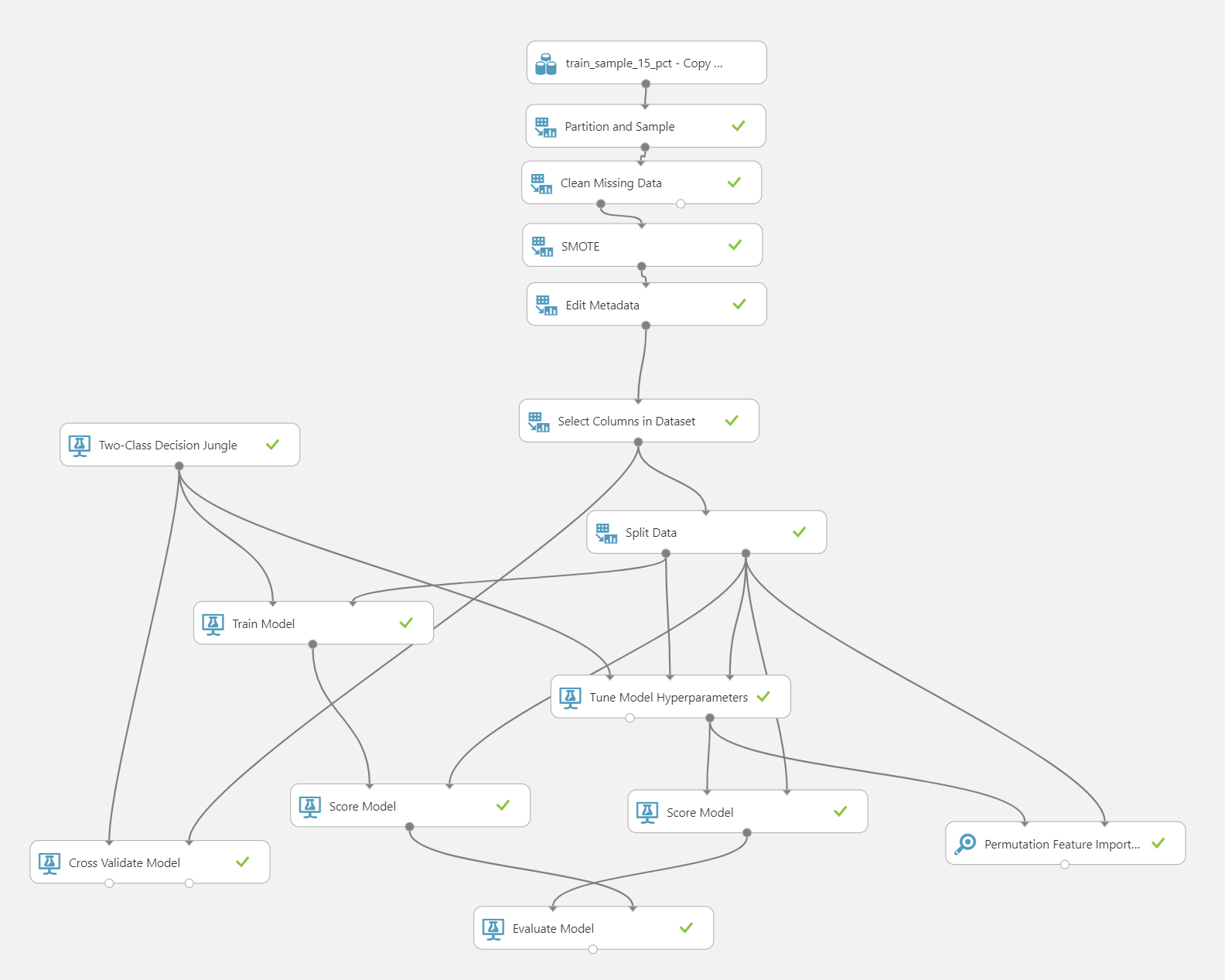


Figure 4. Two-Class Decision Jungle model on Azure ML

In this model, we used 8% of the sample, setting up SMOTE to 5000%, using 70:30 Split train, setting cross validation as 10 folds, using tune model hyperparameters, score and evaluate models.

* Result #1: AUC = 0.606



Figure 5. Two-Class Decision Jungle AUC score

* Result #2 with tune hyperparameters, AUC = 0.905

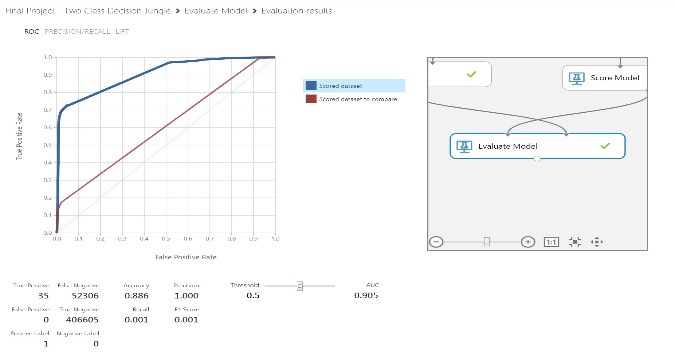


Figure 6. Two-Class Decision Jungle with hyperparameters

**3.2 Two-Class Decision Forest**

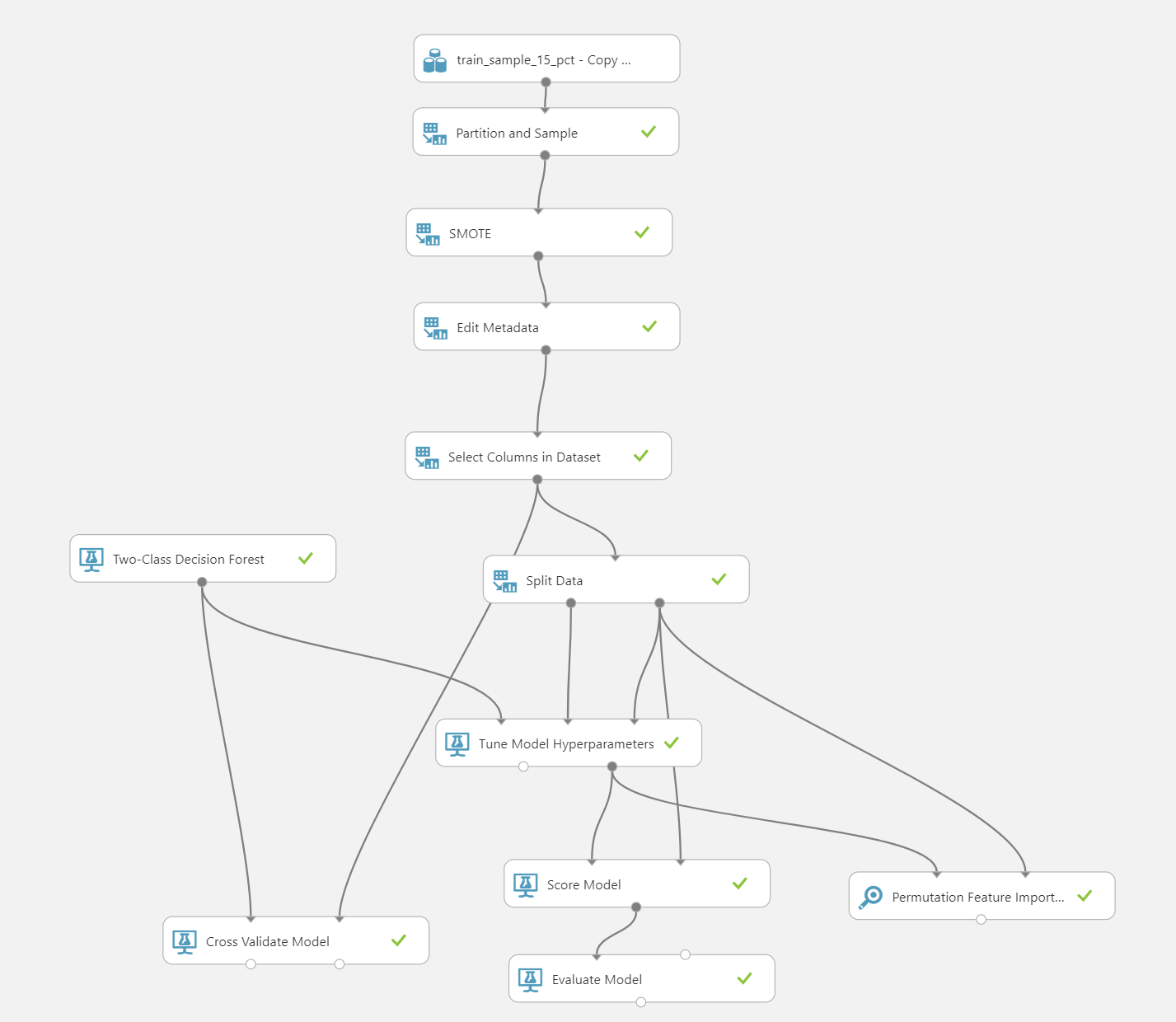


Figure 7. Two-Class Decision Forest Azure ML model

In this model, we used 8% of the sample, setting up SMOTE to 5000%, using 70:30 Split train, setting cross validation as 10 folds, using tune model hyperparameters, score and evaluate models.

* Cross validation: 10 folds
* Result #1: AUC = 0.987, permutation best columns: app and channel, precision = 1.0, TP = 508

A screenshot of a cell phone

Description generated with very high confidence

Figure 8. Two-Class Decision Forest - First AUC

* With permutation feature importance: app, channel. We run the same model but with only the two best columns that Permutation Feature Importance chose. **And the AUC increased from 0.987 to 0.997. However, the precision dropped from 1 to 0.968**
* Improving precision: we moved the threshold in figure 10 from 0.5 to 0.8 and the precision increased from 0.968 to 0.992 with lower FP = 377.
* In figure 9, we have is\_attributed column as **actual values** and scored labels column as **predicted values**.

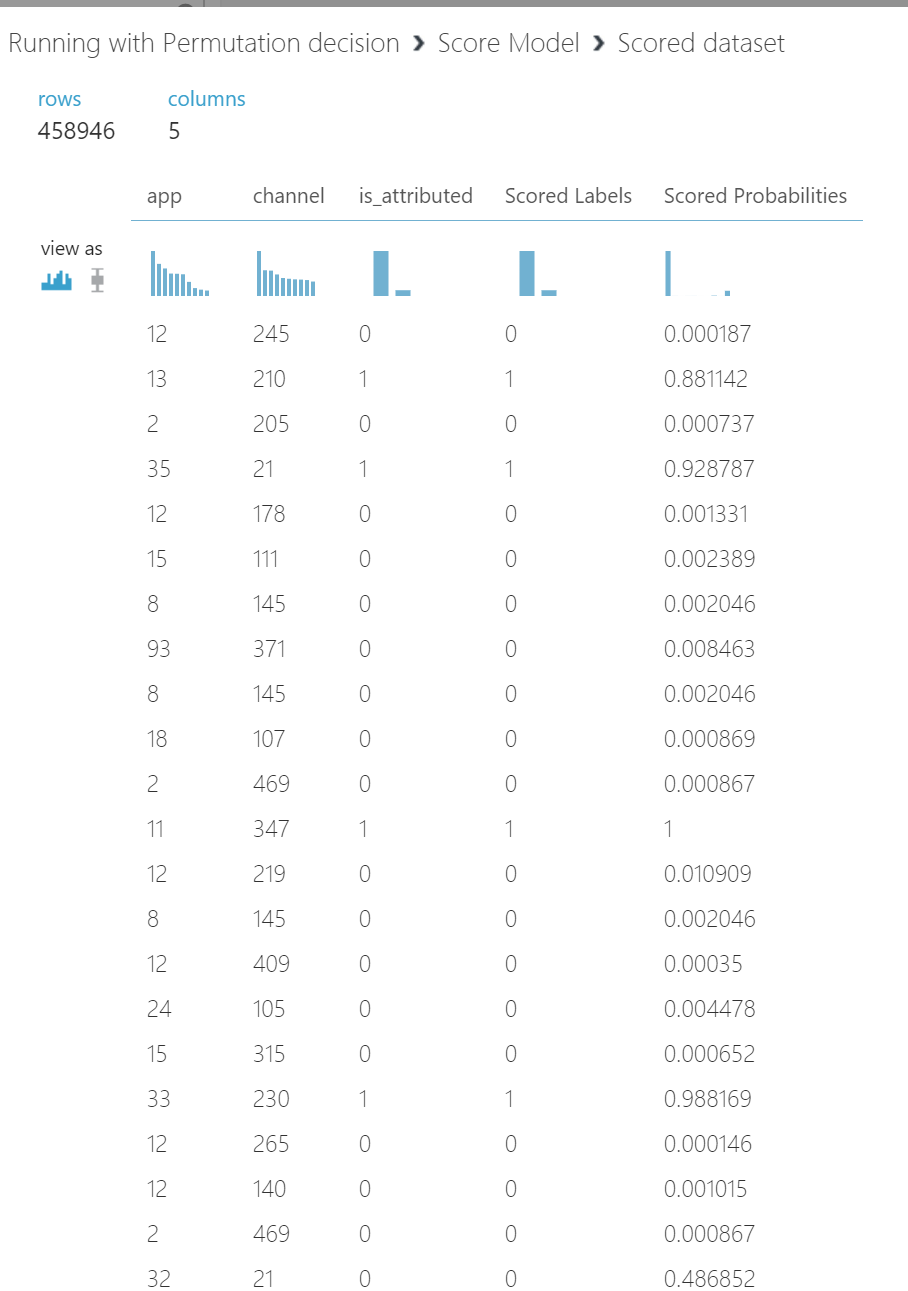


Figure 9. Permutation Feature Importance Result

**3.3 Comparison between two models**

Table 2. Comparison between two-class decision models in Azure ML

**A screenshot of a cell phone

Description generated with very high confidence**

►Two-Class Decision Forest is the best model.

**4. Databricks Spark ML**

We have selected to use Binary Classification for Spark ML. It will predict a category to either downloaded (1) or not downloaded (0). Binary or binomial classification is the task of classifying the elements of a given set into two groups (predicting which group each one belongs to) on the basis of a classification rule. The two main classes that we used are Decision tree classifier and random forest classifier. Below is the combination of features used to analyze data:

A close up of a logo

Description generated with very high confidence

For defining and grouping different features, we used group\_by and join functions to select different groups of data to analyze. Next, we split/trained the data set, defined the pipeline and assigned the pipeline values to the Decision Tree Classifier and Random Forest Classifier models. In order to tune our models, we set the parameters (maxDepth, maxBins) for paramGrid builder and used TrainValidationSplit to tune our models with 80% of training data and 20% for validation. Data size in Databricks will be 1GB. The final steps are testing the models, exploring the confusion matrix and determining the RMSE.

**4.1 Decision Tree Classifier**

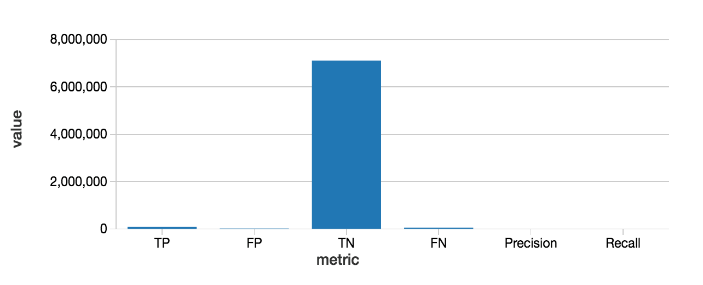


Figure 20. Decision Tree Classifier Confusion Matrix

**4.2 Random Forest Classifier**

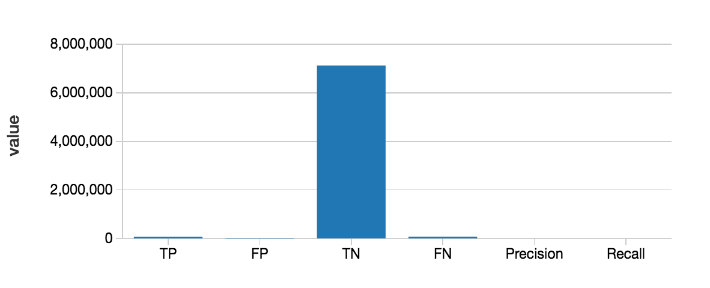


Figure 11. Random Forest Classifier Confusion matrix

**4.3 Comparison between two models**

Table 3. Comparison between two classifier models in Spark ML

A screenshot of a cell phone

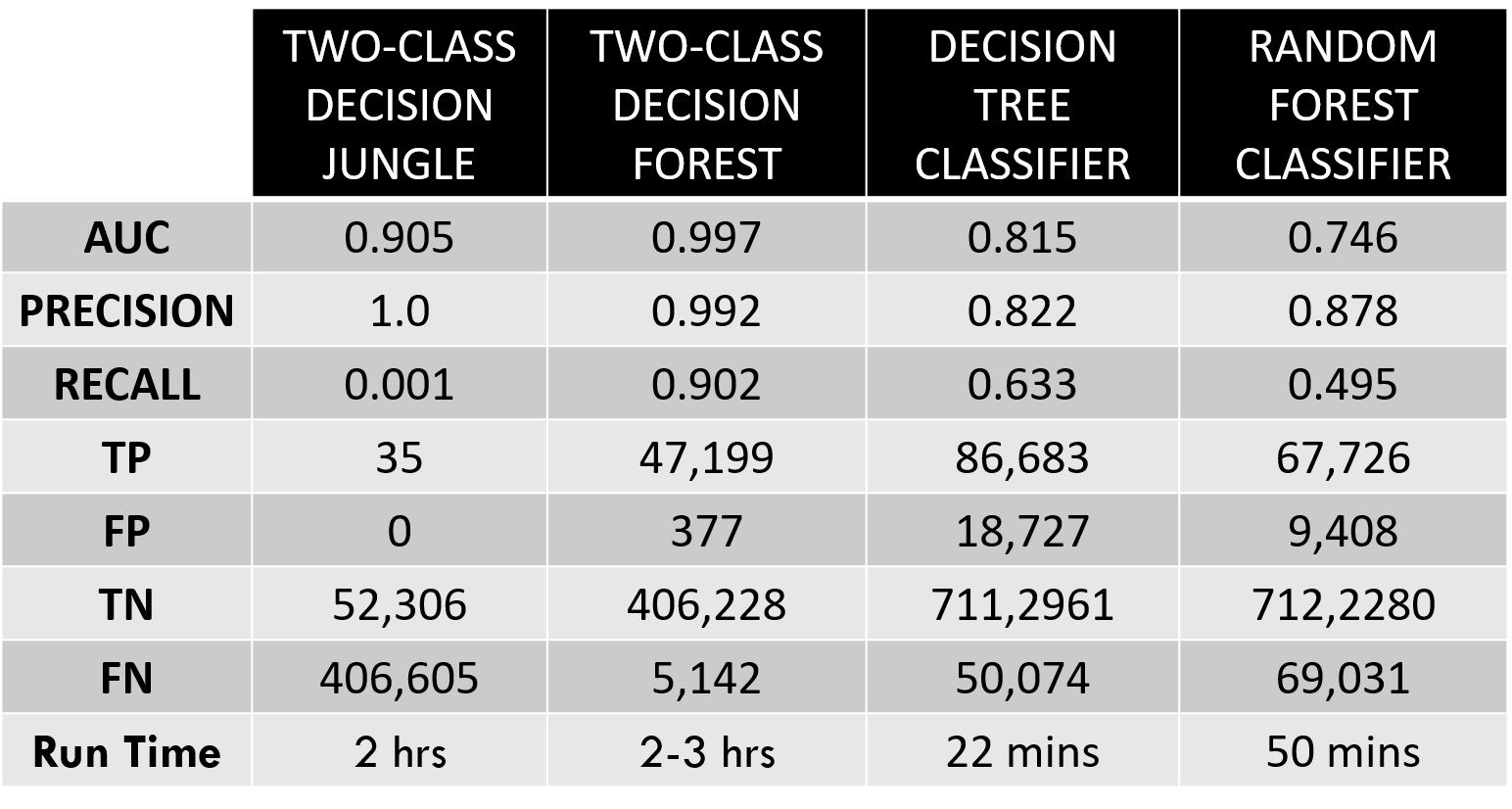
Description generated with very high confidence

►Decision Tree Classifier is the best model.

**5. Comparing all four models and conclusion**

We have tested more than 20 experiments on Azure ML and Spark ML in order to conduct our final results. In Spark ML, we decided to use the subscription version to increase the running speed and prevent cluster shutdown. The results received from all four models are phenomenal.

Table 4. Final comparison of all four models



►Two-Class Decision Forest is the best model.

In conclusion, our experiments have shown a high precision rate of predicting who downloaded or not downloaded an app after clicking an ad. The best model we would recommend using for TalkingData is the Two-Class Decision Forest. By categorizing who downloaded or not downloaded the app, we would know which IP is fraudulent and which is not. Therefore, companies in China could better target their audiences and implement more efficient marketing plans. It will then avoid fraudulent practices from illegitimate accounts and save money.

**6. Github URL**

* <https://github.com/ngupta8>

**7. References**

* <https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data>
* <https://blogs.msdn.microsoft.com/andreasderuiter/2015/02/09/performance-measures-in-azure-ml-accuracy-precision-recall-and-f1-score/>
* <https://docs.microsoft.com/en-us/azure/machine-learning/studio/>
* <https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice>
* <https://docs.databricks.com/spark/latest/mllib/binary-classification-mllib-pipelines.html>