Dwarkadas J Sanghvi College of Engineering

Final Project Report

on

Fraud detection in mobile ADs using ML

Team members:

NAME	SAP ID
Rajus Nagwekar	60002180080
Sarthak Mistry	60002180098
Shrenik Ganguli	60002180105

Guided by:

Industry Mentor:

Table of Contents

Executive Summary

Click fraud is the act of illegally clicking on pay-per-click (PPC) ads to increase site revenue or to exhaust a company's advertising budget. Click fraud is different from invalid clicks (those that are repeated or made by the ad's host/publisher) in that it is intentional, malicious, and has no potential for the ad to result in a sale. Click fraud happens with pay-per-click advertising and may involve either a human, a computer program, or an automated script pretending to be a legitimate user and clicking on paid search advertising with no intention of purchasing something. Fraud risk is everywhere, but for companies that advertise online, click fraud can happen at an overwhelming volume, resulting in misleading click data and wasted money. Ad channels can drive up costs by simply clicking on the ad at a large scale. With over 5.3 billion smart mobile devices in active use every month, the world therefore suffers from huge volumes of fraudulent traffic

Background

Aim

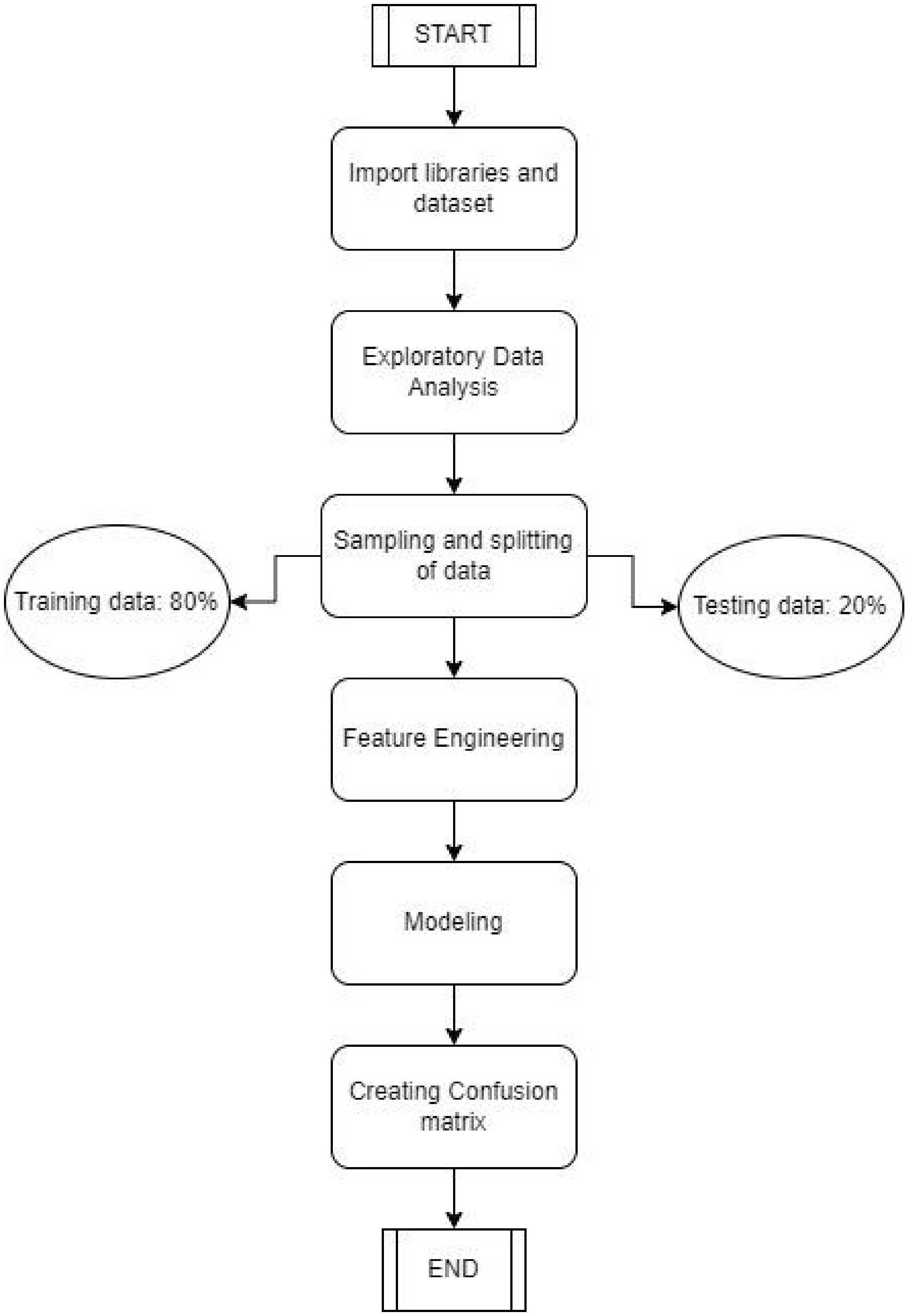
In this project we aim to build an algorithm that predicts whether a user will download an app after clicking a mobile app ad by analyzing and finding patterns on user behavior such as the type of device owned by them, the id of the advertisement publisher. By doing so, we aim at finding such fraudulent companies and blacklisting them as prevention is always better than cure.

Technologies

The only facilities that would be required for this work would be:

- Laptop/PC
- Jupyter Notebook/Google Colab
- Open source python libraries like Numpy and TensorFlow

Software Architecture



SYSTEM

METHODOLOGY COMPARISON

Talking data, China's largest independent big data service platform, covers over 70% of active mobile devices nationwide. They handle 3 billion clicks per day, of which 90% are potentially fraudulent. The dataset to be used for this project is provided by Talking Data. It consists of 10000 data points and 8 features.

- ip
- app
- device
- os
- channel
- click_time
- attributed_time
- is_attributed

We plan to implement the following algorithms:

- **Decision tree**
- **AdaBoost algorithm**
- **XGBoost algorithm**

Decision Tree

● It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome.

● In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

● The decisions or the test are performed on the basis of features of the given dataset.

● It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

● It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

● In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.

● A decision tree simply asks a question and based on the answer (Yes/No), it further splits the tree into subtrees.

Steps:

● **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.

● **Step-2:** Find the best attribute in the dataset using Attribute Selection Measure (ASM).

● **Step-3:** Divide the S into subsets that contain possible values for the best attributes.

● **Step-4:** Generate the decision tree node, which contains the best attribute.

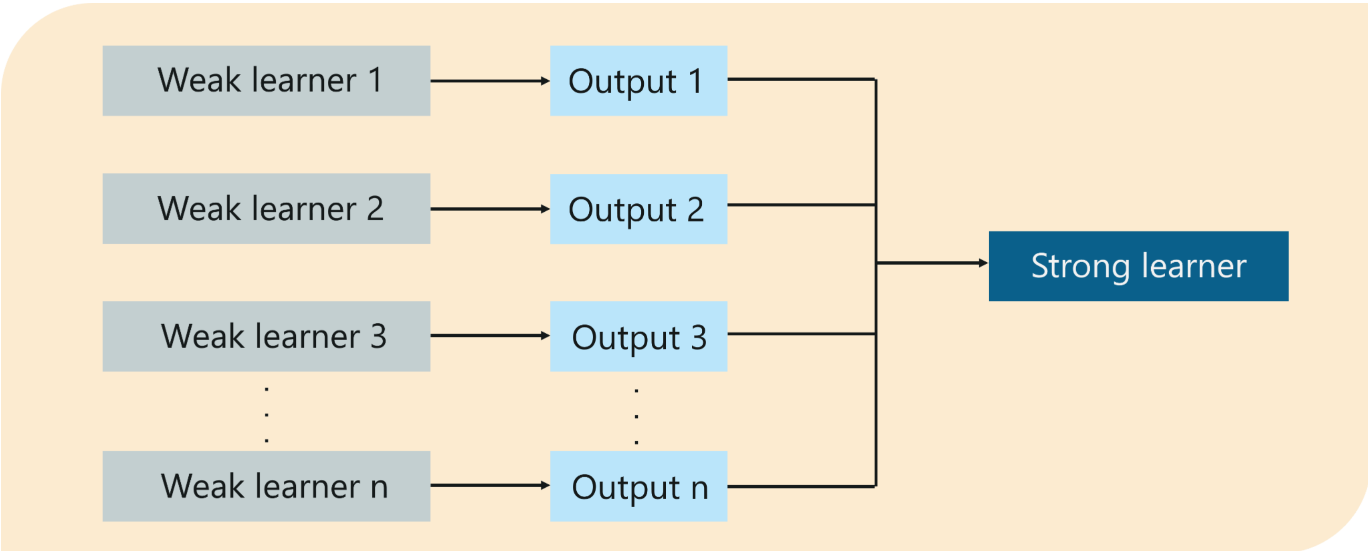
● **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and call the final node as a leaf node.

Formulas:

1. **Information Gain**= Entropy(S)- [(Weighted Avg) *Entropy (each feature)
2. **Entropy(s)**= -P(yes)log2 P(yes)- P(no) log2 P(no)
3. **Gini Index**= 1- ∑jPj2

Boosting

- Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers.
- It is done by building a model by using weak models in series.
- Firstly, a model is built from the training data.
- Then the second model is built which tries to correct the errors present in the first model.
- This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.



What Is Boosting – Boosting Machine Learning

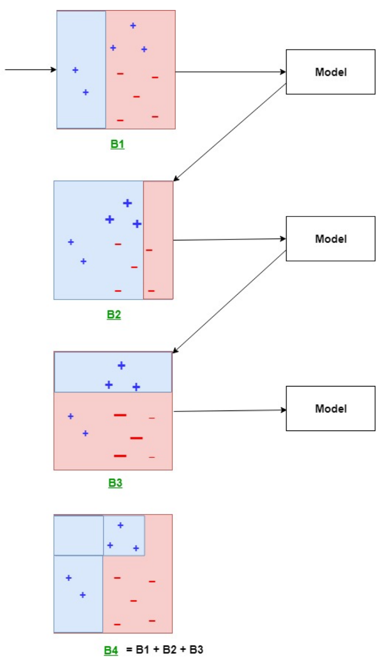
AdaBoost

- AdaBoost was the first really successful boosting algorithm developed for the purpose of binary classification.
- *AdaBoost* is short for *Adaptive Boosting* and is a very popular boosting technique that combines multiple "weak classifiers" into a single "strong classifier".

AdaBoost Algorithm

1. Initialise the dataset and assign equal weight to each of the data point.
2. Provide this as input to the model and identify the wrongly classified data points.
3. Increase the weight of the wrongly classified data points.
4. if (got required results)
     Goto step 5
   else
     Goto step 2

5. End



Explanation:

The above diagram explains the AdaBoost algorithm in a very simple way. Let's try to understand it in a stepwise process:

- **B1** consists of 10 data points which consist of two types namely plus(+) and minus(-) and 5 of which are plus(+) and the other 5 are minus(-) and each one has been assigned equal weight initially. The first model tries to classify the data points and generates a vertical separator line but it wrongly classifies 3 plus(+) as minus(-).
- **B2** consists of the 10 data points from the previous model in which the 3 wrongly classified plus(+) are weighted more so that the current model tries more to classify these pluses(+) correctly. This model generates a vertical separator line that correctly classifies the previously wrongly classified pluses(+) but in this attempt, it wrongly classifies three minuses(-).
- **B3** consists of the 10 data points from the previous model in which the 3 wrongly classified minus(-) are weighted more so that the current model tries more to classify these minuses(-) correctly. This model generates a horizontal separator line that correctly classifies the previously wrongly classified minuses(-).
- **B4** combines together B1, B2, and B3 in order to build a strong prediction model which is much better than any individual model used.

XGBoost

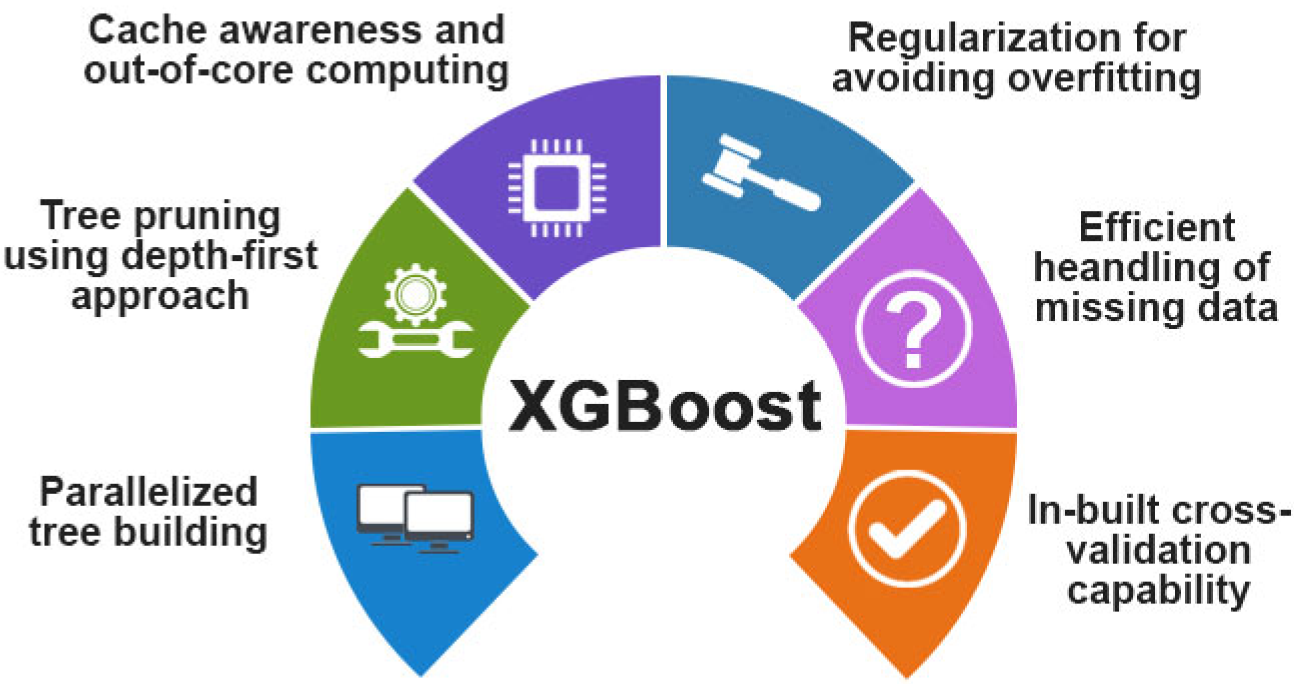
- XGBoost stands for Extreme Gradient Boosting, which was proposed by the researchers at the University of Washington.
- It is a library written in C++ which optimizes the training for Gradient Boosting.
- In this algorithm, decision trees are created in sequential form.
- Weights play an important role in XGBoost.
- Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results.
- The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree.
- These individual classifiers/predictors then ensemble to give a strong and more precise model.
- It can work on regression, classification, rank and user-defined prediction problems.

Optimization and Improvement

System Optimization:

1. **Regularization**: Since the ensembling of decisions, trees can sometimes lead to very complex. XGBoost uses both Lasso and Ridge Regression regularization to penalize the highly complex model.
2. **Parallelization and Cache block**: In XGboost, we cannot train multiple trees parallel, but it can generate the different nodes of tree parallel. For that, data needs to be sorted in order. In order to reduce the cost of sorting, it stores the data in blocks. It stored the data in the compressed column format, with each column sorted by the corresponding feature value. This switch improves algorithmic performance by offsetting any parallelization overheads in computation.
3. **Tree Pruning:** XGBoost uses max_depth parameter as specified the stopping criteria for the splitting of the branch, and starts pruning trees backward. This depth-first approach improves computational performance significantly.

4. **Cache-Awareness and Out-of-score computation**: This algorithm has been designed to make use of hardware resources efficiently. This is accomplished by cache awareness by allocating internal buffers in each thread to store gradient statistics. Further enhancements such as ‘out-of-core computing optimize available disk space while handling big data-frames that do not fit into memory. In out-of-core computation, Xgboost tries to minimize the dataset by compressing it.
5. **Sparsity Awareness**: XGBoost can handle sparse data that may be generated from preprocessing steps or missing values. It uses a special split finding algorithm that is incorporated into it that can handle different types of sparsity patterns.
6. **Weighted Quantile Sketch:** XGBoost has in-built the distributed weighted quantile sketch algorithm that makes it easier to effectively find the optimal split points among weighted datasets.
7. **Cross-validation**: XGboost implementation comes with a built-in cross-validation method. This helps the algorithm prevents overfitting when the dataset is not that big,



```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

from xgboost import XGBClassifier
```

```
df= pd.read_csv("/home/takud/Downloads/kaggledata/train_sample.csv")
```

```
print("Count of rows and column are: " , df.shape)
```

```
Count of rows and column are:  (100000, 8)
```

Dataset Columns

```
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
```

```
df.head()
```

```
       ip  app  device  os  channel           click_time
attributed_time  \
0   87540   12       1  13      497  2017-11-07 09:30:38
NaN
1  105560   25       1  17      259  2017-11-07 13:40:27
NaN
2  101424   12       1  19      212  2017-11-07 18:05:24
NaN
3   94584   13       1  13      477  2017-11-07 04:58:08
NaN
4   68413   12       1   1      178  2017-11-09 09:00:09
NaN

   is_attributed
0              0
1              0
```

```
2             0
3             0
4             0
```

Data Analysis

```
df.describe()
```

```
                  ip           app         device             os  \
count  100000.000000  100000.00000  100000.000000  100000.000000   
mean    91255.879670      12.04788      21.771250      22.818280   
std     69835.553661      14.94150     259.667767      55.943136   
min         9.000000       1.00000       0.000000       0.000000   
25%     40552.000000       3.00000       1.000000      13.000000   
50%     79827.000000      12.00000       1.000000      18.000000   
75%    118252.000000      15.00000       1.000000      19.000000   
max    364757.000000     551.00000    3867.000000     866.000000   

             channel  is_attributed  
count  100000.000000  100000.000000  
mean      268.832460       0.002270  
std       129.724248       0.047591  
min         3.000000       0.000000  
25%       145.000000       0.000000  
50%       258.000000       0.000000  
75%       379.000000       0.000000  
max       498.000000       1.000000  
```

```python
for i in df.columns:
    cnt = len(df[i].unique())
    print(i,":",cnt)
```

```
ip : 34857
app : 161
device : 100
os : 130
channel : 161
click_time : 80350
attributed_time : 228
is_attributed : 2
```

```python
col = ['ip','app','device','os','channel','is_attributed']
for i in col:
    df[i]=df[i].astype('category')
```

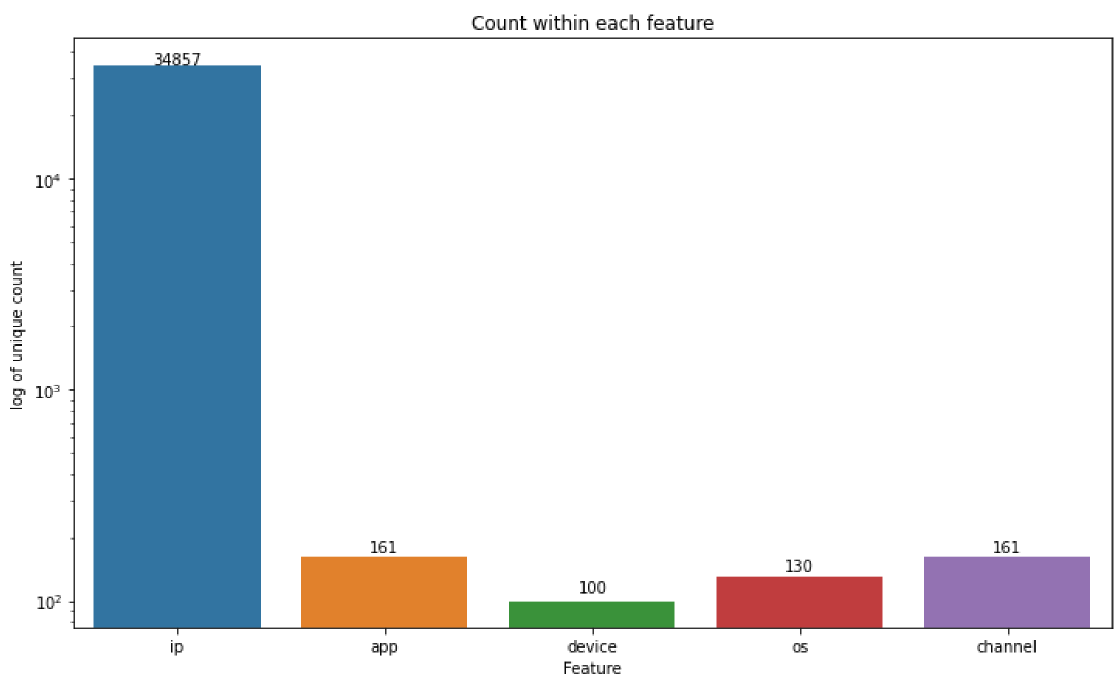
```python
df['click_time']=pd.to_datetime(df['click_time'])
df['attributed_time']=pd.to_datetime(df['attributed_time'])
```

```python
df.info()
```

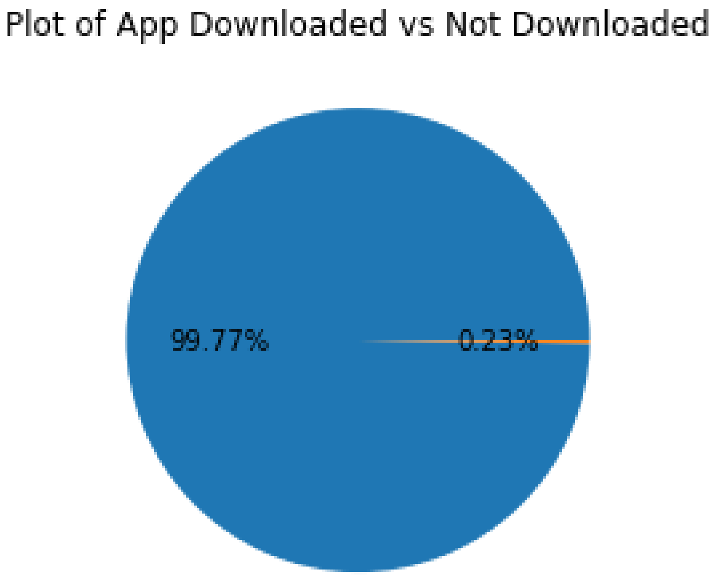
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
```

```
Data columns (total 8 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   ip               100000 non-null  category
 1   app              100000 non-null  category
 2   device           100000 non-null  category
 3   os               100000 non-null  category
 4   channel          100000 non-null  category
 5   click_time       100000 non-null  datetime64[ns]
 6   attributed_time  227 non-null     datetime64[ns]
 7   is_attributed    100000 non-null  category
dtypes: category(6), datetime64[ns](2)
memory usage: 4.0 MB
```

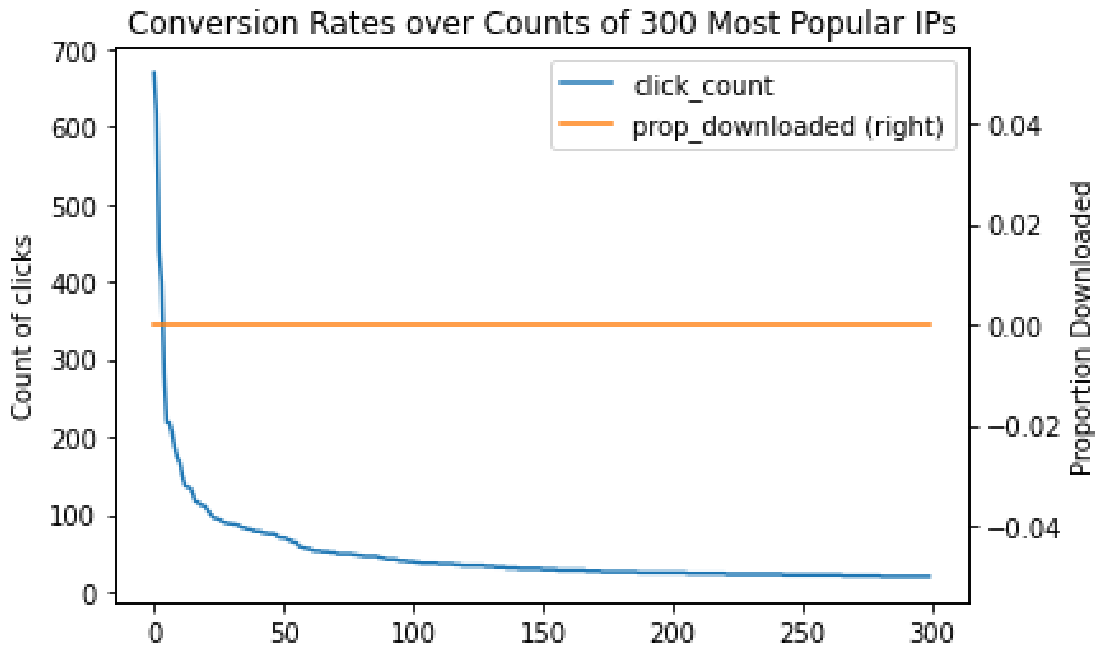
```
col = ['ip','app','device','os','channel']
cnt = [len(df[i].unique()) for i in col]

plt.figure(figsize=(12,7))
ax=sns.barplot(x=col, y=cnt, log= True)
ax.set(xlabel='Feature', ylabel='log of unique count',title="Count
within each feature")
for p, uni in zip(ax.patches, cnt):
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2., height + 10, uni, ha="center")
plt.show()
```




```
plt.pie(df['is_attributed'].value_counts(normalize=True)*100,autopct='
%1.2f%%')
plt.title("Plot of App Downloaded vs Not Downloaded")
plt.show()
```

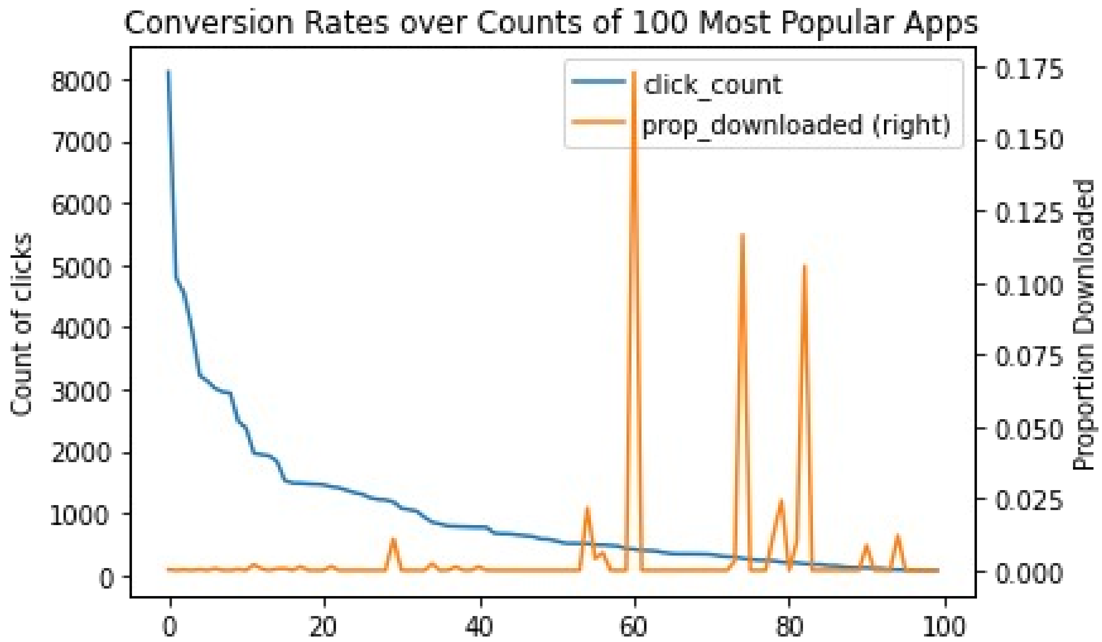



```
df['is_attributed']=df['is_attributed'].astype(int)
prop = df[['ip', 'is_attributed']].groupby('ip',
as_index=False).median().sort_values('is_attributed', ascending=False)

counts = df[['ip', 'is_attributed']].groupby('ip',
as_index=False).count().sort_values('is_attributed', ascending=False)

merge = counts.merge(prop, on='ip', how='left')
merge.columns = ['ip', 'click_count', 'prop_downloaded']

ax = merge[:300].plot(secondary_y='prop_downloaded')
plt.title('Conversion Rates over Counts of 300 Most Popular IPs')
ax.set(ylabel='Count of clicks')
plt.ylabel('Proportion Downloaded')
plt.show()
```




```
proportion = df[['channel', 'is_attributed']].groupby('channel',
as_index=False).mean().sort_values('is_attributed', ascending=False)
counts = df[['channel', 'is_attributed']].groupby('channel',
as_index=False).count().sort_values('is_attributed', ascending=False)
merge = counts.merge(proportion, on='channel', how='left')
merge.columns = ['channel', 'click_count', 'prop_downloaded']
ax = merge[:100].plot(secondary_y='prop_downloaded')
plt.title('Conversion Rates over Counts of 100 Most Popular Apps')
ax.set(ylabel='Count of clicks')
plt.ylabel('Proportion Downloaded')
plt.plot()
```

```
[]
```




```
df["datetime"]=pd.to_datetime(df["click_time"])
df["day_of_week"]=df["datetime"].dt.dayofweek
df["day_of_year"]=df["datetime"].dt.dayofyear
df["month"]=df["datetime"].dt.month
df["hour"]=df["datetime"].dt.hour

df['ip']=df['ip'].astype(int)
df['app']=df['app'].astype(int)
df['device']=df['device'].astype(int)
df['os']=df['os'].astype(int)
df['channel']=df['channel'].astype(int)

df=df.drop(["click_time","datetime","attributed_time"], axis=1)

X=df.drop("is_attributed",axis=1)
Y=df[["is_attributed"]]

x1,x2,y1,y2=train_test_split(X,Y, test_size=0.25, stratify=Y,
random_state=100)

print(y1.mean())
print(y2.mean())

is_attributed    0.002267
dtype: float64
is_attributed    0.00228
dtype: float64
```

AdaBoost

```
#Base Estimator
tree = DecisionTreeClassifier(max_depth=2)
```

```python
#Adaboost using base estimator - tree

ada_model =  AdaBoostClassifier(base_estimator=tree, n_estimators=600,
learning_rate=1.5, algorithm="SAMME")

ada_model.fit(x1,y1)

y_pred = ada_model.predict_proba(x2)

y_pred[:10]
```

```
array([[0.53026676, 0.46973324],
       [0.52230484, 0.47769516],
       [0.53207638, 0.46792362],
       [0.53510079, 0.46489921],
       [0.52804229, 0.47195771],
       [0.5253575 , 0.4746425 ],
       [0.52997573, 0.47002427],
       [0.52794369, 0.47205631],
       [0.52352081, 0.47647919],
       [0.51909813, 0.48090187]])
```

```python
ROC_Score = metrics.roc_auc_score(y2,y_pred[:,1])
print("ROC Score of AdaBoost Model: ", ROC_Score)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/home/takud/Downloads/ibm_project.ipynb Cell 23' in <cell line: 1>()
----> <a href='vscode-notebook-cell:/home/takud/Downloads/ibm_project.ipynb#ch0000023?line=0'>1</a> ROC_Score = metrics.roc_auc_score(y2,y_pred[:,1])
      <a href='vscode-notebook-cell:/home/takud/Downloads/ibm_project.ipynb#ch0000023?line=1'>2</a> print("ROC Score of AdaBoost Model: ", ROC_Score)

NameError: name 'y2' is not defined
```

```python
parameter = {"base_estimator__max_depth":[2,3], "n_estimators":
[100,300,500]}

tree= DecisionTreeClassifier()

adaboostmodel = AdaBoostClassifier(base_estimator=tree,
learning_rate=0.9, algorithm="SAMME")

fold=3
```

```
grid_search_cv = GridSearchCV(adaboostmodel, cv=fold,
param_grid=parameter, scoring='roc_auc', return_train_score=True,
verbose=1)
grid_search_cv.fit(x1,y1)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits

GridSearchCV(cv=3,
             estimator=AdaBoostClassifier(algorithm='SAMME',

base_estimator=DecisionTreeClassifier(),
                                          learning_rate=0.9),
             param_grid={'base_estimator__max_depth': [2, 3],
                         'n_estimators': [100, 300, 500]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

```
ada_cv_result = pd.DataFrame(grid_search_cv.cv_results_)
ada_cv_result
```

```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       3.763095      0.098628         0.123628        0.011940
1      10.519879      0.090545         0.295343        0.015654
2      17.795266      0.111225         0.523209        0.014498
3       5.017527      0.128979         0.118456        0.006820
4      15.104142      0.230655         0.309712        0.035777
5      26.057564      0.219347         0.528709        0.023174

  param_base_estimator__max_depth param_n_estimators  \
0                               2                100
1                               2                300
2                               2                500
3                               3                100
4                               3                300
5                               3                500

                                              params
split0_test_score  \
0  {'base_estimator__max_depth': 2, 'n_estimators...
0.973905
1  {'base_estimator__max_depth': 2, 'n_estimators...
0.975754
2  {'base_estimator__max_depth': 2, 'n_estimators...
0.977250
3  {'base_estimator__max_depth': 3, 'n_estimators...
0.964230
4  {'base_estimator__max_depth': 3, 'n_estimators...
0.968313
5  {'base_estimator__max_depth': 3, 'n_estimators...
0.969704

   split1_test_score  split2_test_score  mean_test_score
```

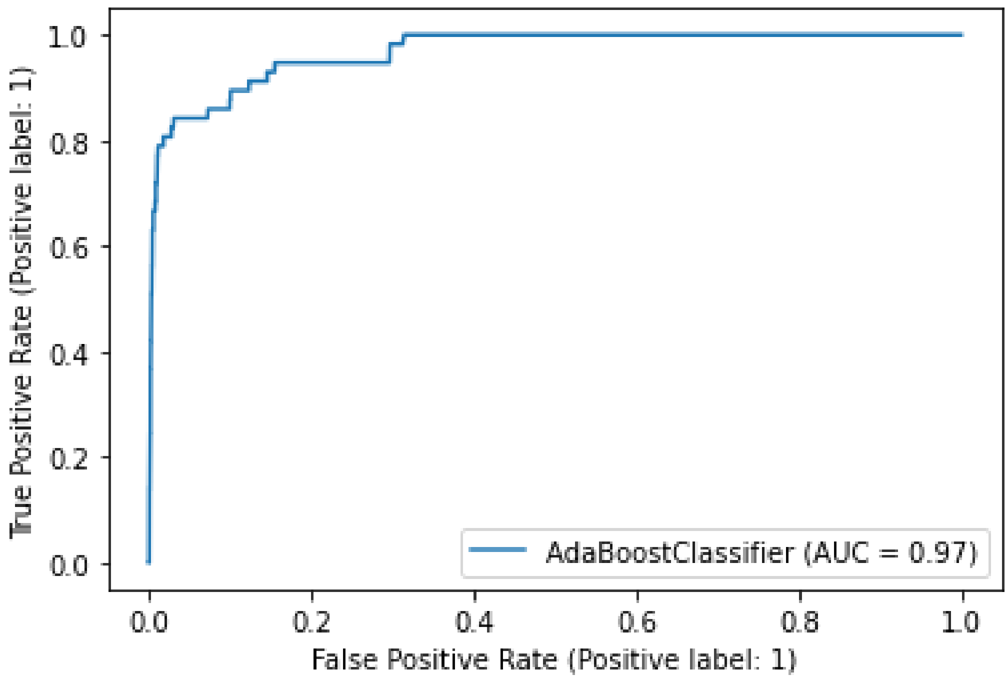
```
std_test_score  \
0          0.926124          0.945589         0.948539
0.019618
1          0.933316          0.941703         0.950258
0.018351
2          0.926629          0.938247         0.947375
0.021651
3          0.918649          0.953138         0.945339
0.019408
4          0.923209          0.951110         0.947544
0.018585
5          0.927578          0.952118         0.949800
0.017276

   rank_test_score  split0_train_score  split1_train_score  \
0                3            0.995584            0.997201
1                1            0.998416            0.998585
2                5            0.998875            0.999205
3                6            0.999193            0.999739
4                4            0.999951            0.999998
5                2            0.999997            1.000000

   split2_train_score  mean_train_score  std_train_score
0            0.995843          0.996209         0.000709
1            0.997709          0.998237         0.000379
2            0.998511          0.998863         0.000283
3            0.998955          0.999296         0.000328
4            0.999919          0.999956         0.000032
5            0.999999          0.999999         0.000001
```

```
tree = DecisionTreeClassifier(max_depth=2)

ada_model1 =
AdaBoostClassifier(base_estimator=tree,learning_rate=0.5,n_estimators=
100,algorithm="SAMME")

ada_model1.fit(x1,y1)
y_pred1 = ada_model1.predict_proba(x2)

ROC_Score=metrics.roc_auc_score(y2,y_pred1[:,1])
print("ROC Score of Hyperparameter Tuned AdaBoost Model: ", ROC_Score)
```

```
/home/takud/.local/lib/python3.8/site-packages/sklearn/utils/
validation.py:1111: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

ROC Score of Hyperparameter Tuned AdaBoost Model:  0.968179027129223
```

```
metrics.plot_roc_curve(ada_model1,x2,y2)
plt.show()
```

```
/home/takud/.local/lib/python3.8/site-packages/sklearn/utils/
deprecation.py:87: FutureWarning: Function plot_roc_curve is
deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and
will be removed in 1.2. Use one of the class
methods: :meth:`sklearn.metric.RocCurveDisplay.from_predictions`
or :meth:`sklearn.metric.RocCurveDisplay.from_estimator`.
  warnings.warn(msg, category=FutureWarning)
```



XGBoost

```
XGB_model = XGBClassifier()
XGB_model.fit(x1,y1)

y_pred3 =XGB_model.predict_proba(x2)
y_pred3[:10]
```

```
array([[9.9998963e-01, 1.0348637e-05],
       [9.9998719e-01, 1.2833507e-05],
       [9.9999648e-01, 3.5333121e-06],
       [9.9999213e-01, 7.8918056e-06],
       [9.9997818e-01, 2.1817492e-05],
       [9.9994433e-01, 5.5651140e-05],
       [9.9994344e-01, 5.6572928e-05],
       [9.9997544e-01, 2.4584380e-05],
       [9.9996036e-01, 3.9621922e-05],
       [9.9991095e-01, 8.9042456e-05]], dtype=float32)
```

```
ROC_Score=metrics.roc_auc_score(y2,y_pred3[:,1])
print("ROC Score of XGBoost Model :%.2f%%" % (ROC_Score * 100.0) )
```

```
ROC Score of XGBoost Model :93.64%
```

```
fold = 3

parameter = {"learning_rate":[0.1,0.3,0.5], "subsample":[0.3,0.6,0.8],
"n_estimators":[100,200,300,500], "max_depth":[2,3,4]}

xgb_model = XGBClassifier()

grid_xgb_model = GridSearchCV(xgb_model, param_grid=parameter,
cv=fold, scoring="roc_auc",return_train_score=True, verbose=0)

grid_xgb_model.fit(x1,y1)
```

```
GridSearchCV(cv=3,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None,
colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False,
eval_metric=None,
                                     gamma=None, gpu_id=None,
grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None, max_bin=None,
                                     max_ca...
                                     max_leaves=None,
min_child_weight=None,
                                     missing=nan,
monotone_constraints=None,
                                     n_estimators=100, n_jobs=None,
                                     num_parallel_tree=None,
predictor=None,
                                     random_state=None,
reg_alpha=None,
                                     reg_lambda=None, ...),
             param_grid={'learning_rate': [0.1, 0.3, 0.5],
                         'max_depth': [2, 3, 4],
                         'n_estimators': [100, 200, 300, 500],
                         'subsample': [0.3, 0.6, 0.8]},
             return_train_score=True, scoring='roc_auc')
```

```
cv_results = pd.DataFrame(grid_xgb_model.cv_results_)
cv_results
```

```
     mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0         0.802222      0.127896         0.012493        0.003198   
1         0.632792      0.030590         0.009509        0.000155   
2         0.611006      0.107436         0.015473        0.005018   
3         0.961475      0.171173         0.013131        0.001418   
4         1.027110      0.019996         0.012963        0.001123   
..             ...           ...              ...             ...   
103       2.884965      0.823878         0.021311        0.001291   
104       2.343053      0.056738         0.021106        0.000192   
105       2.825728      0.144548         0.031081        0.004350   
106       3.560662      0.052154         0.029588        0.002571   
107       3.592764      0.081641         0.028153        0.001325   

    param_learning_rate param_max_depth param_n_estimators
param_subsample  \
0                   0.1               2                100
0.3   
1                   0.1               2                100
0.6   
2                   0.1               2                100
0.8   
3                   0.1               2                200
0.3   
4                   0.1               2                200
0.6   
..                  ...             ...                ...
...   
103                 0.5               4                300
0.6   
104                 0.5               4                300
0.8   
105                 0.5               4                500
0.3   
106                 0.5               4                500
0.6   
107                 0.5               4                500
0.8   

                                                params
split0_test_score  \
0    {'learning_rate': 0.1, 'max_depth': 2, 'n_esti...
0.975522   
1    {'learning_rate': 0.1, 'max_depth': 2, 'n_esti...
0.968320   
2    {'learning_rate': 0.1, 'max_depth': 2, 'n_esti...
0.972795   
3    {'learning_rate': 0.1, 'max_depth': 2, 'n_esti...
0.968942   
4    {'learning_rate': 0.1, 'max_depth': 2, 'n_esti...
0.972135   
```

```
..                                                  ...
...
103  {'learning_rate': 0.5, 'max_depth': 4, 'n_esti...
0.959345
104  {'learning_rate': 0.5, 'max_depth': 4, 'n_esti...
0.967138
105  {'learning_rate': 0.5, 'max_depth': 4, 'n_esti...
0.961503
106  {'learning_rate': 0.5, 'max_depth': 4, 'n_esti...
0.958963
107  {'learning_rate': 0.5, 'max_depth': 4, 'n_esti...
0.965995

     split1_test_score  split2_test_score  mean_test_score
std_test_score  \
0             0.921039           0.949011         0.948524
0.022245
1             0.922168           0.950990         0.947159
0.019035
2             0.921168           0.950408         0.948123
0.021139
3             0.927231           0.953933         0.950035
0.017250
4             0.916268           0.960561         0.949655
0.024076
..                 ...                ...              ...
...
103           0.937452           0.944139         0.946979
0.009161
104           0.914188           0.940874         0.940733
0.021617
105           0.914090           0.927469         0.934354
0.019959
106           0.935620           0.941824         0.945469
0.009872
107           0.916203           0.940283         0.940827
0.020331

     rank_test_score  split0_train_score  split1_train_score  \
0                 68            0.951795            0.969239
1                 78            0.959006            0.969987
2                 71            0.957380            0.970163
3                 52            0.968351            0.981386
4                 55            0.968706            0.983028
..               ...                 ...                 ...
103               81            1.000000            1.000000
104              100            1.000000            1.000000
105              105            1.000000            0.999999
106               85            1.000000            1.000000
107               99            1.000000            1.000000
```

```
     split2_train_score  mean_train_score  std_train_score
0              0.961922          0.960985     7.152306e-03
1              0.960527          0.963173     4.857809e-03
2              0.957980          0.961841     5.889471e-03
3              0.974526          0.974754     5.324220e-03
4              0.973275          0.975003     5.973174e-03
..                  ...               ...              ...
103            1.000000          1.000000     0.000000e+00
104            1.000000          1.000000     0.000000e+00
105            1.000000          1.000000     2.508703e-07
106            1.000000          1.000000     0.000000e+00
107            1.000000          1.000000     0.000000e+00

[108 rows x 20 columns]
```

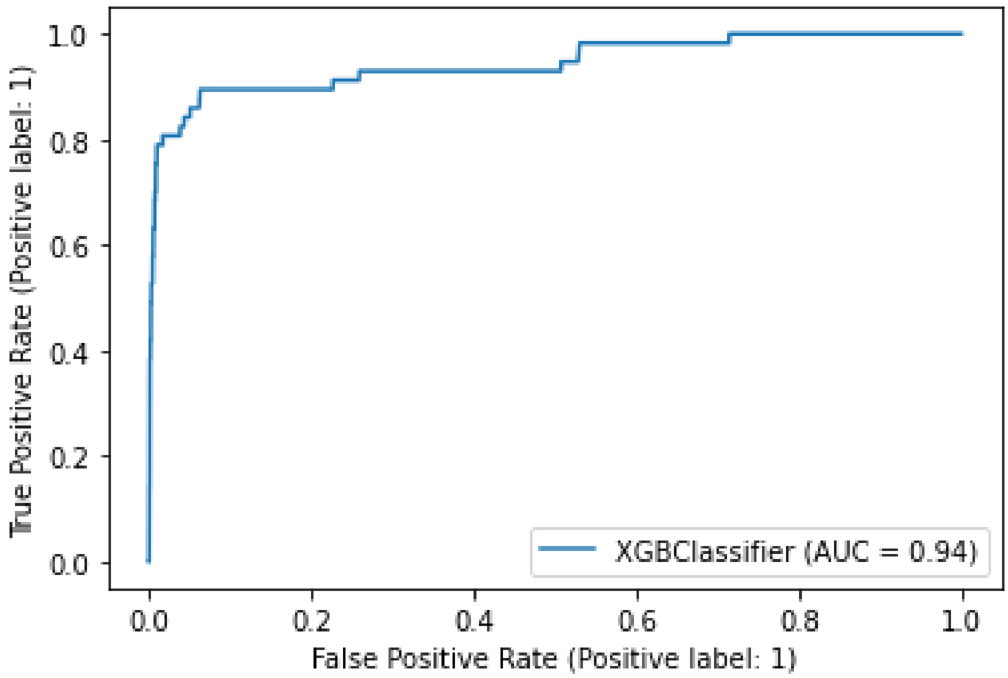
```
XGBC_model = XGBClassifier(max_depth=2, n_estimators=100,
learning_rate=0.1, subsample=0.6)
XGBC_model.fit(x1,y1)
y_pred4=XGBC_model.predict_proba(x2)
y_pred4[:10]
```

```
array([[9.9377602e-01, 6.2239817e-03],
       [9.9898797e-01, 1.0120556e-03],
       [9.9981016e-01, 1.8983467e-04],
       [9.9977374e-01, 2.2623497e-04],
       [9.9911630e-01, 8.8369526e-04],
       [9.9750990e-01, 2.4900995e-03],
       [9.9754852e-01, 2.4514508e-03],
       [9.9972457e-01, 2.7542457e-04],
       [9.9968284e-01, 3.1715786e-04],
       [9.9876481e-01, 1.2352125e-03]], dtype=float32)
```

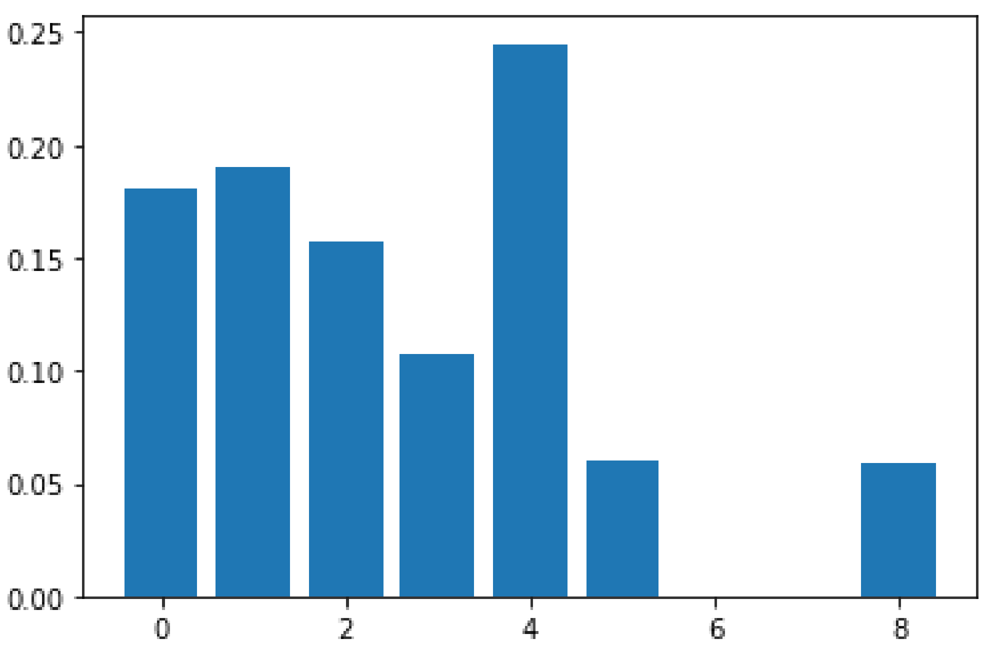
```
ROC_Score=metrics.roc_auc_score(y2,y_pred4[:,1])
print("ROC Score of Hyperparameter Tunned XGBoost Model :%.2f%%" %
(ROC_Score * 100.0) )
```

```
ROC Score of Hyperparameter Tunned XGBoost Model :94.44%
```

```
metrics.plot_roc_curve(XGBC_model,x2,y2)
plt.show()
```




```
plt.bar(range(len(XGBC_model.feature_importances_)),
XGBC_model.feature_importances_)
plt.show()
```




```
# feature importance
importance = dict(zip(x1.columns, XGBC_model.feature_importances_))
importance
```

```
{'ip': 0.18024725,
 'app': 0.19025756,
 'device': 0.15740265,
 'os': 0.10750766,
 'channel': 0.24497178,
 'day_of_week': 0.060586657,
 'day_of_year': 0.0,
 'month': 0.0,
 'hour': 0.05902649}
```

```
test = pd.read_csv("/home/takud/Downloads/kaggledata/test.csv")
print("Count of rows and column are: " , test.shape)
```

```
Count of rows and column are:  (18790469, 7)
```

```
test["datetime"]=pd.to_datetime(test["click_time"])
test["day_of_week"]=test["datetime"].dt.dayofweek
test["day_of_year"]=test["datetime"].dt.dayofyear
test["month"]=test["datetime"].dt.month
test["hour"]=test["datetime"].dt.hour
test['ip']=test['ip'].astype(int)
test['app']=test['app'].astype(int)
test['device']=test['device'].astype(int)
test['os']=test['os'].astype(int)
test['channel']=test['channel'].astype(int)

test_df=test.drop(["click_time","datetime","click_id"], axis=1)

test_df.head()
```

```
       ip  app  device  os  channel  day_of_week  day_of_year  month
hour
0    5744    9       1   3      107            4          314     11
4
1  119901    9       1   3      466            4          314     11
4
2   72287   21       1  19      128            4          314     11
4
3   78477   15       1  13      111            4          314     11
4
4  123080   12       1  13      328            4          314     11
4
```

```
final_y_ada= XGBC_model.predict_proba(test_df)
sub1 = pd.DataFrame()
sub1['click_id'] = test['click_id']
sub1['is_attributed'] = final_y_ada[:, 1]
sub1.head()
```

```
   click_id  is_attributed
0         0       0.000878
```

1	1	0.000447
2	2	0.001518
3	3	0.001273
4	4	0.000320

Conclusion

Therefore we have successfully predicted a potential click fraud by using decision tree algorithm and two boosting algorithms namely Adaboost and xgboost algorithm out of which Adaboost algorithm performs better than xgboost. We can implement this project to avoid and be secure from potential click fraud.

Further development or research

Since machine learning is a very popular field among academicians as well as industry experts, there is a huge scope of innovation. Experimentation with different algorithms and models can help your business in detecting fraud. Machine learning techniques are obviously more reliable than human review and transaction rules. The machine learning solutions are efficient, scalable and process a large number of transactions in real time. But extracting data and training data sets for correct prediction is a tough task.

REFERENCES

[1] Pradheepan Raghavan and Neamat El Gayar, “Fraud Detection using Machine Learning and Deep Learning”, Conference: 2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)

[2] Riwa Mouawi, Mariette Awad, Ali Chehab and Imad H. El Hajj, “Towards a Machine Learning

Approach for Detecting Click Fraud in Mobile Advertising”, Conference: 2018 International Conference on Innovations in Information Technology (IIT)

[3] “Application of Machine Learning on Fraud App Detection” 1 Syed Abdul Moeed, Assistant Professor, Department of CSE, KITS-Warangal, TS, India 2G.Ashmitha, Assistant Professor, Department of CSE, KITS-Warangal, TS, India 3Dr. P. Niranjan, Professor, Department of CSE, KITS-Warangal, TS, India

[4] Follow the Trail: “Machine Learning for Fraud Detection in Fintech Applications” Branka Stojanovi´c 1,*, Josip Boži´c 1 , Katharina Hofer-Schmitz 1 , Kai Nahrgang 1 , Andreas Weber 2 , Atta Badii 3 , Maheshkumar Sundaram 3 , Elliot Jordan 3 and Joel Runevic 3

[5] “Mobile Money Fraud Prediction”—A Cross-Case Analysis on the Efficiency of Support Vector Machines, Gradient Boosted Decision Trees, and Naïve Bayes Algorithms by Francis Effirim Botchey 1,2,Zhen Qin 1,* andKwesi Hughes-Lartey

[6] “Ranking fraud detection for mobile apps: a holistic view” by Hengshu Zhu, Hui Xiong, Yong Ge and Enhong Chen

[7] “Detecting Fraud Apps using Sentiment Research” Mandava Rama Rao, Nandhini Kannan, CH V S Nihanth

[8] “An Enhanced Secure Deep Learning Algorithm for Fraud Detection in Wireless Communication” by Sumaya Sanober, Izhar Alam, Sagar Pande, Farrukh Arslan, Kantilal Pitambar Rane, Bhupesh Kumar Singh, Aditya Khamparia and Mohammad Shabaz