IE4092
Machine learning for cyber security
4th Year, 1st Semester

Final Examination
Type 3 – Assignment type exam

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

06.10.2020

Declaration

I certify that this report does not incorporate without acknowledgement, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief it does not contain any material previously published or written by another person, except where due reference is made in text.

Registration Number : IT17013642

Name : S.N Wijesinghe

Contact no: 071 2263399

TalkingData AdTracking Fraud Detection Challenge

fraud risk is all over the place, however for organizations that promote on the web, click fraud can occur at a staggering volume, bringing about deceiving click information and wasted money. Advertisement channels can drive up costs by just tapping on the advertisement at a huge scope. With more than 1 billion smart mobile phones in dynamic utilize each month, China is the biggest portable market on the planet and accordingly experiences enormous volumes of fraudulent traffic.

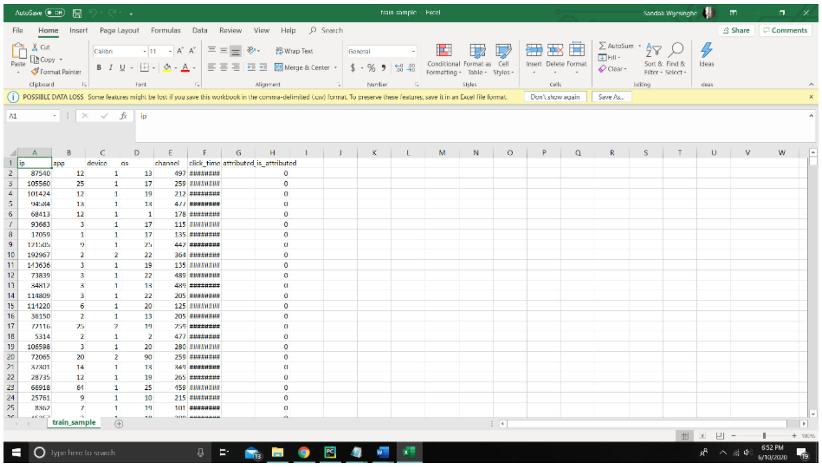
TalkingData, China's largest independent big data service platform, covers over 70% of active mobile devices nationwide. They handle 3 billion clicks for each day, of which 90% are possibly fraudulent. Their current approach to prevent click fraud for app developers is to measure the journey of a user's click across their portfolio, and flag IP addresses who produce lots of clicks, but never end up installing apps. With this data, they've manufactured an IP blacklist and device blacklist

About dataset

Worked on Kaggle talking data adtracking challenge dataset.which contains the below coloumns:

- 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

The features(ip,app, os and channel)are encoded.



POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

A1 ip

	A	B	C	D	E	F	G	H
1	ip	app	device	os	channel	click_time	attributed_	is_attributed
2	87540	12	1	13	497	########		0
3	105560	25	1	17	259	########		0
4	101424	12	1	19	212	########		0
5	94584	13	1	13	477	########		0
6	68413	12	1	1	178	########		0
7	93663	3	1	17	115	########		0
8	17059	1	1	17	135	########		0
9	121505	9	1	25	442	########		0
10	192967	2	2	22	364	########		0
11	140636	3	1	19	135	########		0
12	73839	3	1	22	489	########		0
13	34812	3	1	13	489	########		0
14	114809	3	1	22	205	########		0
15	114220	6	1	20	125	########		0
16	36150	2	1	13	205	########		0
17	72116	25	2	19	259	########		0
18	5314	2	1	2	477	########		0
19	106598	3	1	20	280	########		0
20	72065	20	2	90	259	########		0
21	37301	14	1	13	349	########		0
22	28735	12	1	19	265	########		0
23	66918	64	1	25	459	########		0
24	25761	9	1	10	215	########		0
25	8362	7	1	19	101	########		0

Machine Learning-Based Approaches

Density-Based Anomaly Detection

Density-based outlier detection method investigates the density of an object and that of its neighbors. Here, an object is identified as an outlier if its density is relatively much lower than that of its neighbors. The nearest set of data points are evaluated using a score, which could be Euclidian distance or a similar measure dependent on the type of the data (categorical or numerical). They could be broadly classified into two algorithms:

K-nearest neighbor: k-NN is a simple, non-parametric lazy learning technique used to classify data based on similarities in distance metrics such as Euclidian, Manhattan, Minkowski, or Hamming distance.

Relative density of data: This is better known as local outlier factor (LOF). This concept is based on a distance metric called reachability distance.

Clustering-Based Anomaly Detection

Clustering is one of the most popular concepts in the domain of unsupervised learning.

Assumption: Data points that are similar tend to belong to similar groups or clusters, as determined by their distance from local centroids.

K-means is a widely used clustering algorithm. It creates 'k' similar clusters of data points. Data instances that fall outside of these groups could potentially be marked as anomalies.

Support Vector Machine-Based Anomaly Detection

- A support vector machine is an effective technique for detecting anomalies which is usually associated with supervised learning, but sometimes unsupervised problems can also be detected (OneClassCVM).
- The algorithm learns a soft boundary in order to cluster the normal data instances using the training set, and then, using the testing instance, it tunes itself to identify the abnormalities that fall outside the learned region.
- Depending on the use case, the output of an anomaly detector could be numeric scalar values for filtering on domain-specific thresholds or textual labels (such as binary/multi labels).

In PyCharm we are going to take the talking data adtracking fraud detection as the case study for understanding this concept in detail using the following Anomaly Detection Techniques name

- **Isolation Forest Anomaly Detection Algorithm.**
- **Density-Based Anomaly Detection (Local Outlier Factor) Algorithm.**

Methodology

- First, we obtained our dataset from Kaggle, a data analysis website which provides datasets.

- Platform used in this project was PyCharm.

- Importing of the necessary libraries with the packages

```
import pickle
import sys
import numpy
import matplotlib
import pandas
import scipy
import seaborn
import sklearn
```

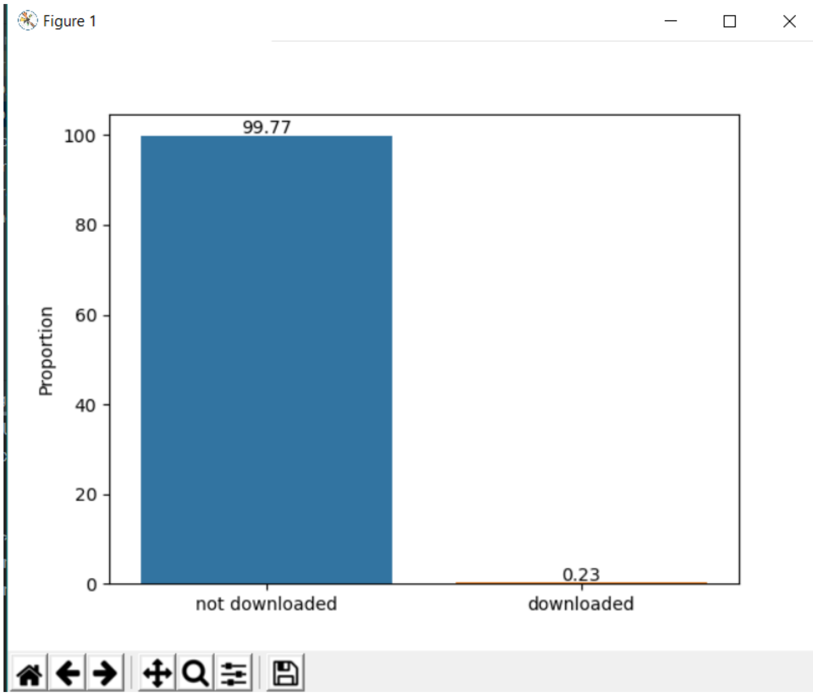
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

- loading the dataset and print the first five rows of the dataframe

```
data = pd.read_csv('train_sample.csv')
data.head()
```

- check the distribution of data based on class labels i.e. what percent of points belong to class 0 and class 1.

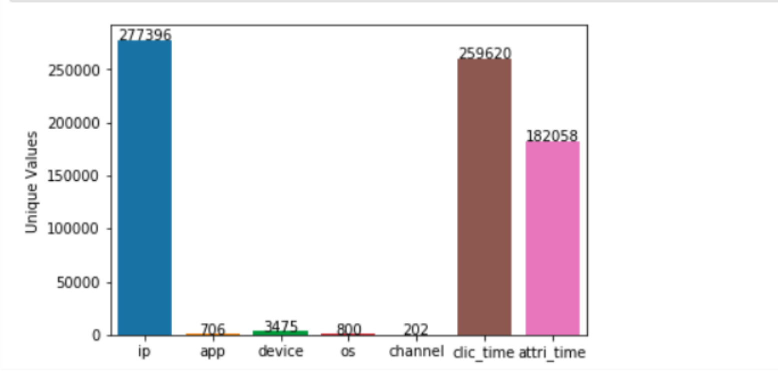
```
plot = sns.barplot(['not downloaded','downloaded'],[percentage*100,(1-
percentage)*100])
# display the proponataly of the data through bar graph
plot.set(ylabel ='Proportion')
for i in range(2):
    a = plot.patches[i]
    height = a.get_height()
    value = abs(percentage-i)
    plot.text(a.get_x()+a.get_width()/2., height+0.5, round(value*100, 2), ha="center")
plt.show()
```



We can see that the dataset is highly imbalance. It is one of the common problem in machine learning.

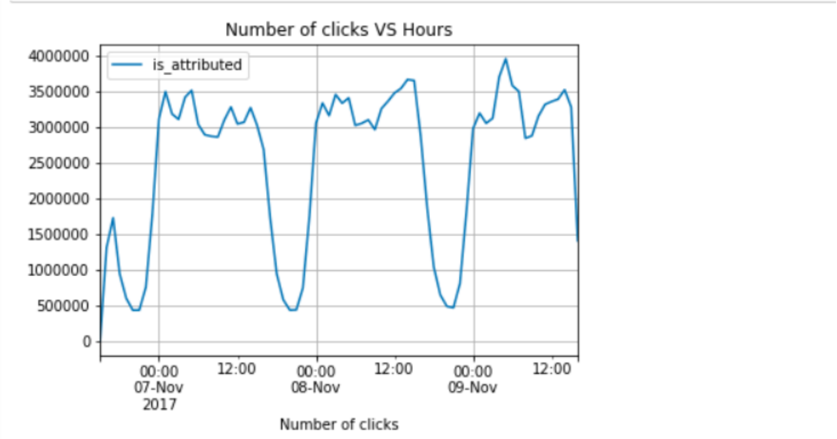
```
unique_values = []

for x in df.columns:
    unique_values.append(len(df[x].unique()))
    unique_values.pop()

plot = sns.barplot(['ip', 'app', 'device', 'os', 'channel', 'clic_time', 'attri_time'], unique_values)
plot.set(ylabel='Unique Values')

for i in range(len(unique_values)):
    a = plot.patches[i]
    height = a.get_height()
    value = abs(percentage - i)
    plot.text(a.get_x()+a.get_width()/2., height+1, unique_values[i], ha="center")
plt.show()
```

- This displays the unique values for features.



- We should also check the distribution of click every hour. It will tell us, at what time of the day the clicks are higher and what time of day the clicks are lower. With this information we can create time based features like time interval

```
#plot to show no of clicks vs hours
df['roundoff_click']=df['click_time'].dt.round('H')
df[('roundoff_click','is_attributed')].groupby(['roundoff_click'], as_index=True).count().plot
plt.xlabel('number of clicks')
plt.title('number of clicks vs hours')
plt.grid()
```



- No of fraud cases and valid cases were identified. An outlier fraction was used to calculate the fraudulent cases to the valid cases.

```
Fraud = data[data['is_attributed'] == 1]
Valid = data[data['is_attributed'] == 0]
outlier_fraction = len(Fraud) / len(Valid)
print(outlier_fraction)
print('Fraud cases:{}'.format(len(Fraud)))
print('valid cases:{}'.format(len(Valid)))
```

- getting all the coloumns from the dataset and filter the coloumns to remove the data we do not need

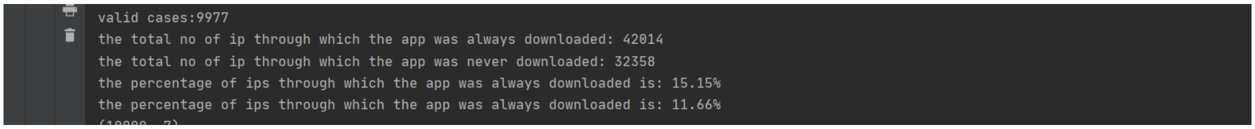
```
#get all the coloumns from the dataframe
columns = data.columns.tolist()

#filter the coloumns to remove the data we do not need
columns = [c for c in columns if c not in ["is_attributed"]]
```

- store the variable we’ll be predicting on

```
target = "is_attributed"
```

- check if there are ip’s which always download the app after clicking and ip’s which never download the app after clicking. This can be very useful information.

```
ip_1= data.ip[data.is_attributed == 1]
set_of_ip_1= set(ip_1.unique())
ip_0= data.ip[data.is_attributed == 0]
set_of_ip_0= set(ip_0.unique())

ip_download = set_of_ip_1 - set_of_ip_0

ip_fraudulent =set_of_ip_0 -set_of_ip_1
print('the total no of ip through which the app was always downloaded: ', len(ip_download))
print('the total no of ip through which the app was never downloaded: ',len(ip_fraudulent))
print('the percentage of ips through which the app was always downloaded is: ',round
((len(ip_download)/277396)*100,2),"%")
print('the percentage of ips through which the app was always downloaded is: ',round
((len(ip_download)/277396)*100,2),"%")
```



```
valid cases:9977
the total no of ip through which the app was always downloaded: 42014
the total no of ip through which the app was never downloaded: 32358
the percentage of ips through which the app was always downloaded is: 15.15%
the percentage of ips through which the app was always downloaded is: 11.66%
(10000, 7)
```

Applying the algorithm to the project

The data set has been preprocessed and is ready to be trained. Imported several packages using sklearn for isolation forest and local outlier factor.

Built two anomaly detection models :- Isolation Forest and Local Outlier Factor. . A comparison is made between 2 models.

Isolation forest: Isolation forest is a machine learning algorithm for anomaly detection. It's an unsupervised learning algorithm that identifies anomaly by isolating outliers in the data.

Local outlier factor: The Local Outlier Factor (LOF) algorithm is an unsupervised anomaly detection method which computes the local density deviation of a given data point with respect to its neighbors. It considers as outliers the samples that have a substantially lower density than their neighbors.

The two algorithms are put into a dictionary of classifiers

```
classifiers = {
    "Isolation Forest": IsolationForest(max_samples=len(X),
                                        contamination=outlier_fraction,
                                        random_state=state),
    "Local Outlier Factor": LocalOutlierFactor(
        n_neighbors=20,
        contamination=outlier_fraction)
}
```

Storing the model

- pickle was used to store the model. Pickle is used for serializing and deserializing python object structures.

  ```
  •pickle.dump(clf, open('model.pkl', 'wb'))
  model = pickle.load(open('model.pkl', 'rb'))
  ```

- need to use ‵wb′ ('b' for binary) during file writing and ‵rb′ during file opening.