Report

Final Project

Hadeer Mamoduh Abdelfattah Mohammed
Nada Abdellatef Shaker Seddik
Mostafa Mahmoud Abdelwahab Nofal
GRUOP | 11

Objective

We want to use machine learning models to identify fake tasks. Then we implement the GAN neural network. It's divided into generator to generate fake tasks and the discriminator to identify whether they are fake or real tasks.

Import the libraries

```
[1] !pip install plotly
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (5.5.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from plotly) (1.15.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-packages (from plotly) (8.0.1)
```

```
[2]
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report, ConfusionMatrixDisplay, accuracy_score, confusion_matrix
from sklearn.ensemble import VotingClassifier
from imblearn.over_sampling import RandomOverSampler

import plotly.express as express

from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential, Model
from keras.layers import Reshape
from keras.layers import Dense
from keras.layers import LeakyReLU
from keras.layers.normalization.batch_normalization import BatchNormalization
from keras.layers import Input, Flatten
```

1. Read the dataset

```
[38] data = pd.read_csv("MCSDatasetNEXTCONLab.csv")
```

```
[39] data
```

	ID	Latitude	Longitude	Day	Hour	Minute	Duration	RemainingTime	Resources	Coverage	OnPeakHours	GridNumber	Ligitimacy
0	1	45.442142	-75.303369	1	4	13	40	40	9	91	0	131380	1
1	1	45.442154	-75.304366	1	4	23	40	30	9	91	0	131380	1
2	1	45.442104	-75.303963	1	4	33	40	20	9	91	0	121996	1
3	1	45.441868	-75.303577	1	4	43	40	10	9	91	0	121996	1
4	2	45.447727	-75.147722	2	15	49	30	30	5	47	0	140784	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
14479	3999	45.445303	-75.165596	2	1	18	20	20	10	80	0	131397	1
14480	3999	45.445574	-75.165168	2	1	28	20	10	10	80	0	131397	1
14481	4000	45.436682	-75.152416	0	12	21	30	30	4	63	0	122015	1
14482	4000	45.436978	-75.153278	0	12	31	30	20	4	63	0	122015	1
14483	4000	45.436983	-75.153240	0	12	41	30	10	4	63	0	122015	1

14484 rows × 13 columns

Splitting the dataset to features and label

Features

```
[40] x = data.iloc[:,:-1]
     y = data.iloc[:,-1:]
```

```
[41] x
```

	ID	Latitude	Longitude	Day	Hour	Minute	Duration	RemainingTime	Resources	Coverage	OnPeakHours	GridNumber
0	1	45.442142	-75.303369	1	4	13	40	40	9	91	0	131380
1	1	45.442154	-75.304366	1	4	23	40	30	9	91	0	131380
2	1	45.442104	-75.303963	1	4	33	40	20	9	91	0	121996
3	1	45.441868	-75.303577	1	4	43	40	10	9	91	0	121996
4	2	45.447727	-75.147722	2	15	49	30	30	5	47	0	140784
...	...	...	...	...	...	...	...	...	...	...	...	...
14479	3999	45.445303	-75.165596	2	1	18	20	20	10	80	0	131397
14480	3999	45.445574	-75.165168	2	1	28	20	10	10	80	0	131397
14481	4000	45.436682	-75.152416	0	12	21	30	30	4	63	0	122015
14482	4000	45.436978	-75.153278	0	12	31	30	20	4	63	0	122015
14483	4000	45.436983	-75.153240	0	12	41	30	10	4	63	0	122015

14484 rows × 12 columns

Label

```
[42] y
```

	Ligitimacy
0	1
1	1
2	1
3	1
4	1
...	...
14479	1
14480	1
14481	1
14482	1
14483	1

14484 rows × 1 columns

2. Splitting the dataset to train and test

```
[38] 1 x_train, x_test, y_train, y_test = train_test_split(x, y, train_size = 0.8)
```

```
[39] 1 x_train
```

	ID	Latitude	Longitude	Day	Hour	Minute	Duration	RemainingTime	Resources	Coverage	OnPeakHours	GridNumber
7914	2206	45.504761	-75.214953	2	23	44	50	20	1	79	0	234615
1225	334	45.518957	-75.229820	0	21	35	60	20	7	43	0	262765
10484	2909	45.520886	-75.215634	0	22	53	10	10	5	85	0	262767
7385	2066	45.402016	-75.272772	5	4	48	50	50	1	76	0	56312
959	262	45.409482	-75.210547	2	22	17	30	10	7	72	0	75088
...	...	...	...	...	...	...	...	...	...	...	...	...
4647	1276	45.535918	-75.173689	3	22	14	20	20	4	100	0	290924
3913	1068	45.387016	-75.175443	1	4	24	60	60	7	89	0	28172
752	209	45.469609	-75.154622	4	15	38	60	40	8	53	0	178319
6162	1724	45.404649	-75.246293	5	11	41	50	10	9	46	0	65699
4032	1099	45.552751	-75.164319	5	7	3	60	50	7	69	1	319078

11587 rows × 12 columns

From the documentation of the dataset, we will see that there is data imbalance. The legitimate tasks have 12,578 row and the fake tasks have only 1,897 rows, so we will do oversampling to try to balance the train data.

```
1 x_train_resampled
```

	ID	Latitude	Longitude	Day	Hour	Minute	Duration	RemainingTime	Resources	Coverage	OnPeakHours	GridNumber
0	2206	45.504761	-75.214953	2	23	44	50	20	1	79	0	234615
1	334	45.518957	-75.229820	0	21	35	60	20	7	43	0	262765
2	2909	45.520886	-75.215634	0	22	53	10	10	5	85	0	262767
3	2066	45.402016	-75.272772	5	4	48	50	50	1	76	0	56312
4	262	45.409482	-75.210547	2	22	17	30	10	7	72	0	75088
...	...	...	...	...	...	...	...	...	...	...	...	...
20125	1341	45.467918	-75.154732	1	8	34	10	10	8	68	1	168935
20126	2788	45.468814	-75.152152	2	8	16	40	10	3	45	1	168935
20127	654	45.485833	-75.218040	4	13	51	50	20	10	32	0	206463
20128	3861	45.487734	-75.218539	2	12	42	60	10	7	55	0	206463
20129	1564	45.468238	-75.153630	1	7	53	50	30	8	70	1	168935

20130 rows × 12 columns

3. Apply models

Our first model is Random Forest

```python
RF_model = RandomForestClassifier(random_state=42)
RF_model = RF_model.fit(x_train_resampled, y_train_resampled)
y_pred_RF = RF_model.predict(x_test)
```

4. Prediction of Random Forest

```
[15] y_pred_RF

array([1, 1, 1, ..., 1, 1, 1])
```

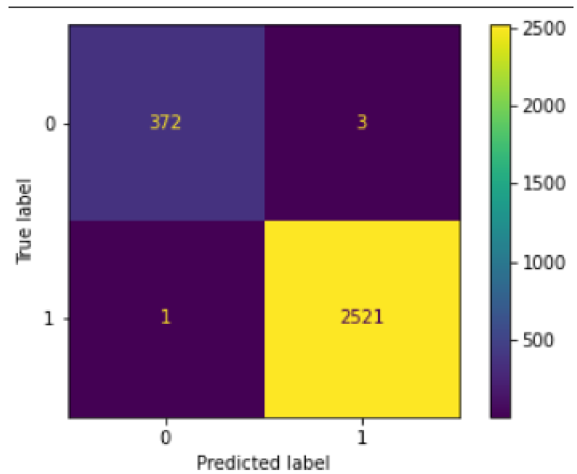
Classification Report for Random Forest

```python
[4]
def cal_accuracies(y_test, y_pred):
  print('\nClassification Report:\n')
  print(classification_report(y_test, y_pred, target_names = ["Class 0", "legit Class 1"]))
  print('\nConfusion Matrix:\n')
  cm = confusion_matrix(y_test, y_pred)
  print(cm)
  print('\nAccuracy Score:\n')
  accuracy = accuracy_score(y_test, y_pred)
  print(accuracy)
  print('\Confusion Matrix Display:\n')
  print(ConfusionMatrixDisplay(cm).plot())
  return accuracy
```

Classification Report for Random Forest

```
RF_acc = cal_accuracies(y_test, y_pred_RF)

Classification Report:

               precision    recall  f1-score   support

      Class 0       1.00      0.99      0.99       375
legit Class 1       1.00      1.00      1.00      2522

     accuracy                           1.00      2897
    macro avg       1.00      1.00      1.00      2897
 weighted avg       1.00      1.00      1.00      2897


Confusion Matrix:

[[ 372    3]
 [   1 2521]]

Accuracy Score:

0.9986192613047981
```

Confusion Matrix



Our second model is AdaBoost

```
[17]  AD_model  = AdaBoostClassifier(n_estimators=100, random_state=42)
      AD_model = AD_model.fit(x_train_resampled, y_train_resampled)
      y_pred_AB = AD_model.predict(x_test)
```

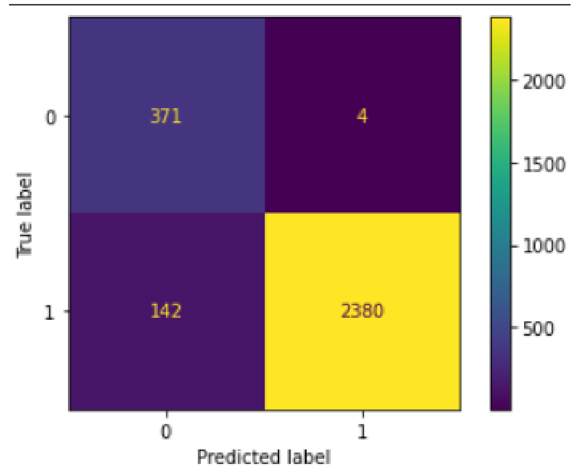
Prediction of Adaboost

```
[18]  y_pred_AB

    array([1, 1, 1, ..., 1, 1, 1])
```

Classification Report for AdaBoost

```
AB_acc = cal_accuracies(y_test, y_pred_AB)

Classification Report:

                precision    recall  f1-score   support

       Class 0       0.72      0.99      0.84       375
 legit Class 1       1.00      0.94      0.97      2522

      accuracy                           0.95      2897
     macro avg       0.86      0.97      0.90      2897
  weighted avg       0.96      0.95      0.95      2897


Confusion Matrix:

[[ 371    4]
 [ 142 2380]]

Accuracy Score:

0.9496030376251294
```

Confusion Matrix



5. Plot the comparison between the accuracies of our models

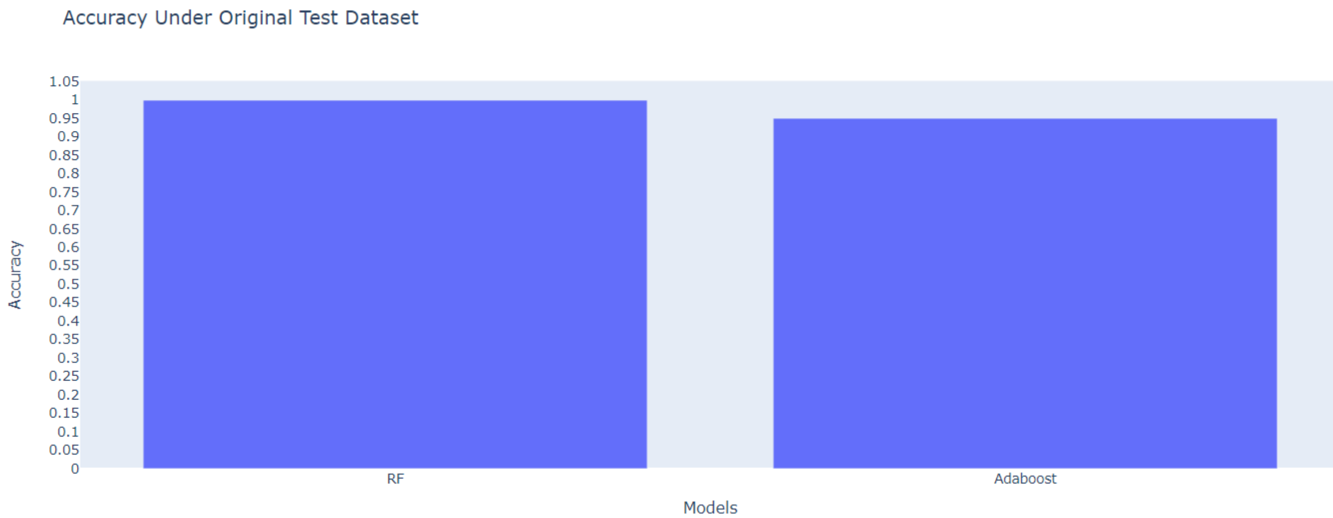


Table to compare the accuracies of the models before the GAN.

Models	Accuracy
Random Forest	99.8%
AdaBoost	94.9%

6. GAN Model

First, we used MinMaxScaler to scale our data.

```
[49] scaler = MinMaxScaler()
     x_train_scaled = scaler.fit_transform(x_train_resampled)
     x_test_scaled = scaler.transform(x_test)

[50] x_test_scaled

array([[0.96324081, 0.92083502, 0.49990828, ..., 0.4       , 0.        ,
        0.89997416],
       [0.31657914, 0.68199069, 0.14896806, ..., 0.25714286, 1.        ,
        0.67495864],
       [0.34133533, 0.86389702, 0.91261469, ..., 0.22857143, 1.        ,
        0.85000919],
       ...,
       [0.8432108 , 0.44909557, 0.67515125, ..., 0.34285714, 0.        ,
        0.45001505],
       [0.57289322, 0.66896582, 0.60065162, ..., 0.67142857, 0.        ,
        0.64999747],
       [0.21830458, 0.59749968, 0.17413294, ..., 0.47142857, 1.        ,
        0.57496743]])
```

Discriminator Function

```
[22]
def define_discriminator():
  model = Sequential()
  #input layer
  model.add(Flatten(input_shape=x_train_scaled.shape[1:]))

  def add_discriminator_block(neurons, alpha = 0.3):
    model.add(Dense(neurons))
    model.add(LeakyReLU(alpha))

  add_discriminator_block(256)
  add_discriminator_block(512)

  model.add(Dense(1, activation='sigmoid'))

  # model.summary()

  data = Input(shape=x_train_scaled.shape[1:])
  output = model(data)

  return Model(data, output)
```

Generator Function

```
[23]
noise_shape = (100,)

#latent_dim is the dimension of the latent vector (e.g., 100)
def define_generator():
  model = Sequential()

  def add_generator_block(neurons, alpha = 0.3):
    model.add(Dense(neurons))
    model.add(LeakyReLU(alpha))
    model.add(BatchNormalization())

  #input layer
  model.add(Dense(256, input_shape=noise_shape))
  model.add(LeakyReLU(alpha=.3))
  model.add(BatchNormalization())

  add_generator_block(512)
  add_generator_block(1024)

  model.add(Dense(np.prod(x_train_scaled.shape[1:]), activation='tanh'))
  #This is my output layer which should represent my fake generated image
  model.add(Reshape(x_train_scaled.shape[1:]))

  # model.summary()

  noise = Input(shape=noise_shape)
  data = model(noise)

  return Model(noise, data)
```

7. Apply the provided training dataset to GAN

```
[24]
def train(epochs, batch, generator, discriminator, combined):
  half_batch = int(batch / 2)
  for epoch in range(epochs):

    # train discriminator
    indices = np.random.randint(0, x_train_scaled.shape[0], half_batch)
    data = x_train_scaled[indices]

    #normal distribution noise
    noise = np.random.normal(0, 1, (half_batch, 100))
    fake_data = generator.predict(noise)

    discriminator_loss_real = discriminator.train_on_batch(data, np.ones((half_batch, 1)))
    discriminator_loss_fake = discriminator.train_on_batch(fake_data, np.zeros((half_batch, 1)))
    discriminator_loss = .5 * np.add(discriminator_loss_real, discriminator_loss_fake)

    # train generator
    noise = np.random.normal(0,1, (batch, 100))
    valid_y = np.array([1]*batch)
    generator_loss = combined.train_on_batch(noise, valid_y)

    print("epoch: %d Discriminator loss: %f - Generator loss: %f" % (epoch+1,
                                                                      discriminator_loss[0],
                                                                      generator_loss))
```

Implementation of the GAN

```
[25]
discriminator = define_discriminator()
discriminator.compile(loss="binary_crossentropy", optimizer= "adam", metrics=["accuracy"])

generator = define_generator()
generator.compile(loss="binary_crossentropy", optimizer= "adam")

noise = Input(shape=(100,))
fake_data = generator(noise)

discriminator.trainable = False

valid = discriminator(fake_data)

combined = Model(noise, valid)
combined.compile(loss="binary_crossentropy", optimizer= "adam")

train(2000, 32, generator, discriminator, combined)
```

The last 10 epochs of the training

```
epoch: 490 Discriminator loss: 0.199105 - Generator loss: 2.533022
epoch: 491 Discriminator loss: 0.349513 - Generator loss: 2.282841
epoch: 492 Discriminator loss: 0.398211 - Generator loss: 2.892312
epoch: 493 Discriminator loss: 0.302201 - Generator loss: 2.841201
epoch: 494 Discriminator loss: 0.241699 - Generator loss: 2.663990
epoch: 495 Discriminator loss: 0.231818 - Generator loss: 2.473982
epoch: 496 Discriminator loss: 0.291553 - Generator loss: 1.945768
epoch: 497 Discriminator loss: 0.260521 - Generator loss: 2.410531
epoch: 498 Discriminator loss: 0.217342 - Generator loss: 2.958851
epoch: 499 Discriminator loss: 0.234307 - Generator loss: 2.808013
epoch: 500 Discriminator loss: 0.244910 - Generator loss: 3.147508
```

8. Generate synthetic fake tasks via Generator network in GAN

Generate the fake data based on normal distribution and the output will be of size (batch size, 100)

```
new_fake_scaled = generator.predict(np.random.normal(0, 1, (1000, 100)))
pd.DataFrame(new_fake_scaled)
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.998731	-0.928944	0.623888	0.892624	0.781644	0.882846	0.152395	0.132759	-0.509205	-0.050465	-0.982832	-0.885357
1	0.774669	0.999996	0.611965	0.734027	0.534774	0.869223	0.999923	0.214865	0.999565	0.993074	0.999941	0.999936
2	0.619393	0.999939	0.510608	0.601337	0.498849	0.654515	0.999467	0.206532	0.997307	0.985556	0.999517	0.999519
3	0.847083	1.000000	0.485383	0.900767	0.514376	0.896539	0.999996	0.638939	0.999987	0.999238	0.999998	0.999992
4	0.974898	0.835256	0.313175	0.511008	0.799219	0.145615	0.758353	0.016510	0.049509	-0.182934	0.166972	0.969837
...	...	...	...	...	...	...	...	...	...	...	...	...
995	0.399686	1.000000	0.601850	0.740838	0.438788	0.882673	0.999997	0.479042	0.999913	0.999258	0.999998	0.999966
996	0.046256	1.000000	0.820530	0.931043	0.078248	0.995315	1.000000	0.695652	1.000000	1.000000	1.000000	1.000000
997	0.485851	0.999995	0.529999	0.856772	0.726184	0.629337	0.999890	0.375936	0.999803	0.994280	0.999977	0.999953
998	0.999633	-0.959580	0.924238	0.994148	0.715244	0.997967	0.960750	0.506445	0.763460	0.018045	-0.998679	-0.980485
999	0.999293	-0.576449	0.807124	0.977813	0.798653	0.965751	0.975212	0.281496	0.645050	-0.097006	-0.950722	0.338552

1000 rows × 12 columns

Adding column names to fake data

```
new_fake_scaled = pd.DataFrame(new_fake_scaled, columns=['ID', 'Latitude','Longitude',
                                                         'Day', 'Hour', 'Minute',
                                                         'Duration','RemainingTime',  'Resources' ,
                                                         'Coverage',  'OnPeakHours',  'GridNumber'])

new_fake_scaled
```

	ID	Latitude	Longitude	Day	Hour	Minute	Duration	RemainingTime	Resources	Coverage	OnPeakHours	GridNumber
0	0.998731	-0.928944	0.623888	0.892624	0.781644	0.882846	0.152395	0.132759	-0.509205	-0.050465	-0.982832	-0.885357
1	0.774669	0.999996	0.611965	0.734027	0.534774	0.869223	0.999923	0.214865	0.999565	0.993074	0.999941	0.999936
2	0.619393	0.999939	0.510608	0.601337	0.498849	0.654515	0.999467	0.206532	0.997307	0.985556	0.999517	0.999519
3	0.847083	1.000000	0.485383	0.900767	0.514376	0.896539	0.999996	0.638939	0.999987	0.999238	0.999998	0.999992
4	0.974898	0.835256	0.313175	0.511008	0.799219	0.145615	0.758353	0.016510	0.049509	-0.182934	0.166972	0.969837
...	...	...	...	...	...	...	...	...	...	...	...	...
995	0.399686	1.000000	0.601850	0.740838	0.438788	0.882673	0.999997	0.479042	0.999913	0.999258	0.999998	0.999966
996	0.046256	1.000000	0.820530	0.931043	0.078248	0.995315	1.000000	0.695652	1.000000	1.000000	1.000000	1.000000
997	0.485851	0.999995	0.529999	0.856772	0.726184	0.629337	0.999890	0.375936	0.999803	0.994280	0.999977	0.999953
998	0.999633	-0.959580	0.924238	0.994148	0.715244	0.997967	0.960750	0.506445	0.763460	0.018045	-0.998679	-0.980485
999	0.999293	-0.576449	0.807124	0.977813	0.798653	0.965751	0.975212	0.281496	0.645050	-0.097006	-0.950722	0.338552

1000 rows × 12 columns

Original scaled test data

```
x_test_scaled = pd.DataFrame(x_test_scaled, columns=['ID', 'Latitude','Longitude',
                                                      'Day', 'Hour', 'Minute',
                                                      'Duration','RemainingTime',  'Resources' ,
                                                      'Coverage',  'OnPeakHours',  'GridNumber'])

x_test_scaled
```

	ID	Latitude	Longitude	Day	Hour	Minute	Duration	RemainingTime	Resources	Coverage	OnPeakHours	GridNumber
0	0.963241	0.920835	0.499908	0.833333	0.521739	0.813559	0.2	0.2	0.555556	0.400000	0.0	0.899974
1	0.316579	0.681991	0.148968	0.333333	0.434783	0.237288	0.6	0.4	1.000000	0.257143	1.0	0.674959
2	0.341335	0.863897	0.912615	0.500000	0.304348	0.491525	1.0	0.4	1.000000	0.228571	1.0	0.850009
3	0.481620	0.168476	0.346074	0.166667	0.478261	0.762712	0.8	0.4	0.777778	0.357143	0.0	0.150007
4	0.894474	0.716124	0.427511	0.666667	0.913043	0.966102	1.0	0.2	0.111111	0.514286	0.0	0.699978
...	...	...	...	...	...	...	...	...	...	...	...	...
2892	0.278570	0.313376	0.697648	0.333333	1.000000	0.694915	0.2	0.2	1.000000	0.614286	0.0	0.300027
2893	0.770443	0.570796	0.322278	0.166667	0.608696	0.644068	0.6	0.6	0.888889	0.971429	0.0	0.549980
2894	0.843211	0.449096	0.675151	0.333333	0.695652	0.271186	0.6	0.2	0.222222	0.342857	0.0	0.450015
2895	0.572893	0.668966	0.600652	0.333333	0.173913	0.559322	1.0	0.8	0.666667	0.671429	0.0	0.649997
2896	0.218305	0.597500	0.174133	0.833333	0.304348	0.779661	0.8	0.0	0.444444	0.471429	1.0	0.574967

2897 rows × 12 columns

9. Mix the generated fake tasks with the original test dataset to obtain a new test dataset

```
x_mixed_data = pd.concat([x_test_scaled, new_fake_scaled])
x_mixed_data
```

	ID	Latitude	Longitude	Day	Hour	Minute	Duration	RemainingTime	Resources	Coverage	OnPeakHours	GridNumber
0	0.963241	0.920835	0.499908	0.833333	0.521739	0.813559	0.200000	0.200000	0.555556	0.400000	0.000000	0.899974
1	0.316579	0.681991	0.148968	0.333333	0.434783	0.237288	0.600000	0.400000	1.000000	0.257143	1.000000	0.674959
2	0.341335	0.863897	0.912615	0.500000	0.304348	0.491525	1.000000	0.400000	1.000000	0.228571	1.000000	0.850009
3	0.481620	0.168476	0.346074	0.166667	0.478261	0.762712	0.800000	0.400000	0.777778	0.357143	0.000000	0.150007
4	0.894474	0.716124	0.427511	0.666667	0.913043	0.966102	1.000000	0.200000	0.111111	0.514286	0.000000	0.699978
...	...	...	...	...	...	...	...	...	...	...	...	...
995	0.399686	1.000000	0.601850	0.740838	0.438788	0.882673	0.999997	0.479042	0.999913	0.999258	0.999998	0.999966
996	0.046256	1.000000	0.820530	0.931043	0.078248	0.995315	1.000000	0.695652	1.000000	1.000000	1.000000	1.000000
997	0.485851	0.999995	0.529999	0.856772	0.726184	0.629337	0.999890	0.375936	0.999803	0.994280	0.999977	0.999953
998	0.999633	-0.959580	0.924238	0.994148	0.715244	0.997967	0.960750	0.506445	0.763460	0.018045	-0.998679	-0.980485
999	0.999293	-0.576449	0.807124	0.977813	0.798653	0.965751	0.975212	0.281496	0.645050	-0.097006	-0.950722	0.338552

3897 rows × 12 columns

We applied the scaler inverse transform and used absolute value function to remove the negative signs and round the values to change it to integer format except for the “Longitude” column like the original data.

```
[30]
x_mixed_data = pd.DataFrame(scaler.inverse_transform(x_mixed_data), columns=['ID', 'Latitude','Longitude',
                                                                              'Day', 'Hour', 'Minute',
                                                                              'Duration','RemainingTime',  'Resources' ,
                                                                              'Coverage',  'OnPeakHours',  'GridNumber'])


x_mixed_data.ID = x_mixed_data.ID.abs().round().astype(int)
x_mixed_data_final = x_mixed_data.iloc[:,3:].abs().round().astype(int)

x_mixed_data_final.insert(0, "ID", x_mixed_data.iloc[:,0], True)
x_mixed_data_final.insert(1, "Latitude", x_mixed_data.iloc[:,1], True)
x_mixed_data_final.insert(2, "Longitude", x_mixed_data.iloc[:,2], True)

x_mixed_data_final
```

	ID	Latitude	Longitude	Day	Hour	Minute	Duration	RemainingTime	Resources	Coverage	OnPeakHours	GridNumber
0	3853	45.567007	-75.211538	5	12	48	20	20	6	58	0	337840
1	1267	45.514863	-75.297589	2	10	14	40	30	10	48	1	253373
2	1366	45.554577	-75.110342	3	7	29	60	30	10	46	1	319084
3	1927	45.402752	-75.249258	1	11	45	50	30	8	55	0	56315
4	3578	45.522315	-75.229290	4	21	57	60	20	2	66	0	262765
...	...	...	...	...	...	...	...	...	...	...	...	...
3892	1599	45.584291	-75.186542	4	10	52	60	34	10	100	1	375375
3893	186	45.584291	-75.132921	6	2	59	60	45	10	100	1	375388
3894	1944	45.584290	-75.204159	5	17	37	60	29	10	100	1	375370
3895	3999	45.156475	-75.107492	6	16	59	58	35	8	31	1	368052
3896	3997	45.240120	-75.136208	6	18	57	59	24	7	23	1	127092

3897 rows × 12 columns

We added the label 0 (Real) to the data which was generated before and added it to the original test label.

```
[84]
x = pd.Series([0]*1000)
y_mixed_data = y_test.Ligitimacy.append(x)
y_mixed_data

13922    1
4605     0
4970     1
6873     0
12842    1
        ..
995      0
996      0
997      0
998      0
999      0
Length: 3897, dtype: int64
```

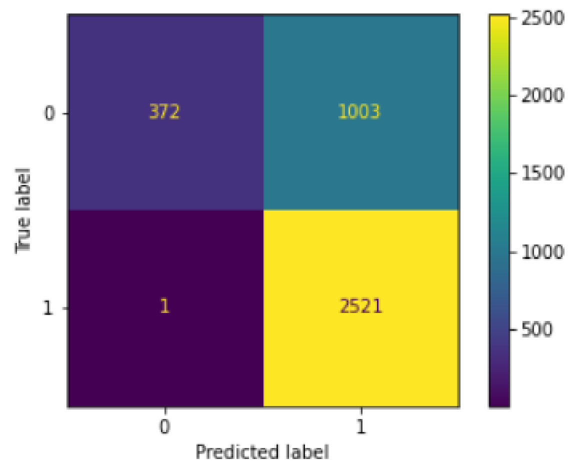
10. Obtain Adaboost and RF detection performance using the new test dataset and present results in bar chart without the discriminator.

Now we will apply the two models with our mixed data. For the Random Forest model.

Classification Report for Random Forest

```
[85]  1 y_mixed_pred_RF = RF_model.predict(x_mixed_data_final)

      1 RF_mixed_acc = cal_accuracies(y_mixed_data, y_mixed_pred_RF)

    Classification Report:

                   precision    recall  f1-score   support

         Class 0       1.00      0.27      0.43      1375
    legit Class 1       0.72      1.00      0.83      2522

        accuracy                           0.74      3897
       macro avg       0.86      0.64      0.63      3897
     weighted avg       0.81      0.74      0.69      3897


    Confusion Matrix:

    [[ 372 1003]
     [   1 2521]]

    Accuracy Score:

    0.7423659225044906
```

Confusion Matrix

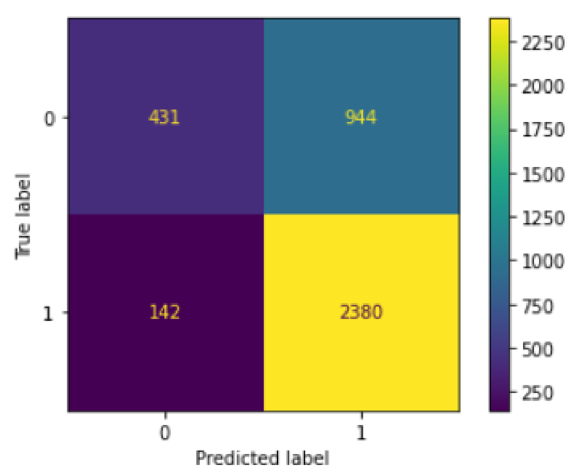


Applying Adaboost with our mixed data

Classification Report for Adaboost

```
[87]  1 y_mixed_pred_AB = AD_model.predict(x_mixed_data_final)

      1 AB_mixed_acc = cal_accuracies(y_mixed_data, y_mixed_pred_AB)

    Classification Report:

                   precision    recall  f1-score   support

         Class 0       0.75      0.31      0.44      1375
    legit Class 1       0.72      0.94      0.81      2522

        accuracy                           0.72      3897
       macro avg       0.73      0.63      0.63      3897
     weighted avg       0.73      0.72      0.68      3897


    Confusion Matrix:

    [[ 431  944]
     [ 142 2380]]

    Accuracy Score:

    0.7213240954580447
```

Confusion Matrix



Plot the comparison between the accuracies of our models

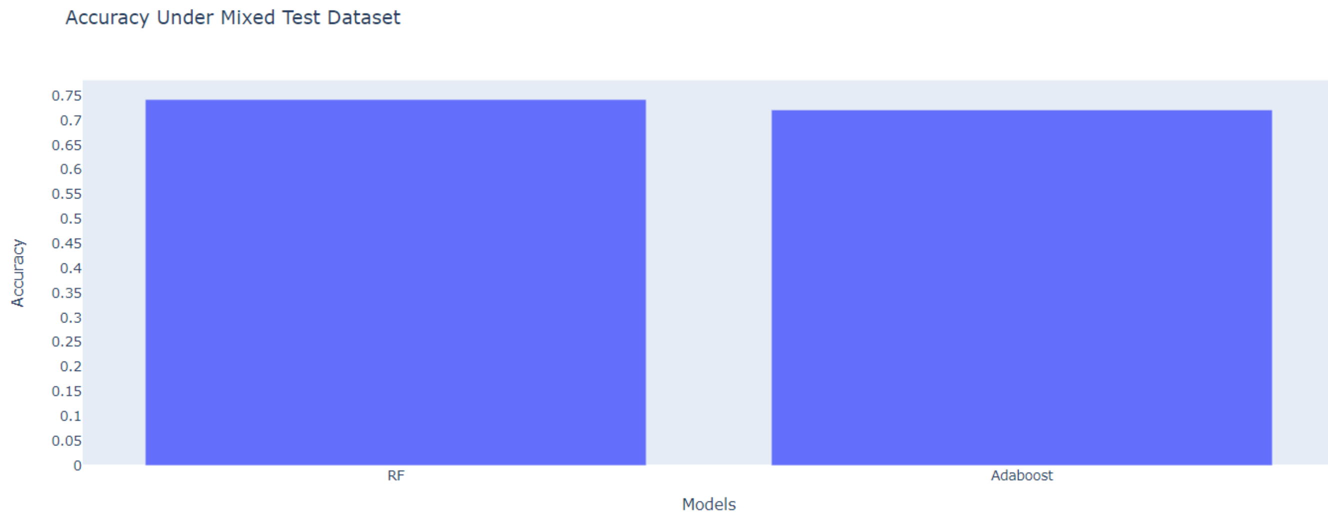


Table to compare the accuracies of the mixed data after the generator.

Models	Accuracy
Random Forest	74.2%
AdaBoost	72.1%

11. According to the cascade detection framework, as shown in Figure 1, verify the cascade framework performance and show results in bar chart

We passed the mixed data to the discriminator, and these are the probabilities of the predictions.

```
[90]  1 discriminator_pred = discriminator.predict(scaler.transform(x_mixed_data_final))

[91]  1 discriminator_pred

     array([[0.57477736],
            [0.93588215],
            [0.70250297],
            ...,
            [0.26107824],
            [0.00316095],
            [0.42230248]], dtype=float32)
```

Then, we implemented a condition that round the probabilities which are bigger than 0.5 to 1, otherwise 0.

```
[92]
for i in range(len(discriminator_pred)):
  if discriminator_pred[i] >= 0.5:
    discriminator_pred[i] = 1
  else:
    discriminator_pred[i] = 0

discriminator_pred
```

```
array([[1.],
       [1.],
       [1.],
       ...,
       [0.],
       [0.],
       [0.]], dtype=float32)
```

Now, we added the discriminator prediction and the original label to our mixed data.

```
[93]
x_mixed_data_final['Ligitimacy'] = y_mixed_data.to_numpy()
x_mixed_data_final['d_prediction'] = discriminator_pred.reshape(-1).astype(int)
```

```
x_mixed_data_final
```

	ID	Latitude	Longitude	Day	Hour	Minute	Duration	RemainingTime	Resources	Coverage	OnPeakHours	GridNumber	Ligitimacy	d_prediction
0	3853	45.567007	-75.211538	5	12	48	20	20	6	58	0	337840	1	1
1	1267	45.514863	-75.297589	2	10	14	40	30	10	48	1	253373	0	1
2	1366	45.554577	-75.110342	3	7	29	60	30	10	46	1	319084	1	1
3	1927	45.402752	-75.249258	1	11	45	50	30	8	55	0	56315	0	1
4	3578	45.522315	-75.229290	4	21	57	60	20	2	66	0	262765	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3892	1599	45.584291	-75.186542	4	10	52	60	34	10	100	1	375375	0	0
3893	186	45.584291	-75.132921	6	2	59	60	45	10	100	1	375388	0	0
3894	1944	45.584290	-75.204159	5	17	37	60	29	10	100	1	375370	0	0
3895	3999	45.156475	-75.107492	6	16	59	58	35	8	31	1	368052	0	0
3896	3997	45.240120	-75.136208	6	18	57	59	24	7	23	1	127092	0	0

3897 rows × 14 columns

We want to get all the true tasks from the discriminator prediction, so we filtered the rows which equals the prediction 1.

```
[95]  1 real_data_df = x_mixed_data_final[x_mixed_data_final.d_prediction==1]
      2 real_data_df.d_prediction.sum()

    2716
```

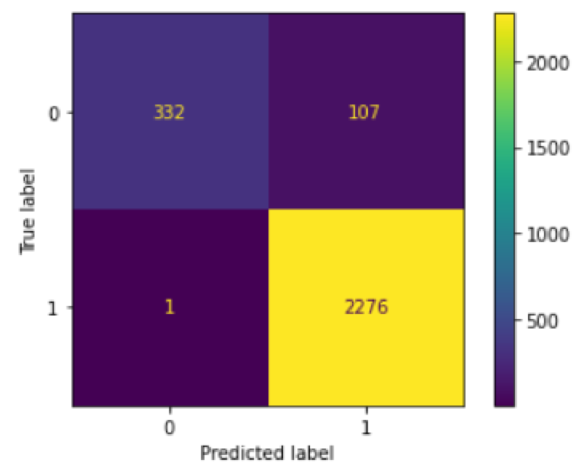
Applying the Random Forest model with our filtered data.

```
[96]  1 y_d_pred_RF = RF_model.predict(real_data_df.iloc[:,:-2])
      2 y_d_pred_RF

    array([1, 0, 1, ..., 1, 1, 1])
```

Classification Report for Random Forest

```
  1 RF_d_acc = cal_accuracies(y_actual, y_d_pred_RF)

Classification Report:

                precision    recall  f1-score   support

       Class 0       1.00      0.76      0.86       439
 legit Class 1       0.96      1.00      0.98      2277

      accuracy                           0.96      2716
     macro avg       0.98      0.88      0.92      2716
  weighted avg       0.96      0.96      0.96      2716


Confusion Matrix:

[[ 332  107]
 [   1 2276]]

Accuracy Score:

0.9602356406480118
```

Confusion Matrix

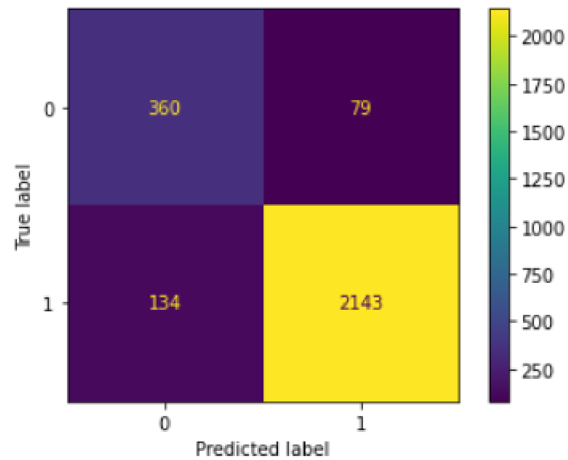


Applying the Adaboost model with our filtered data.

Classification Report for Adaboost

```
[100]  1 AB_d_acc = cal_accuracies(y_actual, y_d_pred_AB)

     Classification Report:

                    precision    recall  f1-score   support

          Class 0       0.73      0.82      0.77       439
     legit Class 1       0.96      0.94      0.95      2277

         accuracy                           0.92      2716
        macro avg       0.85      0.88      0.86      2716
     weighted avg       0.93      0.92      0.92      2716


     Confusion Matrix:

     [[ 360   79]
      [ 134 2143]]

     Accuracy Score:

     0.9215758468335787
```

Confusion Matrix



Plot the comparison between the accuracies of our models

Table to compare the accuracies of the mixed data after the generator.

Models	Accuracy
Random Forest	96%
AdaBoost	92.1%

Conclusion

We noticed that the **Random Forest** model performed better than the **Adaboost** model in all the above cases (*Traditional ML methods, GAN Generated Method and the Cascade framework methods*). The difference between GAN generated data methods and the Cascade framework methods is that the one layer classifier (**RF and Adaboost models**) performed poor compared with the two layer classifier (first layer was the discriminator and the second one was the ML models) which has the best accuracy as shown which was **96%** for Random forest with the discriminator where the Random forest without the discriminator was **74.2%** and that makes sense because the discriminator performed as a filtration layer that give more accurate real results.