2022 Summer ELG 5142 Ubiquitous Sensing and Smart City Project

Group-18

1. Download MCS dataset which is used in assignment 2:

	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
5	2	45.447747	-75.147951	2	15	59	30	20	5	47	0	140784	1
6	2	45.447790	-75.148303	2	16	9	30	10	5	47	0	140784	1
7	3	45.508896	-75.259807	2	12	27	30	30	4	43	0	243994	1
8	3	45.508748	-75.260652	2	12	37	30	20	4	43	0	243994	1
9	3	45.508082	-75.260380	2	12	47	30	10	4	43	0	243994	1

```
# showing the number of rows of target [0 or 1]
dataset["Ligitimacy"].value_counts()
```

```
1    12587
0     1897
Name: Ligitimacy, dtype: int64
```

We removed useless ID column.

```
[ ]  #remove ID coulmn
     X.drop("ID",axis=1,inplace=True)
```

2. Split the dataset into two for training (80%) and test (20%)

```
#split dataset into trianing and testing with 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split( X_over, y_over, test_size=0.2, random_state=42)
```

We made scaling to make all features at the same range

```
# Scaling the data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(-1,1))
X_train = scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)
```

3. Implement classic classifiers (Adaboost and RF)
4. Train Adaboost and RF via training dataset
5. Verify detection performance using test dataset and present results comparison in bar chart

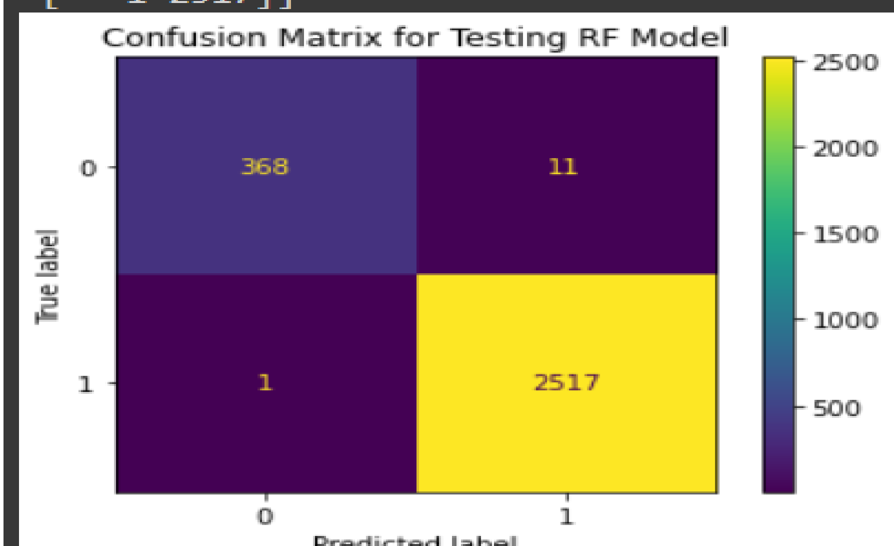
Use RF and Adaboost classification models separately using training dataset.

Random Forest Classifier

```
[ ] #Random Forest Classifier
    #We got best performance at max_depth= 23
    RFclf = RandomForestClassifier(max_depth=23, random_state=0)
    RFclf.fit(X_train, y_train)
```

```
[ ] # Testing accuracy of RF
    ypredRF_testing =RFclf.predict(X_test)
    RandomForest_test_acc = accuracy_score(y_test, ypredRF_testing)
    print(f'Testing accuracy of Random Forest: {RandomForest_test_acc*100}%')
    print('Classification Report for RF')
    print(classification_report(y_test, ypredRF_testing))
    getConfusionMatrix(RFclf,X_test,y_test,"Confusion Matrix for Testing RF Model")
```

```
Testing accuracy of Random Forest: 99.58577839143942%
Classification Report for RF
              precision    recall  f1-score   support

           0       1.00      0.97      0.98       379
           1       1.00      1.00      1.00      2518

    accuracy                           1.00      2897
   macro avg       1.00      0.99      0.99      2897
weighted avg       1.00      1.00      1.00      2897

[[ 368   11]
 [   1 2517]]
```

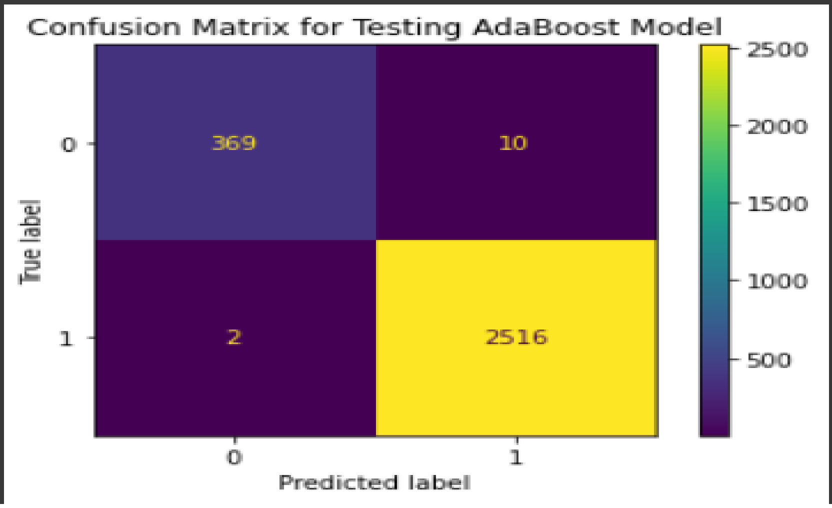


AdaBoost Classifier

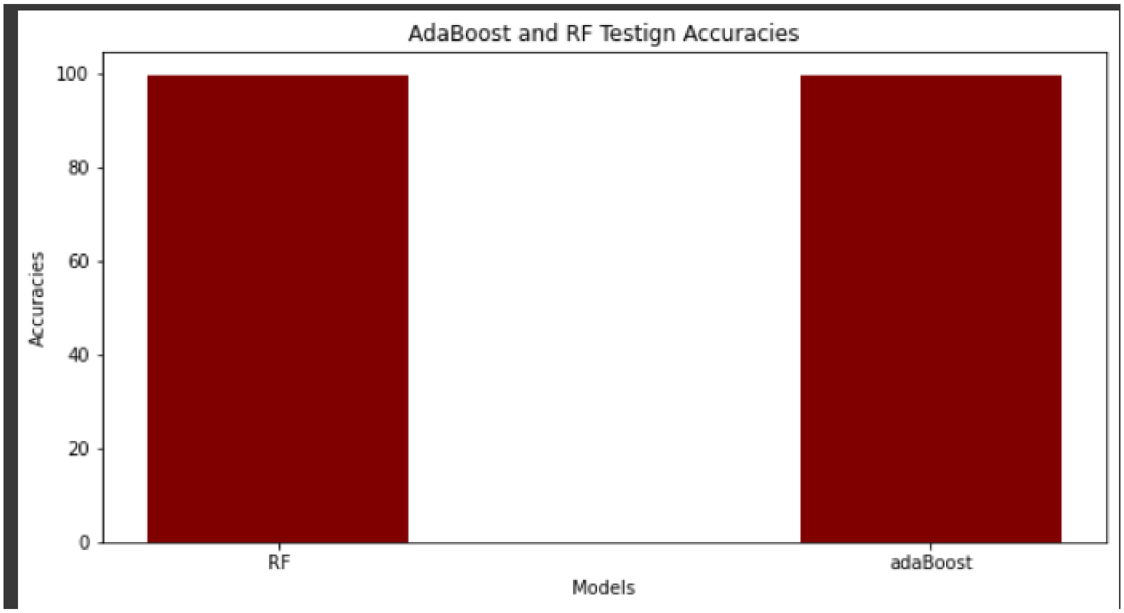
```
#AdaBoost Classifier
#We got best performance at n_estimators=2380
Adaclf = AdaBoostClassifier(n_estimators=2380, random_state=0)
Adaclf.fit(X_train, y_train)
```

```
[ ] # testing accuracy of Adaboost
    ypredAda_testing=Adaclf.predict(X_test)
    adaBoost_test_acc = accuracy_score(y_test, ypredAda_testing)
    print(f'Testing accuracy of Adaboost: {adaBoost_test_acc*100}%')
    print('Classification Report for Adaboost')
    print(classification_report(y_test, ypredAda_testing))
    getConfusionMatrix(Adaclf,X_test,y_test,"Confusion Matrix for Testing AdaBoost Model")
```

```
Testing accuracy of Adaboost: 99.58577839143942%
Classification Report for Adaboost
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       379
           1       1.00      1.00      1.00      2518

    accuracy                           1.00      2897
   macro avg       1.00      0.99      0.99      2897
weighted avg       1.00      1.00      1.00      2897

[[ 369   10]
 [   2 2516]]
```



Present results comparison in bar chart:



6. Implement a CGAN model

After reading the implementation manner of GAN model from [1] we made some updates on the model to make it suitable for our case.

Install required libraries:

```
!pip install keras
!pip install tensorflow
!pip install git+https://github.com/tensorflow/docs
```

Import required libraries:

```
from keras.layers import Activation, Dropout, Flatten, Dense, Input, LeakyReLU
from keras.layers import BatchNormalization
from keras.layers import concatenate, multiply
from keras.layers import Embedding
from keras.models import Model, Sequential
from tensorflow.keras.utils import to_categorical
import tensorflow
import keras
from tensorflow.keras.optimizers import Adam, RMSprop, SGD
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow_docs.vis import embed
import matplotlib.pyplot as plt
import tensorflow as tf
```

```
[ ] #some initialization for GAN Model
    batch_size = 64
    num_channels = 11 # number of our features
    num_classes = 2 #number of classes
    latent_dim = 128
```

We converted a class vector (y_train) to binary class matrix. We used tf.data.Dataset.from_tensor_slices () on X_train and labels to create tf.data.Dataset (dataset).

```
labels = keras.utils.to_categorical(y_train, 2)

# Create tf.data.Dataset.
dataset = tf.data.Dataset.from_tensor_slices((X_train ,labels))
dataset = dataset.shuffle(buffer_size=1024).batch(batch_size)

print(f"Shape of training set: {X_train.shape}")
print(f"Shape of training set: {labels.shape}")
```

```
Shape of training set: (11587, 11)
Shape of training set: (11587, 2)
```

We created the discriminator to discover the generator fault:

```
# the discriminator.
discriminator = keras.Sequential(
    [
        keras.layers.InputLayer(input_shape =   discriminator_in_channels),
        layers.Dense(512),
        layers.LeakyReLU(alpha=0.2),

        layers.Dense(512),
        layers.LeakyReLU(alpha=0.2),
        layers.Dropout(0.2),

        layers.Dense(1 ,activation="sigmoid")


    ],
    name="discriminator",
)
```

We created the generator to generate samples:

```
# Create the generator.
generator = keras.Sequential(
    [
        keras.layers.InputLayer((generator_in_channels)),
        layers.Dense(256),
        layers.LeakyReLU(alpha=0.2),
        layers.BatchNormalization(momentum=0.8),
        layers.Dense(512),
        layers.LeakyReLU(alpha=0.2),
        layers.BatchNormalization(momentum=0.8),
        layers.Dense(11)
    ],
    name="generator",
)
```

We created the conditional of GAN model:

```
#Creating a ConditionalGAN model
class ConditionalGAN(keras.Model):
    def __init__(self, discriminator, generator, latent_dim):
        super(ConditionalGAN, self).__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.latent_dim = latent_dim
        self.gen_loss_tracker = keras.metrics.Mean(name="generator_loss")
        self.disc_loss_tracker = keras.metrics.Mean(name="discriminator_loss")

    @property
    def metrics(self):
        return [self.gen_loss_tracker, self.disc_loss_tracker]

    def compile(self, d_optimizer, g_optimizer, loss_fn):
        super(ConditionalGAN, self).compile()
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer
        self.loss_fn = loss_fn

    def train_step(self, data):
        # Unpack the data.
        real_Tasks, one_hot_labels = data
        real_Tasks = tf.cast(real_Tasks,tf.float32)
        # Add dummy dimensions to the labels so that they can be concatenated with
        # the Tasks. This is for the discriminator.

        Task_one_hot_labels = one_hot_labels[:, :, None, None]
        Task_one_hot_labels = tf.reshape(
            Task_one_hot_labels, (-1,  num_classes)
        )

        # Sample random points in the latent space and concatenate the labels.
        # This is for the generator.
        batch_size = tf.shape(real_Tasks)[0]
        random_latent_vectors = tf.random.normal(shape=(batch_size, self.latent_dim))
        random_vector_labels = tf.concat(
            [random_latent_vectors, one_hot_labels], axis=1
        )
```

```python
# Decode the noise (guided by labels) to fake Tasks.
generated_Tasks = self.generator(random_vector_labels)

# Combine them with real Tasks. Note that we are concatenating the labels
# with these Tasks here.

fake_Task_and_labels = tf.concat([generated_Tasks, Task_one_hot_labels], -1)

real_Task_and_labels = tf.concat([real_Tasks, Task_one_hot_labels], -1)

combined_Tasks = tf.concat(
    [fake_Task_and_labels, real_Task_and_labels], axis=0
)

# Assemble labels discriminating real from fake Tasks.
labels = tf.concat(
    [tf.ones((batch_size, 1)), tf.zeros((batch_size, 1))], axis=0
)

# Train the discriminator.
with tf.GradientTape() as tape:

    predictions = self.discriminator(combined_Tasks)
    d_loss = self.loss_fn(labels, predictions)
grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
self.d_optimizer.apply_gradients(
    zip(grads, self.discriminator.trainable_weights)
)

# Sample random points in the latent space.
random_latent_vectors = tf.random.normal(shape=(batch_size, self.latent_dim))
random_vector_labels = tf.concat(
    [random_latent_vectors, one_hot_labels], axis=1
)

# Assemble labels that say "all real Tasks".
misleading_labels = tf.zeros((batch_size, 1))

# Train the generator (note that we should *not* update the weights
# of the discriminator)!
with tf.GradientTape() as tape:
    fake_Tasks = self.generator(random_vector_labels)
    fake_Task_and_labels = tf.concat([fake_Tasks, Task_one_hot_labels], -1)

    predictions = self.discriminator(fake_Task_and_labels)
    g_loss = self.loss_fn(misleading_labels, predictions)
grads = tape.gradient(g_loss, self.generator.trainable_weights)
self.g_optimizer.apply_gradients(zip(grads, self.generator.trainable_weights))

# Monitor loss.
self.gen_loss_tracker.update_state(g_loss)
self.disc_loss_tracker.update_state(d_loss)
return {
    "g_loss": self.gen_loss_tracker.result(),
    "d_loss": self.disc_loss_tracker.result(),
}
```

7. Apply the provided training dataset to CGAN
(The training dataset can be the same as you used in assignment 2)

We trained the conditional GAN on 50 epochs:

```
#Training the Conditional GAN

cond_gan = ConditionalGAN(
    discriminator=discriminator, generator=generator, latent_dim=latent_dim
)
cond_gan.compile(
    d_optimizer=keras.optimizers.Adam(learning_rate=0.0003),
    g_optimizer=keras.optimizers.Adam(learning_rate=0.0003),
    loss_fn=keras.losses.BinaryCrossentropy(),
)

cond_gan.fit(dataset, epochs=50)
```

The last 10 result of training of the conditional GAN:

```
Epoch 40/50
182/182 [==============================] - 1s 5ms/step - g_loss: 0.9538 - d_loss: 0.6240
Epoch 41/50
182/182 [==============================] - 1s 5ms/step - g_loss: 0.9317 - d_loss: 0.6196
Epoch 42/50
182/182 [==============================] - 1s 5ms/step - g_loss: 0.9826 - d_loss: 0.6150
Epoch 43/50
182/182 [==============================] - 1s 5ms/step - g_loss: 1.0268 - d_loss: 0.6101
Epoch 44/50
182/182 [==============================] - 1s 5ms/step - g_loss: 0.9645 - d_loss: 0.5977
Epoch 45/50
182/182 [==============================] - 1s 5ms/step - g_loss: 0.9972 - d_loss: 0.6032
Epoch 46/50
182/182 [==============================] - 1s 5ms/step - g_loss: 1.0005 - d_loss: 0.5472
Epoch 47/50
182/182 [==============================] - 1s 5ms/step - g_loss: 0.9449 - d_loss: 0.6269
Epoch 48/50
182/182 [==============================] - 1s 5ms/step - g_loss: 0.9833 - d_loss: 0.5678
Epoch 49/50
182/182 [==============================] - 1s 5ms/step - g_loss: 0.9812 - d_loss: 0.6676
Epoch 50/50
182/182 [==============================] - 1s 5ms/step - g_loss: 1.1078 - d_loss: 0.6643
```

8. Generate synthetic fake tasks via Generator network in CGAN after the training procedure

We generated fake 1000 samples to merge them with the original data and test our GAN model on the final combination:

```
#We first extract the trained generator from our Conditiona GAN.
trained_gen = cond_gan.generator

# Choose the number of intermediate functions that would be generated in
# between the interpolation + 2 (start and last functions).
num_interpolation = 1000  # @param {type:"integer"}

# Sample noise for the interpolation.
interpolation_noise = tf.random.normal(shape=(1, latent_dim))
interpolation_noise = tf.repeat(interpolation_noise, repeats=num_interpolation)
interpolation_noise = tf.reshape(interpolation_noise, (num_interpolation, latent_dim))


def interpolate_class(Class):
    # Convert the start and end labels to one-hot encoded vectors.
    first_label = keras.utils.to_categorical([Class]*num_interpolation, num_classes)
    noise_and_labels = tf.concat([interpolation_noise, first_label], 1)
    fake = trained_gen.predict(noise_and_labels)
    return fake

class_ = 1
fake_Functions = interpolate_class(class_)
```

num_interpolation: 1000

+ Code + Text

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

```
X_test_conbination_withoutCascade = np.concatenate((fake_Functions,X_test) , axis = 0)
y_test_conbination_withoutCascade =  np.concatenate((y_test["Ligitimacy"].values,np.zeros(num_interpolation)) , axis = 0).astype("int")
```

```
[[ 0.91811556  0.42507055  0.31489873 ... -1.3154926  -1.0451561
   0.9116875 ]
 [ 0.91811556  0.42507055  0.31489873 ... -1.3154926  -1.0451561
   0.9116875 ]
 [ 0.91811556  0.42507055  0.31489873 ... -1.3154926  -1.0451561
   0.9116875 ]
 ...
 [ 0.47818416  0.08184417  0.0404586  ... -0.57149327 -1.0558331
   0.39634234]
 [ 0.47818416  0.08184417  0.0404586  ... -0.57149327 -1.0558331
   0.39634234]
 [ 0.47818416  0.08184417  0.0404586  ... -0.57149327 -1.0558331
   0.39634234]]
```

10. Obtain Adaboost and RF detection performance using the new test dataset and present results in bar chart
(This step doesn’t consider Discriminator for filtering synthetic samples).

Random Forest Classifier

```
# RF Testing accuracy of our result data that combined the original data plut the generated data before cascading
ypredRF_testing_conbination_withoutCascade =RFclf.predict(X_test_conbination_withoutCascade)
RandomForest_test_acc_conbination_withoutCascade =
            accuracy_score(y_test_conbination_withoutCascade, ypredRF_testing_conbination_withoutCascade)
print(f'Testing accuracy of Random Forest of our result data that combined the original data plut the generated data before cascading:
                {RandomForest_test_acc_conbination_withoutCascade*100}%')
print('Classification Report for RF of our result data that combined the original data plut the generated data before cascading')
print(classification_report(y_test_conbination_withoutCascade, ypredRF_testing_conbination_withoutCascade))
getConfusionMatrix(RFclf,X_test_conbination_withoutCascade,y_test_conbination_withoutCascade,
     "Confusion Matrix for Testing RF Model of our result data that combined the original data plut the generated data before cascading")
```

```
Testing accuracy of Random Forest of our result data that combined the original data plut the generated data before cascading: 63.61303566846293%
Classification Report for RF of our result data that combined the original data plut the generated data before cascading
              precision    recall  f1-score   support

           0       0.45      0.12      0.19      1379
           1       0.66      0.92      0.77      2518

    accuracy                           0.64      3897
   macro avg       0.55      0.52      0.48      3897
weighted avg       0.58      0.64      0.56      3897
```

```
[[ 165 1214]
 [ 204 2314]]
```

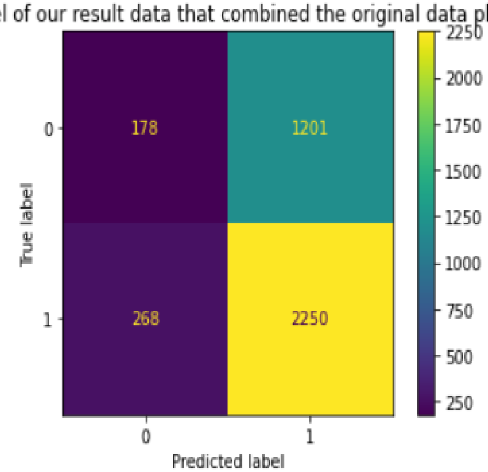
AdaBoost Classifier

```
# Adaboost Testing accuracy of our result data that combined the original data plut the generated data before cascading
ypredAda_testing_conbination_withoutCascade =Adaclf.predict(X_test_conbination_withoutCascade)
Ada_test_acc_conbination_withoutCascade = accuracy_score(y_test_conbination_withoutCascade,
                                                          ypredAda_testing_conbination_withoutCascade)
print(f'Testing accuracy of Adaboost of our result data that combined \
 the original data plut the generated data before cascading: {Ada_test_acc_conbination_withoutCascade*100}%')
print('Classification Report for Adaboost of our result data that combined the original data plut the generated data before cascading')
print(classification_report(y_test_conbination_withoutCascade, ypredAda_testing_conbination_withoutCascade))
getConfusionMatrix(Adaclf,X_test_conbination_withoutCascade,y_test_conbination_withoutCascade,
    "Confusion Matrix for Testing Adaboost Model of our result data that combined the original data plut the generated data before cascading")
```

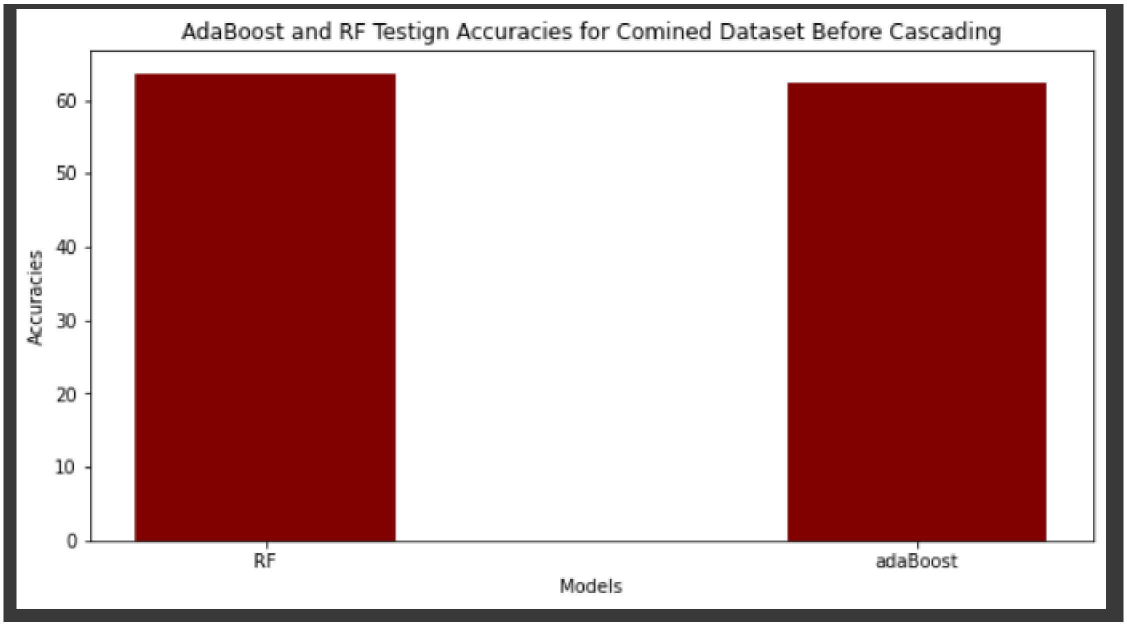
```
Testing accuracy of Adaboost of our result data that combined the original data plut the generated data before cascading: 63.81832178598922%
Classification Report for Adaboost of our result data that combined the original data plut the generated data before cascading
              precision    recall  f1-score   support

           0       0.46      0.12      0.19      1379
           1       0.66      0.92      0.77      2518

    accuracy                           0.64      3897
   macro avg       0.56      0.52      0.48      3897
weighted avg       0.59      0.64      0.56      3897
```

```
[[ 178 1201]
 [ 268 2250]]
```

Confusion Matrix for Testing Adaboost Model of our result data that combined the original data plut the generated data before cascading



Present results comparison in the accuracy in bar chart:



The above graph shows that the accuracy of two models is lower than the performance of original training data (without using GAN model).

11. According to the cascade detection framework, as shown in Figure 1, verify the cascade framework performance and show results in bar chart. Consider the Discriminator to as the first level classifier and RF/Adaboost as the second level classifier.

Cascading Process to Enhance Testing Results

```
#prepare our data to the discriminator
y_one_hot = keras.utils.to_categorical(y_test_conbination_withoutCascade, 2)
data_for_discriminator = tf.concat([X_test_conbination_withoutCascade, y_one_hot], axis=1 )
print(len(data_for_discriminator))
#run the discrimintaor as first level classifier
discriminator_pred = cond_gan.discriminator.predict(data_for_discriminator)
#convert the output to labels of 0 and 1
discriminator_pred = np.round(discriminator_pred)
data_index_that_predicted_from_discriminator = np.where(discriminator_pred == 1)[0]
print(data_index_that_predicted_from_discriminator)
print(data_index_that_predicted_from_discriminator.shape)
X_test_conbination_Cascade = X_test_conbination_withoutCascade[data_index_that_predicted_from_discriminator]
print(X_test_conbination_Cascade.shape)
y_test_conbination_Cascade = np.ones(X_test_conbination_Cascade.shape[0]).astype("int")
```

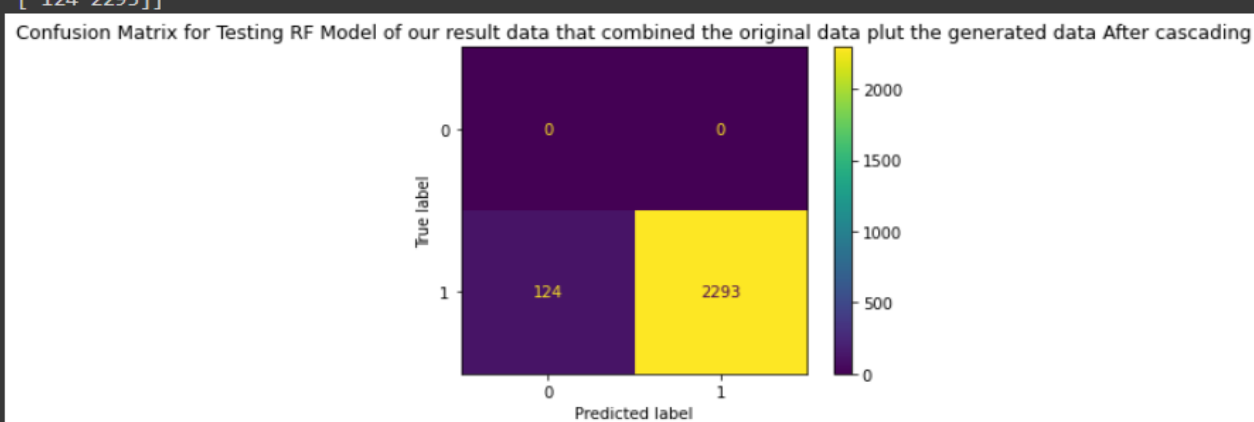
Use previous RF and Adaboost classification models separately to predict our using training + GAN generated dataset after cascading .

Random Forest Classifier

```
# RF Testing accuracy of our result data that combined the original data plut the generated data After cascading
ypredRF_testing_conbination_Cascade =RFclf.predict(X_test_conbination_Cascade)
RandomForest_test_acc_conbination_Cascade = accuracy_score(y_test_conbination_Cascade, ypredRF_testing_conbination_Cascade)
print(f'Testing accuracy of Random Forest of our result data that combined the original data plut the generated data After cascading:
                {RandomForest_test_acc_conbination_Cascade*100}%')
print('Classification Report for RF of our result data that combined the original data plut the generated data After cascading')
print(classification_report(y_test_conbination_Cascade, ypredRF_testing_conbination_Cascade))
getConfusionMatrix(RFclf,X_test_conbination_Cascade,y_test_conbination_Cascade,
   "Confusion Matrix for Testing RF Model of our result data that combined the original data plut the generated data After cascading")
```

```
Testing accuracy of Random Forest of our result data that combined the original data plut the generated data After cascading: 94.86967314853125%
Classification Report for RF of our result data that combined the original data plut the generated data After cascading
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       1.00      0.95      0.97      2417

    accuracy                           0.95      2417
   macro avg       0.50      0.47      0.49      2417
weighted avg       1.00      0.95      0.97      2417
```

```
[[   0    0]
 [ 124 2293]]
```



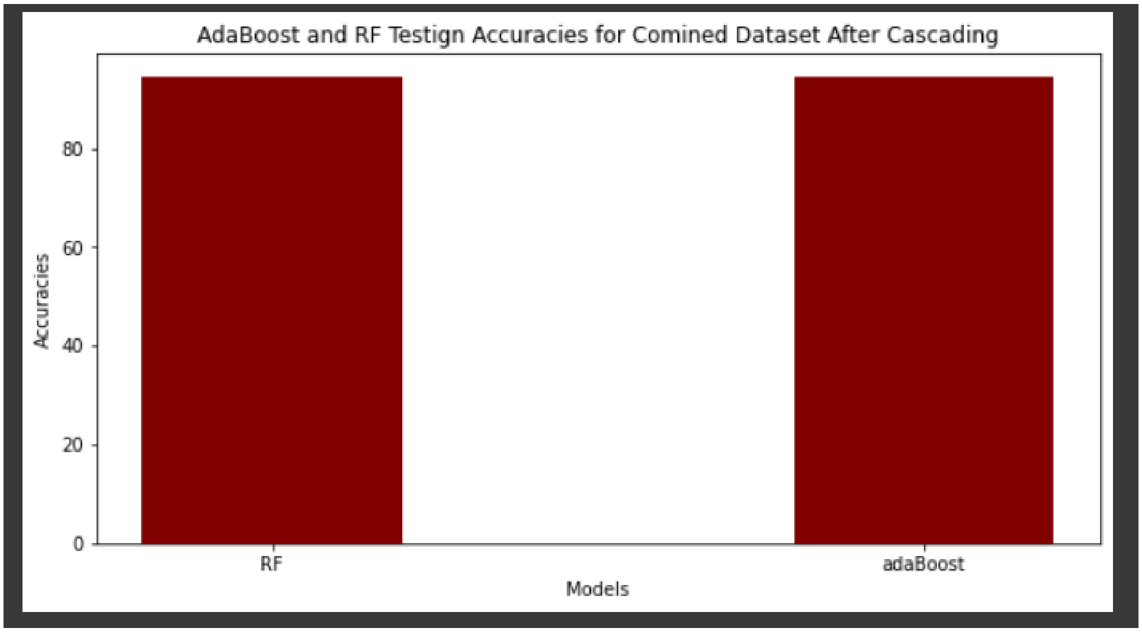
AdaBoost Classifier

```python
# Adaboost Testing accuracy of our result data that combined the original data plut the generated data After cascading
ypredAda_testing_conbination_Cascade =Adaclf.predict(X_test_conbination_Cascade)
Ada_test_acc_conbination_Cascade = accuracy_score(y_test_conbination_Cascade, ypredAda_testing_conbination_Cascade)
print(f'Testing accuracy of Adaboost of our result data that combined the original data plut the generated data After cascading:
{Ada_test_acc_conbination_Cascade*100}%')
print('Classification Report for Adaboost of our result data that combined the original data plut the generated data After cascading')
print(classification_report(y_test_conbination_Cascade, ypredAda_testing_conbination_Cascade))
getConfusionMatrix(Adaclf,X_test_conbination_Cascade,y_test_conbination_Cascade,
"Confusion Matrix for Testing Adaboost Model of our result data that combined the original data plut the generated data After cascading")
```

```
Testing accuracy of Adaboost of our result data that combined the original data plut the generated data After cascading: 94.82829954489036%
Classification Report for Adaboost of our result data that combined the original data plut the generated data After cascading
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       1.00      0.95      0.97      2417

    accuracy                           0.95      2417
   macro avg       0.50      0.47      0.49      2417
weighted avg       1.00      0.95      0.97      2417
```

```
[[   0    0]
 [ 125 2292]]
```

Present results comparison in the accuracy in bar chart:



The above graph shows that the accuracy of two models becomes well instead of the bad performance in Question [10] because of using the discriminator (cascading).

References:

[1] https://keras.io/examples/generative/conditional_gan/