ELG5142: Ubiquitous Sensing / Smart Cities

Group: 23

Prepared by:

Hossam Mahmoud (300327269)

Maryam Almashad (300327220)

Nada Abo-Elfotoh (300327211)

1. Overview

The main objective of this task to generate fakes tasks using CGAN. By comparing accuracy of three dataset using Ml Classifier RF and Adaboost .

1) Original Data set after split it into training dataset(80%)and test dataset (20%)
2) Mixed Data set with Fake tasks (Without Cascade Framework)
3) Dataset After Discriminator detect fake tasks (With Cascade Framework)

That will help MCS platform to verify fake Task Detection

2. Methodology

1. Import important libraries

Import Important Libraries

```
import subprocess

import self as self
import sns as sns
import  tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import voting as voting
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier, VotingClassifie
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
try :
    from imblearn.over_sampling import RandomOverSampler
except:
    print("installing over_sampling ... ")
    subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'imblearn'])
```

2. Split Original dataset and train Rf and Adaboost

```
##Implement  classic classifier Adaboot and RD

def models(model, x, y):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
    model = model.fit(x_train, y_train)
    y_train_pred = model.predict(x_train)
    y_test_pred = model.predict(x_test)
    accuracy_train = accuracy_score(y_train, y_train_pred)
    accuracy_test = accuracy_score(y_test, y_test_pred)
    report_test = classification_report(y_test, y_test_pred)
    return (model, y_train_pred, y_test_pred, accuracy_train, accuracy_test, report_test)

RF, AB = RandomForestClassifier(), AdaBoostClassifier()
#Rf
model_RF, y_train_pred_RF, y_test_pred_RF, accuracy_train_RF, accuracy_test_RF, report_RF = models(RF, x, y)
#AB
model_AB, y_train_pred_AB, y_test_pred_AB, accuracy_train_AB, accuracy_test_AB, report_AB = models(AB, x, y)
```

- **Rf accuracy**

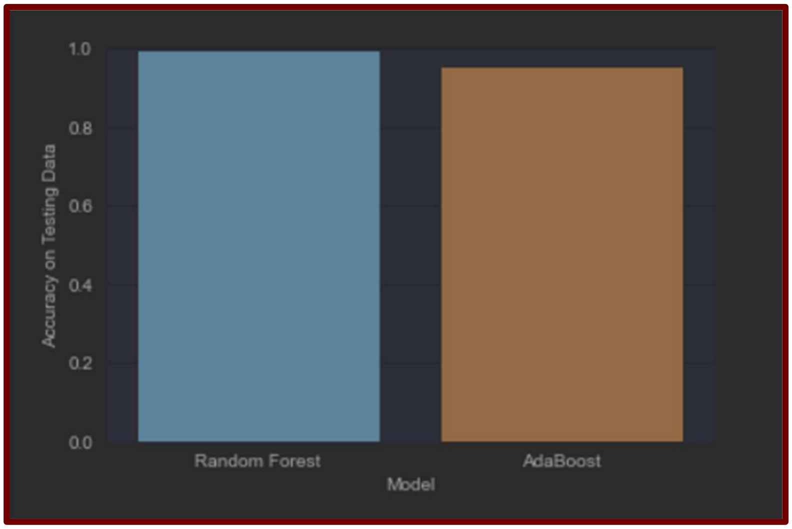
```
print(report_RF)

              precision    recall  f1-score   support

           0       1.00      0.97      0.99       354
           1       1.00      1.00      1.00      2543

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

accuracy_test_RF

1.0
```

- **AB accuracy**

```
print(report_AB)

              precision    recall  f1-score   support

           0       0.87      0.74      0.80       354
           1       0.96      0.98      0.97      2543

    accuracy                           0.95      2897
   macro avg       0.92      0.86      0.89      2897
weighted avg       0.95      0.95      0.95      2897

accuracy_test_AB

0.9540904383845358
```

- **Bar chart for both RF and AB**

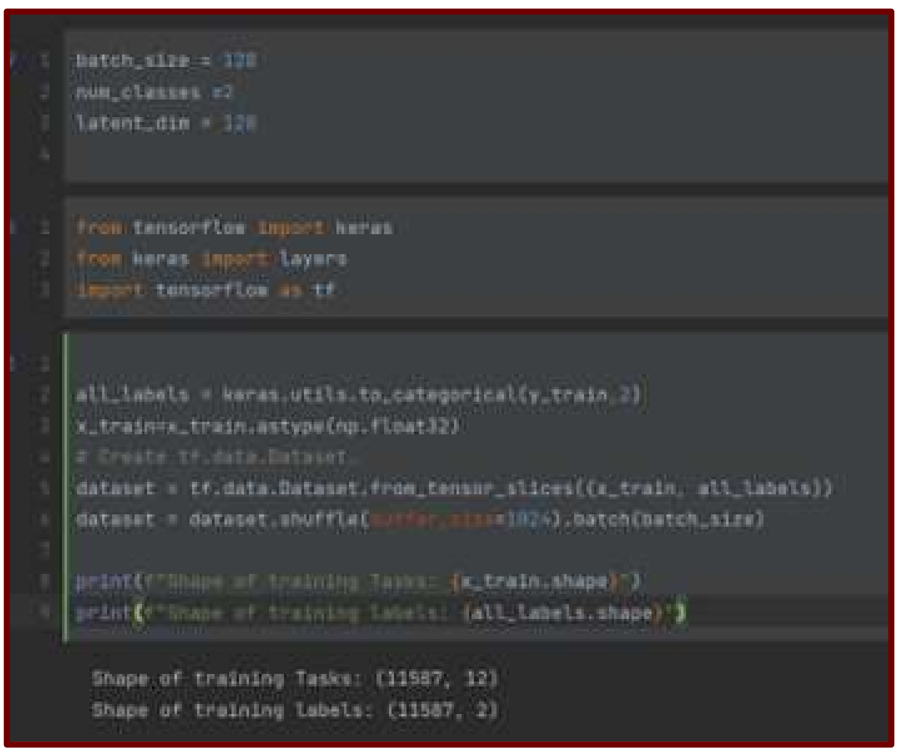
```
## Bar Chart for RF and AB

from sklearn.ensemble import VotingClassifier

df_2 = pd.DataFrame({'Model':['Random Forest', 'AdaBoost'],
'Accuracy on Training Data':[accuracy_train_RF, accuracy_train_AB,
                            ],
'Accuracy on Testing Data':[accuracy_test_RF, accuracy_test_AB,
                           ] ,
})

import seaborn as sns

#for testing data
bar_plot_test = sns.barplot(data = df_2, x = 'Model', y = 'Accuracy on Testing Data')
bar_plot_test.set_ylim(0, 1)
```



- **As we see the comparison Between RF and AdaBoost on Original Data set.**

3. Preparation for CGAN

➢ **Create constant and hyperparameter.**



```
batch_size = 128
num_classes =2
latent_dim = 128
```

```
from tensorflow import keras
from keras import layers
import tensorflow as tf
```

```

all_labels = keras.utils.to_categorical(y_train,2)
x_train=x_train.astype(np.float32)
# Create tf.data.Dataset.
dataset = tf.data.Dataset.from_tensor_slices((x_train, all_labels))
dataset = dataset.shuffle(buffer_size=1024).batch(batch_size)

print(f"Shape of training Tasks: {x_train.shape}")
print(f"Shape of training labels: {all_labels.shape}")
```

```
Shape of training Tasks: (11587, 12)
Shape of training labels: (11587, 2)
```

- **Calculating the number of input channel for the generator and discriminator.**

```
generator_in_channels = latent_dim + num_classes
discriminator_in_channels = 12 + num_classes
print(generator_in_channels, discriminator_in_channels)
```

```
130 14
```

- **Create Discriminator.**

```
discriminator = keras.Sequential(
    [
        keras.layers.Input(shape=(discriminator_in_channels,1)),
        layers.Conv1D(64, 3, strides=2, padding="same",),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv1D(128, 3, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.GlobalMaxPooling1D(),
        layers.Dense(1,activation='sigmoid'),
    ],
    name="discriminator",
)
discriminator.summary()
```

```
Model: "discriminator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_12 (Conv1D)          (None, 7, 64)             256

 leaky_re_lu_29 (LeakyReLU)  (None, 7, 64)             0

 conv1d_13 (Conv1D)          (None, 4, 128)            24704

 leaky_re_lu_30 (LeakyReLU)  (None, 4, 128)            0
```

➢ Create Generator.

```
generator = keras.Sequential(
    [
        keras.layers.InputLayer((generator_in_channels,)),
        layers.Dense(256),
        layers.LeakyReLU(alpha=0.2),
        layers.Dense(128),
        layers.LeakyReLU(alpha=0.2),
        layers.Dense(128,),
        layers.LeakyReLU(alpha=0.2),
        layers.Dense(12,),
    ],
    name="generator",
)
generator.summary()
```

```
Model: "generator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_19 (Dense)            (None, 256)               33536

 leaky_re_lu_31 (LeakyReLU)  (None, 256)               0

 dense_20 (Dense)            (None, 128)               32896

 leaky_re_lu_32 (LeakyReLU)  (None, 128)               0
```

➢ CGAN 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
```

```
# 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
)

# 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_tasks_and_labels = tf.concat([generated_Tasks, one_hot_labels],axis=1)
real_task_with_labels=tf.concat([real_Tasks,one_hot_labels],axis=1)

combined_tasks = tf.concat(
    [fake_tasks_and_labels, real_task_with_labels], axis=0
)

# Assemble labels discriminating real from fake tasks.
labels = tf.concat(
    [tf.zeros((batch_size, 1)), tf.ones((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 images".
misleading_labels = tf.ones((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_tasks_and_labels = tf.concat([fake_tasks, one_hot_labels], -1)
    predictions = self.discriminator(fake_tasks_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(),
        }
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=20)
```

```
91/91 [==============================] - 2s 9ms/step - g_loss: 0.7316 - d_loss: 0.6717
Epoch 2/20
91/91 [==============================] - 1s 9ms/step - g_loss: 1.1182 - d_loss: 0.6587
Epoch 3/20
91/91 [==============================] - 1s 9ms/step - g_loss: 0.6585 - d_loss: 0.7349
Epoch 4/20
91/91 [==============================] - 1s 9ms/step - g_loss: 0.6081 - d_loss: 0.7384
Epoch 5/20
91/91 [==============================] - 1s 9ms/step - g_loss: 0.6730 - d_loss: 0.6956
Epoch 6/20
91/91 [==============================] - 1s 9ms/step - g_loss: 0.6658 - d_loss: 0.6992
```

- **Dataset with fake tasks.**

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

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

# Sample noise for the interpolation.
noise = tf.random.normal(shape=(num_tasks, latent_dim))
noise_with_label=tf.concat([noise,keras.utils.to_categorical([0]*num_tasks,2)],1)

fake_tasks=trained_gen.predict(noise_with_label)
fake_tasks=pd.DataFrame(scaler.inverse_transform(fake_tasks),columns=df.columns[0:12])
fake_tasks
```

32/32 [==============================] - 0s 1ms/step

	ID	Latitude	Longitude	Day	Hour	Minute	Duration	RemainingTime	Resources	Coverage	OnPeakHours	GridNumber
0	1997.067261	45.485867	-75.183136	3.509515	13.716654	24.126377	43.870907	31.259037	5.830646	70.852638	0.167047	214820.890625
1	1925.019531	45.463028	-75.165688	4.073295	10.758926	19.898832	45.134293	24.398106	5.609287	64.971771	0.077406	215842.437500
2	2109.645752	45.466522	-75.187050	3.449309	13.422576	27.190350	38.353382	26.597759	5.490960	63.130047	0.169969	201926.125000
3	2216.646240	45.522606	-75.152603	4.166408	10.943135	15.141571	45.212379	18.648891	5.703578	71.478745	-0.159658	223745.406250
4	2592.276123	45.436474	-75.174400	2.964803	14.573986	27.297588	44.229679	28.196201	5.150286	67.107292	0.293332	191632.046875
5	2286.334961	45.447697	-75.170753	2.759243	14.104823	27.193169	39.529278	34.381466	5.370993	72.136246	0.480977	204095.406250
6	2653.166992	45.448437	-75.188255	2.729698	15.478775	29.220001	40.784500	33.863560	5.922963	68.834732	0.421625	181707.156250
7	2835.236084	45.470509	-75.183586	3.194003	11.807152	21.332096	41.153904	23.671106	5.726243	72.015976	0.116603	194649.531250

4. Mixed dataset Without Discriminator ("Without Cascade Framework").

- **Create constant and hyperparameter.**

```python
New_x_Test = np.concatenate((fake_tasks,x_test) , axis = 0)
New_y_Test =  np.concatenate((y_test.values,np.zeros(num_tasks)), axis = 0).astype("int")
New_x_Test
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	1997.067261	45.485867	-75.183136	3.509515	13.716654	24.126377	43.870907	31.259037	5.830646	70.852638	0.167047	214820.890625
1	1925.019531	45.463028	-75.165688	4.073295	10.758926	19.898832	45.134293	24.398106	5.609287	64.971771	0.077406	215842.437500
2	2109.645752	45.466522	-75.187050	3.449309	13.422576	27.190350	38.353382	26.597759	5.490960	63.130047	0.169969	201926.125000
3	2216.646240	45.522606	-75.152603	4.166408	10.943135	15.141571	45.212379	18.648891	5.703578	71.478745	-0.159658	223745.406250
4	2592.276123	45.436474	-75.174400	2.964803	14.573986	27.297588	44.229679	28.196201	5.150286	67.107292	0.293332	191632.046875
5	2286.334961	45.447697	-75.170753	2.759243	14.104823	27.193169	39.529278	34.381466	5.370993	72.136246	0.480977	204095.406250
6	2653.166992	45.448437	-75.188255	2.729698	15.478775	29.220001	40.784500	33.863560	5.922963	68.834732	0.421625	181707.156250
7	2835.236084	45.470509	-75.183586	3.194003	11.807152	21.332096	41.153904	23.671106	5.726243	72.015976	0.116603	194649.531250

- **Ml Classifier Model (RF and Adaboost).**

```python
def models(model, x, y):
    y_test_pred_mix = model.predict(x)
    accuracy_test = accuracy_score(y, y_test_pred_mix)
    report_test = classification_report(y,y_test_pred_mix)
    return y_test_pred_mix, accuracy_test, report_test
```

```python

#Rf
y_test_pred_RF_mix, accuracy_test_RF_mix, report_RF_mix = models(model_RF, New_x_Test, New_y_Test)
#AB
y_test_pred_AB_mix, accuracy_test_AB_mix, report_AB_mix = models(model_AB,New_x_Test, New_y_Test)
```

- **RF accuracy**

```
print(report_RF_mix)
```

```
              precision    recall  f1-score   support

           0       0.07      0.00      0.00      1354
           1       0.65      0.99      0.79      2543

    accuracy                           0.65      3897
   macro avg       0.36      0.50      0.39      3897
weighted avg       0.45      0.65      0.51      3897
```

```
accuracy_test_RF_mix
```

```
0.649217346676931
```

- **Adaboost Accuracy**

```
print(report_AB_mix)
```

```
              precision    recall  f1-score   support

           0       0.05      0.00      0.00      1354
           1       0.65      0.99      0.79      2543

    accuracy                           0.65      3897
   macro avg       0.35      0.50      0.39      3897
weighted avg       0.44      0.65      0.51      3897
```

```
accuracy_test_AB_mix
```

```
0.6481909160892995
```

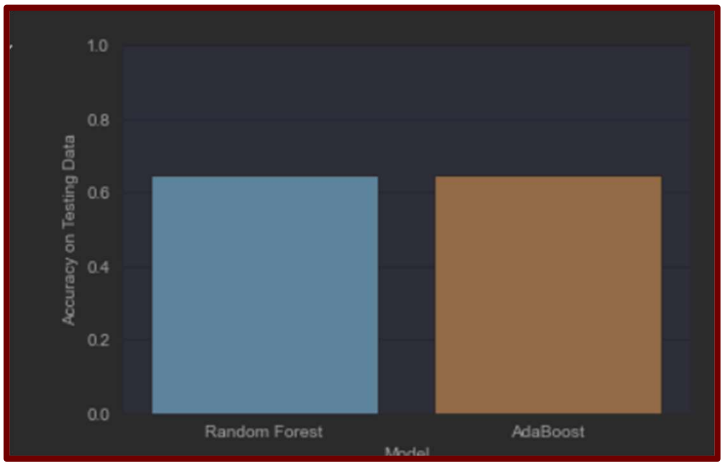
- **As we see the accuracy with fake task lower that Original Dataset.**

➢ **Bar Chart**

```
df_new = pd.DataFrame({'Model':['Random Forest', 'AdaBoost',],
'Accuracy on Testing Data':[accuracy_test_RF_mix, accuracy_test_AB_mix,
                              ] ,
})
```

```
#for testing data
bar_plot_test = sns.barplot(data = df_new, x = 'Model', y = 'Accuracy on Testing Data')
bar_plot_test.set_ylim(0,1)
```

```
(0.0, 1.0)
```



- **After we create mixed test data set with fake tasks fitted it with Random Forest and Adaboost and give it lower accuracy than the original test data set.**

5. **With Discriminator/ cascade framework without Fake tasks**

```
trained_desc=cond_gan.discriminator
new_labels=keras.utils.to_categorical(New_y_Test,2)
New_x_Test_with_labels=tf.concat([New_x_Test,new_labels],axis=1)
y_pred_new=trained_desc.predict(New_x_Test_with_labels)
y_pred_new=np.apply_along_axis(lambda x:1 if x >0.5 else 0 ,axis=1,arr=y_pred_new)
y_pred_new_df=pd.DataFrame(y_pred_new)
y_pred_new_df
```

122/122 [==============================] - 0s 1ms/step

	0
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1

3897 rows × 1 columns Open in new tab

- **After discriminator able to distinguish fake task and Real task we choose Real task only.**

➢ **Real Tasks**

```
Real_tasks=y_pred_new_df[y_pred_new_df[0]==1]
Real_tasks
```

	0
39	1
40	1
41	1
42	1
43	1
44	1
45	1
46	1

➢ Cascade framework Data set.

```
New_x_Test_df=pd.DataFrame(New_x_Test)
New_x_Test_df
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	1997.067261	45.485867	-75.183136	3.509515	13.716654	24.126377	43.870907	31.259037	5.830646	70.852638	0.167047	214820.890625
1	1925.019531	45.463028	-75.165688	4.073295	10.758926	19.898832	45.134293	24.398106	5.609287	64.971771	0.077406	215842.437500
2	2109.645752	45.466522	-75.187050	3.449309	13.422576	27.190350	38.353382	26.597759	5.490960	63.130047	0.169969	201926.125000
3	2216.646240	45.522606	-75.152603	4.166408	10.943135	15.141571	45.212379	18.648891	5.703578	71.478745	-0.159658	223745.406250
4	2592.276123	45.436474	-75.174400	2.964803	14.573986	27.297588	44.229679	28.196201	5.150286	67.107292	0.293332	191632.046875
5	2286.334961	45.447697	-75.170753	2.759243	14.104823	27.193169	39.529278	34.381466	5.370993	72.136246	0.480977	204095.406250
6	2653.166992	45.448437	-75.188255	2.729698	15.478775	29.220001	40.784500	33.863560	5.922963	68.834732	0.421625	181707.156250
7	2835.236084	45.470509	-75.183586	3.194003	11.807152	21.332096	41.153904	23.671106	5.726243	72.015976	0.116603	194649.531250

➢ Features.

```
Real_Features=New_x_Test_df.iloc[Real_tasks.index,:]
Real_Features
```

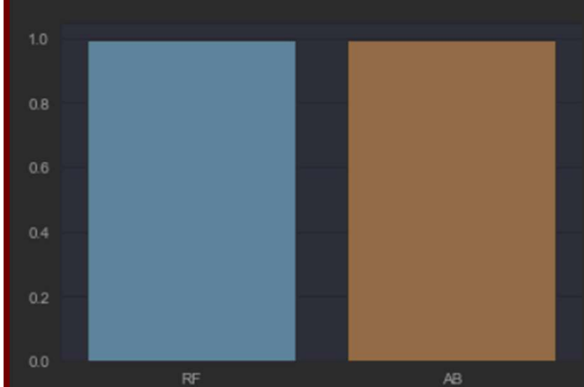
	0	1	2	3	4	5	6	7	8	9	10	11
0	1997.067261	45.485867	-75.183136	3.509515	13.716654	24.126377	43.870907	31.259037	5.830646	70.852638	0.167047	214820.890625
1	1925.019531	45.463028	-75.165688	4.073295	10.758926	19.898832	45.134293	24.398106	5.609287	64.971771	0.077406	215842.437500
2	2109.645752	45.466522	-75.187050	3.449309	13.422576	27.190350	38.353382	26.597759	5.490960	63.130047	0.169969	201926.125000
3	2216.646240	45.522606	-75.152603	4.166408	10.943135	15.141571	45.212379	18.648891	5.703578	71.478745	-0.159658	223745.406250
4	2592.276123	45.436474	-75.174400	2.964803	14.573986	27.297588	44.229679	28.196201	5.150286	67.107292	0.293332	191632.046875
5	2286.334961	45.447697	-75.170753	2.759243	14.104823	27.193169	39.529278	34.381466	5.370993	72.136246	0.480977	204095.406250
6	2653.166992	45.448437	-75.188255	2.729698	15.478775	29.220001	40.784500	33.863560	5.922963	68.834732	0.421625	181707.156250
7	2835.236084	45.470509	-75.183586	3.194003	11.807152	21.332096	41.153904	23.671106	5.726243	72.015976	0.116603	194649.531250

➢ RF and AB

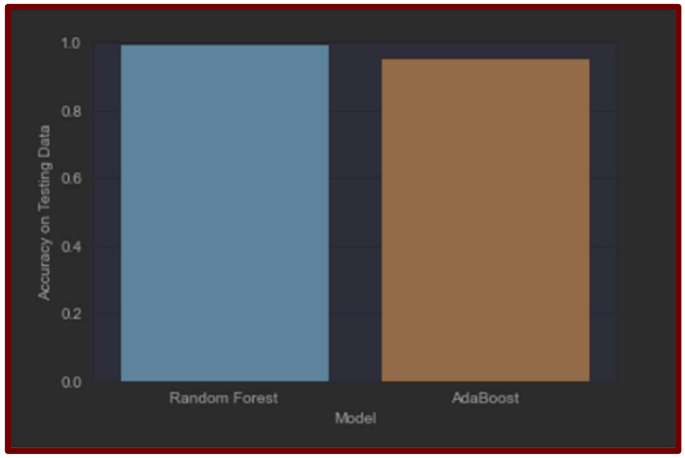
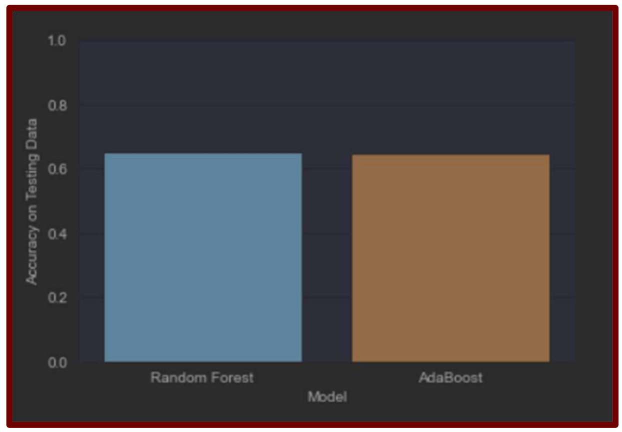
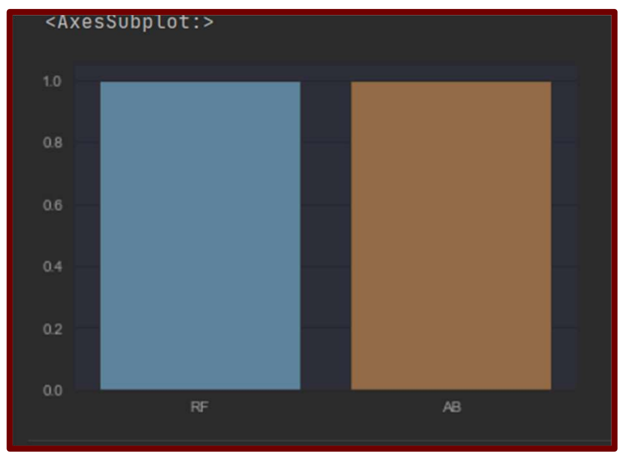
```
Cascade_RF=model_RF.score(Real_Features,Real_tasks)

C:\Users\nadai\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(

Cascade_AB=model_AB.score(Real_Features,Real_tasks)

C:\Users\nadai\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but
AdaBoostClassifier was fitted with feature names
  warnings.warn(
```

➢ Bar Chart

```
sns.barplot(x=["RF","AB"],y=[Cascade_RF,Cascade_AB])

<AxesSubplot:>
```



- **Comparison Between three data set:**

Original Dataset	
Without Cascade Framework	
Cascade Framework	

✓ **Due to these graphs we conclude that Discriminator able to remove fake tasks and this help two classifier model (Rf and AB) to classify correctly with high accuracy.**