uOttawa

ELG 5142:
Ubiquitous Sensing / Smart Cities
Term Project

Dr. Burak Kantarci

GROUP 14

Table of Contents

Part 1 3
A) 4
B) 6
C) 8
Part 2 10
Part 3 11
Appendix 12
Implementation of CGAN: 12
Applying provided training dataset to CGAN: 17
Generating synthetic fake tasks via generator: 18
Testing the discriminator: 18

Table of Figures

Figure 1 Legitimacy counts 3
Figure 2 Data Split 3
Figure 3 RF classification report and confusion matrix 5
Figure 4 Adaboost classification report and confusion matrix 5
Figure 5 RF and adaboost comparison 5
Figure 6 Mixed data' shapes 6
Figure 7 Adaboost classification report and confusion matrix (mixed data) 6
Figure 8 RF classification report and confusion matrix (mixed data) 6
Figure 9 RF and adaboost comparison (mixed data) 7
Figure 10 Discriminator's classification report and confusion matrix 7
Figure 11 Adaboost classification report and confusion matrix (cascaded) 9
Figure 12 RF classification report and confusion matrix (cascaded) 9
Figure 13 RF and adaboost comparison (Cascaded) 9
Figure 14 RF and adaboost comparison (original data) 10
Figure 15 RF and adaboost comparison (mixed data) 10
Figure 16 RF and adaboost comparison (cascaded) 10
Figure 17 Discriminator and generator losses in training 11
Figure 18 Discriminator model architecture 13
Figure 19 Generator model architecture 14
Figure 20 Discriminator classification report and confusion matrix 19

Part 1

Checking the number of legitimate tasks:

```
df['Ligitimacy'].value_counts() #Check the legitimacy counts
```

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

Figure 1 Legitimacy counts

Dropping the ID column:

```
df=df.drop('ID',axis=1) #drop the ID column
```

Applying Scaling:

```
#Scaling
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
scaler = StandardScaler()
X=scaler.fit_transform(X)
```

Splitting the dataset:

```
#Splitting the dataset 80% for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 random_state=42, stratify=y)
print(X_train.shape)
print(X_test.shape)
```

```
(11587, 11)
(2897, 11)
```

Figure 2 Data Split

Function to test and compare:

It outputs the classification reports and confusion matrices for models and comparison between them regarding their accuracies.

```
def test_and_compare(models,x_test,y_test,labels1 = ['Fake', 'Legitimate
' ],title='campare between models'):
  models_name=[]
  Accuracies=[]
  for model in models:
    print("Evaluated {} model".format(model))
    models_name.append(str(model))
    y_pred=model.predict(x_test)
    print(classification_report(y_test, y_pred))
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=la
bels1)
    disp.plot()
    plt.show()
    Accuracies.append(accuracy_score(y_test, y_pred)*100)
  plt.figure(figsize=(20,10))
  ax=sns.barplot(x=models_name, y=Accuracies)
  ax.set_title(title, size=26)
  counter=0
  for value in Accuracies:

    v=str(np.round(value,2))+' %'

    ax.text(counter,value,v, color='black', ha="center")
    counter+=1
  mi=min(Accuracies)
  ma=max(Accuracies)
  range=ma-mi

  plt.ylim(mi-(range), ma+range)
  plt.show()
```

A)

Random Forest model:

```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

Adaboost model:

```
adaboost = AdaBoostClassifier(n_estimators=100, random_state=0)
adaboost.fit(X_train, y_train)
```

Calling test_and_compare function:

```
test_and_compare([rf,adaboost],X_test,y_test)
```

```
Evaluated RandomForestClassifier() model
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       379
           1       0.99      1.00      1.00      2518

    accuracy                           1.00      2897
   macro avg       1.00      0.98      0.99      2897
weighted avg       1.00      1.00      1.00      2897
```

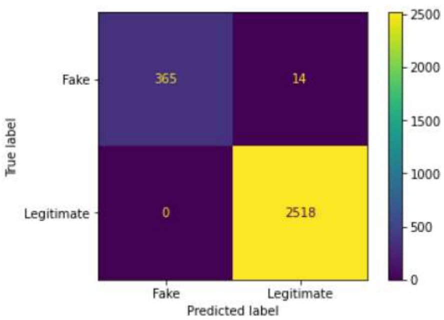

Figure 3 RF classification report and confusion matrix

```
Evaluated AdaBoostClassifier(n_estimators=100, random_state=0) model
              precision    recall  f1-score   support

           0       0.89      0.82      0.85       379
           1       0.97      0.98      0.98      2518

    accuracy                           0.96      2897
   macro avg       0.93      0.90      0.92      2897
weighted avg       0.96      0.96      0.96      2897
```

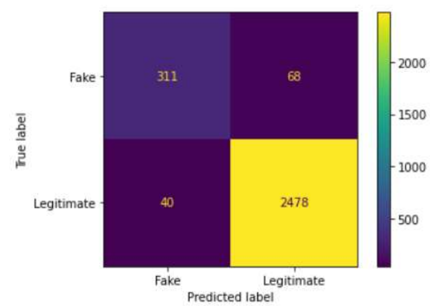

Figure 4 Adaboost classification report and confusion matrix

Models' Accuracies comparison:

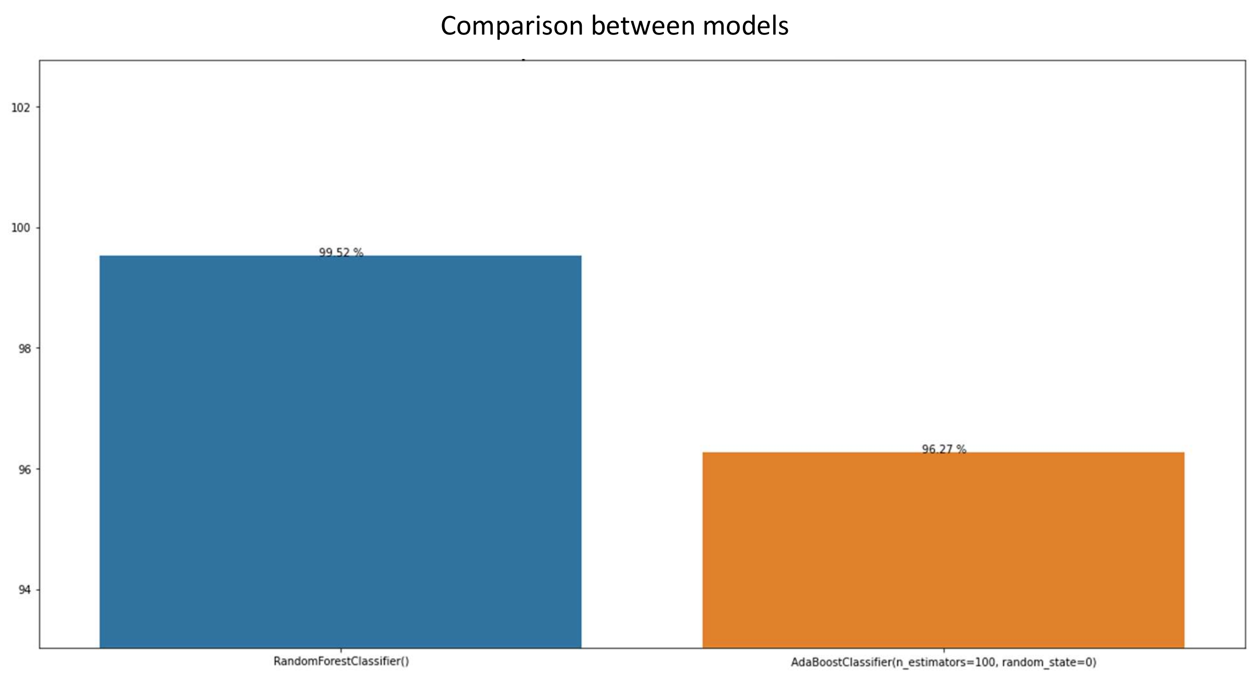

Figure 5 RF and adaboost comparison

B)

After applying CGAN and generating synthetic fake tasks, also the size of synthetic fake tasks equals the size of test dataset, these tasks were mixed with the original test data as follows:

```
mixed_data=np.concatenate((np.array(X_test), fake.reshape(-1,11)), axis=0)
mixed_y=np.concatenate((np.array(y_test), [0]*y_test.shape[0]), axis=0)
```

```
mixed_data.shape
```

```
mixed_y.shape
```

```
1 mixed_data.shape

(5794, 11)

1 mixed_y.shape

(5794,)
```

Figure 6 Mixed data' shapes

Calling test_and_compare function (after data mixture):

```
test_and_compare([rf,adaboost],mixed_data,mixed_y)
```

```
Evaluated RandomForestClassifier() model
              precision    recall  f1-score   support

           0       1.00      0.58      0.74      3276
           1       0.65      1.00      0.79      2518

    accuracy                           0.76      5794
   macro avg       0.82      0.79      0.76      5794
weighted avg       0.85      0.76      0.76      5794
```

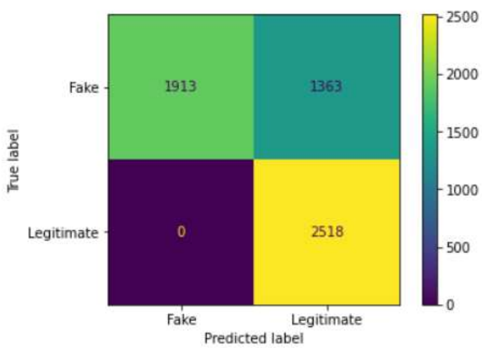

Figure 8 RF classification report and confusion matrix (mixed data)

```
Evaluated AdaBoostClassifier(n_estimators=100, random_state=0) model
              precision    recall  f1-score   support

           0       0.89      0.09      0.17      3276
           1       0.46      0.98      0.62      2518

    accuracy                           0.48      5794
   macro avg       0.67      0.54      0.40      5794
weighted avg       0.70      0.48      0.37      5794
```

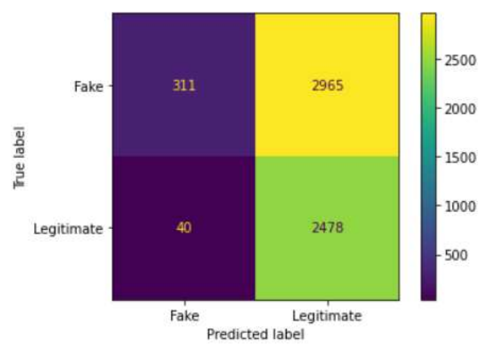

Figure 7 Adaboost classification report and confusion matrix (mixed data)

Models' Accuracies comparison:

Comparison between models

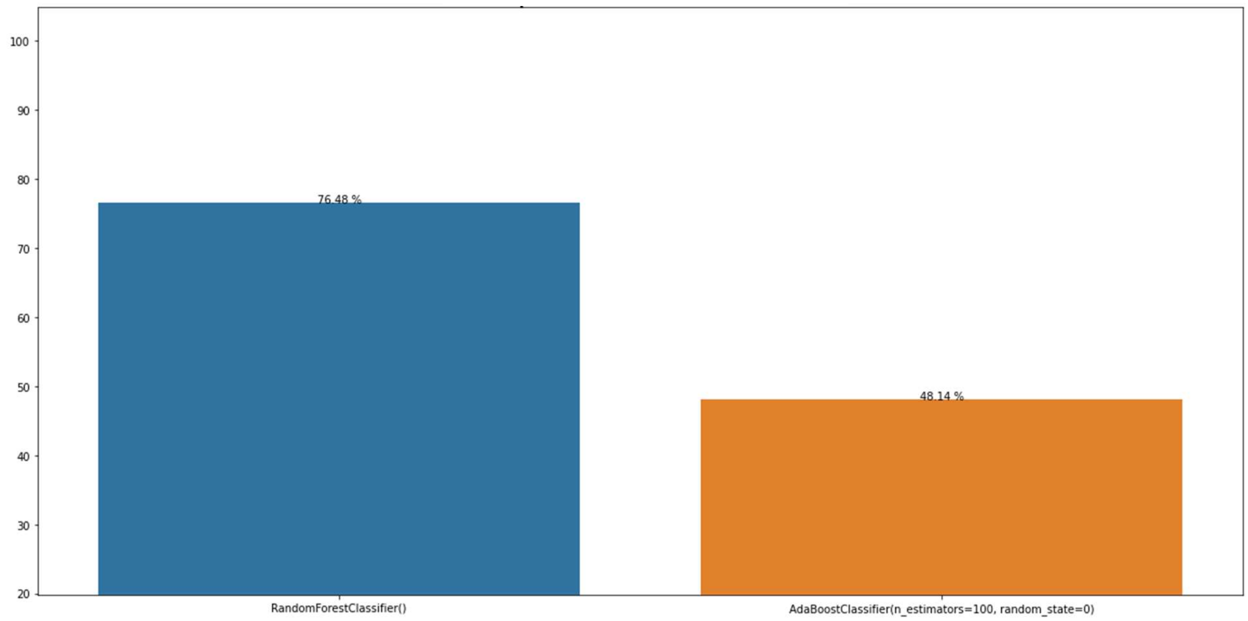

Figure 9 RF and adaboost comparison (mixed data)

N.B: Later on, the discriminator was tested using training data, testing data and generated tasks and resulted this classification report and confusion matrix:

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2897
           1       1.00      1.00      1.00     14484

    accuracy                           1.00     17381
   macro avg       1.00      1.00      1.00     17381
weighted avg       1.00      1.00      1.00     17381
```

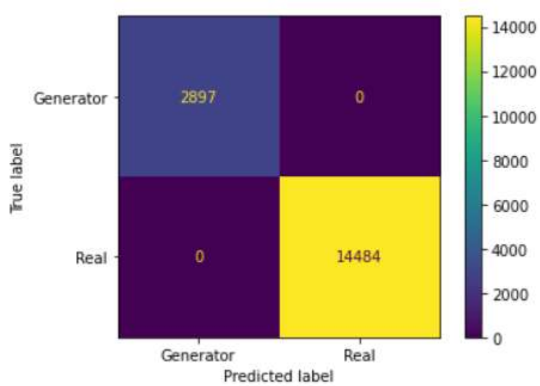

Figure 10 Discriminator's classification report and confusion matrix

C)

Cascade framework test and compare function:

It has the same outputs as “test_and_compare” function but for the cascaded frameworks.

```
def cascade_detection_framework(models,trained_dic,X,Y):
  rows = np.reshape(X, (-1, 11)).astype('float32')
  rows = np.array(rows).reshape(-1,11,1)
  all_labels = keras.utils.to_categorical(Y, 2)


  one_hot_labels = all_labels[:, None, None]

  one_hot_labels = tf.repeat(
      one_hot_labels, repeats=[row_size]
  )

  one_hot_labels = tf.reshape(
      one_hot_labels, (-1, row_size, n_classes)
  )
  # N X 11 X 2
  data_and_labels = tf.concat([rows, one_hot_labels], -1)
  fake_test=trained_dic.predict(data_and_labels)

  y_pred=np.argmax(fake_test,axis=1)
  y_pred=np.array([ 1 if x>0.5 else 0 for x in fake_test])
  x_real=X[y_pred==1]
  y_real=Y[y_pred==1]

  test_and_compare(models,x_real,y_real)
```

Calling cascade_detection_framework function (while using the discriminator):

```
trained_dic=cond_gan.discriminator
cascade_detection_framework([rf,adaboost],trained_dic,mixed_data,mixed_y)
```

```
Evaluated RandomForestClassifier() model
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       379
           1       0.99      1.00      1.00      2518

    accuracy                           1.00      2897
   macro avg       1.00      0.98      0.99      2897
weighted avg       1.00      1.00      1.00      2897
```

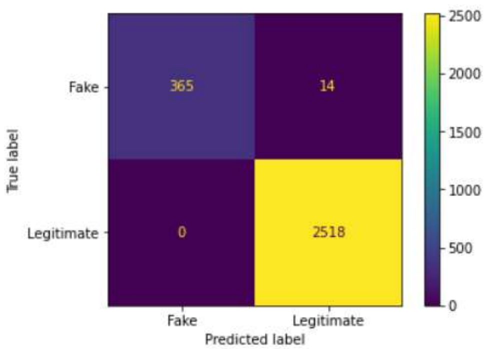

Figure 12 RF classification report and confusion matrix (cascaded)

```
Evaluated AdaBoostClassifier(n_estimators=100, random_state=0) model
              precision    recall  f1-score   support

           0       0.89      0.82      0.85       379
           1       0.97      0.98      0.98      2518

    accuracy                           0.96      2897
   macro avg       0.93      0.90      0.92      2897
weighted avg       0.96      0.96      0.96      2897
```

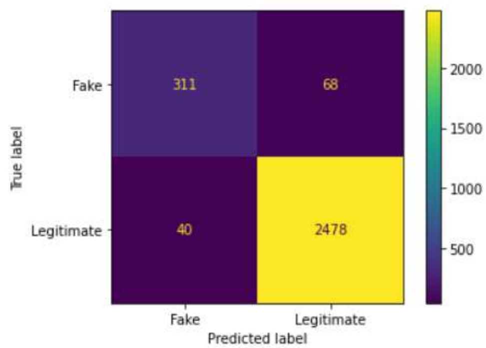

Figure 11 Adaboost classification report and confusion matrix (cascaded)

Models' Accuracies comparison:

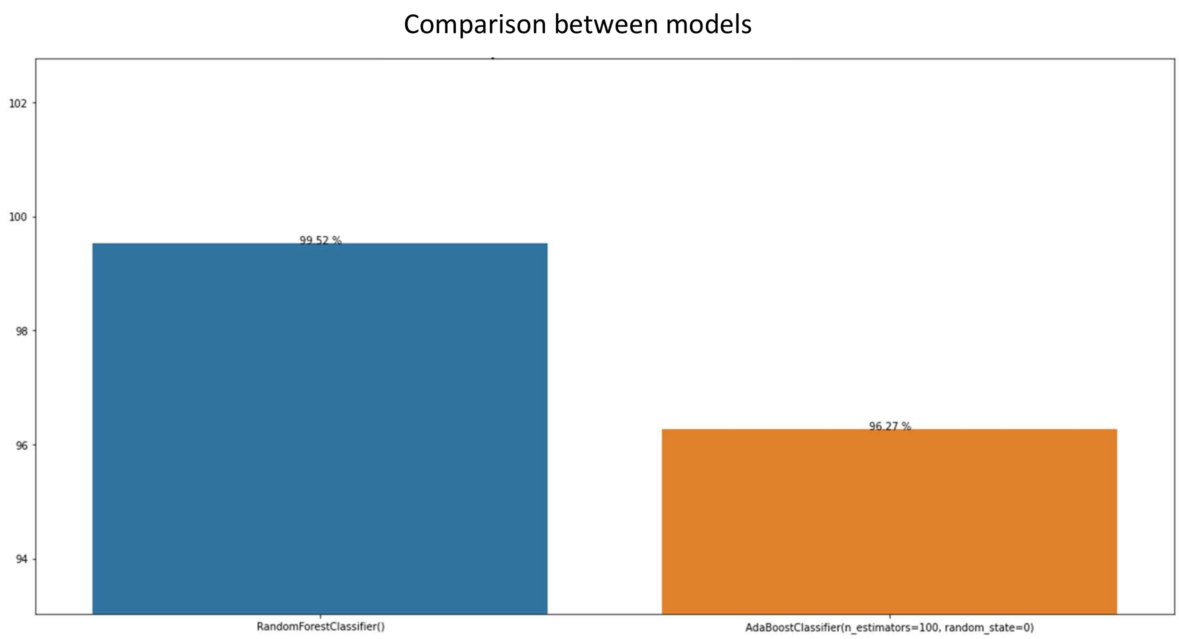

Figure 13 RF and adaboost comparison (Cascaded)

Part 2

As previously mentioned:

Models' Accuracies comparison

(Original data):

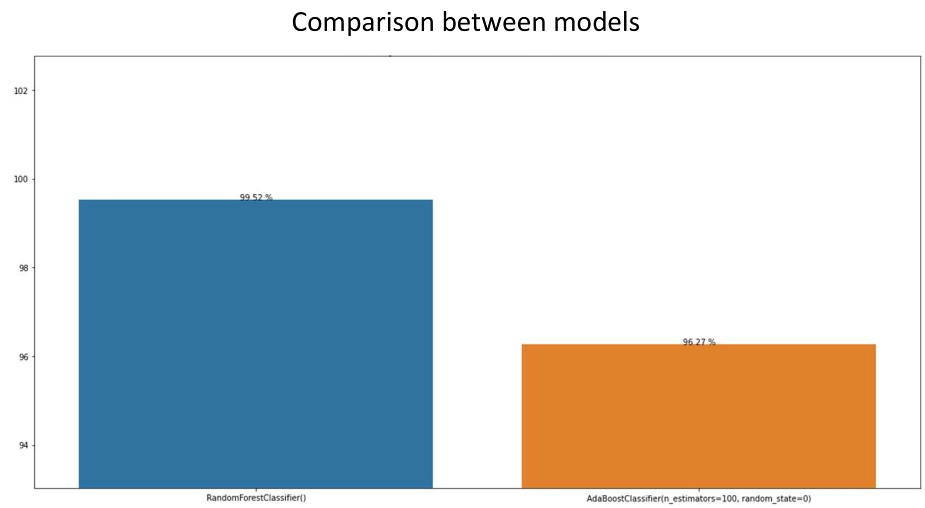

Figure 14 RF and adaboost comparison (original data)

Models' Accuracies comparison

(Mixed data):

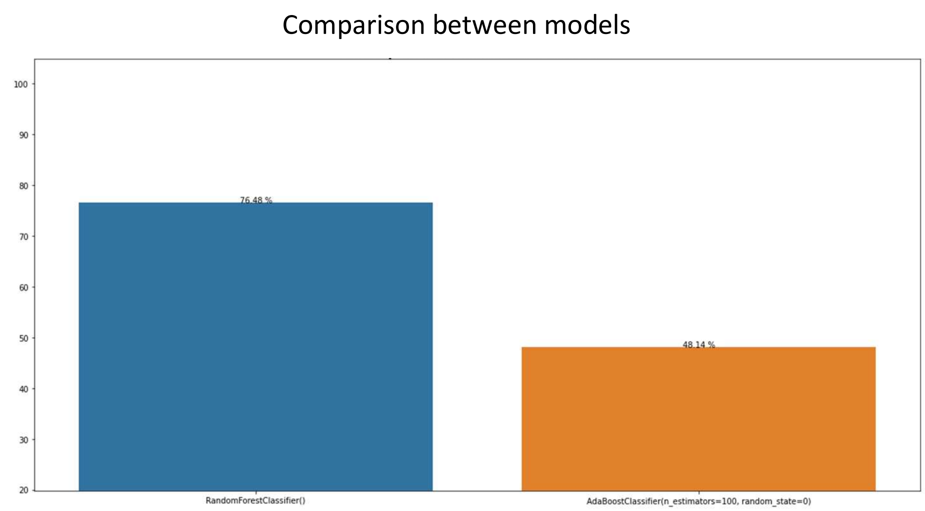

Figure 15 RF and adaboost comparison (mixed data)

Models' Accuracies comparison

(Cascaded):

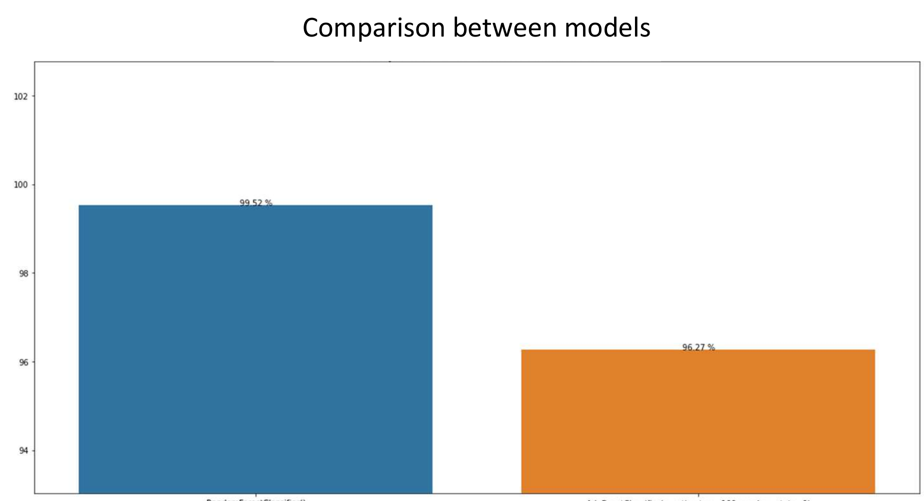

Figure 16 RF and adaboost comparison (cascaded)

Part 3

Generally, after training the classic machine learning classifiers, their accuracies were high enough at 99.52% and 96.27% for random forest and adaboost respectively and they could specify the legitimate from fake tasks. Secondly, after applying the CGAN and generating fake tasks, then, mixing these generated fake tasks with the original data, the accuracies have decreased to 76.48% for random forest and 48.14% for adaboost. Nevertheless, after using the discriminator in CGAN to classify the generated and real tasks to filter the generated ones out and send the real tasks to the machine learning models. While testing the discriminator, it showed accuracy of 100% in classifying the generated and real tasks because, as observed, the generator is considered weak with respect to the discriminator. Lastly, the mixed test dataset was used for cascaded frameworks for both of random forest and adaboost and they both showed 99.52% and 96.27% for random forest and adaboost respectively same as in “A” because the discriminator has the accuracy of 100% so it does the same exact classification, also the following figure shows that the loss of the discriminator (d_loss) in training was much less than that of generator (g_loss).

```
Epoch 3/20
182/182 [==============================] - 5s 26ms/step - g_loss: 11.1259 - d_loss: 2.9026e-05
Epoch 4/20
182/182 [==============================] - 4s 24ms/step - g_loss: 11.8421 - d_loss: 1.5124e-05
Epoch 5/20
182/182 [==============================] - 4s 23ms/step - g_loss: 13.2368 - d_loss: 6.9236e-06
Epoch 6/20
182/182 [==============================] - 4s 23ms/step - g_loss: 15.4571 - d_loss: 2.5487e-06
Epoch 7/20
182/182 [==============================] - 4s 24ms/step - g_loss: 14.7519 - d_loss: 3.6103e-06
Epoch 8/20
182/182 [==============================] - 4s 23ms/step - g_loss: 14.8254 - d_loss: 3.0067e-06
Epoch 9/20
182/182 [==============================] - 4s 21ms/step - g_loss: 15.0579 - d_loss: 2.2230e-06
Epoch 10/20
182/182 [==============================] - 5s 26ms/step - g_loss: 15.2306 - d_loss: 1.8268e-06
Epoch 11/20
182/182 [==============================] - 3s 15ms/step - g_loss: 15.1320 - d_loss: 1.7388e-06
```

Figure 17 Discriminator and generator losses in training

Appendix

Implementation of CGAN:

```
n_classes=2
batch_size = 64
num_channels = 1
row_size = 11
latent_dim = 128
dis_input=num_channels+n_classes
generator_in_channels=latent_dim+n_classes
```

```
discriminator = keras.Sequential(
    [
        keras.layers.InputLayer((11,dis_input)),
        keras.layers.Flatten(),
        #layers.Conv1D(64, 3, strides=2, padding="same"),
        #layers.LeakyReLU(alpha=0.2),
        #layers.GlobalMaxPooling1D(),

        Dense(128, activation='relu'),
        Dense(128, activation='relu'),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ],
    name="discriminator",
)

# Create the generator.
generator = keras.Sequential(
    [
        keras.layers.InputLayer((generator_in_channels)),
        layers.Dense(11 * generator_in_channels),
        layers.LeakyReLU(alpha=0.2),
        layers.Reshape((11, generator_in_channels)),
        layers.Dense(128),
        layers.LeakyReLU(alpha=0.2),
        layers.Dense(64),
        layers.LeakyReLU(alpha=0.2),
        layers.Dense(32),
        layers.LeakyReLU(alpha=0.2),

        layers.Dense(1, activation="sigmoid"),
        layers.Reshape((11, 1)),
    ],
    name="generator",
)
```

Discriminator model architecture:

```
plot_model(discriminator, to_file='model_plot.png', show_shapes=True, sh
ow_layer_names=True)
```

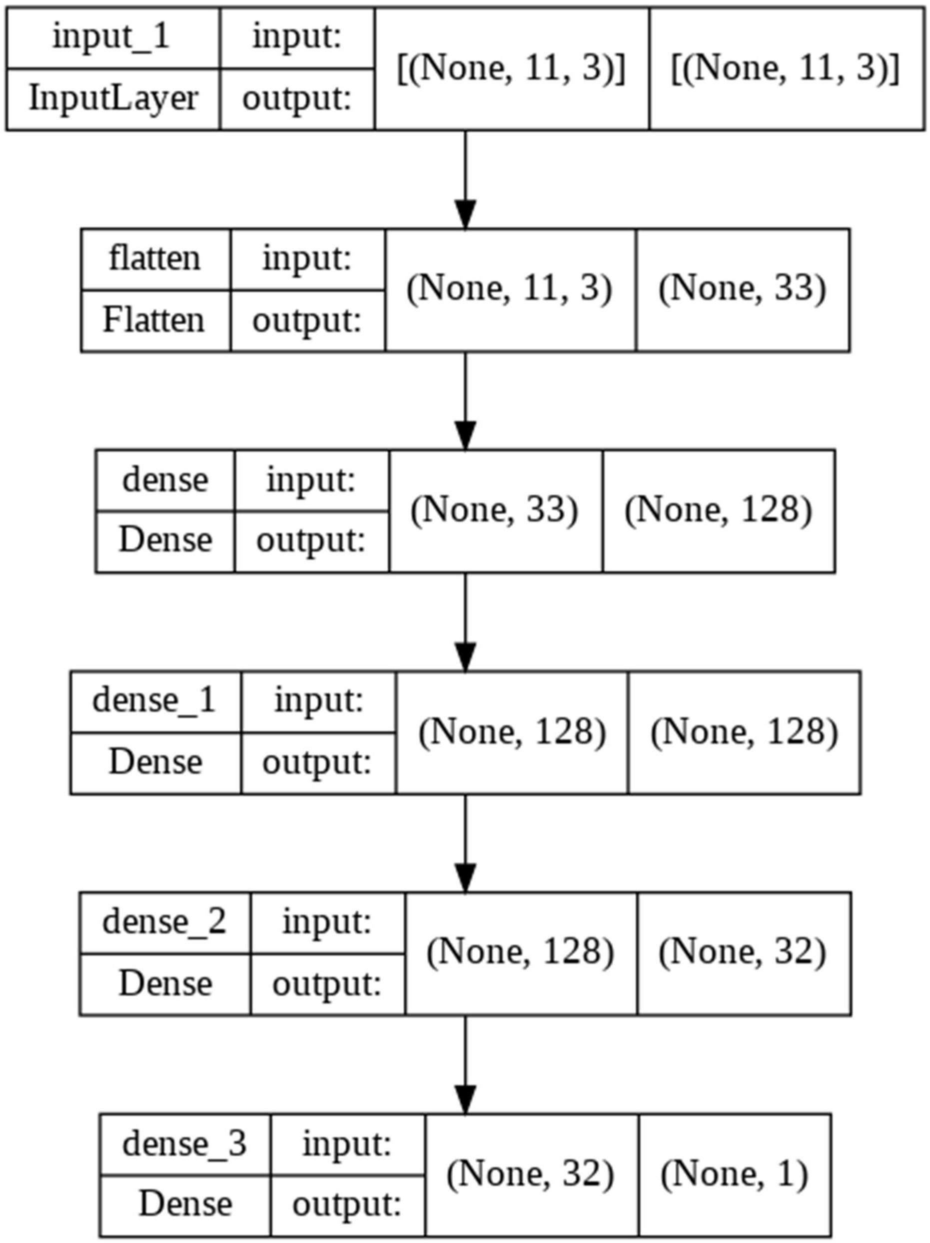

Figure 18 Discriminator model architecture

Generator model architecture:

```
plot_model(generator, to_file='model_plot.png', show_shapes=True, show_l
ayer_names=True)
```

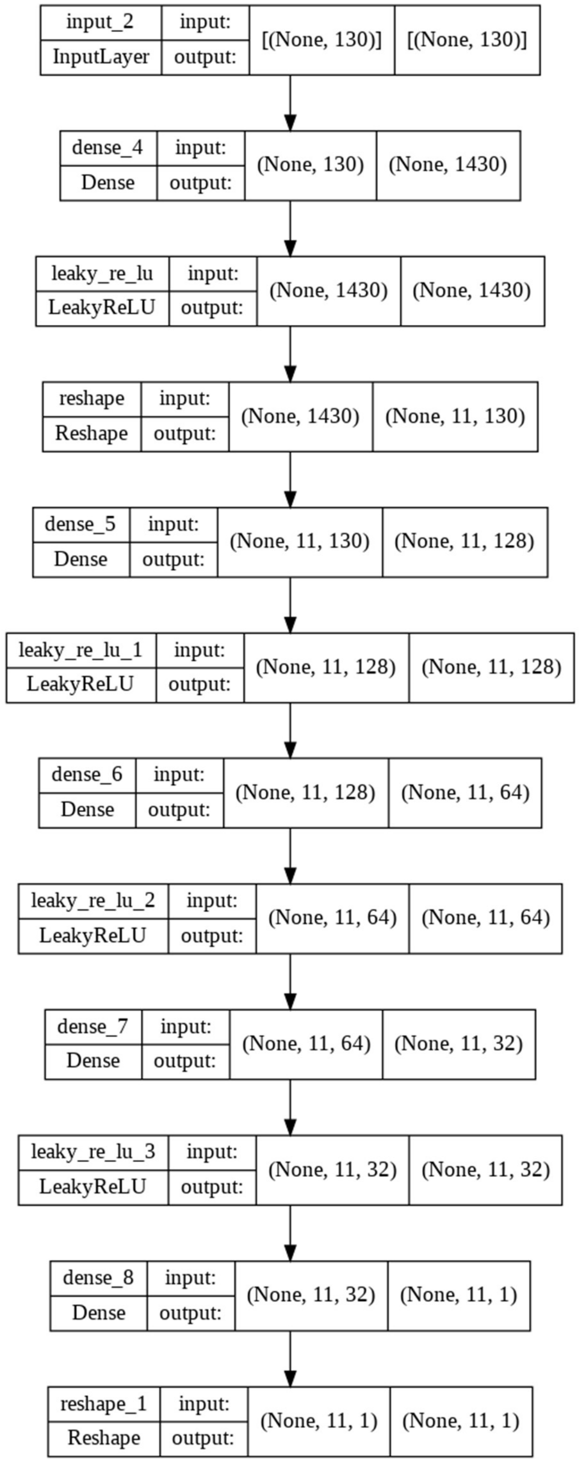

Figure 19 Generator model architecture

Class of CGAN:

```
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):

        real_data, one_hot_labels = data

        row_one_hot_labels = one_hot_labels[:, None, None]
        row_one_hot_labels = tf.repeat(
            row_one_hot_labels, repeats=[row_size]
        )
        row_one_hot_labels = tf.reshape(
            row_one_hot_labels, (-1, row_size, n_classes)
        )
        #    nx11x2
        batch_size = tf.shape(real_data)[0]
        #    n
        random_latent_vectors = tf.random.normal(shape=(batch_size, self
.latent_dim))
        #    nx128


        random_vector_labels = tf.concat(
            [random_latent_vectors, one_hot_labels], axis=1
        )
```

```
        #  nx128 + nx2= nx130

        generated_row = self.generator(random_vector_labels)
        #   nx11X1

        fake_and_labels = tf.concat([generated_row, row_one_hot_labels],
 -1)


        # nx11x1 +nx11x2 =nx11x3
        real_and_labels = tf.concat([real_data, row_one_hot_labels], -1)
        # nx11x1 +nx11x2 =nx11x3
        combined_data = tf.concat(
            [fake_and_labels, real_and_labels], axis=0
        )
        # nx11x3
        labels = tf.concat(
            [tf.zeros((batch_size,1)),tf.ones((batch_size,1))], axis=0
        )
        # 2nx2
        # 0 for generator and 1 for real
        with tf.GradientTape() as tape:
            predictions = self.discriminator(combined_data)
            d_loss = self.loss_fn(labels, predictions)
        # train discriminator

        grads = tape.gradient(d_loss, self.discriminator.trainable_weigh
ts)
        self.d_optimizer.apply_gradients(
            zip(grads, self.discriminator.trainable_weights)
        )


        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
        )

        #nx130

        misleading_labels = tf.ones((batch_size,1))
        # nx2 all ones
        with tf.GradientTape() as tape:
            fake_data = self.generator(random_vector_labels)
```

```
            fake_and_labels = tf.concat([fake_data, row_one_hot_labels],
 -1)
            predictions = self.discriminator(fake_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.train
able_weights))

        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(),
        }
```

Applying provided training dataset to CGAN:

```
# Scale the pixel values to [0, 1] range, add a channel dimension to
# the images, and one-hot encode the labels.
all_digits = np.reshape(X_train, (-1, 11)).astype('float32')
all_digits=np.array(all_digits).reshape(-1,11,1)
all_labels = keras.utils.to_categorical(y_train, 2)
#all_labels=np.array(y_train,dtype="float32").reshape(-1,1)
# Create tf.data.Dataset.
dataset = tf.data.Dataset.from_tensor_slices((all_digits, all_labels))
dataset = dataset.shuffle(buffer_size=1024).batch(batch_size)

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

```
cond_gan = ConditionalGAN(
    discriminator=discriminator, generator=generator, latent_dim=latent_
dim
)
cond_gan.compile(
    d_optimizer=keras.optimizers.Adam(learning_rate=0.01),
    g_optimizer=keras.optimizers.Adam(learning_rate=0.01),
    loss_fn=keras.losses.BinaryCrossentropy(from_logits=True),
)

cond_gan.fit(dataset, epochs=20)
```

Generating synthetic fake tasks via generator:

```
trained_gen = cond_gan.generator
trained_dic=cond_gan.discriminator
num_interpolation =   X_test.shape[0]

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

interpolation_labels = keras.utils.to_categorical([1]*num_interpolation,
 2)
interpolation_labels = tf.reshape(interpolation_labels, (num_interpolati
on, 2))
noise_and_labels = tf.concat([interpolation_noise, interpolation_labels]
, axis=1)

fake = trained_gen.predict(noise_and_labels)
```

Testing the discriminator:

```
y_true_pred=np.concatenate([[1]*y.shape[0],[0]*fake.shape[0]])
y_true_pred.shape
```

```
rows = np.reshape(X, (-1, 11)).astype('float32')
rows = np.array(rows).reshape(-1,11,1)
all_labels = keras.utils.to_categorical(y, 2)


one_hot_labels = all_labels[:, None, None]

one_hot_labels = tf.repeat(
    one_hot_labels, repeats=[row_size]
)

one_hot_labels = tf.reshape(
    one_hot_labels, (-1, row_size, n_classes)
)
# N X 11 X n
data_and_labels = tf.concat([rows, one_hot_labels], -1)

all_labels = keras.utils.to_categorical([0]*fake.shape[0], 2)
```

```
one_hot_labels = all_labels[:, None, None]

one_hot_labels = tf.repeat(
    one_hot_labels, repeats=[row_size]
)

one_hot_labels = tf.reshape(
    one_hot_labels, (-1, row_size, n_classes)
)

fake_and_labels = tf.concat([fake,one_hot_labels], -1)

# nx11x1 +nx11x2 =nx11x3

combined_data = tf.concat(
    [data_and_labels,fake_and_labels], axis=0
)

fake_test=trained_dic.predict(combined_data)

y_pred=np.array([ 1 if x>0.5 else 0 for x in fake_test])
print(classification_report(y_true_pred, y_pred))
cm = confusion_matrix(y_true_pred, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Generator','Real'])
disp.plot()
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2897
           1       1.00      1.00      1.00     14484

    accuracy                           1.00     17381
   macro avg       1.00      1.00      1.00     17381
weighted avg       1.00      1.00      1.00     17381
```

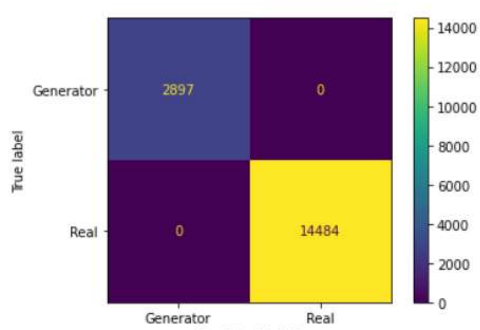

Figure 20 Discriminator classification report and confusion