In [102]:

```
import numpy as np
np.random.seed(1)
from numpy.random import seed
seed(1)
from tensorflow import set_random_seed
set_random_seed(1)

import warnings
import itertools
import pandas as pd
import collections
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
import random


# Plotting packages
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
matplotlib.style.use('ggplot')
from bokeh.plotting import figure, output_notebook, show, ColumnDataSour
from bokeh.models import HoverTool, NumeralTickFormatter
from bokeh.palettes import Set3_12
from bokeh.transform import jitter
from io import StringIO
import re



from tqdm import tqdm, tqdm_notebook

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_auc_score
from sklearn.metrics import (confusion_matrix, precision_recall_curve, a
                             roc_curve, recall_score, classification_rep
from sklearn.metrics import accuracy_score, precision_score

# PyTorch Packages
import torch.nn as nn
from torch.autograd import Variable as V
import torch.nn.functional as F
import torch
from torch.utils.data import DataLoader



from pylab import rcParams
from sklearn.model_selection import train_test_split
from keras.models import Model, load_model
from keras.layers import Input, Dense, Dropout
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import regularizers
%matplotlib inline
```

```
sns.set(style='whitegrid', palette='muted', font_scale=1.5)
rcParams['figure.figsize'] = 14, 8
```

In [103]:

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d %H:%M:%S
df_cpu_utilization_asg_misconfiguration = pd.read_csv('/home/jose/projec
df_ec2_cpu_utilization_5f5533 = pd.read_csv('/home/jose/projects/uwo-ano
```

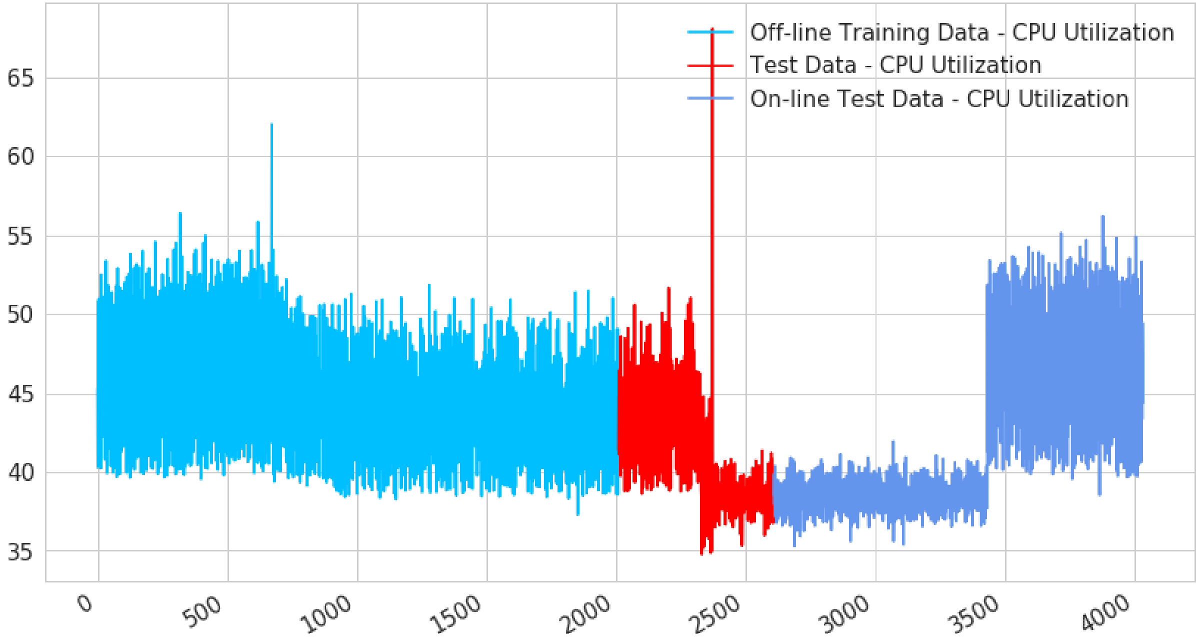
In [104]:

```
#df_ec2_cpu_utilization_5f5533
```

In [105]:

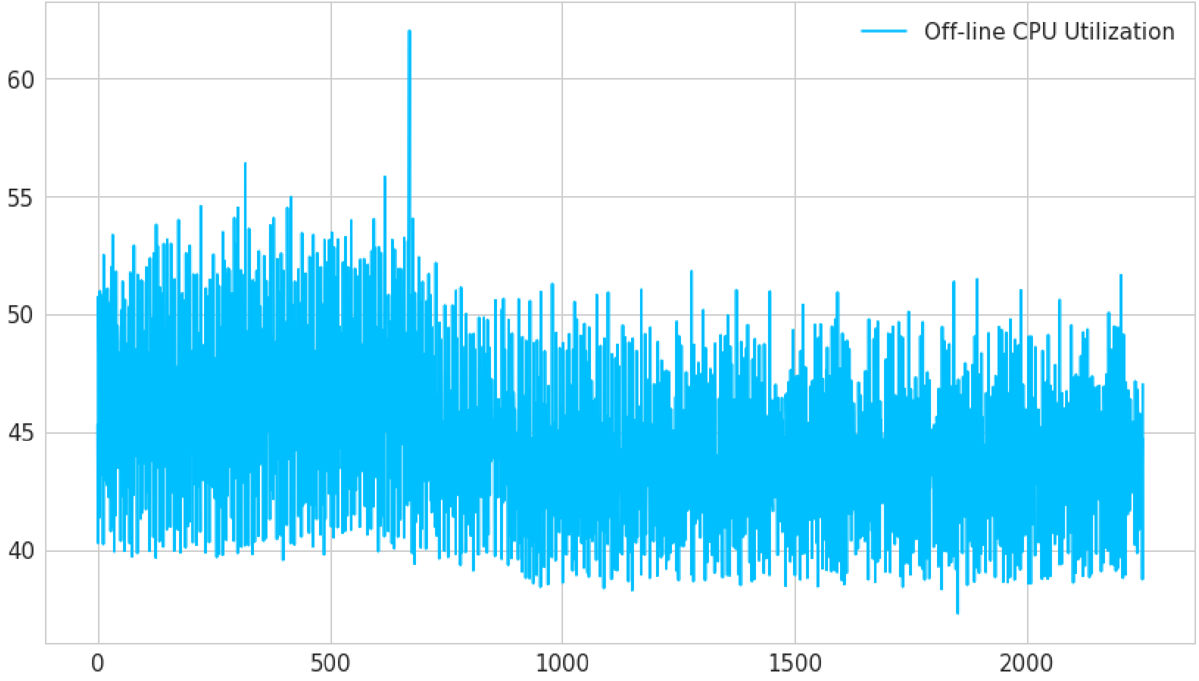
```
df_date = df_ec2_cpu_utilization_5f5533.copy()
df_stage = pd.DataFrame(df_date['2014-02-14 14:27:00':'2014-02-16 16:25:
df_append = df_stage.copy()

rng = pd.date_range('2014-02-28 14:27:00', periods=600, freq='5Min')
df_append = df_append.reset_index(drop=True)
df_append['timestamp'] = rng
df_append = df_append.set_index('timestamp')

df2 = pd.concat([df_ec2_cpu_utilization_5f5533.copy(), df_append.copy()]


df = df2.iloc[600:].copy()

df_plot = df.copy()

df_plot.rename(columns={'value': 'CPU Consumption'}, inplace=True)

df_plot = df_plot.reset_index(drop=True)

df_negative_anomalies = pd.DataFrame(df[:'2014-02-24 18:35:00'].copy())
df_positive_anomalies = pd.DataFrame(df['2014-02-24 18:35:00': ].copy())
print(df_negative_anomalies.shape, df_positive_anomalies.shape)


anomaly_list = ['2014-02-19 00:22:00']
anomaly_list.append('2014-02-24 18:37:00')



df_plot1 = df_plot[:2009].copy()
df_plot2 = df_plot[2009:2609].copy()
df_plot3 = df_plot[2609:].copy()

df_plot1.rename(columns={'value': 'Off-line Training Data - CPU Utilizat
df_plot3.rename(columns={'value': 'Test Data - CPU Utilization'}, inplac
df_plot3.rename(columns={'value': 'On-line Test Data - CPU Utilization'}


plt.plot(df_plot1, color='deepskyblue')
plt.plot(df_plot2, color='red')
plt.plot(df_plot3, color='cornflowerblue')

plt.gcf().autofmt_xdate()
plt.xticks()

plt.legend(['Off-line Training Data - CPU Utilization','Test Data - CPU

plt.show()
```

```
(2330, 1) (1702, 1)
```



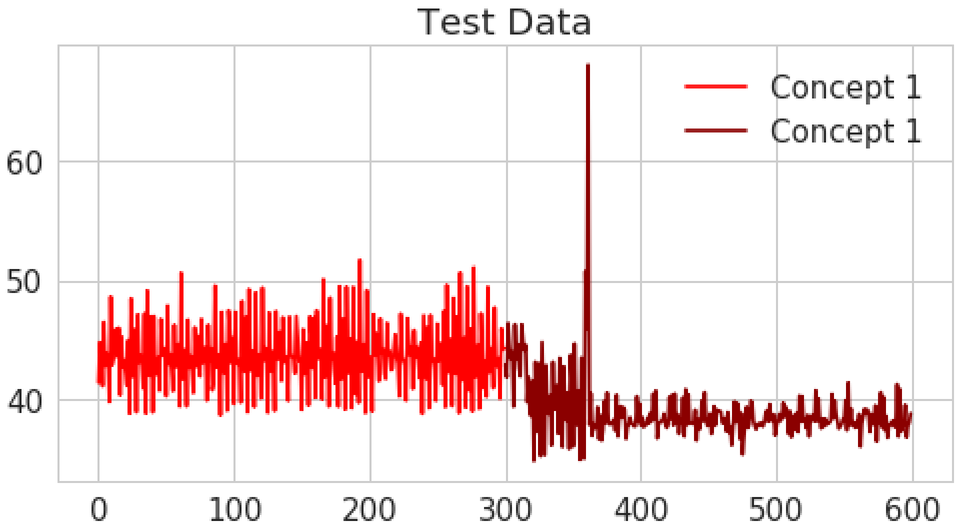
In [106]:

```
df_plot1 = df_plot[:2250].copy()
df_plot1.rename(columns={'value': 'Off-line CPU Utilization'}, inplace=T

plt.plot(df_plot1, color='deepskyblue')

plt.legend(['Off-line CPU Utilization'], loc='upper right')

plt.show()
```



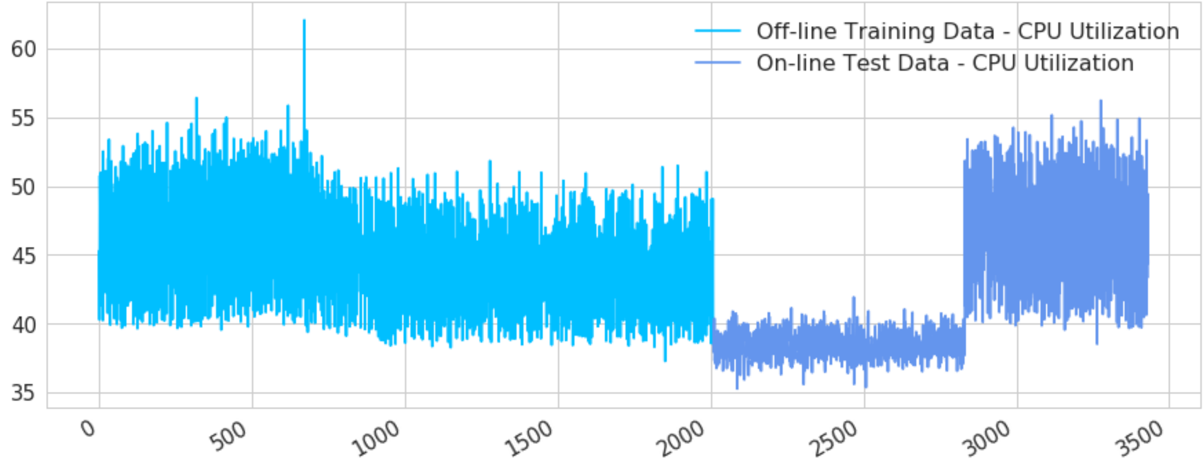
In [107]:

```
#Plto test Dataset

plt.figure(figsize=(8,4))
plot_test = df_plot[2009:2609].copy()
plot_test.reset_index(drop=True, inplace = True)
plot_test1 = plot_test[0:300].copy()
plot_test2 = plot_test[300:].copy()

plt.plot(plot_test1, color ='red')
plt.plot(plot_test2, color ='darkred')


plt.legend(['Concept 1', 'Concept 1'], loc='upper right');

plt.title('Test Data')

plt.show()
```



In [113]:

```
#plot Off-line and On-lie Dtaaset
plt.figure(figsize=(15,6))

df_plot1 = df_plot[:2009].copy()
df_plot3 = df_plot[2609:].copy()

df_plot2 = pd.concat([df_plot1, df_plot3])

df_plot2.reset_index(drop=True, inplace = True)

df_plot1 = df_plot2[:2009].copy()
df_plot3 = df_plot2[2009:].copy()

df_plot1.rename(columns={'value': 'Off-line Training Data - CPU Utilizat
df_plot3.rename(columns={'value': 'On-line Test Data - CPU Utilization'}


plt.plot(df_plot1, color='deepskyblue')
plt.plot(df_plot3, color='cornflowerblue')

plt.gcf().autofmt_xdate()
plt.xticks()

plt.legend(['Off-line Training Data - CPU Utilization', 'On-line Test Da

plt.show()
```



In [8]:

```
skip_window = 3
look_back = 21
look_back_real = look_back - skip_window
```

In [9]:

```
def time_window(dataset, look_back):
    dataX, dataY = [], []

    N = look_back
    x = np.linspace(0,2,N)
    pdf = np.exp(-x**2)
    pdf = pdf/np.sum(pdf)
    n = skip_window

    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]

        #create mask, choose random indices from x according to pdf, set
        indices = np.full(x.shape, False, bool)
        randices = np.random.choice(np.arange(indices.shape[0]), n, repl
        indices[randices] = True

        x_rand_vals = a[randices]
        x_remaining = a[~indices]



        dataX.append(x_remaining)
        dataY.append(dataset[i + look_back, 0])
    return pd.DataFrame(dataX), pd.DataFrame(dataY)
```

In [10]:

```
df_negative_anomalies = df_negative_anomalies.reset_index()
df_positive_anomalies = df_positive_anomalies.reset_index()

dataset = pd.DataFrame(df.copy())
dataset = dataset.reset_index()


dataset_dates = dataset.iloc[look_back: -1 , 0].values

dataset_dates = dataset_dates.reshape(dataset_dates.shape[0], )
dataset_dates.shape

dataset = dataset.drop(['timestamp'], axis=1)

dataset_X, dataset_Y = time_window(dataset.values, look_back)

print(len(dataset_X), len(dataset_Y))
```

```
4010 4010
```

In [11]:

```
dataset_X['timestamp'] = dataset_dates
dataset_X['value'] = dataset_Y
dataset_X['class'] = 0
dataset_Y['class'] = 0
dataset_X['value'] = dataset_Y
```

In [12]:

```
size_positive_anomalies = df_positive_anomalies.shape[0]

dataset_X.iloc[-size_positive_anomalies + 1:, look_back_real +2] = 1
dataset_Y.iloc[-size_positive_anomalies + 1:, 1] = 1
```

In [13]:

```
first_measure_train = dataset_X.iloc[0].name

number_test_negative_anomalies = 300
number_test_postive_anomalies = 300
size_df_negative_anomalies = dataset_X[dataset_X['class']==0].shape[0]
last_measure_train = size_df_negative_anomalies - number_test_negative_an

first_measure_test = last_measure_train
last_measure_test = dataset_X.iloc[first_measure_test].name + number_tes

first_measure_online_test = last_measure_test
last_measure_online_test = dataset_X.iloc[-1].name +1
```

In [14]:

```
print(first_measure_train, last_measure_train, last_measure_train - firs
print(first_measure_test, last_measure_test, last_measure_test - first_me
print(first_measure_online_test, last_measure_online_test, last_measure_o
```

```
0 2009 2009
2009 2609 600
2609 4010 1401
```

In [15]:

```
trainX = pd.DataFrame(dataset_X.iloc[first_measure_train:last_measure_tra
trainY = pd.DataFrame(dataset_Y.iloc[first_measure_train:last_measure_tra
print(trainX.shape, trainY.shape)

testX = pd.DataFrame(dataset_X.iloc[first_measure_test:last_measure_test
testY = pd.DataFrame(dataset_Y.iloc[first_measure_test:last_measure_test
print(testX.shape, testY.shape)

testX_online = pd.DataFrame(dataset_X.iloc[first_measure_online_test:las
testY_online = pd.DataFrame(dataset_Y.iloc[first_measure_online_test:las
testX_online = testX_online.reset_index(drop=True)
print(testX_online.shape, testY_online.shape)
```

```
(2009, 21) (2009, 2)
(600, 21) (600, 2)
(1401, 21) (1401, 2)
```

In [17]:
```
def time_feature_generation(df):
    minutes = 60
    hours = 24
    daysOfWeek = 7

    df['minute'] = pd.DatetimeIndex(df['timestamp']).minute
    df['hour'] = pd.DatetimeIndex(df['timestamp']).hour
    df['date'] = pd.DatetimeIndex(df['timestamp']).day
    df['dayofweek'] = pd.DatetimeIndex(df['timestamp']).dayofweek


    df['sin_minute'] = np.sin(2*np.pi*df.minute/minutes)
    df['cos_minute'] = np.cos(2*np.pi*df.minute/minutes)

    df['sin_hour'] = np.sin(2*np.pi*df.hour/hours)
    df['cos_hour'] = np.cos(2*np.pi*df.hour/hours)


    return df
```

In [18]:
```
trainX = time_feature_generation(trainX)
testX = time_feature_generation(testX)
testX_online = time_feature_generation(testX_online)
```

In [19]:
```
print(trainX.shape, testX.shape, testX_online.shape)
```

```
(2009, 29) (600, 29) (1401, 29)
```

In [20]:
```
def remove_outliers(df_X, df_Y):

        for hour in range(24):
            for minute in range(60):
                window = df_X.loc[(df_X['hour']== hour) &  (df_X['minute

                if not window.empty:

                    q3 = np.percentile(window['value'].values, 75, axis =
                    q1 = np.percentile(window['value'].values, 25, axis =

                    iqr = q3 - q1
                    lower_bound = q1 - (iqr * 1.5)
                    upper_bound = q3 + (iqr * 1.5)

                    df_iqr = window.loc[(window['value'] < lower_bound)

                    if  not df_iqr.empty:
                        for i in df_iqr.index:
                            df_X.loc[i, 'class'] = 1
                            df_Y.loc[i, 'class'] = 1

        return df_X, df_Y
```

In [21]:
```
trainX, trainY = remove_outliers(trainX, trainY)
```

In [22]:
```
def classify_anomalies_train(df_X, df_Y, anomaly_list):

    for i in anomaly_list:

        anomaly = df_X.loc[df_X['timestamp'] == i].index.values.astype(i
        df_X.loc[anomaly, 'class'] = 1
        df_Y.loc[anomaly, 'class'] = 1
        print(df_X.loc[anomaly, :])

    return df_X, df_Y
```

In [23]:
```

trainX, trainY = classify_anomalies_train(trainX, trainY, anomaly_list)
```

```
          0       1      2       3       4       5       6       7  
  8  \
650  48.122  51.914  46.76  44.482  49.664  48.362  46.972  40.502  45.
892

          9     10      11      12      13      14      15      16  
  17  \
650  44.042  49.21  46.572  44.858  48.156  41.878  53.092  50.015  54.
6033

              timestamp   value  class  minute  hour  date  dayofweek  
\
650 2014-02-19 00:22:00  62.056      1      22     0    19          2


     sin_minute  cos_minute  sin_hour  cos_hour
650    0.743145   -0.669131       0.0       1.0
Empty DataFrame
Columns: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
timestamp, value, class, minute, hour, date, dayofweek, sin_minute, cos
_minute, sin_hour, cos_hour]
Index: []
```

In [24]:
```
def remove_anomalies_train(df_X, df_Y):
    if 'class' in df_X.columns:
        df_X = df_X[df_X['class'] == 0]


        df_Y = df_Y[df_Y['class'] == 0]

    else:
        print("Anomalies not classified")
    return df_X, df_Y
```

In [25]:
```
trainX, trainY = remove_anomalies_train(trainX, trainY)
print(trainX.shape, trainY.shape)
```

```
(1936, 29) (1936, 2)
```

In [26]:

```
trainX[trainX['class']==1]
```

Out[26]:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	timestamp	value	class	minut

In [27]:

```
def delete_class_train(df_X):
    if 'class' in df_X.columns:
        df_X = df_X.drop(['class'], axis = 1)
        df_X = df_X.reset_index(drop=True)

    else:
        print("Anomalies not classified")
    return df_X
```

In [28]:

```
trainX = delete_class_train(trainX)

testX = delete_class_train(testX)

testX_online = delete_class_train(testX_online)
```

In [29]:

```
print(trainX.shape, testX.shape, testX_online.shape)
```

```
(1936, 28) (600, 28) (1401, 28)
```

In [30]:

```
def statistics_feature_generation(df):

    iqr = []
    q1 = []
    q3 = []
    median = []
    mean = []


    for i in range(df.shape[0]):
        time_window_columns =df.iloc[i:i+1, 0:look_back_real].values
        median.append(np.percentile(time_window_columns, 50, axis =1))
        q1.append(np.percentile(time_window_columns, 25, axis =1))
        q3.append(np.percentile(time_window_columns, 75, axis =1))
        iqr_stg = np.percentile(time_window_columns, 75, axis =1) - np.p
        iqr.append(iqr_stg)
        mean.append(np.mean(time_window_columns, axis=1))


    #print(iqr)
    iqr =np.array(iqr)
    q1 = np.array(q1)
    q3 = np.array(q3)
    median  = np.array(median)
    mean = np.array(mean)

    df['IQR'] = iqr
    df['mean'] = mean

    df['mean_next'] =  0
    df['mean_previous'] =  0
    df['mean_plus'] =  0
    df['mean_minus'] = 0



    for i in range(df.shape[0]):

        if i ==df.shape[0] - 1:
            df.loc[i, 'mean_next'] =  0
            df.loc[i, 'mean_plus'] = 0
        else:
            df.loc[i, 'mean_next'] = df.loc[i+1, 'mean']
            df.loc[i, 'mean_plus'] = df.loc[i, 'mean_next'] -df.loc[i, '

    df.loc[df.shape[0] - 1, 'mean_plus'] = df.loc[df.shape[0] - 2, 'mean_



    for i in range(df.shape[0]):

        if i == 0:
            df.loc[i, 'mean_previous'] =  0
            df.loc[i, 'mean_minus'] = 0
        else:
            df.loc[i, 'mean_previous'] = df.loc[i-1, 'mean']
            df.loc[i, 'mean_minus'] = df.loc[i, 'mean'] -  df.loc[i, 'mea

    df.loc[0, 'mean_minus'] =df.loc[1, 'mean_minus']

    return df
```

In [31]:

```

trainX = statistics_feature_generation(trainX)

testX = statistics_feature_generation(testX)

testX_online = statistics_feature_generation(testX_online)
```

In [32]:

```
trainX_timestamp = pd.DataFrame(trainX)
testX_timestamp = pd.DataFrame(testX)
testX_online_timestamp = pd.DataFrame(testX_online)
```

In [33]:

```
def remove_columns(df):
    for column in  ['timestamp', 'date', 'minute', 'hour', 'mean_next',

        if column in df.columns:
            df = df.drop([column], axis=1)

    return df
```

In [34]:

```
trainX = remove_columns(trainX)
testX = remove_columns(testX)
testX_online = remove_columns(testX_online)
```

In [35]:

```
testY = testY.drop(0, axis=1)
trainY = trainY.drop(0, axis=1)
testY_online = testY_online.drop(0, axis=1)
```

In [36]:

```
print(trainX.shape, testX.shape, testX_online.shape)
```

```
(1936, 26) (600, 26) (1401, 26)
```

In [37]:

```
scaler = MinMaxScaler()

trainX_scaled = pd.DataFrame(scaler.fit_transform(trainX.values))
testX_scaled = pd.DataFrame(scaler.fit_transform(testX.values))
testX_online_scaled = pd.DataFrame(scaler.fit_transform(testX_online.valu

trainY_scaled = pd.DataFrame((trainY.values))
testY_scaled = pd.DataFrame((testY.values))
testY_online_scaled = pd.DataFrame((testY_online.values))

testX_np = testX_scaled.values
trainX_np = trainX_scaled.values
testX_online_np = testX_online_scaled.values

testY_np = testY_scaled.values
trainY_np = trainY_scaled.values
testY_online_np = testY_online_scaled.values
```

In [38]:

```
input_dim = trainX_np.shape[1]
encoding_dim = int(trainX_np.shape[1]/2)

input_layer = Input(shape=(input_dim, ))

encoder = Dense(encoding_dim,  init = 'glorot_normal', activation='relu'
encoder = Dropout(0.75)(encoder)
encoder = Dense(int(encoding_dim / 2), init = 'glorot_normal', activatio
encoder = Dropout(0.75)(encoder)

decoder = Dense(int(encoding_dim), init = 'glorot_normal', activation='r
decoder = Dropout(0.75)(decoder)
decoder = Dense(input_dim,  init = 'glorot_normal', activation='sigmoid'


autoencoder = Model(inputs=input_layer, outputs=decoder)
```

```
/home/jose/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:
6: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(13,
activation="relu", activity_regularizer=<keras.reg..., kernel_initializ
er="glorot_normal")`

/home/jose/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:
8: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(6,
activation="relu", activity_regularizer=<keras.reg..., kernel_initializ
er="glorot_normal")`

/home/jose/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:
11: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(1
3, activation="relu", activity_regularizer=<keras.reg..., kernel_initia
lizer="glorot_normal")`
  # This is added back by InteractiveShellApp.init_path()
/home/jose/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:
13: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(2
6, activation="sigmoid", kernel_initializer="glorot_normal")`
  del sys.path[0]
```

In [39]:

```
print (autoencoder.summary())
```

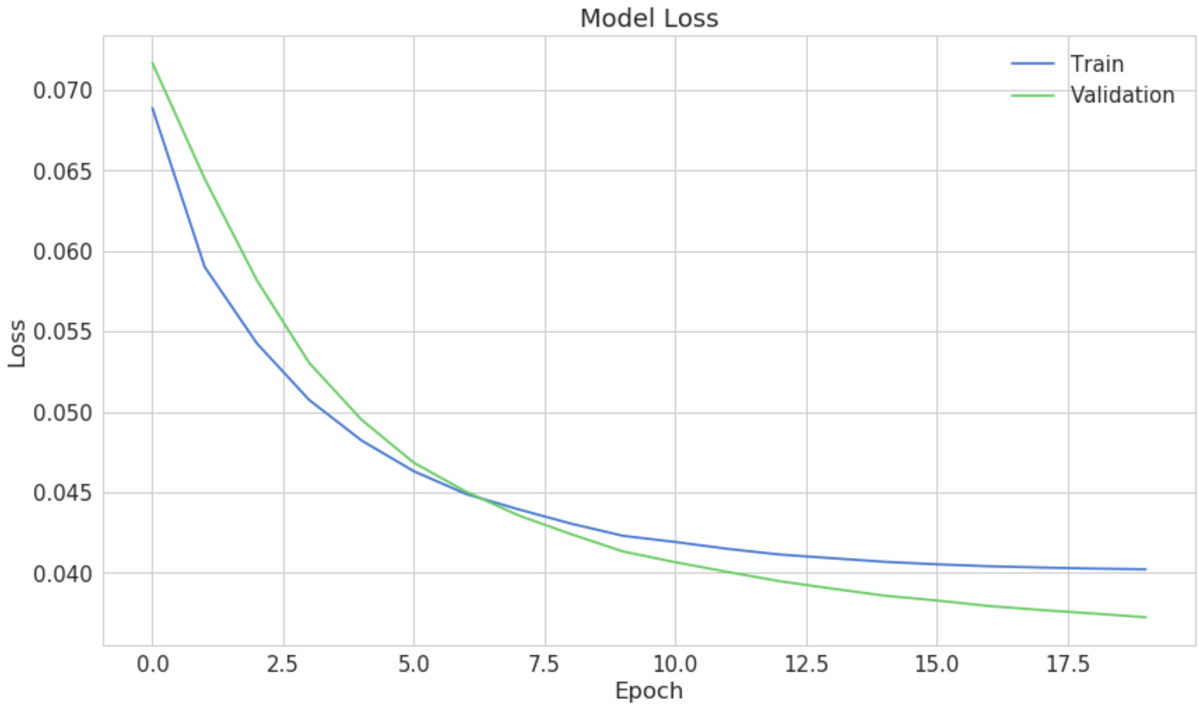
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 26)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 13)                351       
_________________________________________________________________
dropout_1 (Dropout)          (None, 13)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 84        
_________________________________________________________________
dropout_2 (Dropout)          (None, 6)                 0         
_________________________________________________________________
dense_3 (Dense)              (None, 13)                91        
_________________________________________________________________
dropout_3 (Dropout)          (None, 13)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 26)                364       
=================================================================
Total params: 890
Trainable params: 890
Non-trainable params: 0
_________________________________________________________________
None
```

In [40]:

```
del autoencoder

autoencoder = Model(inputs=input_layer, outputs=decoder)

nb_epoch = 20
batch_size = 20

autoencoder.compile(optimizer='adam',
                    loss='mean_squared_error',
                    metrics=['accuracy', 'mse'])


history = autoencoder.fit(trainX_np, trainX_np,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_split=0.1,
                    #validation_data=(testX_np, testX_np),
                    verbose=0).history

autoencoder.save('model.h5')
```

In [41]:

```python
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right');
```



In [42]:

```python
testY_np = testY_np.reshape(testY_np.shape[0],)

preds = autoencoder.predict(testX_np,verbose=1)

error = np.mean(np.power(testX_np - preds, 2), axis=1)

threshold_zeros = np.zeros(testY_np.shape[0])

error_df = pd.DataFrame(data = {'error':error,'true':testY_np, 'threshol

error_df.groupby('true')['error'].describe().reset_index()
```

```
600/600 [==============================] - 0s 88us/step
```

Out[42]:

	true	count	mean	std	min	25%	50%	75%	max
0	0	300.0	0.045642	0.007229	0.033763	0.040598	0.044235	0.049222	0.089270
1	1	300.0	0.056763	0.007577	0.036211	0.054364	0.056586	0.058331	0.093228

In [43]:

```
def format_plot(p, x_label, y_label):
    p.grid.grid_line_color = None
    p.background_fill_color = "whitesmoke"
    p.axis.minor_tick_line_color = None
    p.title.align = 'center'
    p.title.text_font_size = "18px"
    p.xaxis.axis_label = x_label
    p.yaxis.axis_label = y_label
    p.xaxis.axis_label_text_font_size = "14px"
    p.yaxis.axis_label_text_font_size = "14px"
    p.yaxis.axis_line_color = None
    p.yaxis.major_tick_line_color = None
    p.axis.major_label_text_font_size = "12px"
    return p
```

In [44]:

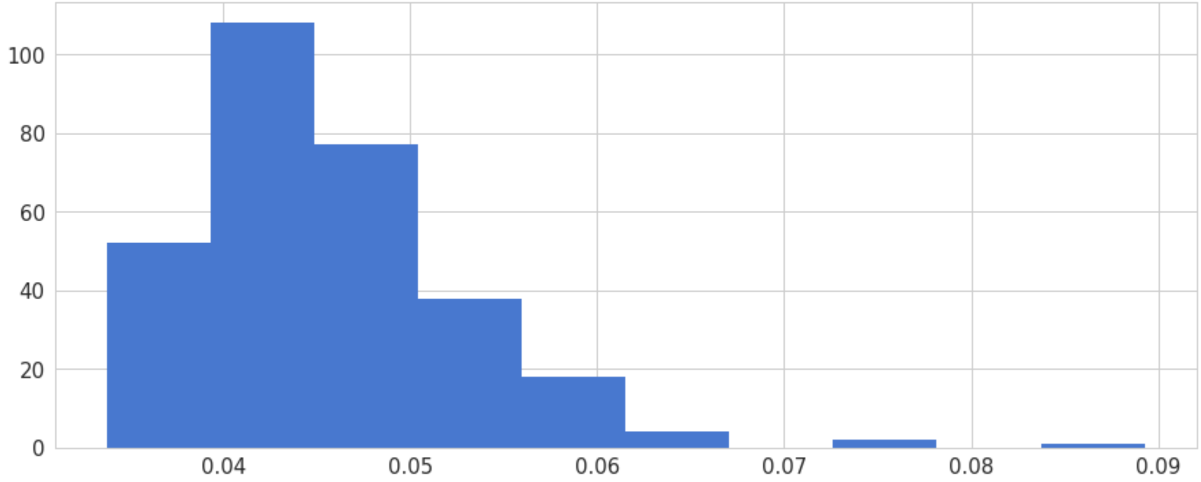
```
def plot_ROC_curve (error):
    fpr, tpr, thresholds = roc_curve(error.true, error.error)
    roc_auc = auc(fpr, tpr)

    source = ColumnDataSource(data=dict(
        fpr = fpr,
        tpr = tpr,
        x = np.linspace(0,1,len(fpr)),
        y = np.linspace(0,1,len(fpr))
    ))

    p = figure(plot_height = 500, plot_width = 500,
               toolbar_location = None,
               title = "Receiver Operating Characteristic")

    j = p.line(x = "x", y = "y",
               color=Set3_12[3],
               line_width = 2,
               line_dash = 'dashed',
               source=source)

    k = p.line(x = "fpr", y = "tpr",
               color=Set3_12[4],
               line_width = 2,
               legend = f'AUC = {roc_auc:0.4f}',
               source=source)

    tips= [
        ("False-Pos", "@fpr{00.0%}"),
        ("True-Pos", "@tpr{00.0%}"),
        ]
    p.add_tools(HoverTool(tooltips=tips, renderers=[k], mode='vline'))

    p = format_plot(p, 'False Positive Rate', 'True Positive Rate')
    p.legend.location = 'bottom_right'

    show(p);
```

In [45]:

```
plot_ROC_curve(error_df)
```

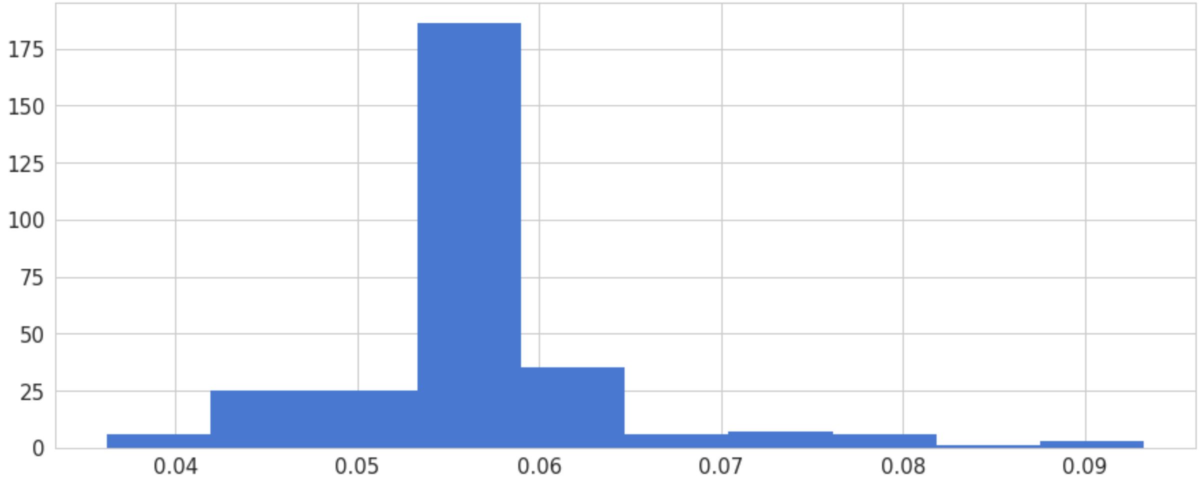
In [46]:

```
#Reconstruction error without anomalyes

fig = plt.figure(figsize=(15,6))
ax = fig.add_subplot(111)
normal_error_df = error_df[(error_df['true']== 0) & (error_df['error'] <
_ = ax.hist(normal_error_df.error.values, bins=10)
```

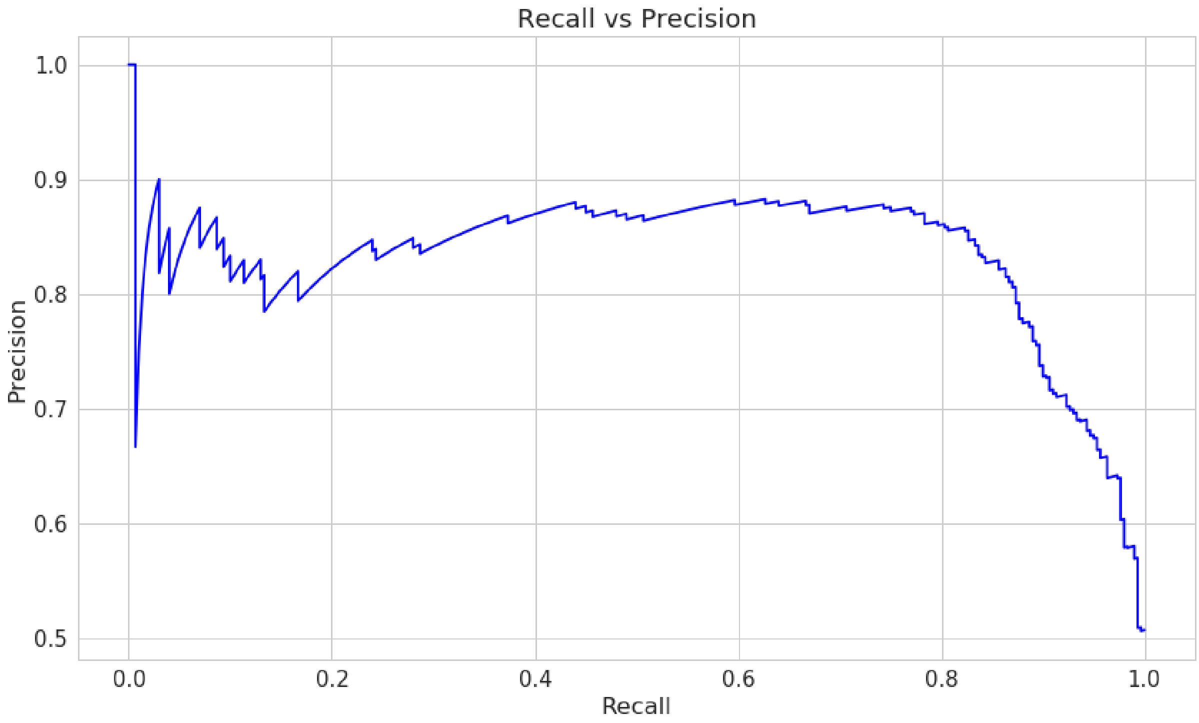


In [47]:

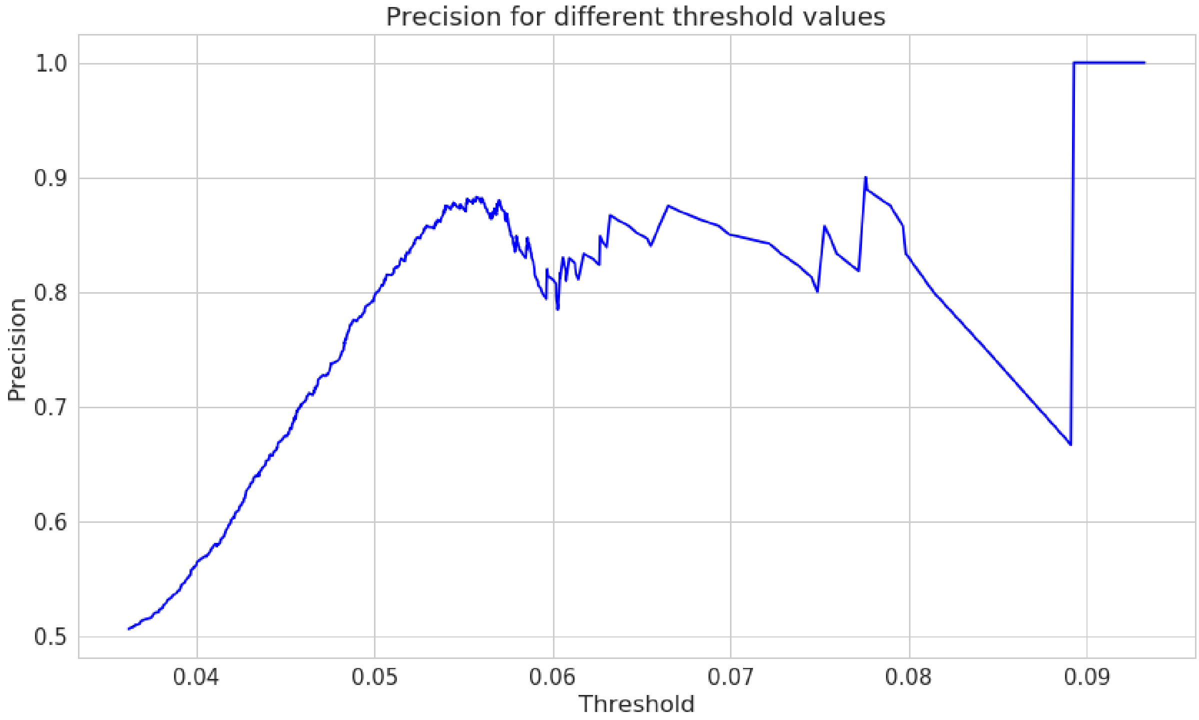
```
#Reconstruction error with anomalyes

fig = plt.figure(figsize=(15,6))
ax = fig.add_subplot(111)
normal_error_df = error_df[(error_df['true']== 1) & (error_df['error'] <
_ = ax.hist(normal_error_df.error.values, bins=10)
```



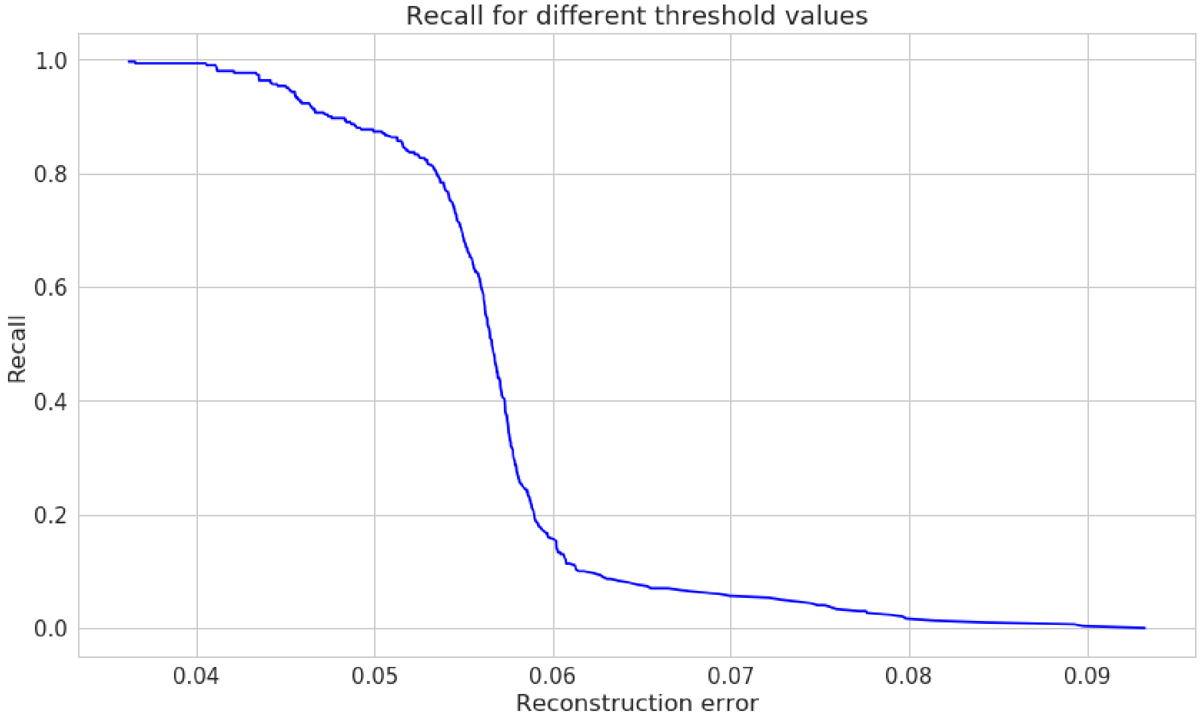
In [48]:
```
precision, recall, th = precision_recall_curve(error_df.true, error_df.e
plt.plot(recall, precision, 'b', label='Precision-Recall curve')
plt.title('Recall vs Precision')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()
```



In [49]:
```
plt.plot(th, precision[1:], 'b', label='Threshold-Precision curve')
plt.title('Precision for different threshold values')
plt.xlabel('Threshold')
plt.ylabel('Precision')
plt.show()
```



In [50]:
```
plt.plot(th, recall[1:], 'b', label='Threshold-Recall curve')
plt.title('Recall for different threshold values')
plt.xlabel('Reconstruction error')
plt.ylabel('Recall')
plt.show()
```



In [51]:

```


temp_df = error_df[error_df['true'] == 1]
threshold_positive = temp_df['error'].mean()  + temp_df['error'].std()
print(f'Threshold: {threshold_positive:.6f}')
```

```
Threshold: 0.064341
```

In [52]:

```
threshold = 0

temp_df = error_df[error_df['true'] == 0]
threshold = temp_df['error'].mean()  + temp_df['error'].std()
print(f'Threshold: {threshold:.6f}')
```

```
Threshold: 0.052871
```

In [53]:

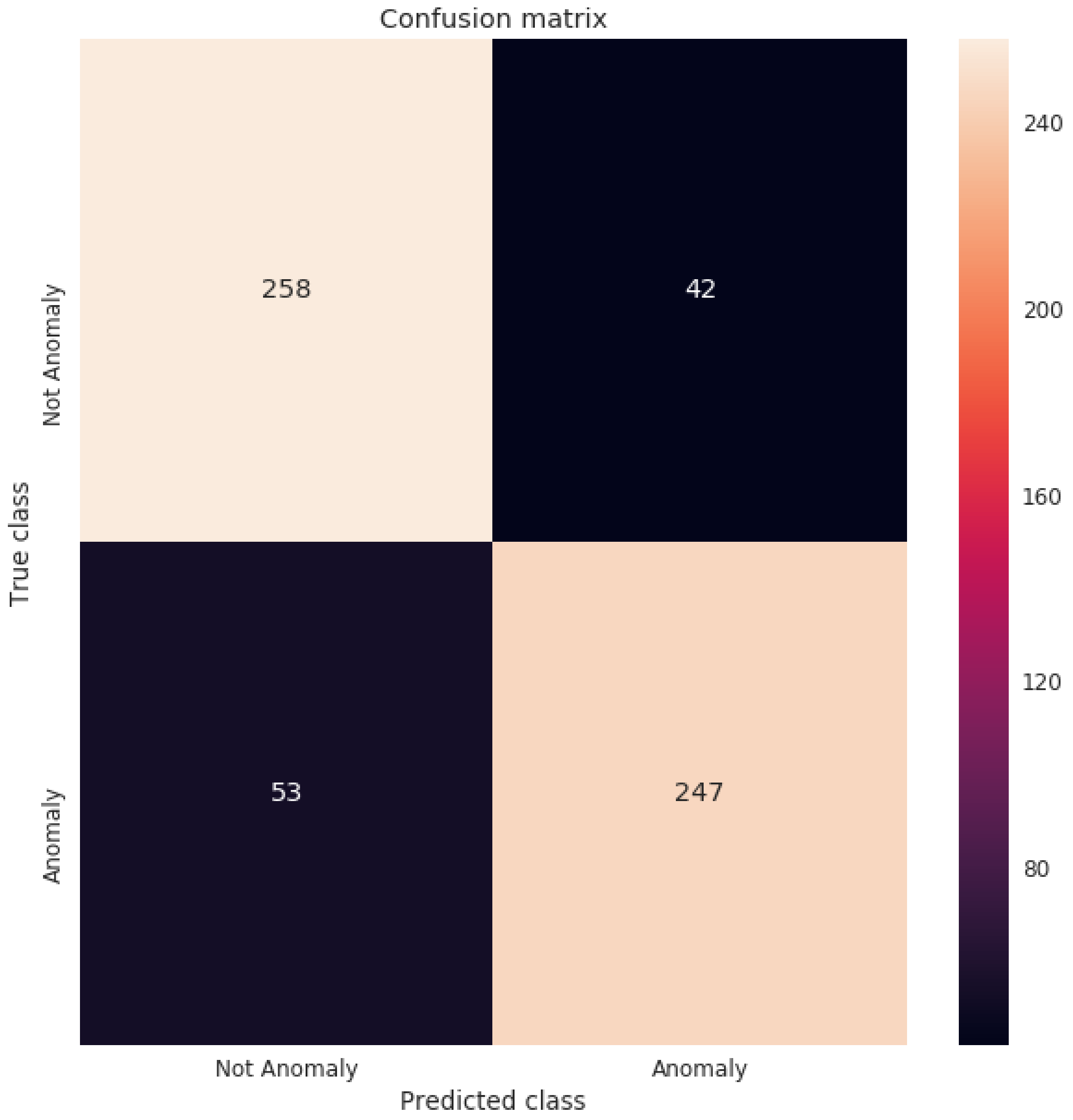
```
error_df['threshold'] = threshold
```

In [55]:

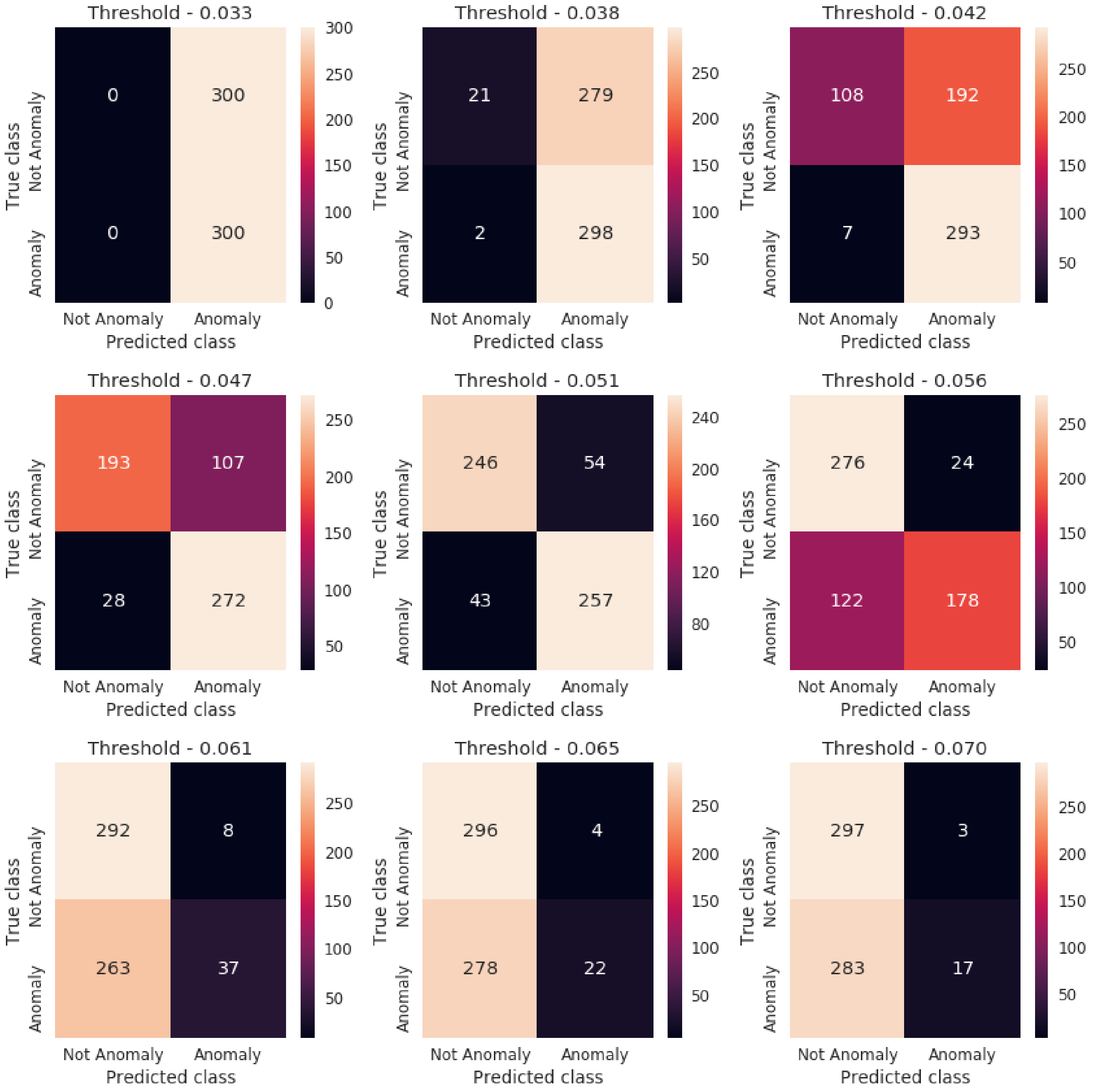
```
y_pred = [1 if e > threshold else 0 for e in error_df.error.values]
print(classification_report(error_df.true.values,y_pred))
```

```
             precision    recall  f1-score   support

          0       0.83      0.86      0.84       300
          1       0.85      0.82      0.84       300

avg / total       0.84      0.84      0.84       600
```

In [56]:
```
conf_matrix = confusion_matrix(error_df.true, y_pred)

sns.set(font_scale = 1.2)
plt.figure(figsize=(10, 10))
sns.heatmap(conf_matrix, xticklabels=['Not Anomaly','Anomaly'], yticklab
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```



In [57]:
```
plt.figure(figsize=(12, 12))
m = []
for thresh in np.linspace(threshold-0.02,0.07,9):
    y_pred = [1 if e > thresh else 0 for e in error_df.error.values]
    conf_matrix = confusion_matrix(error_df.true, y_pred)
    m.append((conf_matrix,thresh))

count = 0
for i in range(3):
    for j in range(3):
        plt.subplot2grid((3, 3), (i, j))
        sns.heatmap(m[count][0], xticklabels=['Not Anomaly','Anomaly'], 
        plt.title(f"Threshold - {m[count][1]:.3f}")
        plt.ylabel('True class')
        plt.xlabel('Predicted class')
        plt.tight_layout()
        count += 1
plt.show()
```



In [58]:
```
def update(model, df_X, nb_epoch, batch_size):

    history = model.fit(df_X, df_X,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    verbose=0).history
    return history
```

In [59]:
```
def predict(model, treshold_pred, df_X, df_Y):

    preds = model.predict(df_X,verbose=0)
    error = np.mean(np.power(df_X - preds, 2), axis=1)

    df_Y= df_Y.reshape(df_Y.shape[0],)

    threshold_zeros = np.zeros(df_Y.shape[0])
    threshold_zeros[:] = treshold_pred

    error= pd.DataFrame(data = {'error':error,'true':df_Y, 'threshold': 

    error['prediction'] = 0

    error['prediction'] = error['error'].apply(lambda x: 1 if x > tresho


    #print (treshold_pred)

    return error
```

In [60]:
```
def calculate_treshold(error):

    temp = error[error['true'] == 0]
    threshold_df = temp['error'].mean()  + temp['error'].std()
    print(f'Threshold: {threshold_df:.6f}')

    error['prediction'] = error['error'].apply(lambda x: 1 if x > thresh

    error['threshold'] =  threshold_df


    return   threshold_df, error
```

In [63]:

```
online_window = 1
df_online_size = testX_online_np.shape[0]
anomalies_count = 0


testX_online_update = np.copy(testX_online_np)
testY_online_update = np.copy(testY_online_np)


testX_online_validate = np.copy(testX_online_np)
testY_online_validate = np.copy(testY_online_np)

threshold_online = threshold

error_anomaly_online_train = error_df[0:0].copy()
check_error_anomaly_online_train = error_df[0:0].copy()

second_context_init = 801

new_model= load_model('model.h5')

context_change = 0

threshold_anomalies = 6

for i in range (df_online_size):

    valueX = np.copy(testX_online_validate[i:online_window+i,:])
    valueY = np.copy(testY_online_validate[i:online_window+i,:])


    error_online_train = predict(new_model, threshold_online, valueX, va
    check_error_anomaly_online_train = pd.concat([check_error_anomaly_on



    if  (error_online_train.loc[0]['error'] >= threshold_online):
        anomalies_count =  anomalies_count + 1



        if (anomalies_count == threshold_anomalies):


            train_onlineX = np.copy(testX_online_update[i-threshold_anom
            train_onlineY = np.copy(testY_online_update[i-threshold_anom
            context_change = context_change + 1


            train_onlineY[:, :] = 0   #daframe with last 5 measures/ ano


            if (i <= second_context_init):


                testY_online_update[i+1:second_context_init,:] = 0 # ent
                history_new_model = update(new_model, train_onlineX, nb_
                new_model.save('model_second.h5')

                error_second_model = predict(new_model, threshold_online

                treshold_second_model,  error_second_model = calculate_t


                new_treshold = treshold_second_model
                threshold_online = new_treshold

                print ("entrou aqui",i, threshold_online )
                anomalies_count = 0

            if (i > second_context_init):

                testY_online_update[i+1:,:] = 0 # entire dataframe with 
                history_new_model = update(new_model, train_onlineX, nb_
                new_model.save('model_third.h5')

                error_online_update = predict(new_model, threshold_onlin

                new_treshold,  error_online_update = calculate_treshold(

                threshold_online = new_treshold

                print ("trainy",i, threshold_online)
                anomalies_count = 0


    else:

        anomalies_count = 0
```

```
Threshold: 0.046248
entrou aqui 77 0.046247727007990694
Threshold: 0.040399
entrou aqui 210 0.04039877644074489
Threshold: 0.036257
entrou aqui 251 0.03625659142240357
Threshold: 0.033653
entrou aqui 409 0.03365318041431244
Threshold: 0.031414
entrou aqui 444 0.03141449364440669
Threshold: 0.029547
entrou aqui 511 0.029547414565639262
Threshold: 0.028265
entrou aqui 603 0.028264678788393373
Threshold: 0.027711
entrou aqui 609 0.027711118423550427
Threshold: 0.026761
entrou aqui 726 0.02676103800625279
Threshold: 0.043253
trainy 807 0.04325307068341215
```

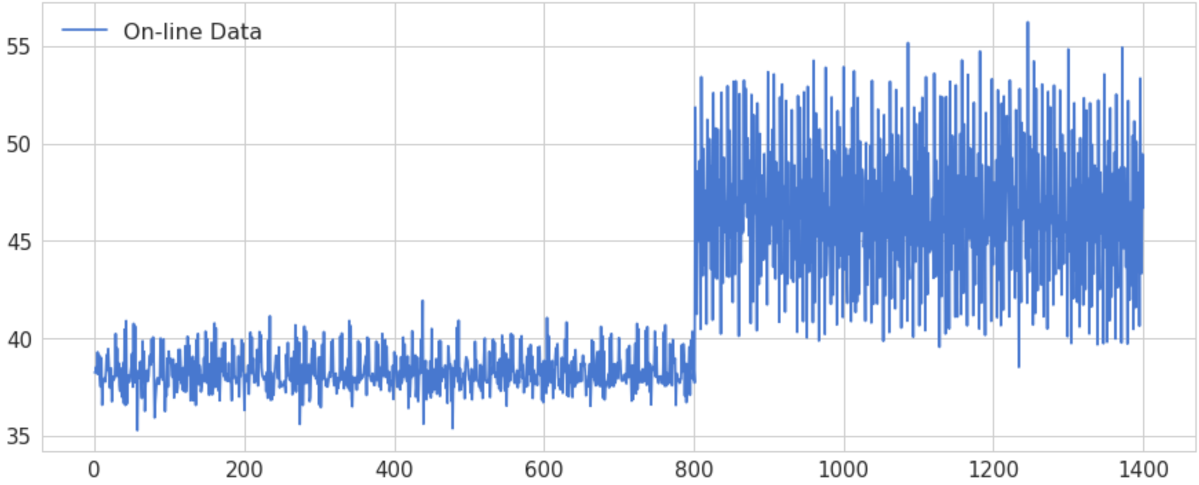
```
Threshold: 0.085053
trainy 813 0.08505257433744108
Threshold: 0.098808
trainy 819 0.09880806920318376
Threshold: 0.100433
trainy 836 0.10043268535258335
Threshold: 0.110319
trainy 877 0.11031902772185109
```

In [66]:

```
check_error_anomaly_online_train['new_true'] = testY_online_update[0:df_
check_error_anomaly_online_train = check_error_anomaly_online_train.rese
```

In [114]:

```
plt.figure(figsize=(15,6))
plt.plot(testX_online_timestamp['value'])
plt.legend(['On-line Data'], loc='upper left', prop={'size': 16})
#plt.title('On-line Dataset')
plt.show()
```

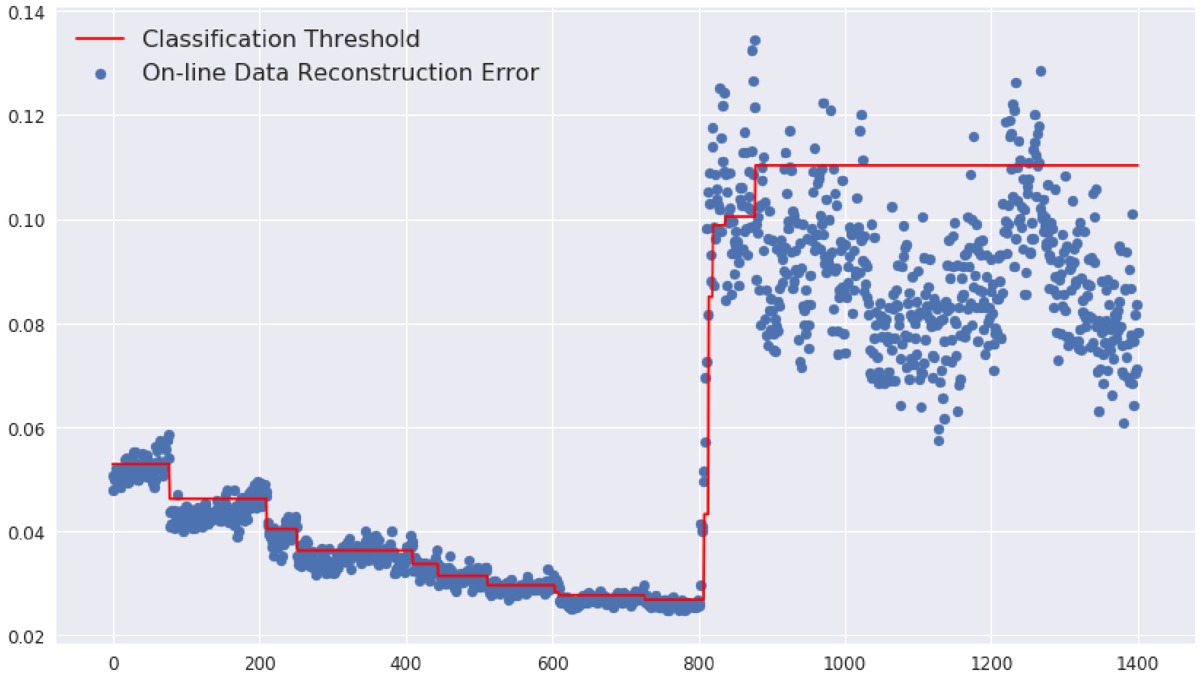


In [99]:

```
plt.plot(check_error_anomaly_online_train['threshold'], color='red')
plt.scatter(check_error_anomaly_online_train.index, check_error_anomaly_
plt.legend(['Classification Threshold', 'On-line Data Reconstruction Err
```

Out[99]:

```
<matplotlib.legend.Legend at 0x7f0a5a809da0>
```



In [75]:

```
first_context = check_error_anomaly_online_train.iloc[0:802,:]
second_context = check_error_anomaly_online_train.iloc[802:,:]
#second_context
```

In [76]:

```
data_validate_modelX = np.copy(testX_online_np[-900:-300,:])
data_validate_modelY = np.copy(testY_online_np[-900:-300,:])
print(data_validate_modelX.shape, data_validate_modelY.shape)

data_validate_modelY[0:300, :] = 0
```

```
(600, 26) (600, 1)
```

In [77]:

```
data1_validate_modelX = np.copy(testX_np)
data1_validate_modelY = np.copy(testY_np)
print(data1_validate_modelX.shape, data1_validate_modelY.shape)
```

```
(600, 26) (600,)
```

In [78]:

```
data1_validate_modelY[0:300] = 1
data1_validate_modelY[300:] = 0
```

In [81]:
```
validate_model_second= load_model('model_second.h5')
error_second_model = predict(validate_model_second, 0, data1_validate_mo
treshold_second_model, error_second_model = calculate_treshold(error_sec
```

```
Threshold: 0.045531
```

In [82]:
```
plot_ROC_curve (error_second_model )
```

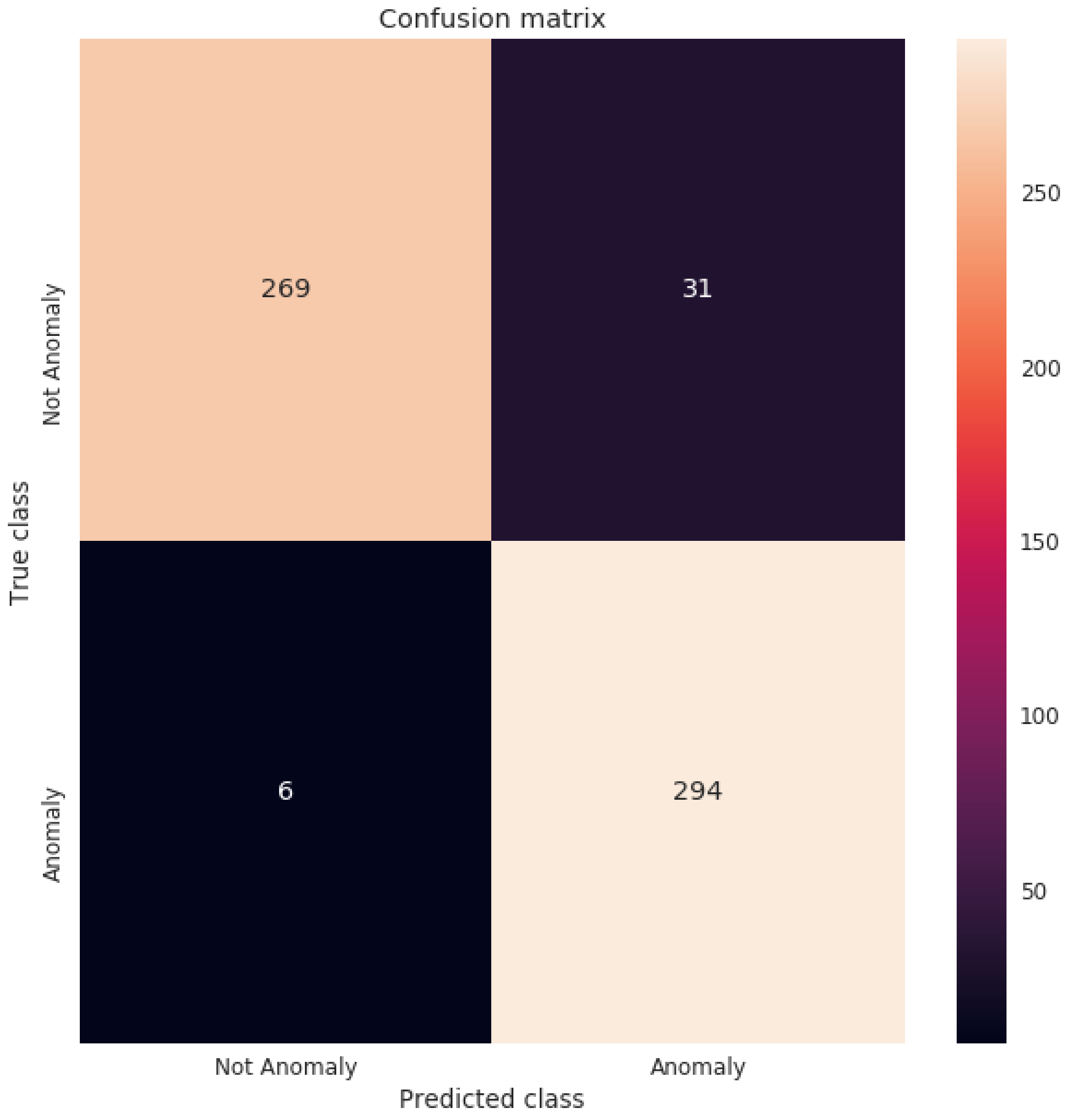
In [84]:
```
y_pred_second_model = [1 if e > treshold_second_model else 0 for e in er
print(classification_report(error_second_model.true.values,y_pred_second_
```

```
             precision    recall  f1-score   support

          0       0.98      0.90      0.94       300
          1       0.90      0.98      0.94       300

avg / total       0.94      0.94      0.94       600
```

In [86]:
```
def report_to_df(report):
    report = re.sub(r" +", " ", report).replace("avg / total", "avg/tota
    report_df = pd.read_csv(StringIO("Classes" + report), sep=' ', index_
    return(report_df)
```

In [87]:
```
report = classification_report(error_second_model.true.values,y_pred_sec
df_precision = report_to_df(report)
precision = df_precision.loc['avg/total']['precision']
```

In [90]:
```
conf_matrix = confusion_matrix(error_second_model.true, y_pred_second_mo

sns.set(font_scale = 1.2)
plt.figure(figsize=(10, 10))
sns.heatmap(conf_matrix, xticklabels=['Not Anomaly','Anomaly'], yticklab
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```



In [92]:
```
data2_validate_modelX = np.copy(testX_np)
data2_validate_modelY = np.copy(testY_np)
print(data2_validate_modelX.shape, data2_validate_modelY.shape)
```

```
(600, 26) (600,)
```

In [93]:
```
validate_model_third= load_model('model_third.h5')

error_third_model = predict(validate_model_third, 0, data2_validate_mode

treshold_third_model, error_third_model = calculate_treshold(error_third_
```

```
Threshold: 0.059196
```

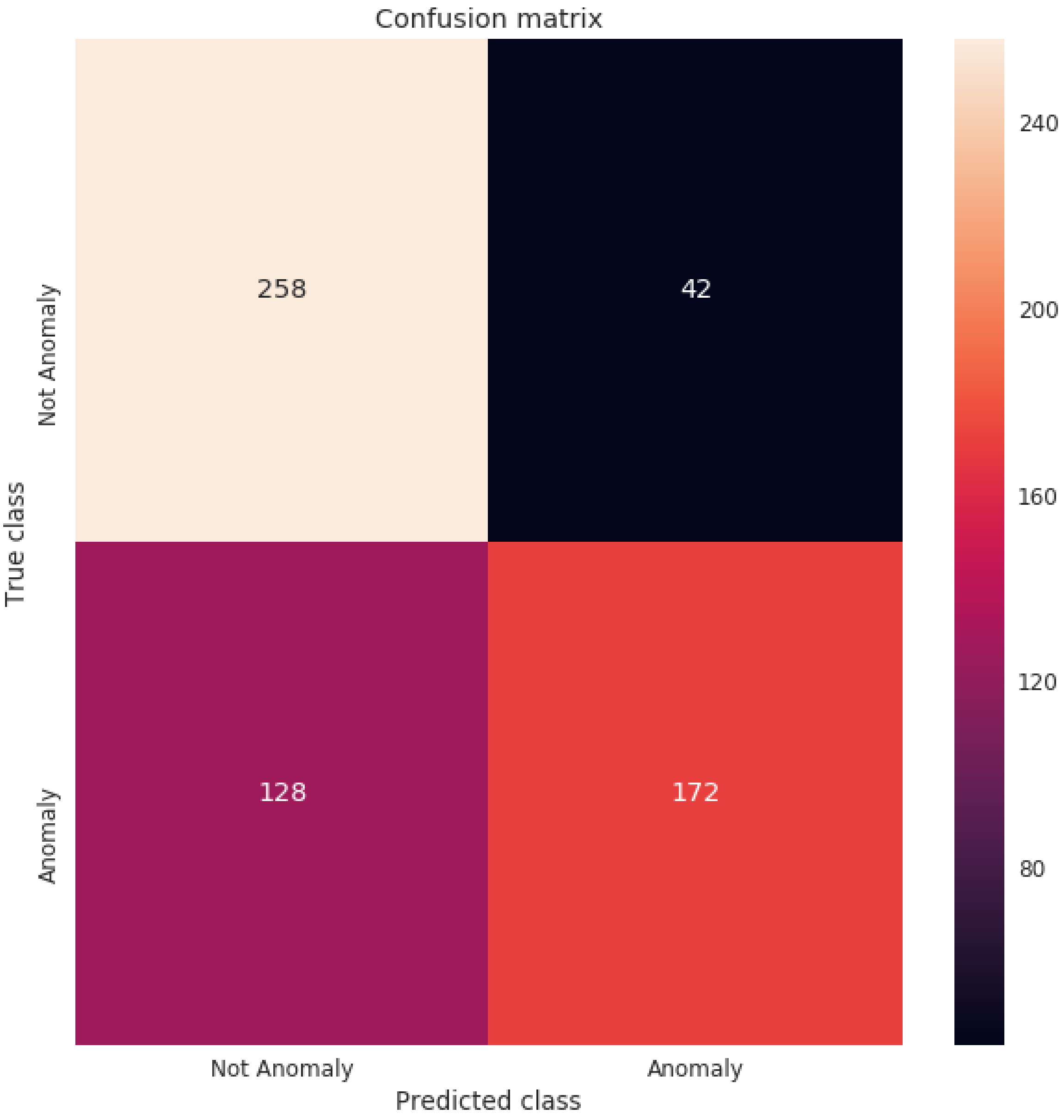
In [94]:
```
plot_ROC_curve (error_third_model)
```

In [95]:
```
y_pred_third_model = [1 if e > treshold_third_model else 0 for e in erro
print(classification_report(error_third_model.true.values,y_pred_third_m
```

```
             precision    recall  f1-score   support

          0       0.67      0.86      0.75       300
          1       0.80      0.57      0.67       300

avg / total       0.74      0.72      0.71       600
```

In [96]:

```
conf_matrix = confusion_matrix(error_third_model.true, y_pred_third_mode

sns.set(font_scale = 1.2)
plt.figure(figsize=(10, 10))
sns.heatmap(conf_matrix, xticklabels=['Not Anomaly','Anomaly'], yticklab
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

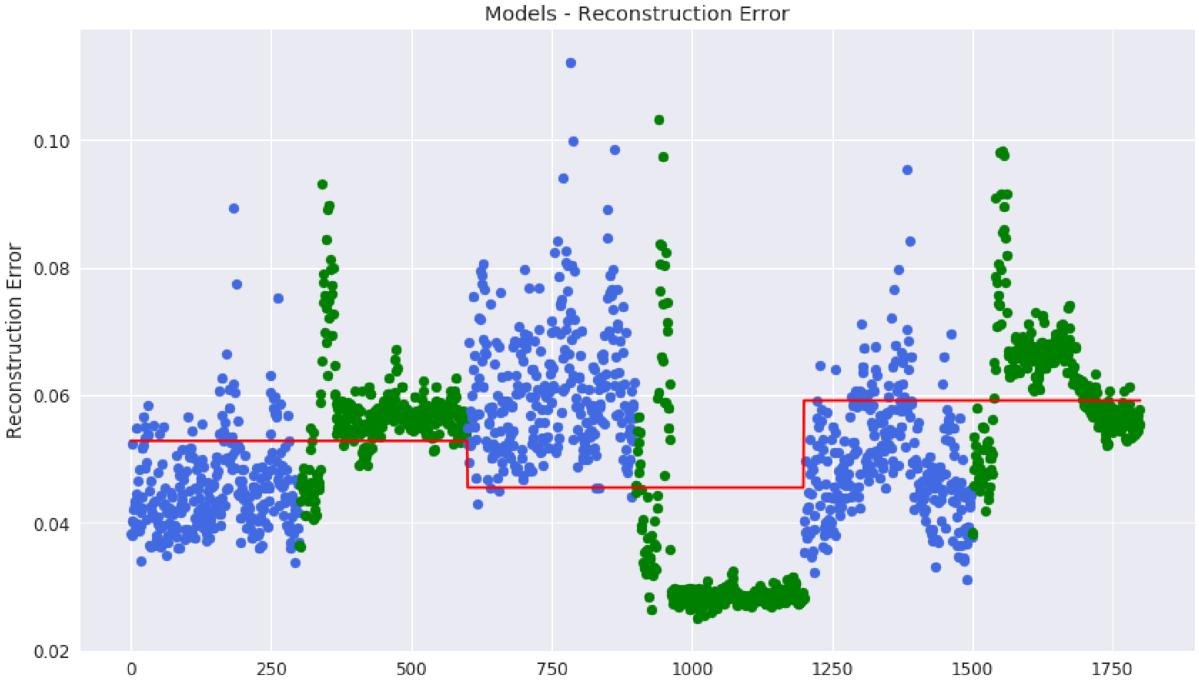


In [97]:

```
error_threshold_plot = error_df[['error', 'threshold']].copy()

error_threshold_plot_temp1 = error_second_model[['error', 'threshold']].
error_threshold_plot_temp2 = error_third_model[['error', 'threshold']].c


error_threshold_plot = pd.concat([error_threshold_plot , error_threshold_
error_threshold_plot = error_threshold_plot.reset_index(drop=True)


error_threshold_plot_1 = error_threshold_plot[0:300].copy()
error_threshold_plot_2 = error_threshold_plot[300:600].copy()


error_threshold_plot_temp1_1 = error_threshold_plot[600:900].copy()
error_threshold_plot_temp1_2 = error_threshold_plot[900:1200].copy()


error_threshold_plot_temp2_1 = error_threshold_plot[1200:1500].copy()
error_threshold_plot_temp2_2 = error_threshold_plot[1500:].copy()


plt.scatter(error_threshold_plot_1.index, error_threshold_plot_1['error'
plt.scatter(error_threshold_plot_2.index, error_threshold_plot_2['error'

plt.scatter(error_threshold_plot_temp1_1.index, error_threshold_plot_tem
plt.scatter(error_threshold_plot_temp1_2.index, error_threshold_plot_tem

plt.scatter(error_threshold_plot_temp2_1.index, error_threshold_plot_tem
plt.scatter(error_threshold_plot_temp2_2.index, error_threshold_plot_tem

plt.ylabel('Reconstruction Error')

plt.title("Models - Reconstruction Error")
plt.plot(error_threshold_plot['threshold'], color='red')
```

Out[97]:

```
[<matplotlib.lines.Line2D at 0x7f0a5a8d96d8>]
```


Models - Reconstruction Error
Reconstruction Error
0.10
0.08
0.06
0.04
0.02
0
250
500
750
1000
1250
1500
1750

```
In [ ]:
```

```
In [ ]:
```