In [1]:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

rawData = pd.read_csv('data.csv')
```

In [2]:

```
#data preprocessing
infoData = pd.DataFrame()
infoData['FLAG'] = rawData['FLAG']
infoData['CONS_NO'] = rawData['CONS_NO']
data = rawData.drop(['FLAG', 'CONS_NO'], axis=1)   #axis 1 column ,axis 0 row
```

In [3]:

```
#droping duplicate row
dropIndex = data[data.duplicated()].index  # duplicates drop
data = data.drop(dropIndex, axis=0)   #droping duplicate value present wen two r
infoData = infoData.drop(dropIndex, axis=0) #droping duplicate index infodata
```

In [4]:

```
#removing row with all zero(Nan) value
zeroIndex = data[(data.sum(axis=1) == 0)].index  # zero rows drop
data = data.drop(zeroIndex, axis=0)
infoData = infoData.drop(zeroIndex, axis=0)
```

In [5]:

```
#change column name to dates(2014/1/1 to 2014-01-01)
data.columns = pd.to_datetime(data.columns)  #columns reindexing according to da

#sort data accoding to date( as previusoly column are unsorted)
data = data.reindex(sorted(data.columns), axis=1)
cols = data.columns
```

In [6]:

```
# reindex row name (as some row has been remove till this step due to duplicate 
data.reset_index(inplace=True, drop=True)  # index sorting
infoData.reset_index(inplace=True, drop=True)

#filling nan value using neighbouring value (middle missing value replace by ave
#and other by maximum 2 distance element)
data = data.interpolate(method='linear', limit=2, limit_direction='both', axis=0
```

In [7]:
```
print(data)
```

```
       2014-01-01  2014-01-02  2014-01-03  2014-01-04  2014-01-05  2014-01-06
\
0           0.000       0.000        0.00       0.000    0.000000       0.000
1           2.900       5.640        6.99       3.320    3.610000       5.350
2           2.900       5.640        6.99       3.320    3.610000       5.350
3           2.900       5.640        6.99       3.320    3.610000       5.350
4           1.505       2.875        3.62       1.795    1.910000       2.775
...           ...         ...         ...         ...         ...         ...
40251       3.540       1.680        1.64       5.440    8.563333       7.450
40252       2.700       0.000        0.00       5.720    6.050000       5.810
40253       0.580       1.160        0.92       0.980    1.540000       1.380
40254      16.890      15.150       19.28      17.190   16.800000      17.480
40255      16.890      15.150       19.28      17.190   16.800000      17.480

       2014-01-07  2014-01-08  2014-01-09  2014-01-10  ...  2016-10-22  \
0        0.000000    0.000000    0.000000    0.000000  ...        7.18
1        4.730000    3.680000    3.530000    3.420000  ...       10.95
2        4.730000    3.680000    3.530000    3.420000  ...       12.81
3        4.730000    3.680000    3.530000    3.420000  ...       14.21
4        2.435000    2.010000    1.880000    1.975000  ...        2.51
...           ...         ...         ...         ...  ...         ...
40251    3.646667    4.806667    6.143333    2.926667  ...        3.27
40252    3.070000    4.040000    5.680000    4.390000  ...        3.84
40253    0.890000    0.700000    1.230000    0.840000  ...        0.99
40254   17.860000   23.990000   12.340000   13.840000  ...       15.64
40255   17.860000   23.990000   12.340000   13.840000  ...       10.56

       2016-10-23  2016-10-24  2016-10-25  2016-10-26  2016-10-27  2016-10-28
\
0            8.07        8.09        9.53        5.48        8.75        9.30
1           17.95       17.83       17.31       21.44       19.09       18.56
2           15.12       17.26       14.91       19.59       20.79       17.95
3           10.22        8.47        6.11        6.10        6.73        7.52
4            2.97        2.93        0.74        0.41        0.42        1.91
...           ...         ...         ...         ...         ...         ...
40251        3.10        2.75        3.01        2.99        2.83        2.54
40252        6.62        3.12        5.16        3.62        4.64        3.71
40253        0.61        0.65        0.55        0.49        0.51        0.79
40254       16.48       13.04       10.39       12.00       11.15       12.22
40255       17.14        8.35        8.68        6.39        7.96        8.13

       2016-10-29  2016-10-30  2016-10-31
0            7.54        9.16        6.74
1           16.25       14.20       13.66
2           19.26       14.46       11.72
3           10.89        9.86        8.72
4            0.42        0.38        0.61
...           ...         ...         ...
40251        3.40        3.59        2.54
40252        6.22        6.05        4.77
40253        0.66        0.39        0.65
40254       13.16       13.33       10.39
40255       11.50        7.16        5.25
```

```
[40256 rows x 1034 columns]
```

In [8]:

```
#removing erronoues value(fixing outliers)
for i in range(data.shape[0]):  # outliers treatment
    m = data.loc[i].mean()
    st = data.loc[i].std()
    data.loc[i] = data.loc[i].mask(data.loc[i] > (m + 2 * st), other=m + 2 * st)
```

In [9]:

```
# save preprocessed data without scaling
data.to_csv(r'visualization.csv', index=False, header=True)  # preprocessed data

#noramalisation process
scale = MinMaxScaler()
scaled = scale.fit_transform(data.values.T).T
mData = pd.DataFrame(data=scaled, columns=data.columns)
preprData = pd.concat([infoData, mData], axis=1, sort=False)  # Back to initial
print("Noramalised data")
print(preprData)

# save preprocessed data after scaling
preprData.to_csv(r'preprocessedR.csv', index=False, header=True)
```

```
Noramalised data
       FLAG                           CONS_NO  2014-01-01 00:00:00  \
0         1  0387DD8A07E07FDA6271170F86AD9151             0.000000
1         1  4B75AC4F2D8434CFF62DB64D0BB43103             0.140053
2         1  B32AC8CC6D5D805AC053557AB05F5343             0.102224
3         1  EDFC78B07BA2908B3395C4EB2304665E             0.144182
4         1  6BCFD78138BC72A9BA1BFB0B79382192             0.178706
...     ...                               ...                  ...
40251     0  F1472871E1AFF49D4289564B6377D76C             0.410818
40252     0  F3C8BBCD2DC26C1E0249DEEF6A4256B7             0.332902
40253     0  A9A0FE83467A680FBFB0DBFC910DF227             0.107701
40254     0  D9A6ADA018FA46A55D5438370456AA45             0.527563
40255     0  F3406636BAD1E6E0826E8EDDC9A1BF00             0.570594

       2014-01-02 00:00:00  2014-01-03 00:00:00  2014-01-04 00:00:00  \
0                 0.000000             0.000000             0.000000
1                 0.272379             0.337576             0.160336
2                 0.198809             0.246396             0.117029
3                 0.280408             0.347527             0.165063
4                 0.341383             0.429846             0.213142
...                    ...                  ...                  ...
40251             0.194965             0.190323             0.631314
40252             0.000000             0.000000             0.705259
40253             0.215401             0.170836             0.181977
40254             0.473214             0.602215             0.536934
40255             0.511812             0.651335             0.580729

       2014-01-05 00:00:00  2014-01-06 00:00:00  2014-01-07 00:00:00  \
0                 0.000000             0.000000             0.000000
1                 0.174342             0.258373             0.228431
2                 0.127252             0.188586             0.166731
3                 0.179481             0.265990             0.235165
4                 0.226797             0.329509             0.289136
...                    ...                  ...                  ...
40251             0.993778             0.864575             0.423197
40252             0.745947             0.716356             0.378522
40253             0.285964             0.256253             0.165265
40254             0.524752             0.545992             0.557861
40255             0.567553             0.590526             0.603363

       2014-01-08 00:00:00  ...  2016-10-22 00:00:00  2016-10-23 00:00:00  \
0                 0.000000  ...             0.265238             0.298116
```

```
1                 0.177722  ...             0.528820             0.866879
2                 0.129719  ...             0.451550             0.532977
3                 0.182961  ...             0.706489             0.508116
4                 0.238671  ...             0.298042             0.352663
...                    ...  ...                  ...                  ...
40251             0.557815  ...             0.379485             0.359756
40252             0.498120  ...             0.473460             0.816226
40253             0.129984  ...             0.183834             0.113271
40254             0.749333  ...             0.488519             0.514757
40255             0.810453  ...             0.356748             0.579040

       2016-10-24 00:00:00  2016-10-25 00:00:00  2016-10-26 00:00:00  \
0                 0.298855             0.352051             0.202438   
1                 0.861084             0.835971             1.000000   
2                 0.608411             0.525574             0.690543   
3                 0.421110             0.303776             0.303278   
4                 0.347914             0.087869             0.048684   
...                    ...                  ...                  ...   
40251             0.319138             0.349311             0.346990   
40252             0.384687             0.636212             0.446335   
40253             0.120699             0.102130             0.090989   
40254             0.407308             0.324534             0.374823   
40255             0.282088             0.293236             0.215873   

       2016-10-27 00:00:00  2016-10-28 00:00:00  2016-10-29 00:00:00  \
0                 0.323236             0.343554             0.278537   
1                 0.921934             0.896338             0.784779   
2                 0.732843             0.632733             0.678911   
3                 0.334601             0.373878             0.541427   
4                 0.049872             0.226797             0.049872   
...                    ...                  ...                  ...   
40251             0.328422             0.294768             0.394571   
40252             0.572098             0.457432             0.766907   
40253             0.094702             0.146696             0.122556   
40254             0.348273             0.381695             0.411056   
40255             0.268912             0.274655             0.388504   

       2016-10-30 00:00:00  2016-10-31 00:00:00  
0                 0.338382             0.248984  
1                 0.685776             0.659697  
2                 0.509712             0.413127  
3                 0.490217             0.433539  
4                 0.045122             0.072433  
...                    ...                  ...  
40251             0.416621             0.294768  
40252             0.745947             0.588127  
40253             0.072419             0.120699  
40254             0.416366             0.324534  
40255             0.241886             0.177360  

[40256 rows x 1036 columns]
```

In [35]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_absolute_error, mean_squared_er
    precision_recall_fscore_support, roc_auc_score
from tensorflow.keras import Sequential
import tensorflow as tf
from tensorflow import keras
from tensorflow.python.keras.layers import Dense, Conv1D, Flatten, Conv2D
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
import numpy as np
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE

tf.random.set_seed(1234)
epochs_number = 1  # number of epochs for the neural networks
test_set_size = 0.05  # percentage of the test size comparing to the whole datase
oversampling_flag = 0  # set to 1 to over-sample the minority class
oversampling_percentage = 0.2  # percentage of the minority class after the overs
```

In [36]:

```
# Definition of functions
def read_data():
    rawData = pd.read_csv('preprocessedR.csv')

    # Setting the target and dropping the unnecessary columns
    y = rawData[['FLAG']]
    X = rawData.drop(['FLAG', 'CONS_NO'], axis=1)

    print('Normal Consumers:                    ', y[y['FLAG'] == 0].count()[0])
    print('Consumers with Fraud:                ', y[y['FLAG'] == 1].count()[0])
    print('Total Consumers:                     ', y.shape[0])
    print("percentage of normal Consumers:     %.2f" % (y[y['FLAG'] == 0].count(

    # columns reindexing according to dates
    X.columns = pd.to_datetime(X.columns)
    X = X.reindex(X.columns, axis=1)

    # Splitting the dataset into training set and test set
    X_train, X_test, y_train, y_test = train_test_split(X, y['FLAG'], test_size=
    print("percentage of normal Consumers in test set:          %.2f" % (y_test

    # Oversampling of minority class to encounter the imbalanced learning
    if oversampling_flag == 1:
        over = SMOTE(sampling_strategy=oversampling_percentage, random_state=0)
        X_train, y_train = over.fit_resample(X_train, y_train)
        print("Oversampling statistics in training set: ")
        print('Normal Consumers:                    ', y_train[y_train == 0].cou
        print('Consumers with Fraud:                ', y_train[y_train == 1].cou
        print("Total Consumers                      ", X_train.shape[0])

    return X_train, X_test, y_train, y_test


def results(y_test, prediction):
    print("Accuracy", 100 * accuracy_score(y_test, prediction))
    print("RMSE:", mean_squared_error(y_test, prediction, squared=False))
    print("MAE:", mean_absolute_error(y_test, prediction))
    print("F1:", 100 * precision_recall_fscore_support(y_test, prediction)[2])
    print("AUC:", 100 * roc_auc_score(y_test, prediction))
    #print(confusion_matrix(y_test, prediction), "\n")
```

In [37]:

```
def ANN(X_train, X_test, y_train, y_test):
    print('Artificial Neural Network:')
    # for i in range(4,100,3):
    #     print("Epoch:",i)

    # Model creation
    model = Sequential()
    model.add(Dense(1000, input_dim=1034, activation='relu'))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss=keras.losses.binary_crossentropy,
                  optimizer='adam',
                  metrics=['accuracy'])

    # model.fit(X_train, y_train, validation_split=0, epochs=i, shuffle=True, ve
    model.fit(X_train, y_train, validation_split=0, epochs=epochs_number, shuffl
    prediction = model.predict_classes(X_test)
    model.summary()
    results(y_test, prediction)


def CNN1D(X_train, X_test, y_train, y_test):
    print('1D - Convolutional Neural Network:')

    # Transforming the dataset into tensors
    X_train = X_train.to_numpy().reshape(X_train.shape[0], X_train.shape[1], 1)
    X_test = X_test.to_numpy().reshape(X_test.shape[0], X_test.shape[1], 1)

    # Model creation
    model = Sequential()
    model.add(Conv1D(100, kernel_size=7, input_shape=(1034, 1), activation='relu
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss=keras.losses.binary_crossentropy,
                  optimizer='adam',
                  metrics=['accuracy'])

    # model.fit(X_train, y_train, epochs=1, validation_split=0.1, shuffle=False,
    model.fit(X_train, y_train, epochs=epochs_number, validation_split=0, shuffl
    prediction = model.predict_classes(X_test)
    model.summary()
    results(y_test, prediction)


def CNN2D(X_train, X_test, y_train, y_test):
    print('2D - Convolutional Neural Network:')

    # Transforming every row of the train set into a 2D array and then into a te
    n_array_X_train = X_train.to_numpy()
    n_array_X_train_extended = np.hstack((n_array_X_train, np.zeros(
        (n_array_X_train.shape[0], 2))))  # adding two empty columns in order to
    # an exact multiple of 7
    week = []
    for i in range(n_array_X_train_extended.shape[0]):
        a = np.reshape(n_array_X_train_extended[i], (-1, 7, 1))
        week.append(a)
    X_train_reshaped = np.array(week)

    # Transforming every row of the train set into a 2D array and then into a te
    n_array_X_test = X_test.to_numpy()  # X_test to 2D - array
    n_array_X_train_extended = np.hstack((n_array_X_test, np.zeros((n_array_X_te
    week2 = []
    for i in range(n_array_X_train_extended.shape[0]):
        b = np.reshape(n_array_X_train_extended[i], (-1, 7, 1))
        week2.append(b)
    X_test_reshaped = np.array(week2)

    input_shape = (1, 148, 7, 1)  # input shape of the tensor

    # Model creation
    model = Sequential()
    model.add(Conv2D(kernel_size=(7, 3), filters=32, input_shape=input_shape[1:]
                     data_format='channels_last'))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss=keras.losses.binary_crossentropy,
                  optimizer='adam',
                  metrics=['accuracy'])
#     model.fit(X_train_reshaped, y_train, validation_split=0.1, epochs=i, s
    model.fit(X_train_reshaped, y_train, validation_split=0.1, epochs=epochs_num

    prediction = model.predict_classes(X_test_reshaped)
    model.summary()
    results(y_test, prediction)


def LR(X_train, X_test, y_train, y_test):
    print('Logistic Regression:')
    model = LogisticRegression(C=1000, max_iter=1000, n_jobs=-1, solver='newton-
    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    results(y_test, prediction)


def DT(X_train, X_test, y_train, y_test):
    print('Decision Tree:')
    model = DecisionTreeClassifier(random_state=0)
    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    results(y_test, prediction)
```

```
def RF(X_train, X_test, y_train, y_test):
    print('Random Forest:')
    model = RandomForestClassifier(n_estimators=100, min_samples_leaf=1, max_feat
                                   random_state=0, n_jobs=-1)
    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    results(y_test, prediction)


def SVM(X_train, X_test, y_train, y_test):
    model = SVC(random_state=0)
    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    results(y_test, prediction)
```

In [ ]:

```
def Combined(X_train, X_test, y_train, y_test):
    print('2D - Convolutional Neural Network:')

    # Transforming every row of the train set into a 2D array and then into a te
    n_array_X_train = X_train.to_numpy()
    n_array_X_train_extended = np.hstack((n_array_X_train, np.zeros(
        (n_array_X_train.shape[0], 2))))  # adding two empty columns in order to
    # an exact multiple of 7
    week = []
    for i in range(n_array_X_train_extended.shape[0]):
        a = np.reshape(n_array_X_train_extended[i], (-1, 7, 1))
        week.append(a)
    X_train_reshaped = np.array(week)

    # Transforming every row of the train set into a 2D array and then into a te
    n_array_X_test = X_test.to_numpy()  # X_test to 2D - array
    n_array_X_train_extended = np.hstack((n_array_X_test, np.zeros((n_array_X_te
    week2 = []
    for i in range(n_array_X_train_extended.shape[0]):
        b = np.reshape(n_array_X_train_extended[i], (-1, 7, 1))
        week2.append(b)
    X_test_reshaped = np.array(week2)

    input_shape = (1, 148, 7, 1)  # input shape of the tensor# Model creation
    model = Sequential()
    model.add(Conv2D(kernel_size=(7, 3), filters=32, input_shape=input_shape[1:],
                     data_format='channels_last'))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss=keras.losses.binary_crossentropy,
                  optimizer='adam',
                  metrics=['accuracy'])
    #     model.fit(X_train_reshaped, y_train, validation_split=0.1, epochs=i, s
    model.fit(X_train_reshaped, y_train, validation_split=0.1, epochs=epochs_num
    prediction = model.predict_classes(X_test_reshaped)
    model.summary()
    results(y_test, prediction)
```

In [38]:

```
# ----Main----
X_train, X_test, y_train, y_test = read_data()
```

```
Normal Consumers:                     36677
Consumers with Fraud:                 3579
Total Consumers:                      40256
percentage of normal Consumers:     91.11 %
percentage of normal Consumers in test set:          91.11 %
```

In [37]:
```
CNN1D(X_train, X_test, y_train, y_test)
```

```
1D - Convolutional Neural Network:
1196/1196 [==============================] - 131s 109ms/step - loss: 0.2792 - a
ccuracy: 0.9073

C:\Users\HP\AppData\Roaming\Python\Python37\site-packages\tensorflow\python\ker
as\engine\sequential.py:450: UserWarning: `model.predict_classes()` is deprecat
ed and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.
predict(x), axis=-1)`,   if your model does multi-class classification   (e.g.
if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astyp
e("int32")`,   if your model does binary classification   (e.g. if it uses a `s
igmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '

Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_1 (Conv1D)            (None, 1028, 100)         800       
_________________________________________________________________
flatten_2 (Flatten)          (None, 102800)            0         
_________________________________________________________________
dense_20 (Dense)             (None, 100)               10280100  
_________________________________________________________________
dense_21 (Dense)             (None, 100)               10100     
_________________________________________________________________
dense_22 (Dense)             (None, 64)                6464      
_________________________________________________________________
dense_23 (Dense)             (None, 1)                 65        
=================================================================
Total params: 10,297,529
Trainable params: 10,297,529
Non-trainable params: 0
_________________________________________________________________
Accuracy 92.05166418281172
RMSE: 0.2819279308119058
MAE: 0.07948335817188276
F1: [95.80052493 25.92592593]
AUC: 57.57586372857813
[[1825    9]
 [ 151   28]]
```

In [38]:
```
CNN2D(X_train, X_test, y_train, y_test)
```

```
2D - Convolutional Neural Network:
1076/1076 [==============================] - 31s 28ms/step - loss: 0.2894 - a
ccuracy: 0.9082 - val_loss: 0.2474 - val_accuracy: 0.9137

C:\Users\HP\AppData\Roaming\Python\Python37\site-packages\tensorflow\python\k
eras\engine\sequential.py:450: UserWarning: `model.predict_classes()` is depr
ecated and will be removed after 2021-01-01. Please use instead:* `np.argmax
(model.predict(x), axis=-1)`,   if your model does multi-class classification
(e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.
5).astype("int32")`,   if your model does binary classification   (e.g. if it
uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '

Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 142, 5, 32)        704
```

In [39]:
```
RF(X_train, X_test, y_train, y_test)
```

```
Random Forest:
Accuracy 91.2568306010929
RMSE: 0.29568850838182914
MAE: 0.08743169398907104
F1: [95.421436   3.2967033]
AUC: 50.83798882681564
[[1834    0]
 [ 176    3]]
```

In [39]:
```
LR(X_train, X_test, y_train, y_test)
```

```
Logistic Regression:
Accuracy 90.80973671137606
RMSE: 0.3031544703385379
MAE: 0.09190263288623944
F1: [95.13797635 16.28959276]
AUC: 54.37362543635732
```

In [40]:
```
DT(X_train, X_test, y_train, y_test)
```

```
Decision Tree:
Accuracy 84.45106805762543
RMSE: 0.39432134030983623
MAE: 0.15548931942374566
F1: [91.37978518 20.75949367]
AUC: 56.68152160006823
[[1659  175]
 [ 138   41]]
```

In [40]:

```
SVM(X_train, X_test, y_train, y_test)
```

```
Accuracy 91.2568306010929
RMSE: 0.29568850838182914
MAE: 0.08743169398907104
F1: [95.421436   3.2967033]
AUC: 50.83798882681564
```

In [20]:

```
def Combined(X_train, X_test, y_train, y_test):
    print('2D - Convolutional Neural Network:')
    # Transforming every row of the train set into a 2D array and then into a te
    n_array_X_train = X_train.to_numpy()
    n_array_X_train_extended = np.hstack((n_array_X_train, np.zeros(
        (n_array_X_train.shape[0], 2))))  # adding two empty columns in order to
    # an exact multiple of 7
    week = []
    for i in range(n_array_X_train_extended.shape[0]):
        a = np.reshape(n_array_X_train_extended[i], (-1, 7, 1))
        week.append(a)
    X_train_reshaped = np.array(week)
    # Transforming every row of the train set into a 2D array and then into a te
    n_array_X_test = X_test.to_numpy()  # X_test to 2D - array
    n_array_X_train_extended = np.hstack((n_array_X_test, np.zeros((n_array_X_te
    week2 = []
    for i in range(n_array_X_train_extended.shape[0]):
        b = np.reshape(n_array_X_train_extended[i], (-1, 7, 1))
        week2.append(b)
    X_test_reshaped = np.array(week2)
    input_shape = (1, 148, 7, 1)  # input shape of the tensor# Model creation
    model1 = Sequential()
    model1.add(Conv2D(kernel_size=(7, 3), filters=32, input_shape=input_shape[1:
                     data_format='channels_last'))
    model1.add(Flatten())
    model1.add(Dense(100, activation='relu'))
    model1.add(Dense(100, activation='relu'))
    model1.add(Dense(64, activation='relu'))
    model1.add(Dense(1, activation='sigmoid'))
    model1.compile(loss=keras.losses.binary_crossentropy,
                  optimizer='adam',
                  metrics=['accuracy'])
    #     model.fit(X_train_reshaped, y_train, validation_split=0.1, epochs=i, sh
    model1.fit(X_train_reshaped, y_train, validation_split=0.1, epochs=epochs_nu
    prediction1 = model1.predict_classes(X_test_reshaped)
    print(prediction1)
    print('1D - Convolutional Neural Network:')
    # Transforming the dataset into tensors
    X_train = X_train.to_numpy().reshape(X_train.shape[0], X_train.shape[1], 1)
    X_test = X_test.to_numpy().reshape(X_test.shape[0], X_test.shape[1], 1)
    # Model creation
    model2 = Sequential()
    model2.add(Conv1D(100, kernel_size=7, input_shape=(1034, 1), activation='rel
    model2.add(Flatten())
    model2.add(Dense(100, activation='relu'))
    model2.add(Dense(100, activation='relu'))
    model2.add(Dense(64, activation='relu'))
    model2.add(Dense(1, activation='sigmoid'))
    model2.compile(loss=keras.losses.binary_crossentropy,
                  optimizer='adam',
                  metrics=['accuracy'])
    # model.fit(X_train, y_train, epochs=1, validation_split=0.1, shuffle=False,
    model2.fit(X_train, y_train, epochs=epochs_number, validation_split=0, shuff
    prediction2 = model2.predict_classes(X_test)
    return prediction1 , prediction2
```

In [21]:

```
prediction1,prediction2=Combined(X_train, X_test, y_train, y_test) ;
```

```
2D - Convolutional Neural Network:
1076/1076 [==============================] - 40s 36ms/step - loss: 0.2936 - acc
uracy: 0.9049 - val_loss: 0.2559 - val_accuracy: 0.9145
[[0]
 [0]
 [0]
 ...
 [0]
 [0]
 [0]]
1D - Convolutional Neural Network:
1196/1196 [==============================] - 177s 147ms/step - loss: 0.2786 - a
ccuracy: 0.9125 - loss: 0.2788
```

In [30]:

```
def deep_and_wideCNN(prediction1,prediction2):
    n=len(prediction2)
    for i in range(n) :
        if(prediction1[i]!=prediction2[i]):
            prediction1[i]=0
    results(y_test, prediction1)
```

In [31]:

```
deep_and_wideCNN(prediction1,prediction2)
```

```
Accuracy 97.55489319423745
RMSE: 0.3906046593873289
MAE: 0.13445106805762544
F1: [98.56367432 15.37113402]
AUC: 80.2701668666955
```

In [ ]: