In [1]:

```python
import numpy as np
import tensorflow as tf
import keras
import scipy.io
import sklearn
import matplotlib
import matplotlib.pyplot as plt
import keras.backend as K

from matplotlib import pyplot as plt

from keras.optimizers import Adam
from keras.models import Model
from keras.layers import AveragePooling1D,MaxPooling1D,UpSampling1D,AveragePooling2D,MaxPooling2D,
UpSampling2D
from keras.models import Sequential
from keras.layers import Input,Conv1D,Conv2D,Conv2DTranspose,Reshape
from keras.layers import Activation,Dense,Dropout,BatchNormalization
from keras.layers import Flatten,Lambda
from keras.utils import np_utils
from keras.callbacks import EarlyStopping

from sklearn import preprocessing,metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
```

```
Using TensorFlow backend.
```

In [2]:

```python
pathAddress="C:\\Users\\HP\\Desktop\\KaunsiElectivesLenaHai\\Neural Network and fuzzy logic BITS F
312"
arr=scipy.io.loadmat(pathAddress+"\\data_for_cnn.mat")
df=arr['ecg_in_window']
label=scipy.io.loadmat(pathAddress+"\\class_label.mat")
labels=label['label']
```

In [3]:

```python
normal=preprocessing.StandardScaler()
df=normal.fit_transform(df)

X=np.expand_dims(df,axis=2)
Y=labels

x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=1)

#print(x_train.shape,x_test.shape)#(800, 1000, 1) (200, 1000, 1)
#print("length of x_train :",len(x_train))# 800
#print("length of y_train :",len(y_train))# 800

#print(x_train)

#print(y_train.shape,y_test.shape)#(800,1) (200,1)
#print(y_train)
y_train=y_train.reshape(y_train.shape[0],y_train.shape[1],1)
y_test=y_test.reshape(y_test.shape[0],y_test.shape[1],1)
#print("length of x_train :",len(x_train))# 800
#print("length of y_train :",len(y_train))# 800
#print(y_train.shape,y_test.shape)#(800, 1, 1) (200, 1, 1)
```

In [4]:

```python
import keras.backend as K

#def Conv1DTranspose(input_tensor, filters, kernel_size, strides=2, padding='same'):
#    x = Lambda(lambda x: K.expand_dims(x, axis=2))(input_tensor)
#    x = Conv2DTranspose(filters=filters, kernel_size=(kernel_size, 1), strides=(strides, 1), padd
```

```
#     x = Conv2DTranspose(filters=filters, kernel_size=(kernel_size, 1), strides=(strides, 1), padd
ing=padding)(x)
#     x = Lambda(lambda x: K.squeeze(x, axis=2))(x)
#     return x
```

In [5]:

```
def build_model(img_shape):
        input_layer = Input(shape=img_shape)

        # encoder
        h = Conv1D(50,1000,activation='relu',input_shape=(1000,1))(input_layer)
        h = MaxPooling1D(pool_size=1, padding='same')(h)
        #h = Flatten()(h)
        #h = Dropout(0.2)(h)

        # decoder
        h = Dense(100,activation='sigmoid')(h)
        h = Dropout(0.2)(h)
        h = UpSampling1D(size=1)(h)

        kernel_size=1
        filters=1
        strides=1
        padding='same'

        h = Lambda(lambda x: K.expand_dims(x, axis=2))(h)
        h = Conv2DTranspose(filters=filters, kernel_size=(kernel_size, 1), strides=(strides, 1), pa
dding=padding)(h)
        output_layer = Lambda(lambda x: K.squeeze(x, axis=2))(h)


        return Model(input_layer, output_layer)
```

In [6]:

```
img_rows = 800
img_cols = 1000
channels = 1
#img_shape = (img_rows, img_cols, channels)
img_shape = ( img_cols, channels)

optimizer = Adam(lr=0.001)

autoencoder_model = build_model(img_shape)
autoencoder_model.compile(loss='mse', optimizer=optimizer,metrics=['mean_squared_error','acc'])
autoencoder_model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 1000, 1)           0         
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1, 50)             50050     
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 1, 50)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 1, 100)            5100      
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 100)            0         
_________________________________________________________________
up_sampling1d_1 (UpSampling1 (None, 1, 100)            0         
_________________________________________________________________
lambda_1 (Lambda)            (None, 1, 1, 100)         0         
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 1, 1, 1)           101       
_________________________________________________________________
lambda_2 (Lambda)            (None, 1, 1)              0         
=================================================================
Total params: 55,251
Trainable params: 55,251
Non-trainable params: 0
```

In [7]:

```
def train_model(autoencoder_model,x_train, y_train, epochs, batch_size=20):
    history=[]
    early_stopping = EarlyStopping(monitor='val_loss',
                                   min_delta=0,
                                   patience=5,
                                   verbose=1,
                                   mode='auto')


    history.append(autoencoder_model.fit(x_train,y_train ,
            epochs=epochs,
            callbacks=[early_stopping]))

    print(history[0].history.keys())

    plt.plot(history[0].history['loss'])

    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

    plt.figure()
    plt.plot(history[0].history['acc'])

    plt.title('Model Accuracy')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()
```

In [8]:

```
def eval_model(autoencoder_model,x_test):
    preds = autoencoder_model.predict(x_test)
    return preds
```

In [9]:

```
numEpochs=300
train_model(autoencoder_model,x_train,y_train,epochs=numEpochs,batch_size=20)
```

```
Epoch 1/300
800/800 [==============================] - 1s 642us/step - loss: 0.3763 - mean_squared_error: 0.37
63 - acc: 0.5275
Epoch 2/300
800/800 [==============================] - 0s 133us/step - loss: 0.3551 - mean_squared_error: 0.35
51 - acc: 0.5512
Epoch 3/300
288/800 [=========>....................] - ETA: 0s - loss: 0.3103 - mean_squared_error: 0.3103 - a
cc: 0.5938
```

```
C:\Users\HP\AppData\Roaming\Python\Python37\site-packages\keras\callbacks\callbacks.py:846:
RuntimeWarning: Early stopping conditioned on metric `val_loss` which is not available. Available
metrics are: loss,mean_squared_error,acc
  (self.monitor, ','.join(list(logs.keys()))), RuntimeWarning
```

```
800/800 [==============================] - 0s 161us/step - loss: 0.3050 - mean_squared_error: 0.30
50 - acc: 0.5850
Epoch 4/300
800/800 [==============================] - 0s 121us/step - loss: 0.3009 - mean_squared_error: 0.30
09 - acc: 0.6112
Epoch 5/300
800/800 [==============================] - 0s 120us/step - loss: 0.2615 - mean_squared_error: 0.26
15 - acc: 0.6625
```

```
Epoch 6/300
800/800 [==============================] - 0s 119us/step - loss: 0.2296 - mean_squared_error: 0.22
96 - acc: 0.6812
Epoch 7/300
800/800 [==============================] - 0s 116us/step - loss: 0.1911 - mean_squared_error: 0.19
11 - acc: 0.7237
Epoch 8/300
800/800 [==============================] - 0s 124us/step - loss: 0.1933 - mean_squared_error: 0.19
33 - acc: 0.7412
Epoch 9/300
800/800 [==============================] - 0s 114us/step - loss: 0.1511 - mean_squared_error: 0.15
11 - acc: 0.8000
Epoch 10/300
800/800 [==============================] - 0s 102us/step - loss: 0.1410 - mean_squared_error: 0.14
10 - acc: 0.8238
Epoch 11/300
800/800 [==============================] - 0s 116us/step - loss: 0.1226 - mean_squared_error: 0.12
26 - acc: 0.8525
Epoch 12/300
800/800 [==============================] - 0s 118us/step - loss: 0.1093 - mean_squared_error: 0.10
93 - acc: 0.8650
Epoch 13/300
800/800 [==============================] - 0s 110us/step - loss: 0.1043 - mean_squared_error: 0.10
43 - acc: 0.8800
Epoch 14/300
800/800 [==============================] - 0s 124us/step - loss: 0.0891 - mean_squared_error: 0.08
91 - acc: 0.8975
Epoch 15/300
800/800 [==============================] - 0s 119us/step - loss: 0.0832 - mean_squared_error: 0.08
32 - acc: 0.9150
Epoch 16/300
800/800 [==============================] - 0s 119us/step - loss: 0.0811 - mean_squared_error: 0.08
11 - acc: 0.9225
Epoch 17/300
800/800 [==============================] - 0s 121us/step - loss: 0.0716 - mean_squared_error: 0.07
16 - acc: 0.9362
Epoch 18/300
800/800 [==============================] - 0s 119us/step - loss: 0.0735 - mean_squared_error: 0.07
35 - acc: 0.9300
Epoch 19/300
800/800 [==============================] - 0s 119us/step - loss: 0.0656 - mean_squared_error: 0.06
56 - acc: 0.9475
Epoch 20/300
800/800 [==============================] - 0s 164us/step - loss: 0.0612 - mean_squared_error: 0.06
12 - acc: 0.9538
Epoch 21/300
800/800 [==============================] - 0s 155us/step - loss: 0.0568 - mean_squared_error: 0.05
68 - acc: 0.9638
Epoch 22/300
800/800 [==============================] - 0s 158us/step - loss: 0.0536 - mean_squared_error: 0.05
36 - acc: 0.9663
Epoch 23/300
800/800 [==============================] - 0s 156us/step - loss: 0.0529 - mean_squared_error: 0.05
29 - acc: 0.9600
Epoch 24/300
800/800 [==============================] - 0s 181us/step - loss: 0.0496 - mean_squared_error: 0.04
96 - acc: 0.9712
Epoch 25/300
800/800 [==============================] - 0s 182us/step - loss: 0.0510 - mean_squared_error: 0.05
10 - acc: 0.9638
Epoch 26/300
800/800 [==============================] - 0s 129us/step - loss: 0.0503 - mean_squared_error: 0.05
03 - acc: 0.9650
Epoch 27/300
800/800 [==============================] - 0s 120us/step - loss: 0.0435 - mean_squared_error: 0.04
35 - acc: 0.9850
Epoch 28/300
800/800 [==============================] - 0s 118us/step - loss: 0.0425 - mean_squared_error: 0.04
25 - acc: 0.9787
Epoch 29/300
800/800 [==============================] - 0s 113us/step - loss: 0.0399 - mean_squared_error: 0.03
99 - acc: 0.9800
Epoch 30/300
800/800 [==============================] - 0s 113us/step - loss: 0.0373 - mean_squared_error: 0.03
73 - acc: 0.9850
Epoch 31/300
800/800 [==============================] - 0s 115us/step - loss: 0.0378 - mean_squared_error: 0.03
```

```
78 - acc: 0.9912
Epoch 32/300
800/800 [==============================] - 0s 124us/step - loss: 0.0363 - mean_squared_error: 0.03
63 - acc: 0.9862
Epoch 33/300
800/800 [==============================] - 0s 117us/step - loss: 0.0340 - mean_squared_error: 0.03
40 - acc: 0.9825
Epoch 34/300
800/800 [==============================] - 0s 116us/step - loss: 0.0356 - mean_squared_error: 0.03
56 - acc: 0.9912
Epoch 35/300
800/800 [==============================] - 0s 128us/step - loss: 0.0332 - mean_squared_error: 0.03
32 - acc: 0.9887
Epoch 36/300
800/800 [==============================] - 0s 117us/step - loss: 0.0329 - mean_squared_error: 0.03
29 - acc: 0.9875
Epoch 37/300
800/800 [==============================] - 0s 113us/step - loss: 0.0305 - mean_squared_error: 0.03
05 - acc: 0.9937
Epoch 38/300
800/800 [==============================] - 0s 108us/step - loss: 0.0276 - mean_squared_error: 0.02
76 - acc: 0.9987
Epoch 39/300
800/800 [==============================] - 0s 118us/step - loss: 0.0268 - mean_squared_error: 0.02
68 - acc: 0.9950
Epoch 40/300
800/800 [==============================] - 0s 118us/step - loss: 0.0267 - mean_squared_error: 0.02
67 - acc: 0.9937
Epoch 41/300
800/800 [==============================] - 0s 127us/step - loss: 0.0272 - mean_squared_error: 0.02
72 - acc: 0.9975
Epoch 42/300
800/800 [==============================] - 0s 115us/step - loss: 0.0263 - mean_squared_error: 0.02
63 - acc: 0.9975
Epoch 43/300
800/800 [==============================] - 0s 116us/step - loss: 0.0261 - mean_squared_error: 0.02
61 - acc: 0.9962
Epoch 44/300
800/800 [==============================] - 0s 113us/step - loss: 0.0241 - mean_squared_error: 0.02
41 - acc: 0.9962
Epoch 45/300
800/800 [==============================] - 0s 141us/step - loss: 0.0240 - mean_squared_error: 0.02
40 - acc: 0.9987
Epoch 46/300
800/800 [==============================] - 0s 111us/step - loss: 0.0243 - mean_squared_error: 0.02
43 - acc: 0.9937
Epoch 47/300
800/800 [==============================] - 0s 123us/step - loss: 0.0200 - mean_squared_error: 0.02
00 - acc: 0.9987
Epoch 48/300
800/800 [==============================] - 0s 124us/step - loss: 0.0210 - mean_squared_error: 0.02
10 - acc: 0.9975
Epoch 49/300
800/800 [==============================] - 0s 117us/step - loss: 0.0188 - mean_squared_error: 0.01
88 - acc: 0.9987
Epoch 50/300
800/800 [==============================] - 0s 112us/step - loss: 0.0220 - mean_squared_error: 0.02
20 - acc: 0.9975
Epoch 51/300
800/800 [==============================] - 0s 148us/step - loss: 0.0189 - mean_squared_error: 0.01
89 - acc: 1.0000
Epoch 52/300
800/800 [==============================] - 0s 113us/step - loss: 0.0177 - mean_squared_error: 0.01
77 - acc: 1.0000
Epoch 53/300
800/800 [==============================] - 0s 116us/step - loss: 0.0172 - mean_squared_error: 0.01
72 - acc: 1.0000
Epoch 54/300
800/800 [==============================] - 0s 107us/step - loss: 0.0189 - mean_squared_error: 0.01
89 - acc: 1.0000
Epoch 55/300
800/800 [==============================] - 0s 111us/step - loss: 0.0181 - mean_squared_error: 0.01
81 - acc: 0.9987
Epoch 56/300
800/800 [==============================] - 0s 112us/step - loss: 0.0177 - mean_squared_error: 0.01
77 - acc: 0.9987
Epoch 57/300
```

```
Epoch 57/300
800/800 [==============================] - 0s 132us/step - loss: 0.0172 - mean_squared_error: 0.01
72 - acc: 1.0000
Epoch 58/300
800/800 [==============================] - 0s 114us/step - loss: 0.0185 - mean_squared_error: 0.01
85 - acc: 0.9975
Epoch 59/300
800/800 [==============================] - 0s 112us/step - loss: 0.0156 - mean_squared_error: 0.01
56 - acc: 0.9987
Epoch 60/300
800/800 [==============================] - 0s 113us/step - loss: 0.0138 - mean_squared_error: 0.01
38 - acc: 1.0000
Epoch 61/300
800/800 [==============================] - 0s 112us/step - loss: 0.0162 - mean_squared_error: 0.01
62 - acc: 1.0000
Epoch 62/300
800/800 [==============================] - 0s 121us/step - loss: 0.0161 - mean_squared_error: 0.01
61 - acc: 1.0000
Epoch 63/300
800/800 [==============================] - 0s 105us/step - loss: 0.0150 - mean_squared_error: 0.01
50 - acc: 1.0000
Epoch 64/300
800/800 [==============================] - 0s 108us/step - loss: 0.0138 - mean_squared_error: 0.01
38 - acc: 1.0000
Epoch 65/300
800/800 [==============================] - 0s 119us/step - loss: 0.0156 - mean_squared_error: 0.01
56 - acc: 1.0000
Epoch 66/300
800/800 [==============================] - 0s 114us/step - loss: 0.0159 - mean_squared_error: 0.01
59 - acc: 0.9987
Epoch 67/300
800/800 [==============================] - 0s 109us/step - loss: 0.0148 - mean_squared_error: 0.01
48 - acc: 1.0000
Epoch 68/300
800/800 [==============================] - 0s 125us/step - loss: 0.0134 - mean_squared_error: 0.01
34 - acc: 1.0000
Epoch 69/300
800/800 [==============================] - 0s 119us/step - loss: 0.0133 - mean_squared_error: 0.01
33 - acc: 0.9987
Epoch 70/300
800/800 [==============================] - 0s 120us/step - loss: 0.0140 - mean_squared_error: 0.01
40 - acc: 0.9987
Epoch 71/300
800/800 [==============================] - 0s 115us/step - loss: 0.0139 - mean_squared_error: 0.01
39 - acc: 1.0000
Epoch 72/300
800/800 [==============================] - 0s 123us/step - loss: 0.0123 - mean_squared_error: 0.01
23 - acc: 1.0000
Epoch 73/300
800/800 [==============================] - 0s 128us/step - loss: 0.0123 - mean_squared_error: 0.01
23 - acc: 1.0000
Epoch 74/300
800/800 [==============================] - 0s 118us/step - loss: 0.0125 - mean_squared_error: 0.01
25 - acc: 1.0000
Epoch 75/300
800/800 [==============================] - 0s 113us/step - loss: 0.0123 - mean_squared_error: 0.01
23 - acc: 1.0000
Epoch 76/300
800/800 [==============================] - 0s 123us/step - loss: 0.0156 - mean_squared_error: 0.01
56 - acc: 0.9987
Epoch 77/300
800/800 [==============================] - 0s 154us/step - loss: 0.0143 - mean_squared_error: 0.01
43 - acc: 1.0000
Epoch 78/300
800/800 [==============================] - 0s 146us/step - loss: 0.0119 - mean_squared_error: 0.01
19 - acc: 1.0000
Epoch 79/300
800/800 [==============================] - 0s 155us/step - loss: 0.0117 - mean_squared_error: 0.01
17 - acc: 1.0000
Epoch 80/300
800/800 [==============================] - 0s 158us/step - loss: 0.0118 - mean_squared_error: 0.01
18 - acc: 1.0000
Epoch 81/300
800/800 [==============================] - 0s 164us/step - loss: 0.0115 - mean_squared_error: 0.01
15 - acc: 1.0000
Epoch 82/300
800/800 [==============================] - 0s 155us/step - loss: 0.0114 - mean_squared_error: 0.01
14 - acc: 1.0000
```

```
14 - acc: 1.0000
Epoch 83/300
800/800 [==============================] - 0s 124us/step - loss: 0.0096 - mean_squared_error: 0.00
96 - acc: 1.0000
Epoch 84/300
800/800 [==============================] - 0s 117us/step - loss: 0.0095 - mean_squared_error: 0.00
95 - acc: 1.0000
Epoch 85/300
800/800 [==============================] - 0s 125us/step - loss: 0.0104 - mean_squared_error: 0.01
04 - acc: 1.0000
Epoch 86/300
800/800 [==============================] - 0s 119us/step - loss: 0.0100 - mean_squared_error: 0.01
00 - acc: 1.0000
Epoch 87/300
800/800 [==============================] - 0s 118us/step - loss: 0.0102 - mean_squared_error: 0.01
02 - acc: 1.0000
Epoch 88/300
800/800 [==============================] - 0s 115us/step - loss: 0.0096 - mean_squared_error: 0.00
96 - acc: 1.0000
Epoch 89/300
800/800 [==============================] - 0s 116us/step - loss: 0.0108 - mean_squared_error: 0.01
08 - acc: 1.0000
Epoch 90/300
800/800 [==============================] - 0s 110us/step - loss: 0.0092 - mean_squared_error: 0.00
92 - acc: 1.0000
Epoch 91/300
800/800 [==============================] - 0s 114us/step - loss: 0.0092 - mean_squared_error: 0.00
92 - acc: 1.0000
Epoch 92/300
800/800 [==============================] - 0s 118us/step - loss: 0.0090 - mean_squared_error: 0.00
90 - acc: 1.0000
Epoch 93/300
800/800 [==============================] - 0s 116us/step - loss: 0.0098 - mean_squared_error: 0.00
98 - acc: 1.0000
Epoch 94/300
800/800 [==============================] - 0s 132us/step - loss: 0.0099 - mean_squared_error: 0.00
99 - acc: 1.0000
Epoch 95/300
800/800 [==============================] - 0s 119us/step - loss: 0.0100 - mean_squared_error: 0.01
00 - acc: 1.0000
Epoch 96/300
800/800 [==============================] - 0s 118us/step - loss: 0.0104 - mean_squared_error: 0.01
04 - acc: 1.0000
Epoch 97/300
800/800 [==============================] - 0s 119us/step - loss: 0.0100 - mean_squared_error: 0.01
00 - acc: 1.0000
Epoch 98/300
800/800 [==============================] - 0s 116us/step - loss: 0.0091 - mean_squared_error: 0.00
91 - acc: 1.0000
Epoch 99/300
800/800 [==============================] - 0s 123us/step - loss: 0.0108 - mean_squared_error: 0.01
08 - acc: 1.0000
Epoch 100/300
800/800 [==============================] - 0s 152us/step - loss: 0.0091 - mean_squared_error: 0.00
91 - acc: 1.0000
Epoch 101/300
800/800 [==============================] - 0s 125us/step - loss: 0.0092 - mean_squared_error: 0.00
92 - acc: 1.0000
Epoch 102/300
800/800 [==============================] - 0s 113us/step - loss: 0.0090 - mean_squared_error: 0.00
90 - acc: 1.0000
Epoch 103/300
800/800 [==============================] - 0s 112us/step - loss: 0.0082 - mean_squared_error: 0.00
82 - acc: 1.0000
Epoch 104/300
800/800 [==============================] - 0s 109us/step - loss: 0.0085 - mean_squared_error: 0.00
85 - acc: 1.0000
Epoch 105/300
800/800 [==============================] - 0s 118us/step - loss: 0.0078 - mean_squared_error: 0.00
78 - acc: 1.0000
Epoch 106/300
800/800 [==============================] - 0s 116us/step - loss: 0.0079 - mean_squared_error: 0.00
79 - acc: 1.0000
Epoch 107/300
800/800 [==============================] - 0s 117us/step - loss: 0.0079 - mean_squared_error: 0.00
79 - acc: 1.0000
Epoch 108/300
800/800 [==============================] - 0s 113us/step - loss: 0.0084 - mean_squared_error: 0.00
```

```
800/800 [==============================] - 0s 113us/step - loss: 0.0084 - mean_squared_error: 0.00
84 - acc: 1.0000
Epoch 109/300
800/800 [==============================] - 0s 119us/step - loss: 0.0086 - mean_squared_error: 0.00
86 - acc: 1.0000
Epoch 110/300
800/800 [==============================] - 0s 122us/step - loss: 0.0076 - mean_squared_error: 0.00
76 - acc: 1.0000
Epoch 111/300
800/800 [==============================] - 0s 116us/step - loss: 0.0075 - mean_squared_error: 0.00
75 - acc: 1.0000
Epoch 112/300
800/800 [==============================] - 0s 118us/step - loss: 0.0091 - mean_squared_error: 0.00
91 - acc: 1.0000
Epoch 113/300
800/800 [==============================] - 0s 117us/step - loss: 0.0085 - mean_squared_error: 0.00
85 - acc: 1.0000
Epoch 114/300
800/800 [==============================] - 0s 109us/step - loss: 0.0077 - mean_squared_error: 0.00
77 - acc: 1.0000
Epoch 115/300
800/800 [==============================] - 0s 125us/step - loss: 0.0079 - mean_squared_error: 0.00
79 - acc: 1.0000
Epoch 116/300
800/800 [==============================] - 0s 114us/step - loss: 0.0080 - mean_squared_error: 0.00
80 - acc: 1.0000
Epoch 117/300
800/800 [==============================] - 0s 123us/step - loss: 0.0076 - mean_squared_error: 0.00
76 - acc: 1.0000
Epoch 118/300
800/800 [==============================] - 0s 119us/step - loss: 0.0081 - mean_squared_error: 0.00
81 - acc: 1.0000
Epoch 119/300
800/800 [==============================] - 0s 132us/step - loss: 0.0073 - mean_squared_error: 0.00
73 - acc: 1.0000
Epoch 120/300
800/800 [==============================] - 0s 121us/step - loss: 0.0086 - mean_squared_error: 0.00
86 - acc: 1.0000
Epoch 121/300
800/800 [==============================] - 0s 118us/step - loss: 0.0070 - mean_squared_error: 0.00
70 - acc: 1.0000
Epoch 122/300
800/800 [==============================] - 0s 121us/step - loss: 0.0086 - mean_squared_error: 0.00
86 - acc: 1.0000
Epoch 123/300
800/800 [==============================] - 0s 112us/step - loss: 0.0088 - mean_squared_error: 0.00
88 - acc: 1.0000
Epoch 124/300
800/800 [==============================] - 0s 116us/step - loss: 0.0069 - mean_squared_error: 0.00
69 - acc: 1.0000
Epoch 125/300
800/800 [==============================] - 0s 105us/step - loss: 0.0074 - mean_squared_error: 0.00
74 - acc: 1.0000
Epoch 126/300
800/800 [==============================] - 0s 112us/step - loss: 0.0074 - mean_squared_error: 0.00
74 - acc: 1.0000
Epoch 127/300
800/800 [==============================] - 0s 116us/step - loss: 0.0074 - mean_squared_error: 0.00
74 - acc: 1.0000
Epoch 128/300
800/800 [==============================] - 0s 110us/step - loss: 0.0070 - mean_squared_error: 0.00
70 - acc: 1.0000
Epoch 129/300
800/800 [==============================] - 0s 110us/step - loss: 0.0074 - mean_squared_error: 0.00
74 - acc: 1.0000
Epoch 130/300
800/800 [==============================] - 0s 114us/step - loss: 0.0069 - mean_squared_error: 0.00
69 - acc: 1.0000
Epoch 131/300
800/800 [==============================] - 0s 118us/step - loss: 0.0066 - mean_squared_error: 0.00
66 - acc: 1.0000
Epoch 132/300
800/800 [==============================] - 0s 123us/step - loss: 0.0075 - mean_squared_error: 0.00
75 - acc: 1.0000
Epoch 133/300
800/800 [==============================] - 0s 129us/step - loss: 0.0074 - mean_squared_error: 0.00
74 - acc: 0.9987
Epoch 134/300
```

```
Epoch 134/300
800/800 [==============================] - 0s 158us/step - loss: 0.0073 - mean_squared_error: 0.00
73 - acc: 1.0000
Epoch 135/300
800/800 [==============================] - 0s 134us/step - loss: 0.0075 - mean_squared_error: 0.00
75 - acc: 1.0000
Epoch 136/300
800/800 [==============================] - 0s 166us/step - loss: 0.0066 - mean_squared_error: 0.00
66 - acc: 1.0000
Epoch 137/300
800/800 [==============================] - 0s 147us/step - loss: 0.0065 - mean_squared_error: 0.00
65 - acc: 1.0000
Epoch 138/300
800/800 [==============================] - 0s 161us/step - loss: 0.0074 - mean_squared_error: 0.00
74 - acc: 1.0000
Epoch 139/300
800/800 [==============================] - 0s 145us/step - loss: 0.0064 - mean_squared_error: 0.00
64 - acc: 1.0000
Epoch 140/300
800/800 [==============================] - 0s 121us/step - loss: 0.0068 - mean_squared_error: 0.00
68 - acc: 1.0000
Epoch 141/300
800/800 [==============================] - 0s 113us/step - loss: 0.0059 - mean_squared_error: 0.00
59 - acc: 1.0000
Epoch 142/300
800/800 [==============================] - 0s 115us/step - loss: 0.0060 - mean_squared_error: 0.00
60 - acc: 1.0000
Epoch 143/300
800/800 [==============================] - 0s 111us/step - loss: 0.0063 - mean_squared_error: 0.00
63 - acc: 1.0000
Epoch 144/300
800/800 [==============================] - 0s 117us/step - loss: 0.0068 - mean_squared_error: 0.00
68 - acc: 1.0000
Epoch 145/300
800/800 [==============================] - 0s 123us/step - loss: 0.0065 - mean_squared_error: 0.00
65 - acc: 1.0000
Epoch 146/300
800/800 [==============================] - 0s 119us/step - loss: 0.0071 - mean_squared_error: 0.00
71 - acc: 1.0000
Epoch 147/300
800/800 [==============================] - 0s 121us/step - loss: 0.0065 - mean_squared_error: 0.00
65 - acc: 1.0000
Epoch 148/300
800/800 [==============================] - 0s 112us/step - loss: 0.0067 - mean_squared_error: 0.00
67 - acc: 1.0000
Epoch 149/300
800/800 [==============================] - 0s 156us/step - loss: 0.0066 - mean_squared_error: 0.00
66 - acc: 1.0000
Epoch 150/300
800/800 [==============================] - 0s 113us/step - loss: 0.0057 - mean_squared_error: 0.00
57 - acc: 1.0000
Epoch 151/300
800/800 [==============================] - 0s 120us/step - loss: 0.0057 - mean_squared_error: 0.00
57 - acc: 1.0000
Epoch 152/300
800/800 [==============================] - 0s 119us/step - loss: 0.0065 - mean_squared_error: 0.00
65 - acc: 1.0000
Epoch 153/300
800/800 [==============================] - 0s 120us/step - loss: 0.0061 - mean_squared_error: 0.00
61 - acc: 1.0000
Epoch 154/300
800/800 [==============================] - 0s 117us/step - loss: 0.0051 - mean_squared_error: 0.00
51 - acc: 1.0000
Epoch 155/300
800/800 [==============================] - 0s 119us/step - loss: 0.0059 - mean_squared_error: 0.00
59 - acc: 1.0000
Epoch 156/300
800/800 [==============================] - 0s 115us/step - loss: 0.0058 - mean_squared_error: 0.00
58 - acc: 1.0000
Epoch 157/300
800/800 [==============================] - 0s 114us/step - loss: 0.0062 - mean_squared_error: 0.00
62 - acc: 1.0000
Epoch 158/300
800/800 [==============================] - 0s 114us/step - loss: 0.0059 - mean_squared_error: 0.00
59 - acc: 1.0000
Epoch 159/300
800/800 [==============================] - 0s 120us/step - loss: 0.0055 - mean_squared_error: 0.00
55 - acc: 1.0000
```

```
55 - acc: 1.0000
Epoch 160/300
800/800 [==============================] - 0s 117us/step - loss: 0.0054 - mean_squared_error: 0.00
54 - acc: 1.0000
Epoch 161/300
800/800 [==============================] - 0s 116us/step - loss: 0.0058 - mean_squared_error: 0.00
58 - acc: 1.0000
Epoch 162/300
800/800 [==============================] - 0s 119us/step - loss: 0.0058 - mean_squared_error: 0.00
58 - acc: 1.0000
Epoch 163/300
800/800 [==============================] - 0s 118us/step - loss: 0.0053 - mean_squared_error: 0.00
53 - acc: 1.0000
Epoch 164/300
800/800 [==============================] - 0s 114us/step - loss: 0.0054 - mean_squared_error: 0.00
54 - acc: 1.0000
Epoch 165/300
800/800 [==============================] - 0s 119us/step - loss: 0.0066 - mean_squared_error: 0.00
66 - acc: 1.0000
Epoch 166/300
800/800 [==============================] - 0s 116us/step - loss: 0.0062 - mean_squared_error: 0.00
62 - acc: 1.0000
Epoch 167/300
800/800 [==============================] - 0s 118us/step - loss: 0.0064 - mean_squared_error: 0.00
64 - acc: 1.0000
Epoch 168/300
800/800 [==============================] - 0s 123us/step - loss: 0.0058 - mean_squared_error: 0.00
58 - acc: 1.0000
Epoch 169/300
800/800 [==============================] - 0s 124us/step - loss: 0.0063 - mean_squared_error: 0.00
63 - acc: 1.0000
Epoch 170/300
800/800 [==============================] - 0s 125us/step - loss: 0.0054 - mean_squared_error: 0.00
54 - acc: 1.0000
Epoch 171/300
800/800 [==============================] - 0s 117us/step - loss: 0.0052 - mean_squared_error: 0.00
52 - acc: 1.0000
Epoch 172/300
800/800 [==============================] - 0s 131us/step - loss: 0.0053 - mean_squared_error: 0.00
53 - acc: 1.0000
Epoch 173/300
800/800 [==============================] - 0s 124us/step - loss: 0.0057 - mean_squared_error: 0.00
57 - acc: 1.0000
Epoch 174/300
800/800 [==============================] - 0s 125us/step - loss: 0.0062 - mean_squared_error: 0.00
62 - acc: 1.0000
Epoch 175/300
800/800 [==============================] - 0s 124us/step - loss: 0.0062 - mean_squared_error: 0.00
62 - acc: 1.0000
Epoch 176/300
800/800 [==============================] - 0s 134us/step - loss: 0.0066 - mean_squared_error: 0.00
66 - acc: 1.0000
Epoch 177/300
800/800 [==============================] - 0s 124us/step - loss: 0.0054 - mean_squared_error: 0.00
54 - acc: 1.0000
Epoch 178/300
800/800 [==============================] - 0s 127us/step - loss: 0.0055 - mean_squared_error: 0.00
55 - acc: 1.0000
Epoch 179/300
800/800 [==============================] - 0s 124us/step - loss: 0.0049 - mean_squared_error: 0.00
49 - acc: 1.0000
Epoch 180/300
800/800 [==============================] - 0s 127us/step - loss: 0.0050 - mean_squared_error: 0.00
50 - acc: 1.0000
Epoch 181/300
800/800 [==============================] - 0s 126us/step - loss: 0.0057 - mean_squared_error: 0.00
57 - acc: 1.0000
Epoch 182/300
800/800 [==============================] - 0s 130us/step - loss: 0.0051 - mean_squared_error: 0.00
51 - acc: 1.0000
Epoch 183/300
800/800 [==============================] - 0s 129us/step - loss: 0.0048 - mean_squared_error: 0.00
48 - acc: 1.0000
Epoch 184/300
800/800 [==============================] - 0s 123us/step - loss: 0.0052 - mean_squared_error: 0.00
52 - acc: 1.0000
Epoch 185/300
```

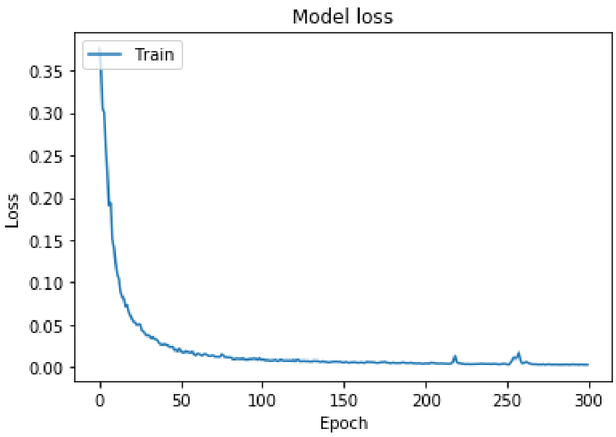
```
800/800 [==============================] - 0s 125us/step - loss: 0.0049 - mean_squared_error: 0.00
49 - acc: 1.0000
Epoch 186/300
800/800 [==============================] - 0s 125us/step - loss: 0.0054 - mean_squared_error: 0.00
54 - acc: 1.0000
Epoch 187/300
800/800 [==============================] - 0s 129us/step - loss: 0.0057 - mean_squared_error: 0.00
57 - acc: 1.0000
Epoch 188/300
800/800 [==============================] - 0s 137us/step - loss: 0.0055 - mean_squared_error: 0.00
55 - acc: 1.0000
Epoch 189/300
800/800 [==============================] - 0s 155us/step - loss: 0.0050 - mean_squared_error: 0.00
50 - acc: 1.0000
Epoch 190/300
800/800 [==============================] - 0s 124us/step - loss: 0.0049 - mean_squared_error: 0.00
49 - acc: 1.0000
Epoch 191/300
800/800 [==============================] - 0s 156us/step - loss: 0.0051 - mean_squared_error: 0.00
51 - acc: 1.0000
Epoch 192/300
800/800 [==============================] - 0s 131us/step - loss: 0.0055 - mean_squared_error: 0.00
55 - acc: 1.0000
Epoch 193/300
800/800 [==============================] - 0s 177us/step - loss: 0.0048 - mean_squared_error: 0.00
48 - acc: 1.0000
Epoch 194/300
800/800 [==============================] - 0s 187us/step - loss: 0.0049 - mean_squared_error: 0.00
49 - acc: 1.0000
Epoch 195/300
800/800 [==============================] - 0s 151us/step - loss: 0.0044 - mean_squared_error: 0.00
44 - acc: 1.0000
Epoch 196/300
800/800 [==============================] - 0s 163us/step - loss: 0.0046 - mean_squared_error: 0.00
46 - acc: 1.0000
Epoch 197/300
800/800 [==============================] - 0s 131us/step - loss: 0.0048 - mean_squared_error: 0.00
48 - acc: 1.0000
Epoch 198/300
800/800 [==============================] - 0s 132us/step - loss: 0.0045 - mean_squared_error: 0.00
45 - acc: 1.0000
Epoch 199/300
800/800 [==============================] - 0s 128us/step - loss: 0.0042 - mean_squared_error: 0.00
42 - acc: 1.0000
Epoch 200/300
800/800 [==============================] - 0s 123us/step - loss: 0.0043 - mean_squared_error: 0.00
43 - acc: 1.0000
Epoch 201/300
800/800 [==============================] - 0s 125us/step - loss: 0.0048 - mean_squared_error: 0.00
48 - acc: 1.0000
Epoch 202/300
800/800 [==============================] - 0s 118us/step - loss: 0.0045 - mean_squared_error: 0.00
45 - acc: 1.0000
Epoch 203/300
800/800 [==============================] - 0s 119us/step - loss: 0.0043 - mean_squared_error: 0.00
43 - acc: 1.0000
Epoch 204/300
800/800 [==============================] - 0s 122us/step - loss: 0.0056 - mean_squared_error: 0.00
56 - acc: 1.0000
Epoch 205/300
800/800 [==============================] - 0s 124us/step - loss: 0.0051 - mean_squared_error: 0.00
51 - acc: 1.0000
Epoch 206/300
800/800 [==============================] - 0s 118us/step - loss: 0.0053 - mean_squared_error: 0.00
53 - acc: 1.0000
Epoch 207/300
800/800 [==============================] - 0s 121us/step - loss: 0.0045 - mean_squared_error: 0.00
45 - acc: 1.0000
Epoch 208/300
800/800 [==============================] - 0s 118us/step - loss: 0.0046 - mean_squared_error: 0.00
46 - acc: 1.0000
Epoch 209/300
800/800 [==============================] - 0s 128us/step - loss: 0.0044 - mean_squared_error: 0.00
44 - acc: 1.0000
Epoch 210/300
800/800 [==============================] - 0s 115us/step - loss: 0.0045 - mean_squared_error: 0.00
45 - acc: 1.0000
```

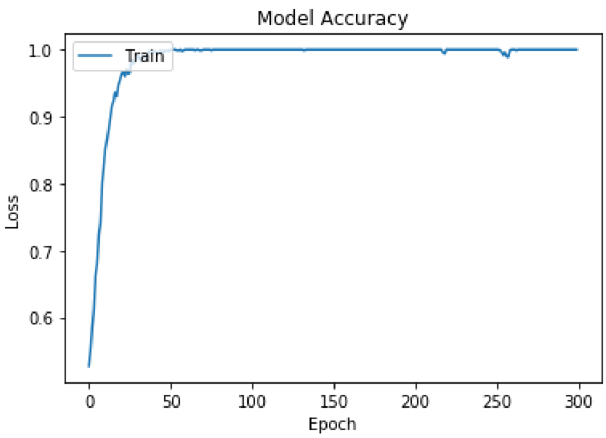
```
Epoch 211/300
800/800 [==============================] - 0s 123us/step - loss: 0.0045 - mean_squared_error: 0.00
45 - acc: 1.0000
Epoch 212/300
800/800 [==============================] - 0s 125us/step - loss: 0.0044 - mean_squared_error: 0.00
44 - acc: 1.0000
Epoch 213/300
800/800 [==============================] - 0s 122us/step - loss: 0.0044 - mean_squared_error: 0.00
44 - acc: 1.0000
Epoch 214/300
800/800 [==============================] - 0s 124us/step - loss: 0.0043 - mean_squared_error: 0.00
43 - acc: 1.0000
Epoch 215/300
800/800 [==============================] - 0s 117us/step - loss: 0.0041 - mean_squared_error: 0.00
41 - acc: 1.0000
Epoch 216/300
800/800 [==============================] - 0s 123us/step - loss: 0.0043 - mean_squared_error: 0.00
43 - acc: 1.0000
Epoch 217/300
800/800 [==============================] - 0s 122us/step - loss: 0.0047 - mean_squared_error: 0.00
47 - acc: 1.0000
Epoch 218/300
800/800 [==============================] - 0s 119us/step - loss: 0.0079 - mean_squared_error: 0.00
79 - acc: 0.9962
Epoch 219/300
800/800 [==============================] - 0s 125us/step - loss: 0.0135 - mean_squared_error: 0.01
35 - acc: 0.9937
Epoch 220/300
800/800 [==============================] - 0s 125us/step - loss: 0.0066 - mean_squared_error: 0.00
66 - acc: 1.0000
Epoch 221/300
800/800 [==============================] - 0s 121us/step - loss: 0.0049 - mean_squared_error: 0.00
49 - acc: 1.0000
Epoch 222/300
800/800 [==============================] - 0s 118us/step - loss: 0.0051 - mean_squared_error: 0.00
51 - acc: 1.0000
Epoch 223/300
800/800 [==============================] - 0s 114us/step - loss: 0.0041 - mean_squared_error: 0.00
41 - acc: 1.0000
Epoch 224/300
800/800 [==============================] - 0s 113us/step - loss: 0.0043 - mean_squared_error: 0.00
43 - acc: 1.0000
Epoch 225/300
800/800 [==============================] - 0s 116us/step - loss: 0.0036 - mean_squared_error: 0.00
36 - acc: 1.0000
Epoch 226/300
800/800 [==============================] - 0s 120us/step - loss: 0.0037 - mean_squared_error: 0.00
37 - acc: 1.0000
Epoch 227/300
800/800 [==============================] - 0s 123us/step - loss: 0.0039 - mean_squared_error: 0.00
39 - acc: 1.0000
Epoch 228/300
800/800 [==============================] - 0s 111us/step - loss: 0.0035 - mean_squared_error: 0.00
35 - acc: 1.0000
Epoch 229/300
800/800 [==============================] - 0s 115us/step - loss: 0.0036 - mean_squared_error: 0.00
36 - acc: 1.0000
Epoch 230/300
800/800 [==============================] - 0s 115us/step - loss: 0.0037 - mean_squared_error: 0.00
37 - acc: 1.0000
Epoch 231/300
800/800 [==============================] - 0s 125us/step - loss: 0.0036 - mean_squared_error: 0.00
36 - acc: 1.0000
Epoch 232/300
800/800 [==============================] - 0s 116us/step - loss: 0.0037 - mean_squared_error: 0.00
37 - acc: 1.0000
Epoch 233/300
800/800 [==============================] - 0s 114us/step - loss: 0.0037 - mean_squared_error: 0.00
37 - acc: 1.0000
Epoch 234/300
800/800 [==============================] - 0s 117us/step - loss: 0.0040 - mean_squared_error: 0.00
40 - acc: 1.0000
Epoch 235/300
800/800 [==============================] - 0s 124us/step - loss: 0.0044 - mean_squared_error: 0.00
44 - acc: 1.0000
Epoch 236/300
800/800 [==============================] - 0s 116us/step - loss: 0.0040 - mean_squared_error: 0.00
```

```
40 - acc: 1.0000
Epoch 237/300
800/800 [==============================] - 0s 117us/step - loss: 0.0039 - mean_squared_error: 0.00
39 - acc: 1.0000
Epoch 238/300
800/800 [==============================] - 0s 117us/step - loss: 0.0040 - mean_squared_error: 0.00
40 - acc: 1.0000
Epoch 239/300
800/800 [==============================] - 0s 120us/step - loss: 0.0039 - mean_squared_error: 0.00
39 - acc: 1.0000
Epoch 240/300
800/800 [==============================] - 0s 123us/step - loss: 0.0041 - mean_squared_error: 0.00
41 - acc: 1.0000
Epoch 241/300
800/800 [==============================] - 0s 111us/step - loss: 0.0035 - mean_squared_error: 0.00
35 - acc: 1.0000
Epoch 242/300
800/800 [==============================] - 0s 120us/step - loss: 0.0041 - mean_squared_error: 0.00
41 - acc: 1.0000
Epoch 243/300
800/800 [==============================] - 0s 120us/step - loss: 0.0039 - mean_squared_error: 0.00
39 - acc: 1.0000
Epoch 244/300
800/800 [==============================] - 0s 122us/step - loss: 0.0040 - mean_squared_error: 0.00
40 - acc: 1.0000
Epoch 245/300
800/800 [==============================] - 0s 144us/step - loss: 0.0036 - mean_squared_error: 0.00
36 - acc: 1.0000
Epoch 246/300
800/800 [==============================] - 0s 165us/step - loss: 0.0034 - mean_squared_error: 0.00
34 - acc: 1.0000
Epoch 247/300
800/800 [==============================] - 0s 132us/step - loss: 0.0035 - mean_squared_error: 0.00
35 - acc: 1.0000
Epoch 248/300
800/800 [==============================] - 0s 147us/step - loss: 0.0040 - mean_squared_error: 0.00
40 - acc: 1.0000
Epoch 249/300
800/800 [==============================] - 0s 155us/step - loss: 0.0042 - mean_squared_error: 0.00
42 - acc: 1.0000
Epoch 250/300
800/800 [==============================] - 0s 127us/step - loss: 0.0041 - mean_squared_error: 0.00
41 - acc: 1.0000
Epoch 251/300
800/800 [==============================] - 0s 123us/step - loss: 0.0033 - mean_squared_error: 0.00
33 - acc: 1.0000
Epoch 252/300
800/800 [==============================] - 0s 121us/step - loss: 0.0031 - mean_squared_error: 0.00
31 - acc: 1.0000
Epoch 253/300
800/800 [==============================] - 0s 116us/step - loss: 0.0057 - mean_squared_error: 0.00
57 - acc: 0.9987
Epoch 254/300
800/800 [==============================] - 0s 113us/step - loss: 0.0085 - mean_squared_error: 0.00
85 - acc: 0.9962
Epoch 255/300
800/800 [==============================] - 0s 112us/step - loss: 0.0117 - mean_squared_error: 0.01
17 - acc: 0.9912
Epoch 256/300
800/800 [==============================] - 0s 119us/step - loss: 0.0105 - mean_squared_error: 0.01
05 - acc: 0.9962
Epoch 257/300
800/800 [==============================] - 0s 117us/step - loss: 0.0122 - mean_squared_error: 0.01
22 - acc: 0.9900
Epoch 258/300
800/800 [==============================] - 0s 122us/step - loss: 0.0174 - mean_squared_error: 0.01
74 - acc: 0.9887
Epoch 259/300
800/800 [==============================] - 0s 127us/step - loss: 0.0089 - mean_squared_error: 0.00
89 - acc: 0.9987
Epoch 260/300
800/800 [==============================] - 0s 123us/step - loss: 0.0048 - mean_squared_error: 0.00
48 - acc: 1.0000
Epoch 261/300
800/800 [==============================] - 0s 128us/step - loss: 0.0050 - mean_squared_error: 0.00
50 - acc: 1.0000
Epoch 262/300
```

```
800/800 [==============================] - 0s 124us/step - loss: 0.0062 - mean_squared_error: 0.00
62 - acc: 1.0000
Epoch 263/300
800/800 [==============================] - 0s 120us/step - loss: 0.0063 - mean_squared_error: 0.00
63 - acc: 0.9987
Epoch 264/300
800/800 [==============================] - 0s 123us/step - loss: 0.0047 - mean_squared_error: 0.00
47 - acc: 1.0000
Epoch 265/300
800/800 [==============================] - 0s 127us/step - loss: 0.0046 - mean_squared_error: 0.00
46 - acc: 1.0000
Epoch 266/300
800/800 [==============================] - 0s 126us/step - loss: 0.0036 - mean_squared_error: 0.00
36 - acc: 1.0000
Epoch 267/300
800/800 [==============================] - 0s 116us/step - loss: 0.0035 - mean_squared_error: 0.00
35 - acc: 1.0000
Epoch 268/300
800/800 [==============================] - 0s 122us/step - loss: 0.0034 - mean_squared_error: 0.00
34 - acc: 1.0000
Epoch 269/300
800/800 [==============================] - 0s 121us/step - loss: 0.0035 - mean_squared_error: 0.00
35 - acc: 1.0000
Epoch 270/300
800/800 [==============================] - 0s 118us/step - loss: 0.0031 - mean_squared_error: 0.00
31 - acc: 1.0000
Epoch 271/300
800/800 [==============================] - 0s 122us/step - loss: 0.0033 - mean_squared_error: 0.00
33 - acc: 1.0000
Epoch 272/300
800/800 [==============================] - 0s 122us/step - loss: 0.0032 - mean_squared_error: 0.00
32 - acc: 1.0000
Epoch 273/300
800/800 [==============================] - 0s 119us/step - loss: 0.0036 - mean_squared_error: 0.00
36 - acc: 1.0000
Epoch 274/300
800/800 [==============================] - 0s 115us/step - loss: 0.0030 - mean_squared_error: 0.00
30 - acc: 1.0000
Epoch 275/300
800/800 [==============================] - 0s 124us/step - loss: 0.0033 - mean_squared_error: 0.00
33 - acc: 1.0000
Epoch 276/300
800/800 [==============================] - 0s 115us/step - loss: 0.0034 - mean_squared_error: 0.00
34 - acc: 1.0000
Epoch 277/300
800/800 [==============================] - 0s 121us/step - loss: 0.0036 - mean_squared_error: 0.00
36 - acc: 1.0000
Epoch 278/300
800/800 [==============================] - 0s 117us/step - loss: 0.0030 - mean_squared_error: 0.00
30 - acc: 1.0000
Epoch 279/300
800/800 [==============================] - 0s 123us/step - loss: 0.0032 - mean_squared_error: 0.00
32 - acc: 1.0000
Epoch 280/300
800/800 [==============================] - 0s 119us/step - loss: 0.0033 - mean_squared_error: 0.00
33 - acc: 1.0000
Epoch 281/300
800/800 [==============================] - 0s 121us/step - loss: 0.0031 - mean_squared_error: 0.00
31 - acc: 1.0000
Epoch 282/300
800/800 [==============================] - 0s 116us/step - loss: 0.0031 - mean_squared_error: 0.00
31 - acc: 1.0000
Epoch 283/300
800/800 [==============================] - 0s 124us/step - loss: 0.0029 - mean_squared_error: 0.00
29 - acc: 1.0000
Epoch 284/300
800/800 [==============================] - 0s 123us/step - loss: 0.0030 - mean_squared_error: 0.00
30 - acc: 1.0000
Epoch 285/300
800/800 [==============================] - 0s 120us/step - loss: 0.0032 - mean_squared_error: 0.00
32 - acc: 1.0000
Epoch 286/300
800/800 [==============================] - 0s 119us/step - loss: 0.0031 - mean_squared_error: 0.00
31 - acc: 1.0000
Epoch 287/300
800/800 [==============================] - 0s 118us/step - loss: 0.0034 - mean_squared_error: 0.00
34 - acc: 1.0000
```

```
Epoch 288/300
800/800 [==============================] - 0s 119us/step - loss: 0.0032 - mean_squared_error: 0.00
32 - acc: 1.0000
Epoch 289/300
800/800 [==============================] - 0s 116us/step - loss: 0.0029 - mean_squared_error: 0.00
29 - acc: 1.0000
Epoch 290/300
800/800 [==============================] - 0s 114us/step - loss: 0.0030 - mean_squared_error: 0.00
30 - acc: 1.0000
Epoch 291/300
800/800 [==============================] - 0s 118us/step - loss: 0.0035 - mean_squared_error: 0.00
35 - acc: 1.0000
Epoch 292/300
800/800 [==============================] - 0s 118us/step - loss: 0.0031 - mean_squared_error: 0.00
31 - acc: 1.0000
Epoch 293/300
800/800 [==============================] - 0s 122us/step - loss: 0.0033 - mean_squared_error: 0.00
33 - acc: 1.0000
Epoch 294/300
800/800 [==============================] - 0s 118us/step - loss: 0.0032 - mean_squared_error: 0.00
32 - acc: 1.0000
Epoch 295/300
800/800 [==============================] - 0s 135us/step - loss: 0.0027 - mean_squared_error: 0.00
27 - acc: 1.0000
Epoch 296/300
800/800 [==============================] - 0s 116us/step - loss: 0.0035 - mean_squared_error: 0.00
35 - acc: 1.0000
Epoch 297/300
800/800 [==============================] - 0s 111us/step - loss: 0.0030 - mean_squared_error: 0.00
30 - acc: 1.0000
Epoch 298/300
800/800 [==============================] - 0s 110us/step - loss: 0.0030 - mean_squared_error: 0.00
30 - acc: 1.0000
Epoch 299/300
800/800 [==============================] - 0s 119us/step - loss: 0.0031 - mean_squared_error: 0.00
31 - acc: 1.0000
Epoch 300/300
800/800 [==============================] - 0s 125us/step - loss: 0.0030 - mean_squared_error: 0.00
30 - acc: 1.0000
dict_keys(['loss', 'mean_squared_error', 'acc'])
```





In [10]:

```
predictions=eval_model(autoencoder_model,x_test)
print("predictions =>",predictions.ravel())
print("y_test =>",y_test.ravel())
```

```
predictions => [ 1.04907846e+00  8.47971618e-01 -1.36626527e-01 -2.05323249e-02
  9.61106271e-02  1.02915645e+00 -2.48379514e-01  9.02010620e-01
 -6.35435730e-02  1.06752664e-02  9.52737570e-01  1.01258588e+00
  5.94216526e-01 -4.26058173e-02  8.15789461e-01  3.33873183e-02
  2.77330875e-01 -2.73034573e-01  9.84727621e-01  1.43073902e-01
  7.75110602e-01  2.29725510e-01  1.15857208e+00  4.58711922e-01
  9.59521055e-01 -2.11697966e-02  4.16784286e-02  9.27541971e-01
  1.07954693e+00 -2.37002969e-04  9.25771773e-01  8.81900430e-01
  5.22562116e-02  1.01889706e+00  3.51743519e-01 -7.22063929e-02
  1.53250039e-01  8.93365622e-01  9.02216434e-01  1.14968204e+00
  9.51241136e-01  1.09081841e+00  1.44211531e-01  3.20255786e-01
 -1.90481246e-02  8.65473807e-01  7.10518003e-01  8.40582848e-01
  6.86021566e-01  1.00215085e-01  1.05978109e-01  8.77248943e-01
  6.32520139e-01 -1.22536466e-01  1.74554557e-01  1.07980222e-02
  1.00210690e+00  9.10771489e-01  4.64332998e-01  9.65136647e-01
  8.58361363e-01 -2.89446831e-01  2.58583009e-01  9.91460979e-01
  1.06469572e+00 -6.27696663e-02  9.84996736e-01  6.16924524e-01
  1.04589856e+00  3.90107661e-01  1.02696133e+00  5.93394220e-01
  8.73144448e-01  9.04207468e-01  6.00795865e-01  4.48996603e-01
  3.60454768e-02  4.75033522e-01 -2.34963298e-02  1.03839815e+00
  1.41879022e-02  1.06145144e+00  1.07417083e+00  9.91800547e-01
  4.78437781e-01  1.50338739e-01  1.00256956e+00  9.43817437e-01
  9.67346132e-01  1.06562662e+00  8.88677776e-01  1.94549501e-01
  1.04710495e+00  9.59238052e-01  3.25476348e-01  9.36538219e-01
  1.03164542e+00  1.04575574e+00  1.32640138e-01 -3.64977717e-02
  2.03522727e-01  2.80194640e-01  1.01223898e+00  2.25957215e-01
  9.31357384e-01  1.02168417e+00  1.93482637e-02  5.29617071e-01
  7.40970492e-01  5.13399482e-01  8.94673288e-01  8.83237720e-01
  9.91800964e-01  4.35560793e-02  1.88081980e-01  5.34500331e-02
  6.66801929e-01  1.02961755e+00 -1.29310936e-02  6.49900675e-01
  2.36034542e-02  1.87699035e-01  1.51077315e-01  4.03109342e-01
  8.14143866e-02  8.29916835e-01  9.85518634e-01  1.07526648e+00
  1.02883494e+00  9.50184226e-01  1.00454247e+00  9.04185697e-02
  1.01943350e+00  1.68350011e-01  7.61684000e-01  1.13554251e+00
  1.00242412e+00  8.96065593e-01  9.99105126e-02  1.00040567e+00
  8.28711033e-01 -2.22483709e-01  9.63955879e-01  4.65681136e-01
  6.17962122e-01  9.24845099e-01  9.98564303e-01  1.01559138e+00
  1.02526271e+00  1.06274021e+00 -2.46298462e-02  8.09580386e-01
 -1.48896277e-02  9.78438318e-01  1.79052234e-01  1.04131317e+00
  8.84822905e-01  5.15870869e-01  3.28978837e-01  3.24295610e-01
  4.36028689e-02  1.42744333e-01  6.87446177e-01  3.92017305e-01
  9.96353328e-01  2.66154706e-01  1.52502626e-01  8.02561581e-01
  8.77715707e-01 -4.42112535e-02  7.39346445e-01  5.84632635e-01
 -2.08141655e-02  2.95377523e-01  9.83316481e-01  4.96960878e-01
  1.32187426e-01  5.29949844e-01  1.09581327e+00  9.32333887e-01
  2.16071248e-01  7.29077876e-01  9.99849021e-01  5.60066879e-01
  1.03321922e+00  5.46347022e-01  1.61701694e-01  8.65315795e-02
  9.54082549e-01  3.33312452e-02  8.95916343e-01  8.64082396e-01
  2.60475278e-03  1.01270461e+00  1.02155423e+00  7.13452041e-01
  8.34349573e-01  9.53726232e-01  1.04530931e+00  4.78465438e-01]
y_test => [1 1 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0 0
 1 1 1 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 0 1 1 1 0 1 1 0 1 0 0 1
 0 0 0 1 0 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 0 0 0 0 1 0 0 1 0 1 0 1 0
 1 1 0 1 0 1 0 0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1 1 1
 1 1 0 1 0 1 0 1 1 1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 1 1 1 0 0 1 1 1
 0 0 0 1 1 1 1 0 1 1 0 1 1 1 0]
```

In [11]:

```
print(len(predictions.ravel()))
print(len(y_test.ravel()))
```

```
200
200
```

In [12]:

```python
def convert_to_binary(pred_cls):
    for i in range(len(pred_cls)):
        if pred_cls[i]>=0.5:pred_cls[i]=1
        else:pred_cls[i]=0
    return pred_cls
pred_classes=convert_to_binary(predictions.ravel())
```

In [13]:

```python
cm1 = confusion_matrix(y_test.ravel(),pred_classes)
print(cm1)
```

```
[[71 25]
 [12 92]]
```

In [14]:

```python
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

print("For 2D confusion matrix")

tn, fp, fn, tp = cm1.ravel()
accuracy=(tp+tn)/(tp+tn+fp+fn)
accuracy*=100
recall=tp/(tp+fn)
recall*=100
precision=tp/(tp+fp)
precision*=100
f1_score=2*(recall*precision)/(recall+precision)

#print("accuracy : ",accuracy)
#print("recall : ",recall)
#print("precision : ",precision)
#print("f1_score : ",f1_score)

actual=y_test.ravel()
predicted=pred_classes
results = confusion_matrix(actual, predicted)
print('Confusion Matrix :')
print(results)
print('Accuracy Score :',100*accuracy_score(actual, predicted))
print('Report : ')
print(classification_report(actual, predicted) )
```

```
For 2D confusion matrix
Confusion Matrix :
[[71 25]
 [12 92]]
Accuracy Score : 81.5
Report :
              precision    recall  f1-score   support

           0       0.86      0.74      0.79        96
           1       0.79      0.88      0.83       104

    accuracy                           0.81       200
   macro avg       0.82      0.81      0.81       200
weighted avg       0.82      0.81      0.81       200
```

In [ ]:

In [ ]:

In [ ]:

In [1]:

```
import numpy as np
import tensorflow as tf
import keras
import scipy.io
import sklearn
import matplotlib
```

```
Using TensorFlow backend.
```

In [2]:

```
from matplotlib import pyplot as plt

from keras.layers import AveragePooling1D
from keras.models import Sequential
from keras.layers import Conv1D
from keras.layers import Activation,Dense,Dropout
from keras.layers import Flatten
from keras.utils import np_utils

from sklearn import preprocessing,metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
```

In [3]:

```
pathAddress="C:\\Users\\HP\\Desktop\\KaunsiElectivesLenaHai\\Neural Network and fuzzy logic BITS F
312"
```

In [4]:

```
arr=scipy.io.loadmat(pathAddress+"\\data_for_cnn.mat")
df=arr['ecg_in_window']
```

In [5]:

```
df
```

Out[5]:

```
array([[ -55,  -42,  -59, ...,  -35,   -1,  -34],
       [   3,  -27,    0, ...,  118, -111,  121],
       [-111,  121, -109, ...,  -23,  -52,  -29],
       ...,
       [  -4,   29,   -1, ...,  -50,  -64,  -58],
       [ -63,  -54,  -57, ...,  -83, -210,  -84],
       [-213,  -91, -225, ...,   10,  -35,    4]], dtype=int16)
```

In [6]:

```
label=scipy.io.loadmat(pathAddress+"\\class_label.mat")
labels=label['label']
```

In [7]:

```
labels
```

Out[7]:

```
array([[0],
       [0],
       [0],
       [0],
       [0],
```

```
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
```

```
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [0],
```

```
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
```

```
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
        [0],
```

```
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
```

```
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
```

```
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[0],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1]
```

```
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1]
```

```
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
```

```
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
```

```
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
```

```
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
[1],
```

```
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1],
        [1]], dtype=uint8)
```

In [8]:

```
normal=preprocessing.StandardScaler()
```

```
df=normal.fit_transform(df)

X=np.expand_dims(df,axis=2)
Y=labels
```

In [9]:

```
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=1)
```

In [10]:

```
num_epoch=30

#available_activations=['sigmoid','tanh','selu','softsign','relu','linear']
available_activations=['sigmoid','tanh']

models=[]
for i in range(len(available_activations)):
    model=Sequential()
    model.add(Conv1D(50,10,activation='relu',input_shape=(1000,1)))
    model.add(AveragePooling1D(pool_size=2))
    model.add(Conv1D(50,10,activation='relu'))
    model.add(AveragePooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(100,activation=available_activations[i]))
    model.add(Dense(20,activation=available_activations[i]))
    model.add(Dense(1,activation=available_activations[i]))
    models.append(model)
print(str(len(available_activations))+" "+"Number of models according to activation function in av
ailable_activations : ")
for i in range(len(models)):
    print("Model "+str(i)+" : ",models[i])

available_optimizer=['SGD','RMSprop','Adagrad','Adadelta','adam','Adamax','Nadam']
learning_rate=0.1
print("Started compiling........")

for i in range(len(models)):
    print("Model "+str(i)+" : ",i)
    models[i].compile(loss='mse',optimizer=available_optimizer[4],metrics=['mean_squared_error','ac
c'])
```

```
2 Number of models according to activation function in available_activations :
Model 0 :  <keras.engine.sequential.Sequential object at 0x0000025906CF8198>
Model 1 :  <keras.engine.sequential.Sequential object at 0x0000025906E395F8>
Started compiling........
Model 0 :  0
Model 1 :  1
```

In [11]:

```
histories=[]
for h in range(len(models)):
    print("Started fitting.....")
    print("Model "+str(h)+" : ",h)#to remove h=1,3,6,7,8
    histories.append(models[h].fit(x_train,y_train,batch_size=100,epochs=num_epoch))
```

```
Started fitting.....
Model 0 :  0
Epoch 1/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2495 - mean_squared_error: 0.2495
- acc: 0.4975
Epoch 2/30
800/800 [==============================] - 3s 3ms/step - loss: 0.2463 - mean_squared_error: 0.2463
- acc: 0.5537
Epoch 3/30
800/800 [==============================] - 3s 3ms/step - loss: 0.2408 - mean_squared_error: 0.2408
- acc: 0.5825
```

```
Epoch 4/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2386 - mean_squared_error: 0.2386
- acc: 0.5800
Epoch 5/30
800/800 [==============================] - 3s 3ms/step - loss: 0.2334 - mean_squared_error: 0.2334
- acc: 0.6100
Epoch 6/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2287 - mean_squared_error: 0.2287
- acc: 0.6150
Epoch 7/30
800/800 [==============================] - 3s 3ms/step - loss: 0.2210 - mean_squared_error: 0.2210
- acc: 0.6413
Epoch 8/30
800/800 [==============================] - 3s 3ms/step - loss: 0.2123 - mean_squared_error: 0.2123
- acc: 0.6687
Epoch 9/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2030 - mean_squared_error: 0.2030
- acc: 0.7075
Epoch 10/30
800/800 [==============================] - 3s 3ms/step - loss: 0.1904 - mean_squared_error: 0.1904
- acc: 0.7350
Epoch 11/30
800/800 [==============================] - 3s 4ms/step - loss: 0.1834 - mean_squared_error: 0.1834
- acc: 0.7500
Epoch 12/30
800/800 [==============================] - 3s 3ms/step - loss: 0.1737 - mean_squared_error: 0.1737
- acc: 0.7725
Epoch 13/30
800/800 [==============================] - 3s 3ms/step - loss: 0.1625 - mean_squared_error: 0.1625
- acc: 0.7912
Epoch 14/30
800/800 [==============================] - 3s 3ms/step - loss: 0.1506 - mean_squared_error: 0.1506
- acc: 0.8225
Epoch 15/30
800/800 [==============================] - 3s 3ms/step - loss: 0.1368 - mean_squared_error: 0.1368
- acc: 0.8537
Epoch 16/30
800/800 [==============================] - 3s 4ms/step - loss: 0.1307 - mean_squared_error: 0.1307
- acc: 0.8562
Epoch 17/30
800/800 [==============================] - 3s 3ms/step - loss: 0.1239 - mean_squared_error: 0.1239
- acc: 0.8687
Epoch 18/30
800/800 [==============================] - 3s 4ms/step - loss: 0.1149 - mean_squared_error: 0.1149
- acc: 0.8687
Epoch 19/30
800/800 [==============================] - 3s 3ms/step - loss: 0.1075 - mean_squared_error: 0.1075
- acc: 0.8788
Epoch 20/30
800/800 [==============================] - 3s 3ms/step - loss: 0.1015 - mean_squared_error: 0.1015
- acc: 0.8913
Epoch 21/30
800/800 [==============================] - 3s 4ms/step - loss: 0.0957 - mean_squared_error: 0.0957
- acc: 0.8963
Epoch 22/30
800/800 [==============================] - 3s 3ms/step - loss: 0.0928 - mean_squared_error: 0.0928
- acc: 0.8975
Epoch 23/30
800/800 [==============================] - 3s 4ms/step - loss: 0.0840 - mean_squared_error: 0.0840
- acc: 0.9100
Epoch 24/30
800/800 [==============================] - 3s 3ms/step - loss: 0.0782 - mean_squared_error: 0.0782
- acc: 0.9200
Epoch 25/30
800/800 [==============================] - 3s 3ms/step - loss: 0.0770 - mean_squared_error: 0.0770
- acc: 0.9150
Epoch 26/30
800/800 [==============================] - 3s 3ms/step - loss: 0.0710 - mean_squared_error: 0.0710
- acc: 0.9250
Epoch 27/30
800/800 [==============================] - 3s 3ms/step - loss: 0.0689 - mean_squared_error: 0.0689
- acc: 0.9325
Epoch 28/30
800/800 [==============================] - 3s 4ms/step - loss: 0.0653 - mean_squared_error: 0.0653
- acc: 0.9350
Epoch 29/30
800/800 [==============================] - 3s 3ms/step - loss: 0.0642 - mean_squared_error: 0.0642
```

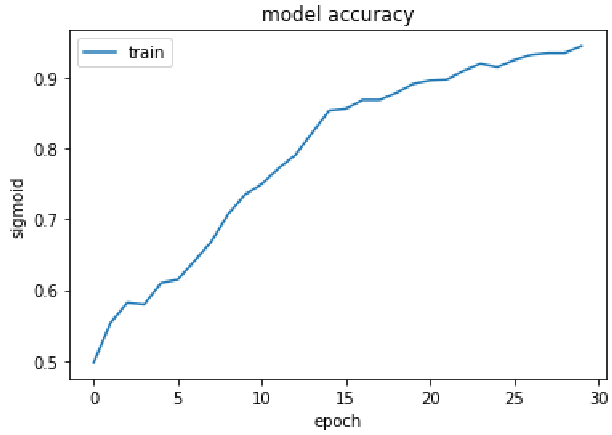
```
- acc: 0.9350
Epoch 30/30
800/800 [==============================] - 3s 4ms/step - loss: 0.0568 - mean_squared_error: 0.0568
- acc: 0.9450
Started fitting.....
Model 1 :  1
Epoch 1/30
800/800 [==============================] - 3s 4ms/step - loss: 0.4919 - mean_squared_error: 0.4919
- acc: 0.5000
Epoch 2/30
800/800 [==============================] - 3s 4ms/step - loss: 0.5047 - mean_squared_error: 0.5047
- acc: 0.4950
Epoch 3/30
800/800 [==============================] - 3s 4ms/step - loss: 0.5046 - mean_squared_error: 0.5046
- acc: 0.4950
Epoch 4/30
800/800 [==============================] - 3s 3ms/step - loss: 0.5044 - mean_squared_error: 0.5044
- acc: 0.4950
Epoch 5/30
800/800 [==============================] - 3s 3ms/step - loss: 0.5041 - mean_squared_error: 0.5041
- acc: 0.4950
Epoch 6/30
800/800 [==============================] - 3s 3ms/step - loss: 0.5035 - mean_squared_error: 0.5035
- acc: 0.4950
Epoch 7/30
800/800 [==============================] - 3s 3ms/step - loss: 0.5014 - mean_squared_error: 0.5014
- acc: 0.4950
Epoch 8/30
800/800 [==============================] - 3s 3ms/step - loss: 0.4939 - mean_squared_error: 0.4939
- acc: 0.4950
Epoch 9/30
800/800 [==============================] - 3s 4ms/step - loss: 0.4571 - mean_squared_error: 0.4571
- acc: 0.4950
Epoch 10/30
800/800 [==============================] - 3s 4ms/step - loss: 0.3463 - mean_squared_error: 0.3463
- acc: 0.4950
Epoch 11/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2541 - mean_squared_error: 0.2541
- acc: 0.5013
Epoch 12/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2503 - mean_squared_error: 0.2503
- acc: 0.5175
Epoch 13/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2447 - mean_squared_error: 0.2447
- acc: 0.5412
Epoch 14/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2368 - mean_squared_error: 0.2368
- acc: 0.5838
Epoch 15/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2333 - mean_squared_error: 0.2333
- acc: 0.6125
Epoch 16/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2400 - mean_squared_error: 0.2400
- acc: 0.5638
Epoch 17/30
800/800 [==============================] - 3s 3ms/step - loss: 0.2302 - mean_squared_error: 0.2302
- acc: 0.5950
Epoch 18/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2179 - mean_squared_error: 0.2179
- acc: 0.6275
Epoch 19/30
800/800 [==============================] - 3s 4ms/step - loss: 0.2012 - mean_squared_error: 0.2012
- acc: 0.6950
Epoch 20/30
800/800 [==============================] - 3s 4ms/step - loss: 0.1742 - mean_squared_error: 0.1742
- acc: 0.7437
Epoch 21/30
800/800 [==============================] - 3s 4ms/step - loss: 0.1415 - mean_squared_error: 0.1415
- acc: 0.8075
Epoch 22/30
800/800 [==============================] - 3s 4ms/step - loss: 0.1173 - mean_squared_error: 0.1173
- acc: 0.8438
Epoch 23/30
800/800 [==============================] - 3s 4ms/step - loss: 0.0994 - mean_squared_error: 0.0994
- acc: 0.8700
Epoch 24/30
800/800 [==============================] - 3s 3ms/step - loss: 0.0903 - mean_squared_error: 0.0903
```

```
- acc: 0.8800
Epoch 25/30
800/800 [==============================] - 3s 4ms/step - loss: 0.0712 - mean_squared_error: 0.0712
- acc: 0.9200
Epoch 26/30
800/800 [==============================] - 3s 4ms/step - loss: 0.0580 - mean_squared_error: 0.0580
- acc: 0.9413
Epoch 27/30
800/800 [==============================] - 3s 4ms/step - loss: 0.0526 - mean_squared_error: 0.0526
- acc: 0.9425
Epoch 28/30
800/800 [==============================] - 3s 4ms/step - loss: 0.0488 - mean_squared_error: 0.0488
- acc: 0.9538
Epoch 29/30
800/800 [==============================] - 3s 4ms/step - loss: 0.0446 - mean_squared_error: 0.0446
- acc: 0.9513
Epoch 30/30
800/800 [==============================] - 3s 4ms/step - loss: 0.0392 - mean_squared_error: 0.0392
- acc: 0.9575
```

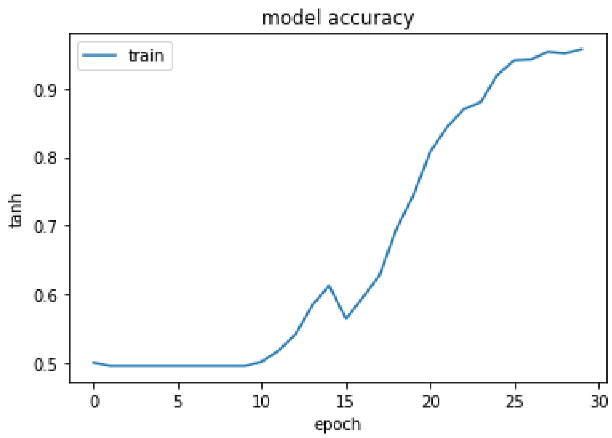
In [12]:

```python
for history in histories:
    print(history.history.keys())
    plt.plot(history.history['acc'])
    plt.title('model accuracy')
    plt.ylabel(available_activations[histories.index(history)])
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
```

```
dict_keys(['loss', 'mean_squared_error', 'acc'])
```




```
dict_keys(['loss', 'mean_squared_error', 'acc'])
```



In [13]:

```python
pre_classes=[]
```

```python
for m in range(len(models)):
    pre_cls=models[m].predict(x_test)
    #for i in range(pre_cls):
     #   if pre_cls[i]>=0.5:
      #       pre_cls[i]=1
      #   else:
       #     pre_cls[i]=0

    pre_classes.append(pre_cls)
```

In [14]:

```python
pre_classes
```

Out[14]:

```
[array([[0.90439963],
        [0.86027145],
        [0.5110382 ],
        [0.1117624 ],
        [0.12512437],
        [0.9085848 ],
        [0.13302767],
        [0.12119031],
        [0.136112  ],
        [0.5892493 ],
        [0.90502965],
        [0.2464943 ],
        [0.85071397],
        [0.10346183],
        [0.4742267 ],
        [0.19423905],
        [0.16059282],
        [0.42449403],
        [0.8930381 ],
        [0.67332006],
        [0.5986308 ],
        [0.903401  ],
        [0.9247646 ],
        [0.5884159 ],
        [0.86116236],
        [0.16793194],
        [0.10421017],
        [0.67061126],
        [0.92140347],
        [0.12117523],
        [0.8599657 ],
        [0.7798533 ],
        [0.29177532],
        [0.92607737],
        [0.1531902 ],
        [0.11578605],
        [0.43951645],
        [0.9044565 ],
        [0.76532197],
        [0.90462935],
        [0.92448455],
        [0.93081254],
        [0.1813634 ],
        [0.15902096],
        [0.1105642 ],
        [0.7012451 ],
        [0.4652852 ],
        [0.5950733 ],
        [0.6469564 ],
        [0.49042845],
        [0.12579826],
        [0.5853909 ],
        [0.11590374],
        [0.15952721],
        [0.12328273],
        [0.25681552],
        [0.94479716],
        [0.9163817 ],
        [0.4836333 ],
```

```
[0.7736126 ],
[0.26259282],
[0.11371428],
[0.13676193],
[0.8957249 ],
[0.9338034 ],
[0.7927536 ],
[0.76262045],
[0.7455189 ],
[0.8228531 ],
[0.26908726],
[0.92204773],
[0.18003505],
[0.12567052],
[0.8553356 ],
[0.45736217],
[0.64084315],
[0.12656954],
[0.70724905],
[0.20780838],
[0.87548673],
[0.17682701],
[0.8934529 ],
[0.92149854],
[0.9385898 ],
[0.19585186],
[0.16827375],
[0.64814293],
[0.8651685 ],
[0.649401  ],
[0.92895883],
[0.6494683 ],
[0.24537903],
[0.89335555],
[0.86927795],
[0.76803994],
[0.77374446],
[0.7334236 ],
[0.9068838 ],
[0.6071971 ],
[0.10417795],
[0.18054682],
[0.3294621 ],
[0.8068044 ],
[0.2495335 ],
[0.20974457],
[0.93578887],
[0.154567  ],
[0.8106657 ],
[0.46345934],
[0.19576481],
[0.86786616],
[0.8079413 ],
[0.8701881 ],
[0.12733701],
[0.3538469 ],
[0.10839307],
[0.9155575 ],
[0.9478786 ],
[0.2307032 ],
[0.59596896],
[0.30929768],
[0.17777008],
[0.12803575],
[0.10786268],
[0.12752998],
[0.75498104],
[0.92885244],
[0.83036226],
[0.8694885 ],
[0.8384381 ],
[0.91340137],
[0.11839706],
[0.62945914],
[0.12694976],
[0.59561145],
[0.90282416],
```

```
[0.73307824],
[0.61459565],
[0.1546295 ],
[0.8837302 ],
[0.93185914],
[0.13529956],
[0.47435775],
[0.15725264],
[0.13887835],
[0.8975543 ],
[0.9276874 ],
[0.7975544 ],
[0.94840825],
[0.9398707 ],
[0.10560557],
[0.7537075 ],
[0.33732685],
[0.79078305],
[0.7342129 ],
[0.78701717],
[0.84873116],
[0.12203228],
[0.13622311],
[0.18402848],
[0.10767168],
[0.261419  ],
[0.17521614],
[0.5618557 ],
[0.82054174],
[0.40018284],
[0.28646618],
[0.83431023],
[0.7063536 ],
[0.13225931],
[0.35448694],
[0.42445084],
[0.2037912 ],
[0.16252151],
[0.5063383 ],
[0.57194036],
[0.11909193],
[0.88380086],
[0.87651014],
[0.7790059 ],
[0.8371399 ],
[0.38819045],
[0.81703043],
[0.12642431],
[0.8329972 ],
[0.36182305],
[0.1648829 ],
[0.13141945],
[0.7467655 ],
[0.1342301 ],
[0.81704   ],
[0.25211054],
[0.8194057 ],
[0.12526095],
[0.34783483],
[0.3893103 ],
[0.8598442 ],
[0.8813135 ],
[0.8855031 ],
[0.10775343]], dtype=float32), array([[ 0.93994933],
[ 0.9614561 ],
[ 0.6537072 ],
[-0.08643335],
[ 0.3586638 ],
[ 0.94314814],
[-0.01102522],
[ 0.45676038],
[ 0.09273316],
[ 0.25772333],
[ 0.9325944 ],
[ 0.17220362],
[ 0.8440013 ],
[ 0.00211747],
```

```
[ 0.39718717],
[ 0.11437977],
[-0.04562643],
[ 0.58223736],
[ 0.9042029 ],
[ 0.84093827],
[ 0.8502221 ],
[ 0.9355726 ],
[ 0.96053374],
[ 0.69967896],
[ 0.90452164],
[ 0.39097875],
[-0.1448662 ],
[ 0.8057305 ],
[ 0.9002039 ],
[ 0.08171769],
[ 0.9067095 ],
[ 0.87294465],
[ 0.39041454],
[ 0.93386066],
[ 0.4403869 ],
[-0.15149483],
[ 0.6152802 ],
[ 0.9373567 ],
[ 0.95988023],
[ 0.9565268 ],
[ 0.9441378 ],
[ 0.9851254 ],
[-0.07147795],
[ 0.41859055],
[-0.01673243],
[ 0.8371896 ],
[ 0.43545106],
[ 0.70766443],
[ 0.5786636 ],
[ 0.43827623],
[ 0.12289744],
[ 0.27983743],
[ 0.40809023],
[ 0.16479813],
[ 0.00827811],
[ 0.24342544],
[ 0.966319  ],
[ 0.9272388 ],
[ 0.6877299 ],
[ 0.87982893],
[ 0.22299276],
[-0.09335505],
[ 0.14640519],
[ 0.95748925],
[ 0.96422   ],
[ 0.9104831 ],
[ 0.68942285],
[ 0.8491992 ],
[ 0.8429856 ],
[ 0.36089227],
[ 0.9007446 ],
[ 0.3096852 ],
[ 0.02001753],
[ 0.9419146 ],
[ 0.44826695],
[ 0.82956254],
[ 0.03826871],
[ 0.5798177 ],
[ 0.07815001],
[ 0.8896292 ],
[-0.01879568],
[ 0.94628656],
[ 0.9100126 ],
[ 0.9359414 ],
[ 0.5375771 ],
[ 0.36996612],
[ 0.8866763 ],
[ 0.9345399 ],
[ 0.5292598 ],
[ 0.970043  ],
[ 0.8275957 ],
```

```
[ 0.66388893],
[ 0.95911676],
[ 0.86813176],
[ 0.8780674 ],
[ 0.90839493],
[ 0.9358105 ],
[ 0.9544464 ],
[ 0.52976775],
[-0.02592514],
[-0.09881848],
[-0.01463253],
[ 0.9046747 ],
[ 0.73809516],
[ 0.00513171],
[ 0.95972025],
[ 0.1527502 ],
[ 0.8372246 ],
[ 0.90285885],
[ 0.4691438 ],
[ 0.8664895 ],
[ 0.8220032 ],
[ 0.8403465 ],
[-0.08719315],
[ 0.34045842],
[ 0.00836826],
[ 0.94341654],
[ 0.9567871 ],
[ 0.09830005],
[ 0.6208601 ],
[ 0.05724577],
[ 0.54230285],
[ 0.02406881],
[ 0.01075795],
[ 0.47694027],
[ 0.8788334 ],
[ 0.96076864],
[ 0.9432168 ],
[ 0.86765295],
[ 0.8777714 ],
[ 0.9568588 ],
[-0.16834512],
[ 0.8394032 ],
[ 0.29106185],
[ 0.70672834],
[ 0.94076824],
[ 0.9042489 ],
[ 0.77786505],
[ 0.0928627 ],
[ 0.9688693 ],
[ 0.80999136],
[-0.09631304],
[ 0.9228758 ],
[ 0.38663295],
[-0.08082683],
[ 0.9455013 ],
[ 0.9614188 ],
[ 0.8001041 ],
[ 0.9781531 ],
[ 0.96520406],
[-0.01777263],
[ 0.85874236],
[ 0.2301726 ],
[ 0.92921335],
[ 0.8828367 ],
[ 0.9420905 ],
[ 0.8730559 ],
[ 0.52012014],
[ 0.1718474 ],
[ 0.29322186],
[-0.01728658],
[ 0.07943096],
[ 0.14741318],
[ 0.94886297],
[ 0.8488778 ],
[ 0.25649464],
[ 0.38746202],
[ 0.94037753],
```

```
       [ 0.87455845],
       [ 0.00404056],
       [ 0.94446445],
       [ 0.4780955 ],
       [-0.07859158],
       [ 0.02432431],
       [ 0.899399  ],
       [ 0.14502329],
       [ 0.02059639],
       [ 0.9312695 ],
       [ 0.95285946],
       [ 0.8328451 ],
       [ 0.8349327 ],
       [-0.01458055],
       [ 0.87817675],
       [ 0.69763   ],
       [ 0.94136965],
       [ 0.4646817 ],
       [ 0.17862886],
       [ 0.24312648],
       [ 0.9476845 ],
       [ 0.37825012],
       [ 0.92094153],
       [ 0.6745453 ],
       [ 0.1068397 ],
       [ 0.49126485],
       [ 0.8026149 ],
       [ 0.74772006],
       [ 0.95721537],
       [ 0.9303328 ],
       [ 0.94064057],
       [-0.099307  ]], dtype=float32)]
```

In [15]:

```
confusion_matrices=[]
predicted_values=[]
for p in pre_classes:
    converted_p_to_binary=p
    converted_p_to_binary=converted_p_to_binary.ravel()

    for i in range(len(converted_p_to_binary)):
        if converted_p_to_binary[i]>=0.5:
            converted_p_to_binary[i]=1
        else:converted_p_to_binary[i]=0

    converted_p_to_binary=np.reshape(converted_p_to_binary,(-1,1))

    predicted_values.append(converted_p_to_binary)

    cm1 = confusion_matrix(y_test,converted_p_to_binary)
    confusion_matrices.append(cm1)
```

In [16]:

```
print("Predicted Values : ")
print(predicted_values)
```

```
Predicted Values :
[array([[1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
```

```
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
```

```
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[0.],
[0.],
[0.],
[1.],
[0.],
[0.],
[1.],
[0.],
[1.],
[0.],
[0.],
[1.],
[1.],
[1.],
[0.],
[0.],
[0.],
[1.],
[1.],
[0.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[1.],
[1.],
[1.],
[1.],
[1.],
[1.],
[0.],
[1.],
[0.],
[1.],
[1.],
[1.],
[1.],
[0.],
[1.],
[1.],
[0.],
[0.],
[0.],
[0.],
[1.],
[1.],
[1.],
[1.],
[1.],
[0.],
[1.],
[0.],
[1.],
[1.],
[1.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[1.],
[1.],
[0.],
[0.],
[1.],
[1.],
```

```
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.]], dtype=float32), array([[1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
```

```
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
```

```
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.]], dtype=float32)]
```

In [17]:

```
print("Confusion Matrices :")
for cm1 in confusion_matrices:
    print(cm1)
```

```
Confusion Matrices :
[[74 22]
 [19 85]]
[[69 27]
 [13 91]]
```

In [18]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

print("For 2D confusion matrix")
for i,cm1 in enumerate(confusion_matrices):
    tn, fp, fn, tp = cm1.ravel()
    accuracy=(tp+tn)/(tp+tn+fp+fn)
    accuracy*=100
    recall=tp/(tp+fn)
    recall*=100
    precision=tp/(tp+fp)
    precision*=100
    f_measure=2*(recall*precision)/(recall+precision)


    print("accuracy : ",accuracy)
    print("recall : ",recall)
    print("precision : ",precision)
    print("f_measure : ",f_measure)

    actual=y_test
    predicted=predicted_values[i]
    results = confusion_matrix(actual, predicted)
    print('Confusion Matrix :')
    print(results)
    print('Accuracy Score :',100*accuracy_score(actual, predicted))
    print('Report : ')
    print(classification_report(actual, predicted) )
```

```
For 2D confusion matrix
accuracy :  79.5
recall :  81.73076923076923
precision :  79.43925233644859
f_measure :  80.56872037914692
Confusion Matrix :
[[74 22]
 [19 85]]
Accuracy Score : 79.5
Report : 
              precision    recall  f1-score   support

           0       0.80      0.77      0.78        96
           1       0.79      0.82      0.81       104

    accuracy                           0.80       200
   macro avg       0.80      0.79      0.79       200
weighted avg       0.80      0.80      0.79       200

accuracy :  80.0
recall :  87.5
precision :  77.11864406779661
f_measure :  81.98198198198199
Confusion Matrix :
[[69 27]
 [13 91]]
Accuracy Score : 80.0
Report : 
              precision    recall  f1-score   support

           0       0.84      0.72      0.78        96
```

```
           1       0.77      0.88      0.82       104

    accuracy                           0.80       200
   macro avg       0.81      0.80      0.80       200
weighted avg       0.80      0.80      0.80       200
```

In [ ]:

```

```

In [ ]:

```

```

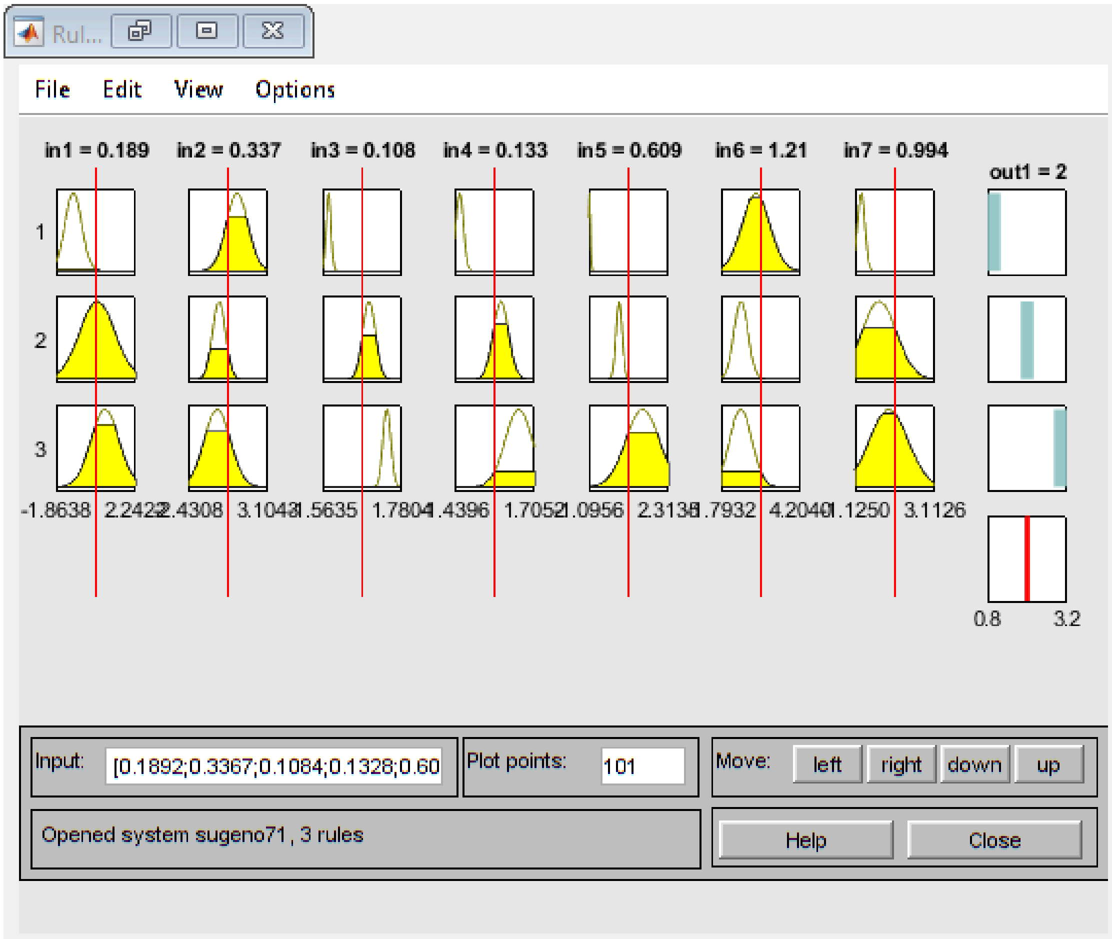
```
clc;
clear all;
close all;

table1 = xlsread('data4.xlsx');
%normalization of data
table1(:,1:end-1) = (table1(:,1:end-1)-mean(table1(:,1:end-1)))./
std(table1(:,1:end-1));

X=table1(:,1:end-1);
Y=table1(:,end);


C = cvpartition(Y,'HoldOut',0.3);
tr = C.training;
te = C.test;
Xtr = X(tr,:);
Xte = X(te,:);
Ytr = Y(tr,:);
Yte = Y(te,:);

input=Xtr;
test=Xte;
target_tr=Ytr;
target_te=Yte;
%first classifier has at epoch 100 recognition rate(98.0952)
epoch=100;
epochs=1000;
class=3;
clustersize=1;
clustsize=2;
[fismat,outputs,recog_tr,recog_te,labels,performance]=scg_nfc(input,target_tr,test
%second classifier has more initial recognition rate(96.1905)
[fismat1,outputs,recog_tr,recog_te,labels,performance]...

 =scg_pow_nfc(input,target_tr,test,target_te,epoch,class,clustersize);

%third classifier has performance 0.046297 at 100 epoch
[fismat3,outputs,recog_tr,recog_te,labels,performance]...

 =scg_nfclass_speedup(input,target_tr,test,target_te,epoch,class,clustersize);

%feature selection
[fismat4,feature,outputs,recog_tr,recog_te,labels,performance]=...
    nfc_feature_select([input;test],
[target_tr;target_te],test,target_te,epochs,class);

%Classification with selected features
[fismat5,outputs,recog_tr,recog_te,labels,performance]...

 =scg_pow_nfc(input(:,feature.selected),target_tr,test(:,feature.selected),target_
```

the classification with NFC is realizing

initial recognation rate= 91.4286 initial perform= 0.119288
epoch 25   recog 95.2381 recog_test 97.7778 performans   0.0667263
epoch 50   recog 97.1429 recog_test 97.7778 performans   0.0459748
epoch 75   recog 98.0952 recog_test 95.5556 performans   0.0424229
epoch 100   recog 98.0952 recog_test 95.5556 performans 0.0416244
the classification with NFC_LH is realizing
initial recognation rate= 91.4286 initial perform= 0.119288
epoch 25   recog 96.1905 recog_test 97.7778 performans   0.053588
epoch 50   recog 98.0952 recog_test 95.5556 performans   0.0427117
epoch 75   recog 98.0952 recog_test 95.5556 performans   0.0417137
epoch 100   recog 98.0952 recog_test 95.5556 performans 0.0404206
initial recognation rate= 91.4286 initial perform= 0.119288
epoch 25   recog_train 95.2381 recog_test 97.7778 performance 0.0720237
epoch 50   recog_train 96.1905 recog_test 97.7778 performance 0.0543865
epoch 75   recog_train 97.1429 recog_test 95.5556 performance 0.044064
epoch 100   recog_train 98.0952 recog_test 95.5556 performance 0.042601
initial recognation rate= 92.6667 initial perform= 0.607901
epoch 25   recog_train 98 recog_test 97.7778 performance 0.0283057
epoch 50   recog_train 98 recog_test 97.7778 performance 0.026438
epoch 75   recog_train 98.6667 recog_test 97.7778 performance 0.0248383
epoch 100   recog_train 99.3333 recog_test 100 performance 0.0229018
epoch 125   recog_train 99.3333 recog_test 100 performance 0.0196366
epoch 150   recog_train 99.3333 recog_test 100 performance 0.0154289
epoch 175   recog_train 99.3333 recog_test 100 performance 0.0150124
epoch 200   recog_train 99.3333 recog_test 100 performance 0.0146258
epoch 225   recog_train 99.3333 recog_test 100 performance 0.0144356
epoch 250   recog_train 99.3333 recog_test 100 performance 0.0141345
epoch 275   recog_train 99.3333 recog_test 100 performance 0.0140335
epoch 300   recog_train 99.3333 recog_test 100 performance 0.0139522
epoch 325   recog_train 99.3333 recog_test 100 performance 0.0137123
epoch 350   recog_train 99.3333 recog_test 100 performance 0.0136514
epoch 375   recog_train 99.3333 recog_test 100 performance 0.0136205

epoch 400 recog_train 99.3333 recog_test 100 performance 0.0135877
epoch 425 recog_train 99.3333 recog_test 100 performance 0.0135643
epoch 450 recog_train 99.3333 recog_test 100 performance 0.013542
epoch 475 recog_train 99.3333 recog_test 100 performance 0.0135048
epoch 500 recog_train 99.3333 recog_test 100 performance 0.0134075
epoch 525 recog_train 99.3333 recog_test 100 performance 0.0133967
epoch 550 recog_train 99.3333 recog_test 100 performance 0.0133876
epoch 575 recog_train 99.3333 recog_test 100 performance 0.0133833
epoch 600 recog_train 99.3333 recog_test 100 performance 0.0133751
epoch 625 recog_train 99.3333 recog_test 100 performance 0.0133494
epoch 650 recog_train 99.3333 recog_test 100 performance 0.0133452
epoch 675 recog_train 99.3333 recog_test 100 performance 0.0133411
epoch 700 recog_train 99.3333 recog_test 100 performance 0.01334
epoch 725 recog_train 99.3333 recog_test 100 performance 0.0133394
epoch 750 recog_train 99.3333 recog_test 100 performance 0.0133389
epoch 775 recog_train 99.3333 recog_test 100 performance 0.0133364
epoch 800 recog_train 99.3333 recog_test 100 performance 0.0133358
epoch 825 recog_train 99.3333 recog_test 100 performance 0.0133355
epoch 850 recog_train 99.3333 recog_test 100 performance 0.013335
epoch 875 recog_train 99.3333 recog_test 100 performance 0.013334
epoch 900 recog_train 99.3333 recog_test 100 performance 0.0133339
epoch 925 recog_train 99.3333 recog_test 100 performance 0.0133338
epoch 950 recog_train 99.3333 recog_test 100 performance 0.0133337
epoch 975 recog_train 99.3333 recog_test 100 performance 0.0133337
epoch 1000 recog_train 99.3333 recog_test 100 performance 0.0133336
the classification with NFC_LH is realizing
initial recognation rate= 96.1905 initial perform= 0.0643495
epoch 25 recog 98.0952 recog_test 97.7778 performans 0.0425365
epoch 50 recog 98.0952 recog_test 97.7778 performans 0.0398737

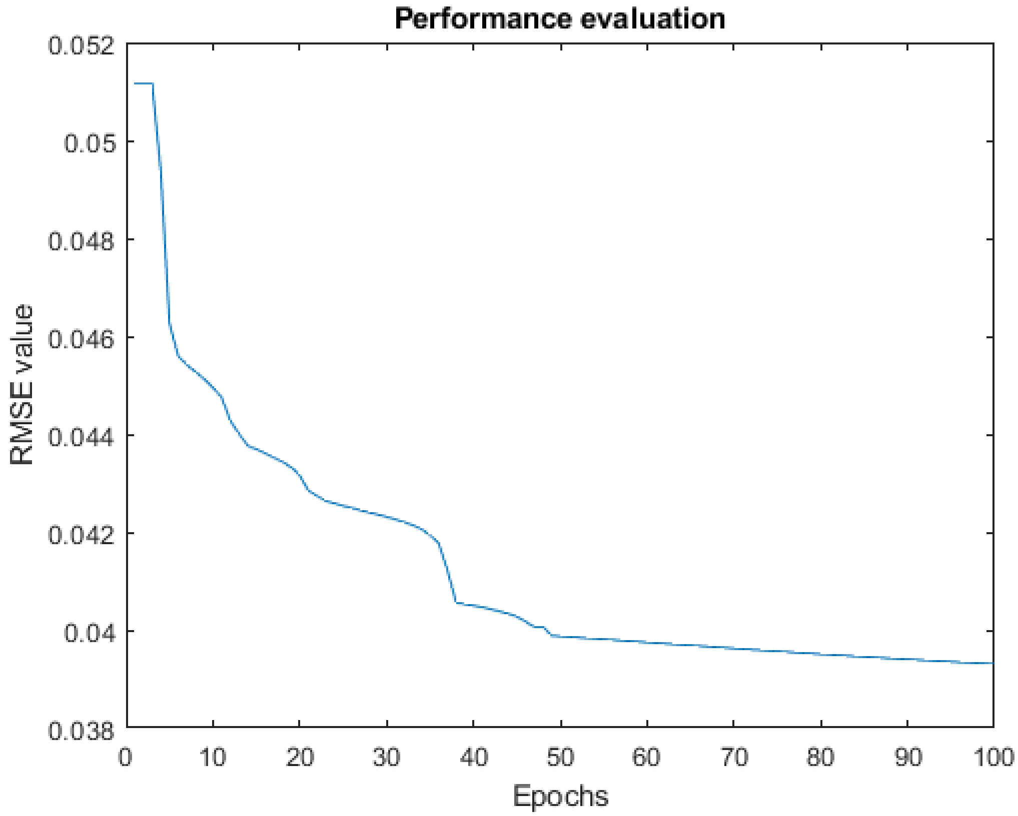
```
epoch 75   recog  98.0952  recog_test  97.7778  performans   0.0395663
epoch 100   recog  98.0952  recog_test  97.7778  performans
 0.0393082
```

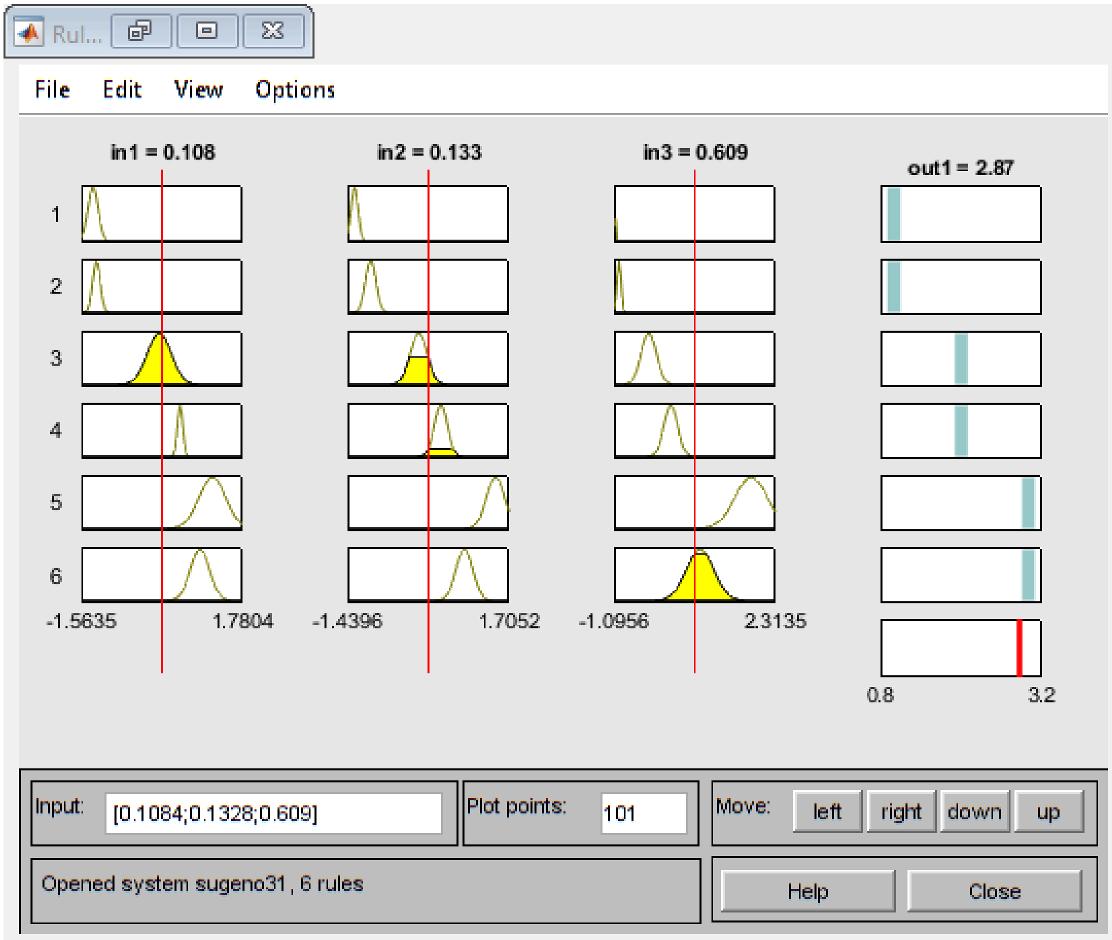


Rul...
File
Edit
View
Options
in1 = 0.189
in2 = 0.337
in3 = 0.108
in4 = 0.133
in5 = 0.609
in6 = 1.21
in7 = 0.994
out1 = 2
1
2
3
-1.8638
2.2422
2.4308
3.1043
1.5635
1.7804
1.4396
1.7052
1.0956
2.3138
1.7932
4.2040
1.1250
3.1126
0.8
3.2
Input:
[0.1892;0.3367;0.1084;0.1328;0.60
Plot points:
101
Move:
left
right
down
up
Opened system sugeno71 , 3 rules
Help
Close


Rul...
File Edit View Options
in1 = 0.189
in2 = 0.337
in3 = 0.108
in4 = 0.133
in5 = 0.609
in6 = 1.21
in7 = 0.994
out1 = 2
1
2
3
-1.8638 2.2422
2.4308 3.1043
1.5635 1.7804
1.4396 1.7052
1.0956 2.3138
1.7932 4.2040
1.1250 3.1126
0.8 3.2
Input: [0.1892;0.3367;0.1084;0.1328;0.60
Plot points: 101
Move: left right down up
Opened system sugeno71 , 3 rules
Help Close


Rule Viewer: sugeno71
File
Edit
View
Options
in1 = 0.31
in2 = 0.337
in3 = 0.108
in4 = 0.133
in5 = 0.609
in6 = 1.21
in7 = 1.3
out1 = 2
1
2
3
-1.8638
2.4832
2.4308
3.1043
1.5635
1.7804
1.4396
1.7052
1.0956
2.3138
1.7932
4.2040
1.1250
3.7254
0.8
3.2
Input:
[0.31;0.3367;0.1084;0.1328;0.609;
Plot points:
101
Move:
left
right
down
up
Opened system sugeno71 , 3 rules
Help
Close


Performance evaluation
RMSE value
Epochs
0.052
0.05
0.048
0.046
0.044
0.042
0.04
0.038
0
10
20
30
40
50
60
70
80
90
100



Published with MATLAB® R2019a

```
function
 [fismat,outputs,recog_tr,recog_te,labels,performance]=scg_nfc(input,target_tr,tes
% In this program, the neuro-fuzzy classifier parameters are adapted
 by Scaled conjugate gradient method.
%INPUTS
%input[N,s2]: training data
%target_tr[N,1]: the target_tr values of training data
%test[m,s2]: test data
%target_te[m,1]: the target values of test data
%stepsize: The maximum iteration number
%class: Number of classes
%clustsize: Number of cluster of each class
%OUTPUTS
%outputs.center[s1,s2]: The center values of Gaussian functions
%outputs.sigma_nf[s1,s2]: The width values of Gaussian functions
%recog_tr: The recognition rate of training data
%recog_te: The recognition rate of test data
%labels.input=out_tr[N,1]: The produced class labels of training data
 obtained from NFC
%labels.test=out_te[m,1]: The produced class labels of test data
 obtained from NFC
%performance: root mean square error of training data
%fismat: Demonstration of NFC in fuzzy viewer.
%       Written by Dr. Bayram Ceti?li Suleyman Demirel University
 Computer
%       Engineeering Isparta Turkey
warning off;
close all;
fprintf('the classification with NFC is realizing\n');
performance=zeros(stepsize,1);
rr=single(zeros(stepsize/25+1,4));
m=size(test,1);
[N,s2] = size(input);
clear data;
center=[];sigma_nf=[];targ=zeros(N,class);w=zeros(clustsize*class,class);w=sparse(
sir=1;
for i=1:class
    [v,vv]=find(target_tr==i);
    temp=input(v,:);
    cent=mean(temp);
    [idc,cc]=kmeans(temp,clustsize,'MaxIter',10);
    center(sir:sir+clustsize-1,:)=cc;
    for j=1:clustsize
        ind=idc==j;
        sigma_nf(sir+j-1,:)=std(temp(ind,:));
        w(sir+j-1,i)=sum(ind)/size(v,1);
    end
    targ(v,i)=1;
    sir=sir+clustsize;
end
for i=1:s2
    ind=sigma_nf(:,i)<=0;
```

```
        sigma_nf(ind,i)=std(input(:,i));
    end
    [s1,s2]=size(center);
    X = zeros(2*s1*s2,1);
    X(1:s1*s2,1)=reshape(center',s1*s2,1);
    X(s1*s2+1:2*s1*s2,1)=reshape(sigma_nf',s1*s2,1);
    % Initial performance
    [gX,out,w]=grad_anfis_aralik(input,center,sigma_nf,w,targ,class);
    [tt,tp]=max(out');out_tr=tp';
    init=sum(target_tr==out_tr)/N*100;
    perf=sum(sum((targ-out).^2))/N;
    rr(1,:)=[0 init init perf];
    fprintf('initial recognation rate= %g initial perform= %g',init,perf);
    fprintf('\n');
    result.center{1}=center;
    result.sigma_nf{1}=sigma_nf;
    result.w{1}=w;
    % Intial gradient and old gradient
    gX_old = gX;
    % Initial search direction and norm
    dX = -gX;
    nrmsqr_dX = dX'*dX;
    norm_dX = sqrt(nrmsqr_dX);
    % Initial training parameters and flag
    sigma=5.0e-5;
    lambda=5.0e-7;
    success = 1;
    lambdab = 0;
    lambdak = lambda;
    num_X=1;
    sir=2;
    %The tarining of NFC with SCG is starting.
    for epoch=1:stepsize
        % If success is true, calculate second order information
        if (success == 1)
            sigmak = sigma/norm_dX;
            X_temp = X + sigmak*dX;
            center=(reshape(X_temp(1:s1*s2,1),s2,s1))';
            sigma_nf=(reshape(X_temp(s1*s2+1:2*s1*s2,1),s2,s1))';
            for i=1:s2
                ind=sigma_nf(:,i)<=0;
                sigma_nf(ind,i)=std(input(:,i));
            end

     [gX_temp,out,w]=grad_anfis_aralik(input,center,sigma_nf,w,targ,class);
            sk = (gX_temp - gX)/sigmak;
            deltak = dX'*sk;
        end
        % Scale deltak
        deltak = deltak + (lambdak - lambdab)*nrmsqr_dX;
        % IF deltak <= 0 then make the Hessian matrix positive definite
        if (deltak <= 0)
            lambdab = 2*(lambdak - deltak/nrmsqr_dX);
            deltak = -deltak + lambdak*nrmsqr_dX;
```

```
        lambdak = lambdab;
    end
    % Calculate step size
    muk = -dX'*gX;
    alphak = muk/deltak;
    if muk==0

[rr(sir,:),recog_tr,recog_te,out_tr,out_te]=performance_measurement(epoch,perf_te
          break;
        break;
    end

    % Calculate the comparison parameter  X_temp = X + alphak*dX;
    X_temp = X + alphak*dX;
    center=(reshape(X_temp(1:s1*s2,1),s2,s1))';
    sigma_nf=(reshape(X_temp(s1*s2+1:2*s1*s2,1),s2,s1))';
    for i=1:s2
        ind=sigma_nf(:,i)<=0;
        sigma_nf(ind,i)=std(input(:,i));
    end
    out=output_anfis_aralik(input,center,sigma_nf,w,class);
    perf_temp=sum(sum((targ-out).^2))/N;
    difk = 2*deltak*(perf - perf_temp)/(muk^2);

    % If difk >= 0 then a successful reduction in error can be made
    if (difk >= 0)
        gX_old = gX;
        X = X_temp;
        center=(reshape(X(1:s1*s2,1),s2,s1))';
        sigma_nf=(reshape(X(s1*s2+1:2*s1*s2,1),s2,s1))';
        for i=1:s2
            ind=sigma_nf(:,i)<=0;
            sigma_nf(ind,i)=std(input(:,i));
        end

[gX,out,w]=grad_anfis_aralik(input,center,sigma_nf,w,targ,class);
        perf=sum(sum((targ-out).^2))/N;
        % Initial gradient and old gradient
        lambdab = 0;
        success = 1;
        perf = perf_temp;

        % Restart the algorithm every num_X iterations
        if rem(epoch,num_X)==0
            dX = -gX;
        else
            betak = (gX'*gX - gX'*gX_old)/muk;
            dX = -gX + betak*dX;
        end
        nrmsqr_dX = dX'*dX;
        norm_dX = sqrt(nrmsqr_dX);

        % If difk >= 0.75, then reduce the scale parameter
        if (difk >= 0.75)
```

```
            lambdak = 0.25*lambdak;
        end
    else
        lambdab = lambdak;
        success = 0;
    end

    % If difk < 0.25, then increase the scale parameter
    if (difk < 0.25)
        lambdak_old=lambdak;
        lambdak = lambdak + deltak*(1 - difk)/nrmsqr_dX;
        if isinf(lambdak)
            lambdak=lambdak_old*1.2;
        end
    end
    performance(epoch,1)=perf;
    if rem(epoch,25)==0 | rem(epoch,stepsize)==0

 [rr(sir,:),recog_tr,recog_te,out_tr,out_te]=performance_measurement(epoch,perf_te
        result.center{sir}=center;
        result.sigma_nf{sir}=sigma_nf;
        result.w{sir}=w;
        sir=sir+1;
        if(rr(sir-2,4)==rr(sir-1,4))
            break;
        end
    end
%gg(epoch,:)=gX;
end
[~,best]=max(rr(:,2));
outputs.center=result.center{best};
outputs.sigma_nf=result.sigma_nf{best};
outputs.w=result.w{best};
labels.input=out_tr;
labels.test=out_te;
figure;plot(performance);
title('Performance evaluation');
xlabel('Epochs');
ylabel('RMSE value');
fismat=nfc_fis(double(input),double(target_tr),double(center),double(sigma_nf),cla
%fismat=nfc_fis(input,target_tr,center,sigma_nf,class,clustsize);
ruleview(fismat);
%Functions
%***************************************************************************
%Calculation of gradients and NFC outputs
function [gX,
 out,w]=grad_anfis_aralik(input,center,sigma_nf,w,targ,class)
[s1,s2]=size(center);
N=size(input,1);
clust=s1/class;
if s2>1
    for i=1:s1
        temp=exp(-0.5*[(input-ones(N,1)*center(i,:)).^2]./
(ones(N,1)*sigma_nf(i,:)).^2);
```

```
        mem(:,i)=[prod([temp]')]';
    end
elseif s2==1
    for i=1:s1
        temp=exp(-0.5*[(input-ones(N,1)*center(i,:)).^2]./
(ones(N,1)*sigma_nf(i,:)).^2);
        mem(:,i)=temp;
    end
end
out_t=mem*w;
top=sum(out_t,2);
ind=top==0;
top(ind)=0.01;
out=out_t./(top*ones(1,class));
tempoc=zeros(s1*s2,1);tempos=zeros(s1*s2,1);
gX=zeros(2*s1*s2,1);
t1=-2*(targ-out);
sira=1;sir=1;
for k=1:class
    for j=1:s1/class
        tempoc(sir:sir+s2-1)=tempoc(sir:sir
+s2-1)+[(input-ones(N,1)*center(sira,:))./
(ones(N,1)*sigma_nf(sira,:).^2)]'*[mem(:,sira).*t1(:,k).*(1-
out(:,k))./top(:,1)*w(sira,k)];
        tempos(sir:sir+s2-1)=tempos(sir:sir
+s2-1)+[(input-ones(N,1)*center(sira,:)).^2./
(ones(N,1)*sigma_nf(sira,:).^3)]'*[mem(:,sira).*t1(:,k).*(1-
out(:,k))./top(:,1)*w(sira,k)];
        sir=sir+s2;sira=sira+1;
    end
end
gX=[tempoc;tempos]/N;
%**************************************************************************
%Calculation of only outputs
function [out]=output_anfis_aralik(input,center,sigma_nf,w,class)
[s1,s2]=size(center);
clustsize=s1/class;
N=size(input,1);
if s2>1
    for i=1:s1
        temp=exp(-0.5*[(input-ones(N,1)*center(i,:)).^2]./
(ones(N,1)*sigma_nf(i,:)).^2);
        mem(:,i)=[prod([temp]')]';
    end
elseif s2==1
    for i=1:s1
        temp=exp(-0.5*[(input-ones(N,1)*center(i,:)).^2]./
(ones(N,1)*sigma_nf(i,:)).^2);
        mem(:,i)=temp;
    end
end
out_t=mem*w;
top=sum(out_t,2);
ind=top==0;
```

```
top(ind)=0.01;
out=out_t./(top*ones(1,class));
%**************************************************************************
function
 [rr,recog_tr,recog_te,out_tr,out_te]=performance_measurement(epoch,perf_temp,out,
N=size(target_tr,1);
m=size(target_te,1);
[tt,tp]=max(out');out_tr=tp';
indx=(out_tr==target_tr);
recog_tr=sum(indx)/N*100;
output=output_anfis_aralik(test,center,sigma_nf,w,class);
[tt,tp]=max(output');out_te=tp';
indx=(out_te==target_te);
recog_te=sum(indx)/m*100;
fprintf('epoch %g   recog  %g  recog_test  %g  performans   %g
\n',epoch,recog_tr,recog_te,perf_temp);
rr=[epoch recog_tr recog_te perf_temp];

function
 [fismat,outputs,recog_tr,recog_te,labels,performance]=scg_pow_nfc(input,target_tr
% In this program, the neuro-fuzzy classifier parameters are adapted
 by Scaled conjugate gradient method.
%Also, the power values are applied to the fuzzy sets and adapted with
 SCG
%method.
%INPUTS
%input[N,s2]: training data
%target_tr[N,1]: the target_tr values of training data
%test[m,s2]: test data
%target_te[m,1]: the target values of test data
%stepsize: The maximum iteration number
%class: Number of classes
%clustsize: Number of cluster of each class
%OUTPUTS
%outputs.center[s1,s2]: The center values of Gaussian functions
%outputs.sigma_nf[s1,s2]: The width values of Gaussian functions
%recog_tr: The recognition rate of training data
%recog_te: The recognition rate of test data
%labels.input=out_tr[N,1]: The produced class labels of training data
 obtained from NFC
%labels.test=out_te[m,1]: The produced class labels of test data
 obtained from NFC
%performance: root mean square error of training data
%outputs.pw[s1,s2]:The power values of Gaussian functions
%fismat: Demonstration of NFC in fuzzy viewer.
%       Written by Dr. Bayram Ceti?li Suleyman Demirel University
 Computer
%       Engineeering Isparta Turkey
close all;
performance=single(zeros(stepsize,1));
warning off;
fprintf('the classification with NFC_LH is realizing\n');
rr=single(zeros(stepsize/25,4));
m=size(test,1);
```

```
[N,s2] = size(input);
input=single(input);test=single(test);
target_tr=uint8(target_tr);target_te=uint8(target_te);
center=single(zeros(clustsize*class,s2));sigma_nf=single(zeros(clustsize*class,s2)
targ=uint8(zeros(N,class));w=single(zeros(clustsize*class,class));
sir=1;
for i=1:class
    [v,vv]=find(target_tr==i);
    temp=input(v,:);
    cent=mean(temp);
    [idc,cc]=kmeans(temp,clustsize,'MaxIter',s2);
    center(sir:sir+clustsize-1,:)=cc;
    for j=1:clustsize
        ind=idc==j;
        sigma_nf(sir+j-1,:)=std(temp(ind,:));
        w(sir+j-1,i)=sum(ind)/size(v,1);
    end
    targ(v,i)=1;
    sir=sir+clustsize;
end
clear cent m1 v1 w1
for i=1:s2
    ind=sigma_nf(:,i)<=0;
    sigma_nf(ind,i)=std(input(:,i));
end
clear ind
[s1,s2]=size(center);
pw=single(1*ones(s1,s2));
X = single(zeros(3*s1*s2,1));
X(1:s1*s2,1)=reshape(center',s1*s2,1);
X(s1*s2+1:2*s1*s2,1)=reshape(sigma_nf',s1*s2,1);
X(2*s1*s2+1:end,1)=reshape(pw',s1*s2,1);
% Initial performance
[gX,out,w]=grad_anfis_aralik(input,center,sigma_nf,pw,w,targ,class);
ind=isnan(gX);
gX(ind)=0.0001;
[tt,tp]=max(out');out_tr=uint8(tp');
init=sum(target_tr==out_tr)/N*100;
perf=sum(sum((single(targ)-out).^2))/N;
rr(1,:)=[1 init init perf];
fprintf('initial recognation rate= %g initial perform= %g',init,perf);
fprintf('\n');
result.center{1}=center;
result.sigma_nf{1}=sigma_nf;
result.w{1}=w;
result.pw{1}=pw;
% Intial gradient and old gradient
gX_old = gX;
% Initial search direction and norm
dX = -gX;
nrmsqr_dX = dX'*dX;
norm_dX = sqrt(nrmsqr_dX);
% Initial training parameters and flag
sigma=5.0e-5;
```

```
lambda=5.0e-7;
success = 1;
lambdab = 0;
lambdak = lambda;
num_X=1;
%The tarining of NFC with SCG is starting.
sir=2;
for epoch=1:stepsize
    % If success is true, calculate second order information
    if (success == 1)
        if norm_dX~=0
            sigmak = sigma/norm_dX;
        else
            sigmak=sigma;
        end
        X_temp = X + sigmak*dX;
        center=(reshape(X_temp(1:s1*s2,1),s2,s1))';
        sigma_nf=(reshape(X_temp(s1*s2+1:2*s1*s2,1),s2,s1))';
        pw=(reshape(X_temp(2*s1*s2+1:end,1),s2,s1))';
        for i=1:s2
            ind=sigma_nf(:,i)<=0;
            sigma_nf(ind,i)=std(input(:,i));
        end

 [gX_temp,out,w]=grad_anfis_aralik(input,center,sigma_nf,pw,w,targ,class);
        ind=isnan(gX_temp);
        gX_temp(ind)=0.0001;
        sk = (gX_temp - gX)/sigmak;
        deltak = dX'*sk;
    end
    % Scale deltak
    deltak = deltak + (lambdak - lambdab)*nrmsqr_dX;
    % IF deltak <= 0 then make the Hessian matrix positive definite
    if (deltak <= 0)
        lambdab = 2*(lambdak - deltak/nrmsqr_dX);
        deltak = -deltak + lambdak*nrmsqr_dX;
        lambdak = lambdab;
    end
    % Calculate step size
    muk = -dX'*gX;
    alphak = muk/deltak;
    if muk==0

 [rr(sir,:),recog_tr,recog_te,out_tr,out_te]=performance_measurement(epoch,perf_te
        break;
    end

    % Calculate the comparison parameter  X_temp = X + alphak*dX;
    X_temp = X + alphak*dX;
    center=(reshape(X_temp(1:s1*s2,1),s2,s1))';
    sigma_nf=(reshape(X_temp(s1*s2+1:2*s1*s2,1),s2,s1))';
    pw=(reshape(X_temp(2*s1*s2+1:end,1),s2,s1))';
    for i=1:s2
        ind=sigma_nf(:,i)<=0;
```

```
        sigma_nf(ind,i)=std(input(:,i));
    end

    out=output_anfis_aralik(input,center,sigma_nf,pw,w,class);
    perf_temp=sum(sum((single(targ)-out).^2))/N;
    difk = 2*deltak*(perf - perf_temp)/(muk^2);

    % If difk >= 0 then a successful reduction in error can be made
    if (difk >= 0)
        gX_old = gX;
        X = X_temp;
        center=(reshape(X(1:s1*s2,1),s2,s1))';
        sigma_nf=(reshape(X(s1*s2+1:2*s1*s2,1),s2,s1))';
        pw=(reshape(X_temp(2*s1*s2+1:end,1),s2,s1))';
        for i=1:s2
            ind=sigma_nf(:,i)<=0;
            sigma_nf(ind,i)=std(input(:,i));
        end


 [gX,out,w]=grad_anfis_aralik(input,center,sigma_nf,pw,w,targ,class);
        ind=isnan(gX);
        gX(ind)=0.0001;
        perf=sum(sum((single(targ)-out).^2))/N;
        % Initial gradient and old gradient
        lambdab = 0;
        success = 1;
        perf = perf_temp;

        % Restart the algorithm every num_X iterations
        if rem(epoch,num_X)==0
            dX = -gX;
        else
            betak = (gX'*gX - gX'*gX_old)/muk;
            dX = -gX + betak*dX;
        end
        nrmsqr_dX = dX'*dX;
        norm_dX = sqrt(nrmsqr_dX);

        % If difk >= 0.75, then reduce the scale parameter
        if (difk >= 0.75)
            lambdak = 0.25*lambdak;
        end
    else
        lambdab = lambdak;
        success = 0;
    end

    % If difk < 0.25, then increase the scale parameter
    if (difk < 0.25)
        lambdak = lambdak + deltak*(1 - difk)/nrmsqr_dX;
    end
    performance(epoch,1)=perf;
    if rem(epoch,25)==0 | rem(epoch,stepsize)==0
```

```
 [rr(sir,:),recog_tr,recog_te,out_tr,out_te]=performance_measurement(epoch,perf_te
        result.center{sir}=center;
        result.sigma_nf{sir}=sigma_nf;
        result.w{sir}=w;
        result.pw{sir}=pw;
        if rr(sir,4)==rr(sir-1,4)
             break;
         end
        sir=sir+1;
    end
    if epoch==stepsize
        ind=find(pw<0);
        pw(ind)=0;
    end
   % gg(epoch,:)=gX;
end
[~,best]=max(rr(:,2));
outputs.center=result.center{best};
outputs.sigma_nf=result.sigma_nf{best};
outputs.w=result.w{best};
outputs.pw=result.pw{best};
labels.input=out_tr;
labels.test=out_te;
figure;plot(performance);
title('Performance evaluation');
xlabel('Epochs');
ylabel('RMSE value');
fismat=nfc_fis(double(input),double(target_tr),double(center),double(sigma_nf),cla
ruleview(fismat);
%Functions
%***************************************************************************
%Calculation of gradients and NFC outputs
function [gX,
 out,w]=grad_anfis_aralik(input,center,sigma_nf,pw,w,targ,class)
[s1,s2]=size(center);
N=size(input,1);temp=single(zeros(N,s2));mem=single(zeros(N,s1));
for i=1:s1
    temp=exp(-0.5*[(input-ones(N,1)*center(i,:)).^2]./
(ones(N,1)*sigma_nf(i,:)).^2);
    mem(:,i)=[prod([temp.^(ones(N,1)*pw(i,:))]')]';
end
ind=isinf(mem);
mem(ind)=1;
ind=isnan(mem);
mem(ind)=1;
ind=sum(mem,2)==0;
mem(ind,1)=1;
out_t=mem*w;
top=single(sum(out_t,2));
ind=top==0;
top(ind)=0.01;
out=out_t./(top*ones(1,class));
tempoc=single(zeros(s1*s2,1));tempos=single(zeros(s1*s2,1));
```

```
tempopw=single(zeros(s1*s2,1));gX=single(zeros(3*s1*s2,1));
t1=-2*(single(targ)-out);
sira=1;sir=1;temp=single(zeros(N,s2));
for k=1:class
    for j=1:s1/class
        temp=exp(-0.5*[(input-ones(N,1)*center(sira,:)).^2]./
(ones(N,1)*sigma_nf(sira,:)).^2);
        [v,vv]=find(temp==0);
        temp(v,vv)=0.000001;
        tempoc(sir:sir+s2-1)=tempoc(sir:sir
+s2-1)+([(input-ones(N,1)*center(sira,:))./
(ones(N,1)*sigma_nf(sira,:).^2)]'*[mem(:,sira).*t1(:,k).*(1-
out(:,k))./top(:,1)*w(sira,k)]).*pw(sira,:)';
        tempos(sir:sir+s2-1)=tempos(sir:sir
+s2-1)+([(input-ones(N,1)*center(sira,:)).^2./
(ones(N,1)*sigma_nf(sira,:).^3)]'*[mem(:,sira).*t1(:,k).*(1-
out(:,k))./top(:,1)*w(sira,k)]).*pw(sira,:)';
        tempopw(sir:sir+s2-1)=tempopw(sir:sir
+s2-1)+log(temp')*[t1(:,k).*mem(:,sira).*(1-out(:,k))./
top(:,1)]*w(sira,k);
        sir=sir+s2;sira=sira+1;
    end
end
gX=[tempoc;tempos;tempopw]/N;
%***************************************************************************
%Calculation of only outputs
function [out]=output_anfis_aralik(input,center,sigma_nf,pw,w,class)
[s1,s2]=size(center);
clustsize=s1/class;
N=size(input,1);
temp=single(zeros(N,s2));mem=single(zeros(N,s1));
for i=1:s1
    temp=exp(-0.5*[(input-ones(N,1)*center(i,:)).^2]./
(ones(N,1)*sigma_nf(i,:)).^2);
    mem(:,i)=[prod([temp.^(ones(N,1)*pw(i,:))]')]';
end
ind=isinf(mem);
mem(ind)=1;
ind=isnan(mem);
mem(ind)=1;
ind=sum(mem,2)==0;
mem(ind,1)=1;
out_t=mem*w;
top=single(sum(out_t,2));
ind=top==0;
top(ind)=0.01;
out=out_t./(top*ones(1,class));
%**************************************************************************
function
 [rr,recog_tr,recog_te,out_tr,out_te]=performance_measurement(epoch,perf,out,targe
N=size(target_tr,1);
m=size(target_te,1);
[tt,tp]=max(out');out_tr=tp';
indx=(out_tr==target_tr);
```

```
recog_tr=sum(indx)/N*100;
output=output_anfis_aralik(test,center,sigma_nf,pw,w,class);
[tt,tp]=max(output');out_te=tp';
indx=(out_te==target_te);
recog_te=sum(indx)/m*100;
fprintf('epoch %g   recog  %g  recog_test  %g  performans   %g
\n',epoch,recog_tr,recog_te,perf);
rr=[epoch recog_tr recog_te perf];

function
 [fismat,outputs,recog_tr,recog_te,labels,performance]=scg_nfclass_speedup(input,t
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% In this program, the neuro-fuzzy classifier parameters are adapted
 by
% Scaled conjugate gradient method. Also if the gradients smoothly
% decrease, the gradients are estimated with LSE instead of directly
% calculation. This operation is satisfied to speed up the algorithm
 for
% medium and large scale problems.
%INPUTS
%input[N,s2]: training data
%target_tr[N,1]: the target values of training data
%test[m,s2]: test data
%target_te[m,1]: the target values of test data
%stepsize: The maximum iteration number
%class: Number of classes
%clustsize: Number of cluster of each class
%OUTPUTS
%center[s1,s2]: The center values of Gaussian functions
%sigma_nf[s1,s2]: The width values of Gaussian functions
%recog_tr: The recognition rate of training data
%recog_te: The recognition rate of test data
%out_tr[N,1]: The produced class labels of training data obtained from
 NFC
%out_te[m,1]: The produced class labels of test data obtained from NFC
%performance: root mean square error of training data
%       Written by Dr. Bayram Ceti?li Suleyman Demirel University
 Computer
%       Engineeering Isparta Turkey
close all;
performance=single(zeros(stepsize,1));
warning off;
rr=single(zeros(stepsize/25,4));
m=size(test,1);
[N,s2] = size(input);
%data=scale([input;test],0.1,1);
%input=data(1:N,:);test=data(N+1:end,:);
input=single(input);test=single(test);
%clear data;
target_tr=uint8(target_tr);target_te=uint8(target_te);
center=single(zeros(clustsize*class,s2));sigma_nf=single(zeros(clustsize*class,s2)
targ=single(zeros(N,class));w=single(zeros(clustsize*class,class));
sir=1;
for i=1:class
```

```
        [v,~]=find(target_tr==i);
        temp=input(v,:);
        [idc,cc]=kmeans(temp,clustsize);
        center(sir:sir+clustsize-1,:)=cc;
        for j=1:clustsize
            ind=idc==j;
            sigma_nf(sir+j-1,:)=std(temp(ind,:));
            w(sir+j-1,i)=sum(ind)/size(v,1);
        end
        targ(v,i)=1;
        sir=sir+clustsize;
    end
    ind=sigma_nf<=0;
    sigma_nf(ind)=0.01;
    clear ind
    [s1,s2]=size(center);
    X = zeros(2*s1*s2,1);
    X(1:s1*s2,1)=reshape(center',s1*s2,1);
    X(s1*s2+1:2*s1*s2,1)=reshape(sigma_nf',s1*s2,1);
    % Initial performance
    [gX,out]=grad_anfis_aralik(input,X,w,targ,class,s1,s2);
    [~,tp]=max(out');out_tr=uint8(tp');
    init=sum(target_tr==out_tr)/N*100;
    perf=sum(sum((single(targ)-out).^2))/N;
    fprintf('initial recognation rate= %g initial perform= %g',init,perf);
    fprintf('\n');
    % Intial gradient and old gradient
    gX_old = gX;
    % Initial search direction and norm
    dX = -gX;
    nrmsqr_dX = dX'*dX;
    norm_dX = sqrt(nrmsqr_dX);
    % Initial training parameters and flag
    sigma=5.0e-5;
    lambda=5.0e-7;
    success = 1;
    lambdab = 0;
    lambdak = lambda;
    num_X=length(X);
    X_tr=X';
    G_tr=gX';
    if (success == 1)
            sigmak = sigma/norm_dX;
            X_temp = X + sigmak*dX;

     [gX_temp,out]=grad_anfis_aralik(input,X_temp,w,targ,class,s1,s2);
            X_tr=[X_tr;X_temp'];
            G_tr=[G_tr;gX_temp'];
            sk = (gX_temp - gX)/sigmak;
            deltak = dX'*sk;
    end
    % Scale deltak
    deltak = deltak + (lambdak - lambdab)*nrmsqr_dX;
    % IF deltak <= 0 then make the Hessian matrix positive definite
```

```
if (deltak <= 0)
    lambdab = 2*(lambdak - deltak/nrmsqr_dX);
    deltak = -deltak + lambdak*nrmsqr_dX;
    lambdak = lambdab;
end
% Calculate step size
muk = -dX'*gX;
alphak = muk/deltak;
% Calculate the comparison parameter  X_temp = X + alphak*dX;
X_temp = X + alphak*dX;
[out,w]=output_anfis_aralik(input,X_temp,w,class,s1,s2,targ);
perf_temp=sum(sum((targ-out).^2))/N;
difk = 2*deltak*(perf - perf_temp)/(muk^2);
% If difk >= 0 then a successful reduction in error can be made
if (difk >= 0)
    gX_old = gX;
    X = X_temp;
    [gX,out]=grad_anfis_aralik(input,X_temp,w,targ,class,s1,s2);
    X_tr=[X_tr;X'];
    G_tr=[G_tr;gX'];
    perf_temp=sum(sum((targ-out).^2))/N;
    % Initial gradient and old gradient
    lambdab = 0;
    success = 1;
    perf = perf_temp;
    % Restart the algorithm every num_X iterations
    dX = -gX;
    nrmsqr_dX = dX'*dX;
    norm_dX = sqrt(nrmsqr_dX);
    % If difk >= 0.75, then reduce the scale parameter
    if (difk >= 0.75)
        lambdak = 0.25*lambdak;
    end
else
    lambdab = lambdak;
    success = 0;
end
% If difk < 0.25, then increase the scale parameter
if (difk < 0.25)
    lambdak = lambdak + deltak*(1 - difk)/nrmsqr_dX;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The training of NFC with SCG algorithm
for epoch=1:stepsize
    % If success is true, calculate second order information
    if (success == 1)
        sigmak = sigma/norm_dX;
        X_temp = X + sigmak*dX;
        ind=X_temp(s1*s2+1:2*s1*s2)<=0;
        X_temp(ind)=0.01;
        [gX_temp]=lse_gradient(num_X,X_temp,X_tr,G_tr);
        if (find(isinf(gX_temp))>=1) | (find(isnan(gX_temp))>=1)
            fprintf('epoch %g  the gradient is calculated as directly
 %g \n',epoch);
```

```
            fprintf('\n');
            if isempty(X_temp)

[gX_temp,out]=grad_anfis_aralik(input,X_temp,w,targ,class,s1,s2);
            else
                X_tr=[X_temp;X_tr];
                G_tr=[gX_temp;G_tr];
                [gX_temp]=lse_gradient(num_X,X_temp,X_tr,G_tr);
            end
        end
        sk = (gX_temp - gX)/sigmak;
        deltak = dX'*sk;
    end
    % Scale deltak
    deltak = deltak + (lambdak - lambdab)*nrmsqr_dX;
    % IF deltak <= 0 then make the Hessian matrix positive definite
    if (deltak <= 0)
        lambdab = 2*(lambdak - deltak/nrmsqr_dX);
        deltak = -deltak + lambdak*nrmsqr_dX;
        lambdak = lambdab;
    end
    % Calculate step size
    muk = -dX'*gX;
    alphak = muk/deltak;
    if muk==0
        break;
    end

    % Calculate the comparison parameter  X_temp = X + alphak*dX;
    X_temp = X + alphak*dX;
    ind=X_temp(s1*s2+1:2*s1*s2)<=0;
    X_temp(ind)=0.01;
    [out,w]=output_anfis_aralik(input,X_temp,w,class,s1,s2,targ);
    perf_temp=sum(sum((single(targ)-out).^2))/N;
    difk = 2*deltak*(perf - perf_temp)/(muk^2);

    % If difk >= 0 then a successful reduction in error can be made
    if (difk >= 0)
        gX_old = gX;
        X = X_temp;
        ind=X(s1*s2+1:2*s1*s2)<=0;
        X(ind)=0.01;
        [gX,out]=grad_anfis_aralik(input,X,w,targ,class,s1,s2);
        if size(X_tr,1)==3
            x_temp=X_tr(1,:);
            X_tr(1,:)=[];
            g_temp=G_tr(1,:);
            G_tr(1,:)=[];
        end
        X_tr=[X_tr;X_temp'];
        G_tr=[G_tr;gX'];
        perf_temp=sum(sum((single(targ)-out).^2))/N;
        % Initial gradient and old gradient
        lambdab = 0;
```

```
            success = 1;
            perf = perf_temp;

            % Restart the algorithm every num_X iterations
            if rem(epoch,num_X)==0
                dX = -gX;
            else
                betak = (gX'*gX - gX'*gX_old)/muk;
                dX = -gX + betak*dX;
            end
            nrmsqr_dX = dX'*dX;
            norm_dX = sqrt(nrmsqr_dX);

            % If difk >= 0.75, then reduce the scale parameter
            if (difk >= 0.75)
                lambdak = 0.25*lambdak;
            end
        else
            lambdab = lambdak;
            success = 0;
        end

        % If difk < 0.25, then increase the scale parameter
        if (difk < 0.25)
            lambdak_old=lambdak;
            lambdak = lambdak + deltak*(1 - difk)/nrmsqr_dX;
            if isinf(lambdak)
                lambdak=lambdak_old*1.2;
            end
        end
        performance(epoch,1)=perf;
        if rem(epoch,25)==0 | rem(epoch,stepsize)==0

   [rr,recog_tr,recog_te,out_tr,out_te]=performance_measurement(rr,epoch,perf_temp,o
            file=['SCG_NFC_results_',date];
            save (file, 'rr', 'recog_tr', 'recog_te', 'X', 'w');
            if rr(end-1,4)==rr(end,4)
                fprintf('\n');
                disp('The gradient does not change, and the program is
   broken');
                break;
            end
        end

    end
    center=(reshape(X(1:s1*s2,1),s2,s1))';
    sigma_nf=(reshape(X(s1*s2+1:2*s1*s2,1),s2,s1))';
    outputs.center=center;
    outputs.sigma_nf=sigma_nf;
    outputs.w=w;
    labels.input=out_tr;
    labels.test=out_te;
    figure;plot(performance);
    title('Performance evaluation');
```

```
xlabel('Epochs');
ylabel('RMSE value');
fismat=nfc_fis(double(input),double(target_tr),double(center),double(sigma_nf),cla
ruleview(fismat);
%Fonctions
%*************************************************************************
%Calculation of gradients and output values together
function [gX, out]=grad_anfis_aralik(input,X,w,targ,class,s1,s2)
N=size(input,1);
mem=single(zeros(N,s1));
for i=1:s1
    mem(:,i)=exp(sum(-0.5*[(input-
ones(N,1)*X((i-1)*s2+1:i*s2,1)').^2]./
(ones(N,1)*X((i-1)*s2+s1*s2+1:i*s2+s1*s2,1)').^2,2));
end
out_t=mem*w;
top=single(sum(out_t,2));
ind=top==0;
top(ind)=0.01;
out=out_t./(top*ones(1,class));
gX=single(zeros(2*s1*s2,1));
t1=-2*(single(targ)-out);
sira=1;sir=1;
for k=1:class
    for j=1:s1/class
        temp=[mem(:,sira).*t1(:,k).*(1-out(:,k))./top(:,1)*w(sira,k)];
        gX(sir:sir+s2-1)=gX(sir:sir+s2-1)+[(input-
ones(N,1)*X((sira-1)*s2+1:sira*s2,1)')./
(ones(N,1)*X((sira-1)*s2+s1*s2+1:sira*s2+s1*s2,1)'.^2)]'*temp;
        gX(s1*s2+sir:s1*s2+sir+s2-1)=gX(s1*s2+sir:s1*s2+sir
+s2-1)+[(input-ones(N,1)*X((sira-1)*s2+1:sira*s2,1)').^2./
(ones(N,1)*X((sira-1)*s2+s1*s2+1:sira*s2+s1*s2,1)'.^3)]'*temp;
        sir=sir+s2;sira=sira+1;
    end
end
gX=gX/N;
%*************************************************************************
%The calculation of only output values
function [out,w]=output_anfis_aralik(input,X,w,class,s1,s2,targ)
if nargin<7
    targ=[];
end
N=size(input,1);
mem=single(zeros(N,s1));
for i=1:s1
    mem(:,i)=exp(sum(-0.5*[(input-
ones(N,1)*X((i-1)*s2+1:i*s2,1)').^2]./
(ones(N,1)*X((i-1)*s2+s1*s2+1:i*s2+s1*s2,1)').^2,2));
end
if isempty(targ)==0 && s1/class>1
    for i=1:class
        [v,vv]=find(targ(:,i)==1);
        [~,vv1]=max(mem(v,[(i-1)*s1/class+1:i*s1/class])');
        for j=1:s1/class
```

```
            w((i-1)*s1/class+j,i)=sum(vv1==j)/size(v,1);
        end
    end
end
out_t=mem*w;
top=single(sum(out_t,2));
ind=top==0;
top(ind)=0.01;
out=out_t./(top*ones(1,class));
%***************************************************************************
function
 [rr,recog_tr,recog_te,out_tr,out_te]=performance_measurement(rr,epoch,perf_temp,o
N=size(target_tr,1);
m=size(target_te,1);
[~,tp]=max(out');out_tr=uint8(tp');
indx=(out_tr==target_tr);
recog_tr=sum(indx)/N*100;
output=output_anfis_aralik(test,X,w,class,s1,s2);
[~,tp]=max(output');out_te=uint8(tp');
indx=(out_te==target_te);
recog_te=sum(indx)/m*100;
fprintf('epoch %g   recog_train  %g  recog_test  %g  performance   %g
\n',epoch,recog_tr,recog_te,perf_temp);
rr=[rr;epoch recog_tr recog_te perf_temp];
if recog_te>99.5
    return;
end
%***************************************************************************
function [gX_temp]=lse_gradient(num_X,X_temp,X_tr,G_tr)
for i=1:num_X
    P=[X_tr(:,i).^2 X_tr(:,i) ones(size(X_tr,1),1)]\G_tr(:,i);
    if isnan(P(1,1))==1 || isinf(P(1,1))==1
        gX_temp(i,1)=G_tr(end,i);
    else
        gX_temp(i,1)=[X_temp(i,1)^2 X_temp(i,1) 1]*P;
    end
end

function
 [fismat,outputs,recog_tr,recog_te,labels,performance]=scg_nfclass_speedup(input,t
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% In this program, the neuro-fuzzy classifier parameters are adapted
 by
% Scaled conjugate gradient method. Also if the gradients smoothly
% decrease, the gradients are estimated with LSE instead of directly
% calculation. This operation is satisfied to speed up the algorithm
 for
% medium and large scale problems.
%INPUTS
%input[N,s2]: training data
%target_tr[N,1]: the target values of training data
%test[m,s2]: test data
%target_te[m,1]: the target values of test data
%stepsize: The maximum iteration number
```

```
%class: Number of classes
%clustsize: Number of cluster of each class
%OUTPUTS
%center[s1,s2]: The center values of Gaussian functions
%sigma_nf[s1,s2]: The width values of Gaussian functions
%recog_tr: The recognition rate of training data
%recog_te: The recognition rate of test data
%out_tr[N,1]: The produced class labels of training data obtained from
 NFC
%out_te[m,1]: The produced class labels of test data obtained from NFC
%performance: root mean square error of training data
%       Written by Dr. Bayram Ceti?li Suleyman Demirel University
 Computer
%       Engineeering Isparta Turkey
close all;
performance=single(zeros(stepsize,1));
warning off;
rr=single(zeros(stepsize/25,4));
m=size(test,1);
[N,s2] = size(input);
%data=scale([input;test],0.1,1);
%input=data(1:N,:);test=data(N+1:end,:);
input=single(input);test=single(test);
%clear data;
target_tr=uint8(target_tr);target_te=uint8(target_te);
center=single(zeros(clustsize*class,s2));sigma_nf=single(zeros(clustsize*class,s2)
targ=single(zeros(N,class));w=single(zeros(clustsize*class,class));
sir=1;
for i=1:class
    [v,~]=find(target_tr==i);
    temp=input(v,:);
    [idc,cc]=kmeans(temp,clustsize);
    center(sir:sir+clustsize-1,:)=cc;
    for j=1:clustsize
        ind=idc==j;
        sigma_nf(sir+j-1,:)=std(temp(ind,:));
        w(sir+j-1,i)=sum(ind)/size(v,1);
    end
    targ(v,i)=1;
    sir=sir+clustsize;
end
ind=sigma_nf<=0;
sigma_nf(ind)=0.01;
clear ind
[s1,s2]=size(center);
X = zeros(2*s1*s2,1);
X(1:s1*s2,1)=reshape(center',s1*s2,1);
X(s1*s2+1:2*s1*s2,1)=reshape(sigma_nf',s1*s2,1);
% Initial performance
[gX,out]=grad_anfis_aralik(input,X,w,targ,class,s1,s2);
[~,tp]=max(out');out_tr=uint8(tp');
init=sum(target_tr==out_tr)/N*100;
perf=sum(sum((single(targ)-out).^2))/N;
fprintf('initial recognation rate= %g initial perform= %g',init,perf);
```

```
fprintf('\n');
% Intial gradient and old gradient
gX_old = gX;
% Initial search direction and norm
dX = -gX;
nrmsqr_dX = dX'*dX;
norm_dX = sqrt(nrmsqr_dX);
% Initial training parameters and flag
sigma=5.0e-5;
lambda=5.0e-7;
success = 1;
lambdab = 0;
lambdak = lambda;
num_X=length(X);
X_tr=X';
G_tr=gX';
if (success == 1)
        sigmak = sigma/norm_dX;
        X_temp = X + sigmak*dX;

 [gX_temp,out]=grad_anfis_aralik(input,X_temp,w,targ,class,s1,s2);
        X_tr=[X_tr;X_temp'];
        G_tr=[G_tr;gX_temp'];
        sk = (gX_temp - gX)/sigmak;
        deltak = dX'*sk;
end
% Scale deltak
deltak = deltak + (lambdak - lambdab)*nrmsqr_dX;
% IF deltak <= 0 then make the Hessian matrix positive definite
if (deltak <= 0)
    lambdab = 2*(lambdak - deltak/nrmsqr_dX);
    deltak = -deltak + lambdak*nrmsqr_dX;
    lambdak = lambdab;
end
% Calculate step size
muk = -dX'*gX;
alphak = muk/deltak;
% Calculate the comparison parameter  X_temp = X + alphak*dX;
X_temp = X + alphak*dX;
[out,w]=output_anfis_aralik(input,X_temp,w,class,s1,s2,targ);
perf_temp=sum(sum((targ-out).^2))/N;
difk = 2*deltak*(perf - perf_temp)/(muk^2);
% If difk >= 0 then a successful reduction in error can be made
if (difk >= 0)
    gX_old = gX;
    X = X_temp;
    [gX,out]=grad_anfis_aralik(input,X_temp,w,targ,class,s1,s2);
    X_tr=[X_tr;X'];
    G_tr=[G_tr;gX'];
    perf_temp=sum(sum((targ-out).^2))/N;
    % Initial gradient and old gradient
    lambdab = 0;
    success = 1;
    perf = perf_temp;
```

```
    % Restart the algorithm every num_X iterations
    dX = -gX;
    nrmsqr_dX = dX'*dX;
    norm_dX = sqrt(nrmsqr_dX);
    % If difk >= 0.75, then reduce the scale parameter
    if (difk >= 0.75)
        lambdak = 0.25*lambdak;
    end
else
    lambdab = lambdak;
    success = 0;
end
% If difk < 0.25, then increase the scale parameter
if (difk < 0.25)
    lambdak = lambdak + deltak*(1 - difk)/nrmsqr_dX;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The training of NFC with SCG algorithm
for epoch=1:stepsize
    % If success is true, calculate second order information
    if (success == 1)
        sigmak = sigma/norm_dX;
        X_temp = X + sigmak*dX;
        ind=X_temp(s1*s2+1:2*s1*s2)<=0;
        X_temp(ind)=0.01;
        [gX_temp]=lse_gradient(num_X,X_temp,X_tr,G_tr);
        if (find(isinf(gX_temp))>=1) | (find(isnan(gX_temp))>=1)
            fprintf('epoch %g  the gradient is calculated as directly
 %g \n',epoch);
            fprintf('\n');
            if isempty(X_temp)

 [gX_temp,out]=grad_anfis_aralik(input,X_temp,w,targ,class,s1,s2);
            else
                X_tr=[X_temp;X_tr];
                G_tr=[gX_temp;G_tr];
                [gX_temp]=lse_gradient(num_X,X_temp,X_tr,G_tr);
            end
        end
        sk = (gX_temp - gX)/sigmak;
        deltak = dX'*sk;
    end
    % Scale deltak
    deltak = deltak + (lambdak - lambdab)*nrmsqr_dX;
    % IF deltak <= 0 then make the Hessian matrix positive definite
    if (deltak <= 0)
        lambdab = 2*(lambdak - deltak/nrmsqr_dX);
        deltak = -deltak + lambdak*nrmsqr_dX;
        lambdak = lambdab;
    end
    % Calculate step size
    muk = -dX'*gX;
    alphak = muk/deltak;
    if muk==0
```

```
    break;
end

% Calculate the comparison parameter  X_temp = X + alphak*dX;
X_temp = X + alphak*dX;
ind=X_temp(s1*s2+1:2*s1*s2)<=0;
X_temp(ind)=0.01;
[out,w]=output_anfis_aralik(input,X_temp,w,class,s1,s2,targ);
perf_temp=sum(sum((single(targ)-out).^2))/N;
difk = 2*deltak*(perf - perf_temp)/(muk^2);

% If difk >= 0 then a successful reduction in error can be made
if (difk >= 0)
    gX_old = gX;
    X = X_temp;
    ind=X(s1*s2+1:2*s1*s2)<=0;
    X(ind)=0.01;
    [gX,out]=grad_anfis_aralik(input,X,w,targ,class,s1,s2);
    if size(X_tr,1)==3
        x_temp=X_tr(1,:);
        X_tr(1,:)=[];
        g_temp=G_tr(1,:);
        G_tr(1,:)=[];
    end
    X_tr=[X_tr;X_temp'];
    G_tr=[G_tr;gX'];
    perf_temp=sum(sum((single(targ)-out).^2))/N;
    % Initial gradient and old gradient
    lambdab = 0;
    success = 1;
    perf = perf_temp;

    % Restart the algorithm every num_X iterations
    if rem(epoch,num_X)==0
        dX = -gX;
    else
        betak = (gX'*gX - gX'*gX_old)/muk;
        dX = -gX + betak*dX;
    end
    nrmsqr_dX = dX'*dX;
    norm_dX = sqrt(nrmsqr_dX);

    % If difk >= 0.75, then reduce the scale parameter
    if (difk >= 0.75)
        lambdak = 0.25*lambdak;
    end
else
    lambdab = lambdak;
    success = 0;
end

% If difk < 0.25, then increase the scale parameter
if (difk < 0.25)
    lambdak_old=lambdak;
```

```
        lambdak = lambdak + deltak*(1 - difk)/nrmsqr_dX;
        if isinf(lambdak)
            lambdak=lambdak_old*1.2;
        end
    end
    performance(epoch,1)=perf;
    if rem(epoch,25)==0 | rem(epoch,stepsize)==0

 [rr,recog_tr,recog_te,out_tr,out_te]=performance_measurement(rr,epoch,perf_temp,o
        file=['SCG_NFC_results_',date];
        save (file, 'rr', 'recog_tr', 'recog_te', 'X', 'w');
        if rr(end-1,4)==rr(end,4)
            fprintf('\n');
            disp('The gradient does not change, and the program is
 broken');
            break;
        end
    end

end
center=(reshape(X(1:s1*s2,1),s2,s1))';
sigma_nf=(reshape(X(s1*s2+1:2*s1*s2,1),s2,s1))';
outputs.center=center;
outputs.sigma_nf=sigma_nf;
outputs.w=w;
labels.input=out_tr;
labels.test=out_te;
figure;plot(performance);
title('Performance evaluation');
xlabel('Epochs');
ylabel('RMSE value');
fismat=nfc_fis(double(input),double(target_tr),double(center),double(sigma_nf),cla
ruleview(fismat);
%Fonctions
%***************************************************************************
%Calculation of gradients and output values together
function [gX, out]=grad_anfis_aralik(input,X,w,targ,class,s1,s2)
N=size(input,1);
mem=single(zeros(N,s1));
for i=1:s1
    mem(:,i)=exp(sum(-0.5*[(input-
ones(N,1)*X((i-1)*s2+1:i*s2,1)').^2]./
(ones(N,1)*X((i-1)*s2+s1*s2+1:i*s2+s1*s2,1)').^2,2));
end
out_t=mem*w;
top=single(sum(out_t,2));
ind=top==0;
top(ind)=0.01;
out=out_t./(top*ones(1,class));
gX=single(zeros(2*s1*s2,1));
t1=-2*(single(targ)-out);
sira=1;sir=1;
for k=1:class
    for j=1:s1/class
```

```
        temp=[mem(:,sira).*t1(:,k).*(1-out(:,k))./top(:,1)*w(sira,k)];
        gX(sir:sir+s2-1)=gX(sir:sir+s2-1)+[(input-
ones(N,1)*X((sira-1)*s2+1:sira*s2,1)')./
(ones(N,1)*X((sira-1)*s2+s1*s2+1:sira*s2+s1*s2,1)'.^2)]'*temp;
        gX(s1*s2+sir:s1*s2+sir+s2-1)=gX(s1*s2+sir:s1*s2+sir
+s2-1)+[(input-ones(N,1)*X((sira-1)*s2+1:sira*s2,1)').^2./
(ones(N,1)*X((sira-1)*s2+s1*s2+1:sira*s2+s1*s2,1)'.^3)]'*temp;
        sir=sir+s2;sira=sira+1;
    end
end
gX=gX/N;
%***************************************************************************
%The calculation of only output values
function [out,w]=output_anfis_aralik(input,X,w,class,s1,s2,targ)
if nargin<7
    targ=[];
end
N=size(input,1);
mem=single(zeros(N,s1));
for i=1:s1
    mem(:,i)=exp(sum(-0.5*[(input-
ones(N,1)*X((i-1)*s2+1:i*s2,1)').^2]./
(ones(N,1)*X((i-1)*s2+s1*s2+1:i*s2+s1*s2,1)').^2,2));
end
if isempty(targ)==0 && s1/class>1
    for i=1:class
        [v,vv]=find(targ(:,i)==1);
        [~,vv1]=max(mem(v,[(i-1)*s1/class+1:i*s1/class])');
        for j=1:s1/class
            w((i-1)*s1/class+j,i)=sum(vv1==j)/size(v,1);
        end
    end
end
out_t=mem*w;
top=single(sum(out_t,2));
ind=top==0;
top(ind)=0.01;
out=out_t./(top*ones(1,class));
%**************************************************************************
function
 [rr,recog_tr,recog_te,out_tr,out_te]=performance_measurement(rr,epoch,perf_temp,o
N=size(target_tr,1);
m=size(target_te,1);
[~,tp]=max(out');out_tr=uint8(tp');
indx=(out_tr==target_tr);
recog_tr=sum(indx)/N*100;
output=output_anfis_aralik(test,X,w,class,s1,s2);
[~,tp]=max(output');out_te=uint8(tp');
indx=(out_te==target_te);
recog_te=sum(indx)/m*100;
fprintf('epoch %g   recog_train  %g  recog_test  %g  performance   %g
\n',epoch,recog_tr,recog_te,perf_temp);
rr=[rr;epoch recog_tr recog_te perf_temp];
if recog_te>99.5
```

```
    return;
end
%*************************************************************************
function [gX_temp]=lse_gradient(num_X,X_temp,X_tr,G_tr)
for i=1:num_X
    P=[X_tr(:,i).^2 X_tr(:,i) ones(size(X_tr,1),1)]\G_tr(:,i);
    if isnan(P(1,1))==1 || isinf(P(1,1))==1
        gX_temp(i,1)=G_tr(end,i);
    else
        gX_temp(i,1)=[X_temp(i,1)^2 X_temp(i,1) 1]*P;
    end
end

function
 [fismat,feature,outputs,recog_tr,recog_te,labels,performance]=nfc_feature_select(
% In this program, the performance of SCG_NFC is improved using power.
%According to the power values of features, some of the features are
%accepted to train or rejected. Note that the centers and widths of
 Gaussian functions are not trained during the NFC training.
%INPUTS
%input[N,s2]: training data
%target[N,1]: the target values of training data
%test[m,s2]: test data
%hedef[m,1]: the target values of test data
%stepsize: The maximum iteration number
%class: Number of classes
%clustsize: Number of cluster of each class
%OUTPUTS
%center[s1,s2]: The center values of Gaussian functions of i-th rule
 and j-th feature
%sigma_nf[s1,s2]: The width values of Gaussian functions of i-th rule
 and j-th feature
%recog: The recognition rate of training data
%recog_test: The recognition rate of test data
%out_t[N,1]: The actual class labels of training data obtained from
 NFC
%output_t[m,1]: The actual class labels of test data obtained from NFC
%performans: meas square error of training data
%pw[s1,s2]:The power values of Gaussian functions of i-th rule and j-
th feature
%warning off;
%bu fonksiyon sadece kuvvetleri kullanarak featurelar?n nas?l se?
ilece?ini
%g?stermektedir.
close all;
performance=zeros(stepsize,1);
sir=2;
rr=single(zeros(stepsize/25+1,4));
m=size(test,1);
[N,s2] = size(input);
center=zeros(class,s2);sigma_nf=zeros(class,s2);
targ=zeros(N,class);w=zeros(class,class);
for i=1:class
    [v,vv]=find(target_tr==i);
```

```
    temp=input(v,:);
    center(i,:)=mean(temp);
    sigma_nf(i,:)=std(temp);
    w(i,i)=1;
    targ(v,i)=1;
end
[s1,s2]=size(center);
pw=0.0001*ones(s1,s2);
X = zeros(s1*s2,1);
X(1:end,1)=reshape(pw',s1*s2,1);
% Initial performance
[gX,out]=grad_anfis_aralik(input,center,sigma_nf,pw,w,targ,class);
[tt,tp]=max(out');out_t=tp';
init=sum(target_tr==out_t)/N*100;
perf=sum(sum((targ-out).^2))/N;
perf_b=perf;
fprintf('initial recognation rate= %g initial perform= %g',init,perf);
rr(1,:)=[0 init init perf];
fprintf('\n');
% Intial gradient and old gradient
gX_old = gX;
% Initial search direction and norm
dX = -gX;
nrmsqr_dX = dX'*dX;
norm_dX = sqrt(nrmsqr_dX);
% Initial training parameters and flag
sigma=5.0e-5;
lambda=5.0e-7;
success = 1;
lambdab = 0;
lambdak = lambda;
num_X=1;
%SCG ile parametrelerin uyarlanarak e?itimin yap?lmas?
tic;
for epoch=1:stepsize
    % If success is true, calculate second order information
    if (success == 1)
        sigmak = sigma/norm_dX;
        X_temp = X + sigmak*dX;
        pw=(reshape(X_temp(1:end,1),s2,s1))';

 [gX_temp,out]=grad_anfis_aralik(input,center,sigma_nf,pw,w,targ,class);
        sk = (gX_temp - gX)/sigmak;
        deltak = dX'*sk;
    end
    % Scale deltak
    deltak = deltak + (lambdak - lambdab)*nrmsqr_dX;
    % IF deltak <= 0 then make the Hessian matrix positive definite
    if (deltak <= 0)
        lambdab = 2*(lambdak - deltak/nrmsqr_dX);
        deltak = -deltak + lambdak*nrmsqr_dX;
        lambdak = lambdab;
    end
    % Calculate step size
```

```
        muk = -dX'*gX;
        alphak = muk/deltak;
        if muk==0
     
[rr(sir,:),recog_tr,recog_te,out_tr,out_te]=performance_measurement(epoch,perf_te
            break;
        end

        % Calculate the comparison parameter  X_temp = X + alphak*dX;
        X_temp = X + alphak*dX;
        pw=(reshape(X_temp(1:end,1),s2,s1))';
        out=output_anfis_aralik(input,center,sigma_nf,pw,w,class);
        perf_temp=sum(sum((targ-out).^2))/N;
        difk = 2*deltak*(perf - perf_temp)/(muk^2);

        % If difk >= 0 then a successful reduction in error can be made
        if (difk >= 0)
            gX_old = gX;
            X = X_temp;
            pw=(reshape(X_temp(1:end,1),s2,s1))';
     
[gX,out]=grad_anfis_aralik(input,center,sigma_nf,pw,w,targ,class);
            perf=sum(sum((targ-out).^2))/N;
            % Initial gradient and old gradient
            lambdab = 0;
            success = 1;
            perf = perf_temp;
            % Restart the algorithm every num_X iterations
            if rem(epoch,num_X)==0
                dX = -gX;
            else
                betak = (gX'*gX - gX'*gX_old)/muk;
                dX = -gX + betak*dX;
            end
            nrmsqr_dX = dX'*dX;
            norm_dX = sqrt(nrmsqr_dX);

            % If difk >= 0.75, then reduce the scale parameter
            if (difk >= 0.75)
                lambdak = 0.25*lambdak;
            end
        else
            lambdab = lambdak;
            success = 0;
        end

        % If difk < 0.25, then increase the scale parameter
        if (difk < 0.25)
            lambdak = lambdak + deltak*(1 - difk)/nrmsqr_dX;
        end
        performance(epoch,1)=perf;
        if rem(epoch,25)==0 | rem(epoch,stepsize)==0
     
[rr(sir,:),recog_tr,recog_te,out_tr,out_te]=performance_measurement(epoch,perf_te
```

```
            sir=sir+1;
            if(rr(sir-2,4)==rr(sir-1,4))
                break;
            end
        end
        [v,vv]=find(pw<=0);
        [v1,vv1]=find(pw>=1);
        d1=(pw<0);d2=(pw>1);
        if (size(v,1)>1 | size(v1,1)>1)
           pw(d1)=0;
           pw(d2)=1;
        end

    end
    pw=(reshape(X(1:s1*s2,1),s2,s1))';
    ind=pw<0;
    pw(ind)=0;
    ind=pw>1;
    pw(ind)=1;
    [members]=membership_f([zeros(1,s2);ones(1,s2)],center,sigma_nf,pw,w,class);
    PW=sum(pw,1);
    [v,vv]=sort(PW,'descend');
    figure;bar(vv,v,0.1);
    title('Feature selection criteria');
    xlabel('features');
    ylabel('total linguistic hedge values');
    feature.index=vv;
    feature.power=v;
    [v,vv]=sort(PW,'descend');
    if size(vv,2)>round(s2/2)
        feature.selected=vv(1:round(s2/2));
    else
        feature.selected=vv;
    end
    ind=pw==0;
    index=sum(ind,1)>=class-1;
    [~,vv]=find(index==1);
    feature.rejected=vv;
    temp=[];
    for i=1:size(feature.selected,2)
        for j=1:size(feature.rejected,2)
            if feature.selected(i)==feature.rejected(j)
                temp=[temp;i];
            end
        end
    end
    feature.selected(temp)=[];
    outputs.center=center;
    outputs.sigma_nf=sigma_nf;
    outputs.pw=pw;
    outputs.w=w;
    outputs.mf=members;
    labels.input=out_tr;
    labels.test=out_te;
```

```
fismat=nfc_fis(double(input),double(target_tr),double(center),double(sigma_nf),cla
ruleview(fismat);
figure;plot(performance);
title('Performance evaluation');
xlabel('Epochs');
ylabel('RMSE value');
%Functions
%**************************************************************************
%Calculation of gradients and NFC outputs
function [gX,
 out]=grad_anfis_aralik(input,center,sigma_nf,pw,w,targ,class)
[s1,s2]=size(center);
N=size(input,1);
for i=1:s1
    temp=exp(-0.5*[(input-ones(N,1)*center(i,:)).^2]./
(ones(N,1)*sigma_nf(i,:)).^2);
    mem(:,i)=[prod([temp.^(ones(N,1)*pw(i,:))]')]';
end
ind=isinf(mem);
mem(ind)=1;
ind=isnan(mem);
mem(ind)=1;
ind=sum(mem,2)==0;
mem(ind,1)=1;
out_t=mem*w;
top=sum(out_t,2);
out=out_t./(top*ones(1,class));
tempopw=zeros(s1*s2,1);gX=zeros(s1*s2,1);
t1=-2*(targ-out);
sira=1;sir=1;
for k=1:class
    temp=zeros(N,s2);
    for j=1:s1/class
        temp(:,:)=exp(-0.5*[(input-ones(N,1)*center(sira,:)).^2]./
(ones(N,1)*sigma_nf(sira,:)).^2);
        [v,vv]=find(temp==0);
        temp(v,vv)=0.000001;
        tempopw(sir:sir+s2-1)=tempopw(sir:sir
+s2-1)+log(temp')*[t1(:,k).*mem(:,sira).*(1-out(:,k))./
top(:,1)]*w(sira,k);
        sir=sir+s2;sira=sira+1;
    end
end
gX=[tempopw]/N;
%**************************************************************************
%Calculation of NFC outputs
function
 [out,mem]=output_anfis_aralik(input,center,sigma_nf,pw,w,class)
[s1,s2]=size(center);
clustsize=s1/class;
N=size(input,1);
for i=1:s1
    temp=exp(-0.5*[(input-ones(N,1)*center(i,:)).^2]./
(ones(N,1)*sigma_nf(i,:)).^2);
```

```
    mem(:,i)=[prod([temp.^(ones(N,1)*pw(i,:))]')]';
end
ind=isinf(mem);
mem(ind)=1;
ind=isnan(mem);
mem(ind)=1;
ind=sum(mem,2)==0;
mem(ind,1)=1;
out_t=mem*w;
top=sum(out_t,2);
out=out_t./(top*ones(1,class));
%***************************************************************************
%Performance measurement
function
 [rr,recog_tr,recog_te,out_tr,out_te]=performance_measurement(epoch,perf_temp,out,
N=size(target_tr,1);
m=size(target_te,1);
[tt,tp]=max(out');out_tr=tp';
indx=(out_tr==target_tr);
recog_tr=sum(indx)/N*100;
output=output_anfis_aralik(test,center,sigma_nf,pw,w,class);
[tt,tp]=max(output');out_te=tp';
indx=(out_te==target_te);
recog_te=sum(indx)/m*100;
fprintf('epoch %g   recog_train  %g  recog_test  %g  performance   %g
\n',epoch,recog_tr,recog_te,perf_temp);
rr=[epoch recog_tr recog_te perf_temp];
%***************************************************************************
%calculation of boundary conditions
function [members]=membership_f(input,center,sigma_nf,pw,w,class)
[s1,s2]=size(center);
clustsize=s1/class;
N=size(input,1);
members=[];
for i=1:s1
    temp=(exp(-0.5*[(input-ones(N,1)*center(i,:)).^2]./
(ones(N,1)*sigma_nf(i,:)).^2)).^(ones(N,1)*pw(i,:));
    members=[members;temp];
end

Error using dbstatus
Error: File: C:\Users\HP\Documents\MATLAB\Assignment_3_Functions.m
 Line: 485 Column: 22
Function with duplicate name "grad_anfis_aralik" cannot be defined.
```

Published with MATLAB® R2019a