NNFL (BITS F312) Assignment 2

2016B5A30572H
K Pranath Reddy

Question 1

The dataset (data5.mat) contains 72 features and the last column is the output (class labels). Design a multilayer perceptron based neural network (two hidden layers) for the classification. You can use both holdout and 5-fold cross-validation approaches for evaluating the performance of the classifier.

Solution :

Code :

```
'''
*** MLP ***
Binary Classification
with two hidden layers

Author :
Pranath Reddy
2016B5A30572H
'''

import pandas as pd
import math
import numpy as np
from mat4py import loadmat
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from sklearn.model_selection import train_test_split

def set(y):
    for i in range(len(y)):
        if(y[i]>0.5):
```

```
            y[i] = 1.0
        if(y[i]<=0.5):
            y[i] = 0.0
    return y

data = loadmat('./data5.mat')
data = pd.DataFrame(data)
data = np.asarray(data)

data_temp = []
for i in range(len(data)):
    data_temp.append(data[i][0])

data_temp = np.asarray(data_temp)
data = data_temp
y = data[:,-1]
x = data[:,:-1]
x = (x - np.mean(x,axis=0))/np.std(x,axis=0)
x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.3)
m = x_tr.shape[0]
n = x_tr.shape[1]

for p in range(50,250,25):
        #p - no of neurons in hidden layer

        batch_size = 128
        epochs = 2000

        model = Sequential() # instantiate the model
        model.add(Dense(p, activation='relu', input_shape=(n,))) # first
hidden layer
        model.add(Dense(p, activation='relu')) # second hidden layer
        model.add(Dense(1, activation='sigmoid')) # output layer

        sgd = optimizers.SGD(lr=0.01)
        #model.compile(loss='mean_squared_error', optimizer=sgd,
metrics=['accuracy'])
```

```
        model.compile(loss='binary_crossentropy', optimizer=sgd,
metrics=['accuracy'])
        model.fit(x_tr, y_tr, batch_size=batch_size, epochs=epochs)
        yp = model.predict(x_ts, batch_size=batch_size)

        yp_temp = []
        for i in range(len(yp)):
            yp_temp.append(yp[i][0])
        yp = yp_temp
        yp = set(yp)

        y_actual = pd.Series(y_ts, name='Actual')
        y_pred = pd.Series(yp, name='Predicted')
        confmat = pd.crosstab(y_actual, y_pred)

        print('Result for hidden neurons : ' + str(p) + ' :')
        print(confmat)
        confmat = np.asarray(confmat)
        tp = confmat[1][1]
        tn = confmat[0][0]
        fp = confmat[0][1]
        fn = confmat[1][0]

        Acc = float(tp+tn)/float(tp+tn+fp+fn)
        SE = float(tp)/float(tp+fn)
        SP = float(tn)/float(tn+fp)

        print('Accuracy : ' + str(Acc))
        print('sensitivity : ' + str(SE))
        print('specificity : ' + str(SP))
```

Q1 Results : (Ran on google colab)

```
Result for hidden neurons : 50 :
Predicted  0.0  1.0
Actual
0.0        303   20
1.0         12  310
Accuracy : 0.9503875968992248
sensitivity : 0.9627329192546584
specificity : 0.9380804953560371
Result for hidden neurons : 75 :
Predicted  0.0  1.0
Actual
0.0        307   16
1.0         12  310
Accuracy : 0.9565891472868217
sensitivity : 0.9627329192546584
specificity : 0.9504643962848297
Result for hidden neurons : 100 :
Predicted  0.0  1.0
Actual
0.0        309   14
1.0          6  316
Accuracy : 0.9689922480620154
sensitivity : 0.9813664596273292
specificity : 0.9566563467492261
Result for hidden neurons : 125 :
Predicted  0.0  1.0
Actual
0.0        303   20
1.0         11  311
Accuracy : 0.951937984496124
sensitivity : 0.9658385093167702
specificity : 0.9380804953560371
Result for hidden neurons : 150 :
Predicted  0.0  1.0
Actual
0.0        309   14
1.0         10  312
Accuracy : 0.9627906976744186
sensitivity : 0.968944099378882
specificity : 0.9566563467492261
Result for hidden neurons : 175 :
Predicted  0.0  1.0
Actual
0.0        309   14
1.0         11  311
Accuracy : 0.9612403100775194
sensitivity : 0.9658385093167702
specificity : 0.9566563467492261
Result for hidden neurons : 200 :
Predicted  0.0  1.0
Actual
0.0        309   14
1.0          7  315
Accuracy : 0.9674418604651163
sensitivity : 0.9782608695652174
specificity : 0.9566563467492261
```

Best Result:

```
Result for hidden neurons : 100 :
Predicted  0.0  1.0
Actual
0.0        309   14
1.0          6  316
Accuracy : 0.9689922480620154
sensitivity : 0.9813664596273292
specificity : 0.9566563467492261
```

Question 2

Implement the radial basis function neural network (RBFNN) for the classification problem. You can use Gaussian, multiquadric and linear kernel functions for the implementation. You can use both holdout and 5-fold cross-validation approaches for evaluating the performance of the classifier. Please use the dataset (data5.mat).

Solution :

Code :

```
'''
*** RBFNN ***
Binary Classification

Author :
Pranath Reddy
2016B5A30572H
'''

import pandas as pd
import math
import numpy as np
from mat4py import loadmat
from sklearn.model_selection import train_test_split

def set(y):
    for i in range(len(y)):
        if(y[i]>0.5):
            y[i] = 1.0
        if(y[i]<=0.5):
            y[i] = 0.0
    return y

data = loadmat('./data5.mat')
data = pd.DataFrame(data)
data = np.asarray(data)
```

```python
data_temp = []
for i in range(len(data)):
    data_temp.append(data[i][0])

data_temp = np.asarray(data_temp)
data = data_temp
y = data[:,-1]
x = data[:,:-1]
x = (x - np.mean(x,axis=0))/np.std(x,axis=0)
x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.3)

# p - no of neurons in hidden layer
p = 300

cval = np.zeros((x_tr.shape[0],1))
indexes = np.random.randint(0,x_tr.shape[0],p)
centers = x_tr[indexes]
l2_norm = np.zeros((x_tr.shape[0],1))
beta = np.zeros((p,1))

def compute_distance(feature_centers,datapoint):
    return np.sum(np.power(datapoint-feature_centers,2),axis=1)

def kmeans(data,num_cluster_centers,epochs=1000):
    cluster = np.zeros((data.shape[0],1))
    center_indexes =
np.random.random_integers(0,data.shape[0],num_cluster_centers)
    feature_centers = data[center_indexes]

    for epoch in range(epochs):
        distances = np.zeros((num_cluster_centers,1))
        for datapoint in range(data.shape[0]):
            distances =
compute_distance(feature_centers,data[datapoint,:])
            cluster_index = np.argmin(distances)
            cluster[datapoint,0] = cluster_index
```

```
        for i in range(num_cluster_centers):
            cluster_points_indices = np.argwhere(cluster == i)
            cluster_points = data[cluster_points_indices[:,0]]
            if cluster_points.shape[0] != 0:
                feature_centers[i] = np.mean(cluster_points,axis=0)

    return feature_centers

centers = kmeans(x_tr,p)

H= np.zeros((x_tr.shape[0],p))
for i in range(x_tr.shape[0]):
    for j in range(p):
        H[i][j] = np.linalg.norm(x_tr[i]-centers[j])

H_test = np.empty((x_ts.shape[0],p), dtype= float)
for i in range(x_ts.shape[0]):
    for j in range(p):
        H_test[i][j] = np.linalg.norm(x_ts[i]-centers[j])

H = np.matrix(H)
print(H.shape)
print(y_tr.shape)
W= np.dot(H.I,y_tr)
print(W.shape)
y_pred = np.dot(H_test,W.T)

y_pred_temp = []
y_pred = np.asarray(y_pred)
for i in range(y_pred.shape[0]):
    y_pred_temp.append(y_pred[i][0])
yp = set(y_pred_temp)

y_actual = pd.Series(y_ts, name='Actual')
y_pred = pd.Series(yp, name='Predicted')
confmat = pd.crosstab(y_actual, y_pred)
```

```
print(confmat)
confmat = np.asarray(confmat)
tp = confmat[1][1]
tn = confmat[0][0]
fp = confmat[0][1]
fn = confmat[1][0]

Acc = float(tp+tn)/float(tp+tn+fp+fn)
SE = float(tp)/float(tp+fn)
SP = float(tn)/float(tn+fp)

print('Accuracy : ' + str(Acc))
print('sensitivity : ' + str(SE))
print('specificity : ' + str(SP))
```

Q2 Result : (Ran on local machine)

```
Predicted   0.0   1.0
Actual
0.0         297    15
1.0          17   316
Accuracy : 0.950387596899
sensitivity : 0.948948948949
specificity : 0.951923076923
```

Question 3

Implement the stacked autoencoder based deep neural network for the classification problem. The deep neural network must contain 3 hidden layers from three autoencoders. You can use data5.mat file and either holdout or 5-fold cross-validation technique for selecting, training and test instances for the classifier. For autoencoder implementation, please use back propagation algorithm which has been already taught in the class.

Solution :

Code :

```
'''
*** Stacked autoencoder ***
Binary Classification

Author :
Pranath Reddy
2016B5A30572H
'''

import pandas as pd
import math
import numpy as np
from mat4py import loadmat
import keras
from keras.models import Sequential
from keras.models import Model
from keras.layers import Dense
from keras.layers import Input
from keras import optimizers
from sklearn.model_selection import train_test_split

def set(y):
    for i in range(len(y)):
        if(y[i]>0.5):
            y[i] = 1.0
        if(y[i]<=0.5):
            y[i] = 0.0
```

```
    return y

data = loadmat('./data5.mat')
data = pd.DataFrame(data)
data = np.asarray(data)

data_temp = []
for i in range(len(data)):
    data_temp.append(data[i][0])

data_temp = np.asarray(data_temp)
data = data_temp
y = data[:,-1]
x = data[:,:-1]
x = (x - np.mean(x,axis=0))/np.std(x,axis=0)
x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.3)
m = x_tr.shape[0]
n = x_tr.shape[1]

batch_size = 128
epochs = 5000

# Pretraining Models
# Layer 1
input1 = Input(shape = (n, ))
encoder1 = Dense(120, activation = 'sigmoid')(input1)
decoder1 = Dense(n, activation = 'sigmoid')(encoder1)

autoencoder1 = Model(input = input1, output = decoder1)
encoder_layer1 = Model(input = input1, output = encoder1)

# Layer 2
input2 = Input(shape = (120,))
encoder2 = Dense(100, activation = 'sigmoid')(input2)
decoder2 = Dense(120, activation = 'sigmoid')(encoder2)

autoencoder2 = Model(input = input2, output = decoder2)
```

```
encoder_layer2 = Model(input = input2, output = encoder2)

# Layer 3
input3 = Input(shape = (100,))
encoder3 = Dense(80, activation = 'sigmoid')(input3)
decoder3 = Dense(100, activation = 'sigmoid')(encoder3)

autoencoder3 = Model(input = input3, output = decoder3)

sgd = optimizers.SGD(lr=0.1)

autoencoder1.compile(loss='binary_crossentropy', optimizer = sgd)
autoencoder2.compile(loss='binary_crossentropy', optimizer = sgd)
autoencoder3.compile(loss='binary_crossentropy', optimizer = sgd)

encoder_layer1.compile(loss='binary_crossentropy', optimizer = sgd)
encoder_layer2.compile(loss='binary_crossentropy', optimizer = sgd)

autoencoder1.fit(x_tr, x_tr, epochs= 1000, batch_size = 512, verbose=0)
layer2_input = encoder_layer1.predict(x_tr)
autoencoder2.fit(layer2_input, layer2_input, epochs= 1000, batch_size =
512, verbose=0)
layer3_input = encoder_layer2.predict(layer2_input)
autoencoder3.fit(layer3_input, layer3_input, epochs= 1000, batch_size =
512, verbose=0)

# Main Model
# stacked Autoencoder

encoder1_sa = Dense(120, activation = 'sigmoid')(input1)
encoder2_sa = Dense(100, activation = 'sigmoid')(encoder1_sa)
encoder3_sa = Dense(80, activation = 'sigmoid')(encoder2_sa)
final_output = Dense(1, activation = 'sigmoid')(encoder3_sa)

stack_autoencoder = Model(input = input1, output = final_output)

stack_autoencoder.compile(loss='binary_crossentropy', optimizer = sgd)
```

```
stack_autoencoder.layers[1].set_weights(autoencoder1.layers[1].get_weights
())
stack_autoencoder.layers[2].set_weights(autoencoder2.layers[1].get_weights
())
stack_autoencoder.layers[3].set_weights(autoencoder3.layers[1].get_weights
())

stack_autoencoder.fit(x_tr, y_tr, batch_size=batch_size, epochs=epochs,
verbose=1)

yp = stack_autoencoder.predict(x_ts, batch_size=batch_size)

yp_temp = []
for i in range(len(yp)):
    yp_temp.append(yp[i][0])
yp = yp_temp
yp = set(yp)

y_actual = pd.Series(y_ts, name='Actual')
y_pred = pd.Series(yp, name='Predicted')
confmat = pd.crosstab(y_actual, y_pred)

print(confmat)
confmat = np.asarray(confmat)
tp = confmat[1][1]
tn = confmat[0][0]
fp = confmat[0][1]
fn = confmat[1][0]

Acc = float(tp+tn)/float(tp+tn+fp+fn)
SE = float(tp)/float(tp+fn)
SP = float(tn)/float(tn+fp)

print('Accuracy : ' + str(Acc))
print('sensitivity : ' + str(SE))
print('specificity : ' + str(SP))
```

Q3 Result : (Ran on google colab)

```
Predicted   0.0   1.0
Actual
0.0         304    10
1.0           8   323
Accuracy : 0.9720930232558139
sensitivity : 0.9758308157099698
specificity : 0.9681528662420382
```

Question 4

Implement extreme learning machine (ELM) classifier for the classification. You can use Gaussian and tanh activation functions. Please select the training and test instances using 5- fold cross-validation technique. Please use the dataset as data5.mat.

Solution :

Code :

```
'''
*** ELM ***
Binary Classification

Author :
Pranath Reddy
2016B5A30572H
'''

import pandas as pd
import math
import numpy as np
from mat4py import loadmat
from sklearn.model_selection import train_test_split
```

```python
def set(y):
    for i in range(len(y)):
        if(y[i]>0.5):
            y[i] = 1.0
        if(y[i]<=0.5):
            y[i] = 0.0
    return y

data = loadmat('./data5.mat')
data = pd.DataFrame(data)
data = np.asarray(data)

data_temp = []
for i in range(len(data)):
    data_temp.append(data[i][0])

data_temp = np.asarray(data_temp)
data = data_temp
y = data[:,-1]
x = data[:,:-1]
x = (x - np.mean(x,axis=0))/np.std(x,axis=0)
x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.3)

for p in range (250,1001,50):

        randommat = np.random.randn(x_tr.shape[1]+1,p)
        H = np.append(np.ones((x_tr.shape[0],1)), x_tr, axis=1)
        H = np.dot(H,randommat)
        H = np.tanh(H)
        H = np.matrix(H)

        y_tr = np.matrix(y_tr)
        W = np.dot(H.I,y_tr.T)

        H_ts = np.append(np.ones((x_ts.shape[0],1)), x_ts, axis=1)
        H_ts = np.dot(H_ts,randommat)
        H_ts = np.tanh(H_ts)
```

```python
H_ts = np.matrix(H_ts)
y_pred = np.dot(H_ts,W)

y_pred_temp = []
y_pred = np.asarray(y_pred)
for i in range(y_pred.shape[0]):
    y_pred_temp.append(y_pred[i][0])
yp = set(y_pred_temp)

y_actual = pd.Series(y_ts, name='Actual')
y_pred = pd.Series(yp, name='Predicted')
confmat = pd.crosstab(y_actual, y_pred)

print('Result for hidden neurons : ' + str(p) + ' :')
print(confmat)
confmat = np.asarray(confmat)
tp = confmat[1][1]
tn = confmat[0][0]
fp = confmat[0][1]
fn = confmat[1][0]

Acc = float(tp+tn)/float(tp+tn+fp+fn)
SE = float(tp)/float(tp+fn)
SP = float(tn)/float(tn+fp)

print('Accuracy : ' + str(Acc))
print('sensitivity : ' + str(SE))
print('specificity : ' + str(SP))
```

Q4 Results : (Ran on local machine)

Assignment-2 — -bash — 108×48

```
Result for hidden neurons : 400 :
Predicted  0.0  1.0
Actual
0.0        278   41
1.0         56  270
Accuracy : 0.849612403101
sensitivity : 0.828220858896
specificity : 0.871473354232
Result for hidden neurons : 450 :
Predicted  0.0  1.0
Actual
0.0        284   35
1.0         44  282
Accuracy : 0.877519379845
sensitivity : 0.865030674847
specificity : 0.890282131661
Result for hidden neurons : 500 :
Predicted  0.0  1.0
Actual
0.0        271   48
1.0         59  267
Accuracy : 0.834108527132
sensitivity : 0.819018404908
specificity : 0.849529780564
Result for hidden neurons : 550 :
Predicted  0.0  1.0
Actual
0.0        260   59
1.0         55  271
Accuracy : 0.823255813953
sensitivity : 0.831288343558
specificity : 0.815047021944
Result for hidden neurons : 600 :
Predicted  0.0  1.0
Actual
0.0        267   52
1.0         43  283
Accuracy : 0.852713178295
sensitivity : 0.868098159509
specificity : 0.836990595611
Result for hidden neurons : 650 :
Predicted  0.0  1.0
Actual
0.0        270   49
1.0         38  288
Accuracy : 0.86511627907
sensitivity : 0.883435582822
specificity : 0.846394984326
```

Best Result:

```
Result for hidden neurons : 450 :
Predicted  0.0  1.0
Actual
0.0        284   35
1.0         44  282
Accuracy : 0.877519379845
sensitivity : 0.865030674847
specificity : 0.890282131661
```

Question 5

Implement a deep neural network, which contains two hidden layers (the hidden layers are obtained from the autoencoders). The last layer will be the ELM layer which means the second hidden layer feature vector is used as input to the ELM classifier. The network can be called as deep layer stacked autoencoder based extreme learning machine. You can use holdout approach for evaluating the performance of the classifier. Please use the dataset (data5.mat).

Solution :

Code :

```
'''
*** Deep layer stacked autoencoder based ELM ***
Binary Classification

Author :
Pranath Reddy
2016B5A30572H
'''

import pandas as pd
import math
import numpy as np
from mat4py import loadmat
import keras
from keras.models import Sequential
from keras.models import Model
from keras.layers import Dense
from keras.layers import Input
from keras import optimizers
from sklearn.model_selection import train_test_split

def set(y):
    for i in range(len(y)):
        if(y[i]>0.5):
            y[i] = 1.0
        if(y[i]<=0.5):
            y[i] = 0.0
```

```python
    return y

data = loadmat('./data5.mat')
data = pd.DataFrame(data)
data = np.asarray(data)

data_temp = []
for i in range(len(data)):
    data_temp.append(data[i][0])

data_temp = np.asarray(data_temp)
data = data_temp
y = data[:,-1]
x = data[:,:-1]
x = (x - np.mean(x,axis=0))/np.std(x,axis=0)
x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.3)
m = x_tr.shape[0]
n = x_tr.shape[1]

# Pretraining Models
# Layer 1
input1 = Input(shape = (n, ))
encoder1 = Dense(120, activation = 'sigmoid')(input1)
decoder1 = Dense(n, activation = 'sigmoid')(encoder1)

autoencoder1 = Model(input = input1, output = decoder1)
encoder_layer1 = Model(input = input1, output = encoder1)

# Layer 2
input2 = Input(shape = (120,))
encoder2 = Dense(100, activation = 'sigmoid')(input2)
decoder2 = Dense(120, activation = 'sigmoid')(encoder2)

autoencoder2 = Model(input = input2, output = decoder2)

sgd = optimizers.SGD(lr=0.1)
```

```
autoencoder1.compile(loss='binary_crossentropy', optimizer = sgd)
autoencoder2.compile(loss='binary_crossentropy', optimizer = sgd)

encoder_layer1.compile(loss='binary_crossentropy', optimizer = sgd)

autoencoder1.fit(x_tr, x_tr, epochs= 2000, batch_size = 128, verbose=0)
layer2_input = encoder_layer1.predict(x_tr)
autoencoder2.fit(layer2_input, layer2_input, epochs= 2000, batch_size =
128, verbose=0)

encoder1_sa = Dense(120, activation = 'sigmoid')(input1)
encoder2_sa = Dense(100, activation = 'sigmoid')(encoder1_sa)

stack_autoencoder = Model(input = input1, output = encoder2_sa)

stack_autoencoder.compile(loss='binary_crossentropy', optimizer = sgd)

stack_autoencoder.layers[1].set_weights(autoencoder1.layers[1].get_weights
())
stack_autoencoder.layers[2].set_weights(autoencoder2.layers[1].get_weights
())

elm_train_input = stack_autoencoder.predict(x_tr, batch_size=128)
elm_test_input = stack_autoencoder.predict(x_ts, batch_size=128)


for p in range (250,1001,50):
        #p - no of neurons in hidden layer

        randommat = np.random.randn(elm_train_input.shape[1]+1,p)
        H = np.append(np.ones((elm_train_input.shape[0],1)),
elm_train_input, axis=1)
        H = np.dot(H,randommat)
        H = np.tanh(H)
        H = np.matrix(H)

        y_tr = np.matrix(y_tr)
```

```
        W = np.dot(H.I,y_tr.T)

        H_ts = np.append(np.ones((elm_test_input.shape[0],1)),
elm_test_input, axis=1)
        H_ts = np.dot(H_ts,randommat)
        H_ts = np.tanh(H_ts)
        H_ts = np.matrix(H_ts)
        y_pred = np.dot(H_ts,W)

        y_pred_temp = []
        y_pred = np.asarray(y_pred)
        for i in range(y_pred.shape[0]):
            y_pred_temp.append(y_pred[i][0])
        yp = set(y_pred_temp)

        y_actual = pd.Series(y_ts, name='Actual')
        y_pred = pd.Series(yp, name='Predicted')
        confmat = pd.crosstab(y_actual, y_pred)

        print('Result for hidden neurons : ' + str(p) + ' :')
        print(confmat)
        confmat = np.asarray(confmat)
        tp = confmat[1][1]
        tn = confmat[0][0]
        fp = confmat[0][1]
        fn = confmat[1][0]

        Acc = float(tp+tn)/float(tp+tn+fp+fn)
        SE = float(tp)/float(tp+fn)
        SP = float(tn)/float(tn+fp)

        print('Accuracy : ' + str(Acc))
        print('sensitivity : ' + str(SE))
        print('specificity : ' + str(SP))
```

Q5 Results : (Ran on google colab)

```
Result for hidden neurons : 350 :
Predicted  0.0  1.0
Actual
0.0        288   53
1.0         38  266
Accuracy : 0.8589147286821706
sensitivity : 0.875
specificity : 0.844574780058651
Result for hidden neurons : 400 :
Predicted  0.0  1.0
Actual
0.0        299   42
1.0         31  273
Accuracy : 0.8868217054263566
sensitivity : 0.8980263157894737
specificity : 0.8768328445747801
Result for hidden neurons : 450 :
Predicted  0.0  1.0
Actual
0.0        289   52
1.0         35  269
Accuracy : 0.8651162790697674
sensitivity : 0.8848684210526315
specificity : 0.8475073313782991
Result for hidden neurons : 500 :
Predicted  0.0  1.0
Actual
0.0        300   41
1.0         33  271
Accuracy : 0.8852713178294573
sensitivity : 0.8914473684210527
specificity : 0.8797653958944281
Result for hidden neurons : 550 :
Predicted  0.0  1.0
Actual
0.0        295   46
1.0         41  263
Accuracy : 0.8651162790697674
sensitivity : 0.8651315789473685
specificity : 0.8651026392961877
Result for hidden neurons : 600 :
Predicted  0.0  1.0
Actual
0.0        295   46
1.0         39  265
Accuracy : 0.8682170542635659
sensitivity : 0.8717105263157895
specificity : 0.8651026392961877
Result for hidden neurons : 650 :
Predicted  0.0  1.0
Actual
0.0        288   53
1.0         38  266
Accuracy : 0.8589147286821706
sensitivity : 0.875
specificity : 0.844574780058651
```

Best Result:

```
Result for hidden neurons : 400 :
Predicted  0.0  1.0
Actual
0.0        299   42
1.0         31  273
Accuracy : 0.8868217054263566
sensitivity : 0.8980263157894737
specificity : 0.8768328445747801
```