## CA Assignment:

## #### Neural Network Ensembles

##### Given :  Two benchmark classification/regression problems:

* Diabetes.csv

The diabetes data set contains the diagnostic data to investigate whether the patient shows signs of diabetes according to World Health Organization criteria such as the 2-hour post-load plasma glucose.

* Winequality-white.csv

 The winequality-white data is related to the white variants of the Portuguese "Vinho Verde" wine. The goal is to model wine quality based on physicochemical tests.

##### Expected :

1. Train a group of different types of NNs using different NN tools to solve the two problems given. (Use 2 different tools to train 2-3 different types of NNs)
2. Work on the two data sets  You may partition each data set into two subsets: eg 75% as training data and 25% as test data
3. Train the NNs to achieve the highest possible classification accuracy or lowest possible MSE.
4. NN ensemble - combine the outputs of individual NNs for final output (you may define certain calculation, such as rule(s) for the integration) Compare the NN performance between the NN ensemble and the individual NNs

#all imports  
import matplotlib.pyplot as plt  
import numpy as np  
import tensorflow as tf  
from sklearn.neural\_network import MLPClassifier  
from sklearn.utils import shuffle  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
from sklearn import preprocessing  
from sklearn.metrics import accuracy\_score, f1\_score  
import pandas as pd  
from neupy import algorithms, estimators, environment,layers  
from sklearn.metrics import confusion\_matrix  
%matplotlib inline

## 1. [ Diabities Problem ]

For Each Attribute: (all numeric-valued) 1. Number of times pregnant 2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test 3. Diastolic blood pressure (mm Hg) 4. Triceps skin fold thickness (mm) 5. 2-Hour serum insulin (mu U/ml) 6. Body mass index (weight in kg/(height in m)^2) 7. Diabetes pedigree function 8. Age (years) 9. Class variable (0 or 1)

Class Distribution: (class value 1 is interpreted as "tested positive for diabetes")

Class Value Number of instances 0 500 1 268

df\_diab = pd.read\_csv('Diabetes.csv')  
df\_diab.columns = ['nop',  
 'pgc',  
 'bp',  
 'sft',  
 'sein',  
 'bmi',  
 'pedig',  
 'age',  
 'cls'  
 ]  
df\_diab.head(5)

<tr style="text-align: right;">  
 <th></th>  
 <th>nop</th>  
 <th>pgc</th>  
 <th>bp</th>  
 <th>sft</th>  
 <th>sein</th>  
 <th>bmi</th>  
 <th>pedig</th>  
 <th>age</th>  
 <th>cls</th>  
</tr>

<tr>  
 <th>0</th>  
 <td>1</td>  
 <td>85</td>  
 <td>66</td>  
 <td>29</td>  
 <td>0</td>  
 <td>26.6</td>  
 <td>0.351</td>  
 <td>31</td>  
 <td>0</td>  
</tr>  
<tr>  
 <th>1</th>  
 <td>8</td>  
 <td>183</td>  
 <td>64</td>  
 <td>0</td>  
 <td>0</td>  
 <td>23.3</td>  
 <td>0.672</td>  
 <td>32</td>  
 <td>1</td>  
</tr>  
<tr>  
 <th>2</th>  
 <td>1</td>  
 <td>89</td>  
 <td>66</td>  
 <td>23</td>  
 <td>94</td>  
 <td>28.1</td>  
 <td>0.167</td>  
 <td>21</td>  
 <td>0</td>  
</tr>  
<tr>  
 <th>3</th>  
 <td>0</td>  
 <td>137</td>  
 <td>40</td>  
 <td>35</td>  
 <td>168</td>  
 <td>43.1</td>  
 <td>2.288</td>  
 <td>33</td>  
 <td>1</td>  
</tr>  
<tr>  
 <th>4</th>  
 <td>5</td>  
 <td>116</td>  
 <td>74</td>  
 <td>0</td>  
 <td>0</td>  
 <td>25.6</td>  
 <td>0.201</td>  
 <td>30</td>  
 <td>0</td>  
</tr>

#total  
print("total size of records")  
np.size(df\_diab)

total size of records  
  
  
  
  
  
6903

X1 = df\_diab.iloc[:,:8]  
y1 = df\_diab['cls']  
X1,y1 = shuffle(X1,y1)  
#X1\_train, X1\_test, y1\_train, y1\_test = train\_test\_split(X1, y1, test\_size=0.33, random\_state=42)  
X1\_train, X1\_test, y1\_train, y1\_test = train\_test\_split(preprocessing.minmax\_scale(X1),preprocessing.minmax\_scale(y1),train\_size=0.70)

/usr/local/anaconda3/envs/carnd-term1/lib/python3.5/site-packages/sklearn/utils/validation.py:429: DataConversionWarning: Data with input dtype int64 was converted to float64.  
 warnings.warn(msg, \_DataConversionWarning)

print("train",X1\_train.shape)  
print("test",X1\_test.shape)  
print("train\_y",y1\_train.shape)

train (536, 8)  
test (231, 8)  
train\_y (536,)

print(y1\_train[0])  
X1\_train[0]

0.0  
  
  
  
  
  
array([ 0.05882353, 0.53768844, 0.55737705, 0.19191919, 0. ,  
 0.39493294, 0.03714774, 0.05 ])

X1\_test[0]

array([ 0.35294118, 0.52763819, 0.57377049, 0.32323232, 0.08037825,  
 0.45901639, 0.01878736, 0.26666667])

theta = 2 # readius  
epsilon = 1e-4  
(size,nf) = X1\_train.shape  
# activation function  
def rce\_activation(X,weights):  
 z = np.dot(X,weights) #distance matrix for d(X,Wi)  
 print("z is",X.shape, weights.shape, z.shape)  
 f = 1 if z <= theta else 0 #threshold  
 return(f)  
  
  
# model  
def rce\_network\_train(X):  
 weights = np.array(X)  
 biases = np.zeros((size,nf))  
 input\_layer = np.matmul(weights.transpose(),X)  
   
 #y = rce\_activation(X,input\_layer)  
 #print('y is ' + y)  
   
   
 print(input\_layer.shape)  
   
 #lamdba = np.zeros((1,nf))  
   
 #for i in range(1,)  
   
 return(input\_layer)

## 1a. GRNN Network - [ Diabities Problem ]

grnn\_nw = algorithms.GRNN(std=0.1, verbose=True)  
print(grnn\_nw)

Main information  
  
[ALGORITHM] GRNN  
  
[OPTION] verbose = True  
[OPTION] epoch\_end\_signal = None  
[OPTION] show\_epoch = 1  
[OPTION] shuffle\_data = False  
[OPTION] step = 0.1  
[OPTION] train\_end\_signal = None  
[OPTION] std = 0.1  
  
GRNN(std=0.1, show\_epoch=None, train\_end\_signal=None, shuffle\_data=None, verbose=True, epoch\_end\_signal=None, step=None)

grnn\_nw.train(X1\_train, y1\_train)

y1\_predicted = grnn\_nw.predict(X1\_test).round()  
  
y1\_predicted[0]

array([ 0.])

#accuracy  
estimators.rmse(y1\_predicted, y1\_test)

0.5425608669746597

#confusion matrix  
confusion\_matrix(y1\_test,y1\_predicted)

array([[128, 26],  
 [ 42, 35]])

from sklearn.metrics import accuracy\_score  
grnn\_acc\_score = accuracy\_score(y1\_test, y1\_predicted)  
print("Grnn accuracy score ", grnn\_acc\_score)

Grnn accuracy score 0.705627705628

## 1b. PNN Network - [ Diabities Problem ]

pnn\_nw = algorithms.PNN(std=10, verbose=False)  
print(pnn\_nw)

PNN(std=10, show\_epoch=1, train\_end\_signal=None, shuffle\_data=False, verbose=False, epoch\_end\_signal=None, batch\_size=128, step=0.1)

pnn\_nw.train(X1\_train, y1\_train)

y1\_pnn\_predicted = pnn\_nw.predict(X1\_test).round()  
y1\_pnn\_predicted[0]

0.0

#accuracy  
estimators.rmse(y1\_pnn\_predicted, y1\_test)

0.5385566730097122

#confusion matrix  
confusion\_matrix(y1\_test,y1\_pnn\_predicted)

array([[130, 24],  
 [ 43, 34]])

pnn\_acc\_score = accuracy\_score(y1\_test, y1\_pnn\_predicted)  
print("Pnn accuracy score ", pnn\_acc\_score)

Pnn accuracy score 0.709956709957

## 1c. RBF - [ Diabities Problem ]

rbf\_nw = algorithms.RBFKMeans(n\_clusters=2, verbose=False)

rbf\_nw.train(X1\_train, epsilon=1e-5)

y1\_rbf\_predicted = rbf\_nw.predict(X1\_test)

confusion\_matrix(y1\_test,y1\_rbf\_predicted)

array([[117, 37],  
 [ 45, 32]])

rbf\_acc\_score = accuracy\_score(y1\_test, y1\_rbf\_predicted)  
print("RBF accuracy score ", rbf\_acc\_score)

RBF accuracy score 0.645021645022

## 1d. Ensemble Learning - [ Diabities Problem ]

from sklearn.ensemble import RandomForestClassifier, VotingClassifier  
from sklearn.neural\_network import MLPClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.svm import SVC  
#clf1 = LogisticRegression(random\_state=1)  
clf2 = RandomForestClassifier(random\_state=1)  
clf3 = GaussianNB()  
clf4 = SVC(kernel='rbf', probability=True)  
mlp\_nw = MLPClassifier(solver='lbfgs', alpha=0.01,max\_iter=2000, hidden\_layer\_sizes=(5, 2), random\_state=1, activation='relu')  
mv\_clf = VotingClassifier(estimators=[('pnn\_nw', pnn\_nw), ('clf2', clf2), ('clf3', clf3),('clf4', clf4),('mlp\_nw', mlp\_nw)], voting='hard')  
#mv\_clf = MajorityVoteClassifier(classifiers=[grnn\_nw,pnn\_nw,rbf\_nw])

mv\_clf = mv\_clf.fit(X1\_train, y1\_train)

X1\_test.shape  
#X1\_train.shape  
#mv\_clf  
#y\_mv\_clf\_predicted = mv\_clf.predict(X1\_test)

(231, 8)

y\_mv\_clf\_predicted = mv\_clf.predict(X1\_test)

y\_mv\_clf\_predicted[0]

0.0

confusion\_matrix(y1\_test,y\_mv\_clf\_predicted)

array([[139, 15],  
 [ 37, 40]])

mv\_clf\_acc\_score = accuracy\_score(y1\_test, y\_mv\_clf\_predicted)  
print("Ensembles accuracy score ", mv\_clf\_acc\_score)

Ensembles accuracy score 0.774891774892

## 1e. MLP - [ Diabities Problem ]

from sklearn.neural\_network import MLPClassifier  
mlp\_nw = MLPClassifier(solver='lbfgs', alpha=0.01,max\_iter=2000, hidden\_layer\_sizes=(5, 2), random\_state=1, activation='relu')  
#sgd

mlp\_model = mlp\_nw.fit(X1\_train, y1\_train)  
mlp\_model

MLPClassifier(activation='relu', alpha=0.01, batch\_size='auto', beta\_1=0.9,  
 beta\_2=0.999, early\_stopping=False, epsilon=1e-08,  
 hidden\_layer\_sizes=(5, 2), learning\_rate='constant',  
 learning\_rate\_init=0.001, max\_iter=2000, momentum=0.9,  
 nesterovs\_momentum=True, power\_t=0.5, random\_state=1, shuffle=True,  
 solver='lbfgs', tol=0.0001, validation\_fraction=0.1, verbose=False,  
 warm\_start=False)

y1\_mlp\_predicted = mlp\_model.predict(X1\_test)

y1\_mlp\_predicted[1]

0.0

confusion\_matrix(y1\_test,y1\_mlp\_predicted)

array([[152, 2],  
 [ 62, 15]])

mlp\_acc\_score = accuracy\_score(y1\_test, y1\_mlp\_predicted)  
print("Pnn accuracy score ", mlp\_acc\_score)

Pnn accuracy score 0.722943722944

## 1f. MLFF with tensorflow - [ Diabities Problem ]

#  
# Parameters  
learning\_rate\_1 = 0.001  
training\_epochs\_1 = 20000  
batch\_size\_1 = 10  
display\_step\_1 = 1000  
  
# Network Parameters  
n\_hidden\_1 = 8 # 1st layer number of features  
n\_hidden\_2 = 8 # 1st layer number of features  
n\_input = 8 # diabities data have 8 features and 1 output with 2 classes  
n\_classes = 2 # 2 classess

# tf Graph input  
x = tf.placeholder("float", [None, n\_input],name="x")  
y = tf.placeholder("float", [None,n\_classes],name="y")

# Store layers weight & bias  
weights = {  
 'h1': tf.Variable(tf.random\_normal([n\_input, n\_hidden\_1])),  
 'h2': tf.Variable(tf.random\_normal([n\_hidden\_1, n\_hidden\_2])),  
 'out': tf.Variable(tf.random\_normal([n\_hidden\_2, n\_classes]))  
}  
biases = {  
 'b1': tf.Variable(tf.random\_normal([n\_hidden\_1])),  
 'b2': tf.Variable(tf.random\_normal([n\_hidden\_2])),  
 'out': tf.Variable(tf.random\_normal([n\_classes]))  
}

from sklearn import preprocessing  
def one\_hot(y\_data) :  
 enc = preprocessing.LabelEncoder()  
 y\_data\_encoded = enc.fit\_transform(y\_data)  
 #print(y\_data\_encoded)  
 a = np.array(y\_data\_encoded, dtype=int)  
 b = np.zeros((a.size, a.max()+1))  
 b[np.arange(a.size),a] = 1  
 #print(b)  
 return b

# Create model  
def multilayer\_perceptron\_tf(x, weights, biases):  
 # Hidden layer with RELU activation  
 layer\_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])  
 layer\_1 = tf.nn.relu(layer\_1)  
 # Hidden layer with RELU activation  
 layer\_2 = tf.add(tf.matmul(layer\_1, weights['h2']), biases['b2'])  
 layer\_2 = tf.nn.relu(layer\_2)  
 # Output layer with linear activation  
 out\_layer = tf.matmul(layer\_2, weights['out']) + biases['out']  
 return out\_layer

# Construct model  
pred = multilayer\_perceptron\_tf(x, weights, biases)  
  
# Define loss and optimizer  
cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=pred, labels=y))  
optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate\_1).minimize(cost)  
  
# Initializing the variables  
init = tf.global\_variables\_initializer()

errors = []  
y1\_train\_h = one\_hot(y1\_train)  
# Launch the graph  
with tf.Session() as sess:  
 sess.run(init)  
  
   
 # Training cycle  
 for epoch in range(training\_epochs\_1):  
 avg\_cost = 0.  
 #print(X1\_train.shape, y1\_train.shape)  
 X1\_train, y1\_train\_h = shuffle(X1\_train,y1\_train\_h)  
 # y1\_train = pd.DataFrame(y1\_train,columns=['cls'])  
 # y1\_train['e'] = pd.Series(0, index=y1\_train.index)  
 #print(X1\_train.shape, y1\_train.shape)  
 # Run optimization op (backprop) and cost op (to get loss value)  
 #print(y\_train.shape)  
 \_, c = sess.run([optimizer, cost], feed\_dict={x: X1\_train, y: y1\_train\_h})  
   
 # Display logs per epoch step  
 if epoch % display\_step\_1 == 0:  
 print("Epoch:", '%04d' % (epoch+1), "cost=", \  
 "{:.9f}".format(c))  
 errors.append(c)  
 print("Optimization Finished!")  
  
 # Test model  
 correct\_prediction = tf.equal(tf.argmax(tf.round(pred), 1), tf.argmax(y, 1))  
 # Calculate accuracy  
 accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))  
 y1\_test\_h = one\_hot(y1\_test)  
   
 mlp\_tf\_acc\_Score = accuracy.eval({x: X1\_test, y: y1\_test\_h})  
 print("Accuracy:", mlp\_tf\_acc\_Score)

Epoch: 0001 cost= 2.092828035  
Epoch: 1001 cost= 0.456179202  
Epoch: 2001 cost= 0.433209538  
Epoch: 3001 cost= 0.410290569  
Epoch: 4001 cost= 0.382849276  
Epoch: 5001 cost= 0.365616709  
Epoch: 6001 cost= 0.354704320  
Epoch: 7001 cost= 0.345349103  
Epoch: 8001 cost= 0.338299721  
Epoch: 9001 cost= 0.332837075  
Epoch: 10001 cost= 0.328014821  
Epoch: 11001 cost= 0.315553159  
Epoch: 12001 cost= 0.308456808  
Epoch: 13001 cost= 0.304505199  
Epoch: 14001 cost= 0.301903009  
Epoch: 15001 cost= 0.300433159  
Epoch: 16001 cost= 0.298976272  
Epoch: 17001 cost= 0.298128486  
Epoch: 18001 cost= 0.296585053  
Epoch: 19001 cost= 0.290240049  
Optimization Finished!  
Accuracy: 0.718615

summary\_1 = pd.DataFrame([[grnn\_acc\_score,pnn\_acc\_score,rbf\_acc\_score,mv\_clf\_acc\_score,mlp\_acc\_score,mlp\_tf\_acc\_Score]])  
summary\_1.columns=['GRNN', 'PNN', 'RBF', 'ENSEMBLE', 'MLP','MLP\_TF']

summary\_1

<tr style="text-align: right;">  
 <th></th>  
 <th>GRNN</th>  
 <th>PNN</th>  
 <th>RBF</th>  
 <th>ENSEMBLE</th>  
 <th>MLP</th>  
 <th>MLP\_TF</th>  
</tr>

<tr>  
 <th>0</th>  
 <td>0.705628</td>  
 <td>0.709957</td>  
 <td>0.645022</td>  
 <td>0.774892</td>  
 <td>0.722944</td>  
 <td>0.718615</td>  
</tr>

As seen above, the best classifier was with **Ensemble** learning.

## 2. Wine Quality

Input variables (based on physicochemical tests): 1. - fixed acidity 2. - volatile acidity 3. - citric acid 4. - residual sugar 5. - chlorides 6. - free sulfur dioxide 7. - total sulfur dioxide 8. - density 9. - pH 10. - sulphates 11. - alcohol Output variable (based on sensory data): 12. - quality (score between 0 and 10)

df\_wines = pd.read\_csv('winequality-white.csv')

df\_wines.head(5)

<tr style="text-align: right;">  
 <th></th>  
 <th>fixed acidity</th>  
 <th>volatile acidity</th>  
 <th>citric acid</th>  
 <th>residual sugar</th>  
 <th>chlorides</th>  
 <th>free sulfur dioxide</th>  
 <th>total sulfur dioxide</th>  
 <th>density</th>  
 <th>pH</th>  
 <th>sulphates</th>  
 <th>alcohol</th>  
 <th>quality</th>  
</tr>

<tr>  
 <th>0</th>  
 <td>7.0</td>  
 <td>0.27</td>  
 <td>0.36</td>  
 <td>20.7</td>  
 <td>0.045</td>  
 <td>45.0</td>  
 <td>170.0</td>  
 <td>1.0010</td>  
 <td>3.00</td>  
 <td>0.45</td>  
 <td>8.8</td>  
 <td>6</td>  
</tr>  
<tr>  
 <th>1</th>  
 <td>6.3</td>  
 <td>0.30</td>  
 <td>0.34</td>  
 <td>1.6</td>  
 <td>0.049</td>  
 <td>14.0</td>  
 <td>132.0</td>  
 <td>0.9940</td>  
 <td>3.30</td>  
 <td>0.49</td>  
 <td>9.5</td>  
 <td>6</td>  
</tr>  
<tr>  
 <th>2</th>  
 <td>8.1</td>  
 <td>0.28</td>  
 <td>0.40</td>  
 <td>6.9</td>  
 <td>0.050</td>  
 <td>30.0</td>  
 <td>97.0</td>  
 <td>0.9951</td>  
 <td>3.26</td>  
 <td>0.44</td>  
 <td>10.1</td>  
 <td>6</td>  
</tr>  
<tr>  
 <th>3</th>  
 <td>7.2</td>  
 <td>0.23</td>  
 <td>0.32</td>  
 <td>8.5</td>  
 <td>0.058</td>  
 <td>47.0</td>  
 <td>186.0</td>  
 <td>0.9956</td>  
 <td>3.19</td>  
 <td>0.40</td>  
 <td>9.9</td>  
 <td>6</td>  
</tr>  
<tr>  
 <th>4</th>  
 <td>7.2</td>  
 <td>0.23</td>  
 <td>0.32</td>  
 <td>8.5</td>  
 <td>0.058</td>  
 <td>47.0</td>  
 <td>186.0</td>  
 <td>0.9956</td>  
 <td>3.19</td>  
 <td>0.40</td>  
 <td>9.9</td>  
 <td>6</td>  
</tr>

X2 = df\_wines.iloc[:,:11]  
y2 = df\_wines['quality']  
X2,y2 = shuffle(X2,y2)  
#X1\_train, X1\_test, y1\_train, y1\_test = train\_test\_split(X1, y1, test\_size=0.33, random\_state=42)  
X2\_train, X2\_test, y2\_train, y2\_test = train\_test\_split(preprocessing.minmax\_scale(X2),preprocessing.minmax\_scale(y2),train\_size=0.70)

/usr/local/anaconda3/envs/carnd-term1/lib/python3.5/site-packages/sklearn/utils/validation.py:429: DataConversionWarning: Data with input dtype int64 was converted to float64.  
 warnings.warn(msg, \_DataConversionWarning)

print("train",X2\_train.shape)  
print("test",X2\_test.shape)  
print("train\_y",y2\_train.shape)

train (3428, 11)  
test (1470, 11)  
train\_y (3428,)

### 2a. MLR ( Multi linear Regression)

from sklearn.neural\_network import MLPRegressor  
mlr\_nw = MLPRegressor(solver='lbfgs', alpha=0.01,max\_iter=2000, hidden\_layer\_sizes=(5, 2), random\_state=1, activation='relu')  
#sgd

mlr\_model = mlr\_nw.fit(X2\_train, y2\_train)  
mlr\_model

MLPRegressor(activation='relu', alpha=0.01, batch\_size='auto', beta\_1=0.9,  
 beta\_2=0.999, early\_stopping=False, epsilon=1e-08,  
 hidden\_layer\_sizes=(5, 2), learning\_rate='constant',  
 learning\_rate\_init=0.001, max\_iter=2000, momentum=0.9,  
 nesterovs\_momentum=True, power\_t=0.5, random\_state=1, shuffle=True,  
 solver='lbfgs', tol=0.0001, validation\_fraction=0.1, verbose=False,  
 warm\_start=False)

y2\_mlr\_predicted = mlr\_model.predict(X2\_test)

y2\_mlr\_predicted[0]

0.44020853688104605

# The mean squared error  
y2\_mlr\_mse = np.mean((y2\_mlr\_predicted - y2\_test) \*\* 2)  
print("Mean squared error: %.4f"  
 % y2\_mlr\_mse)

Mean squared error: 0.0154

# Explained variance score: 1 is perfect prediction  
print('Variance score: %.2f' % mlr\_model.score(X2\_test, y2\_test))

Variance score: 0.27

# Plot outputs  
#plt.scatter(X2\_test[:,1:2], y2\_test, color='black')  
#plt.plot(X2\_test, y2\_mlr\_predicted, color='blue',  
# linewidth=3)  
  
#plt.xticks(())  
#plt.yticks(())

### 2b. GRNN

grnn\_nw\_2 = algorithms.GRNN(std=0.1, verbose=True)  
print(grnn\_nw\_2)

Main information  
  
[ALGORITHM] GRNN  
  
[OPTION] verbose = True  
[OPTION] epoch\_end\_signal = None  
[OPTION] show\_epoch = 1  
[OPTION] shuffle\_data = False  
[OPTION] step = 0.1  
[OPTION] train\_end\_signal = None  
[OPTION] std = 0.1  
  
GRNN(std=0.1, show\_epoch=None, train\_end\_signal=None, shuffle\_data=None, verbose=True, epoch\_end\_signal=None, step=None)

grnn\_nw\_2.train(X2\_train, y2\_train)

y2\_grnn\_predicted = grnn\_nw\_2.predict(X2\_test)

y2\_grnn\_predicted[0]

array([ 0.48209117])

# The mean squared error  
y2\_grnn\_mse = np.mean((y2\_grnn\_predicted - y2\_test) \*\* 2)  
print("Mean squared error: %.4f"  
 % y2\_grnn\_mse)

Mean squared error: 0.0294

### 2c. PNN Network

pnn\_nw\_2 = algorithms.PNN(std=10, verbose=False)  
print(pnn\_nw\_2)

PNN(std=10, show\_epoch=1, train\_end\_signal=None, shuffle\_data=False, verbose=False, epoch\_end\_signal=None, batch\_size=128, step=0.1)

pnn\_nw\_2.train(X2\_train, y2\_train)

y2\_pnn\_predicted = pnn\_nw\_2.predict(X2\_test)

y2\_pnn\_predicted[0]

0.33333333333333326

# The mean squared error  
y2\_pnn\_mse = np.mean((y2\_pnn\_predicted - y2\_test) \*\* 2)  
print("Mean squared error: %.4f"  
 % y2\_pnn\_mse)

Mean squared error: 0.0421

### 2d. RBF Network

rbf\_nw\_2 = algorithms.RBFKMeans(n\_clusters=2, verbose=False)

rbf\_nw\_2.train(X2\_train, epsilon=1e-5)

y2\_rbf\_predicted = rbf\_nw\_2.predict(X2\_test)

y2\_rbf\_predicted[0]

array([ 0.])

#The mean squared error  
y2\_rbf\_mse = np.mean((y2\_rbf\_predicted - y2\_test) \*\* 2)  
print("Mean squared error: %.4f"  
 % y2\_rbf\_mse)

Mean squared error: 0.2690

:( Too high

### 2d. Ensembles

from sklearn.ensemble import AdaBoostRegressor  
#clf2 = RandomForestClassifier(random\_state=1)  
#clf3 = GaussianNB()  
#clf4 = SVC(kernel='rbf', probability=True)  
  
en\_reg = AdaBoostRegressor(base\_estimator=mlr\_nw ,n\_estimators=50)  
#

en\_reg.fit(X2\_train, y2\_train)

AdaBoostRegressor(base\_estimator=MLPRegressor(activation='relu', alpha=0.01, batch\_size='auto', beta\_1=0.9,  
 beta\_2=0.999, early\_stopping=False, epsilon=1e-08,  
 hidden\_layer\_sizes=(5, 2), learning\_rate='constant',  
 learning\_rate\_init=0.001, max\_iter=2000, momentum=0.9,  
 nesterovs\_momentum=True, power\_t=0.5, random\_state=1, shuffle=True,  
 solver='lbfgs', tol=0.0001, validation\_fraction=0.1, verbose=False,  
 warm\_start=False),  
 learning\_rate=1.0, loss='linear', n\_estimators=50,  
 random\_state=None)

y2\_ens\_predicted = en\_reg.predict(X2\_test)

y2\_ens\_predicted[0]

0.45672889913636916

# The mean squared error  
y2\_ens\_mse = np.mean((y2\_ens\_predicted - y2\_test) \*\* 2)  
print("Mean squared error: %.4f"  
 % y2\_ens\_mse)

Mean squared error: 0.0148

### Summary

summary\_2 = pd.DataFrame([[y2\_mlr\_mse,y2\_grnn\_mse,y2\_pnn\_mse,y2\_rbf\_mse,y2\_ens\_mse]])  
summary\_2.columns=['MLR', 'GRNN', 'PNN', 'RBF','ENSEMBLE']

summary\_2

<tr style="text-align: right;">  
 <th></th>  
 <th>MLR</th>  
 <th>GRNN</th>  
 <th>PNN</th>  
 <th>RBF</th>  
 <th>ENSEMBLE</th>  
</tr>

<tr>  
 <th>0</th>  
 <td>0.015443</td>  
 <td>0.029372</td>  
 <td>0.042082</td>  
 <td>0.268967</td>  
 <td>0.014786</td>  
</tr>

So the lowest Mean Square Error (MSE) is again with Ensemble.