COMPUTATIONAL INTELLIGENCE I

GROUP CA: NEURAL NETWORK ENSEMBLES

Submitted by : Gaelan Gu, Sunil Prakash

Date: 30/05/2017

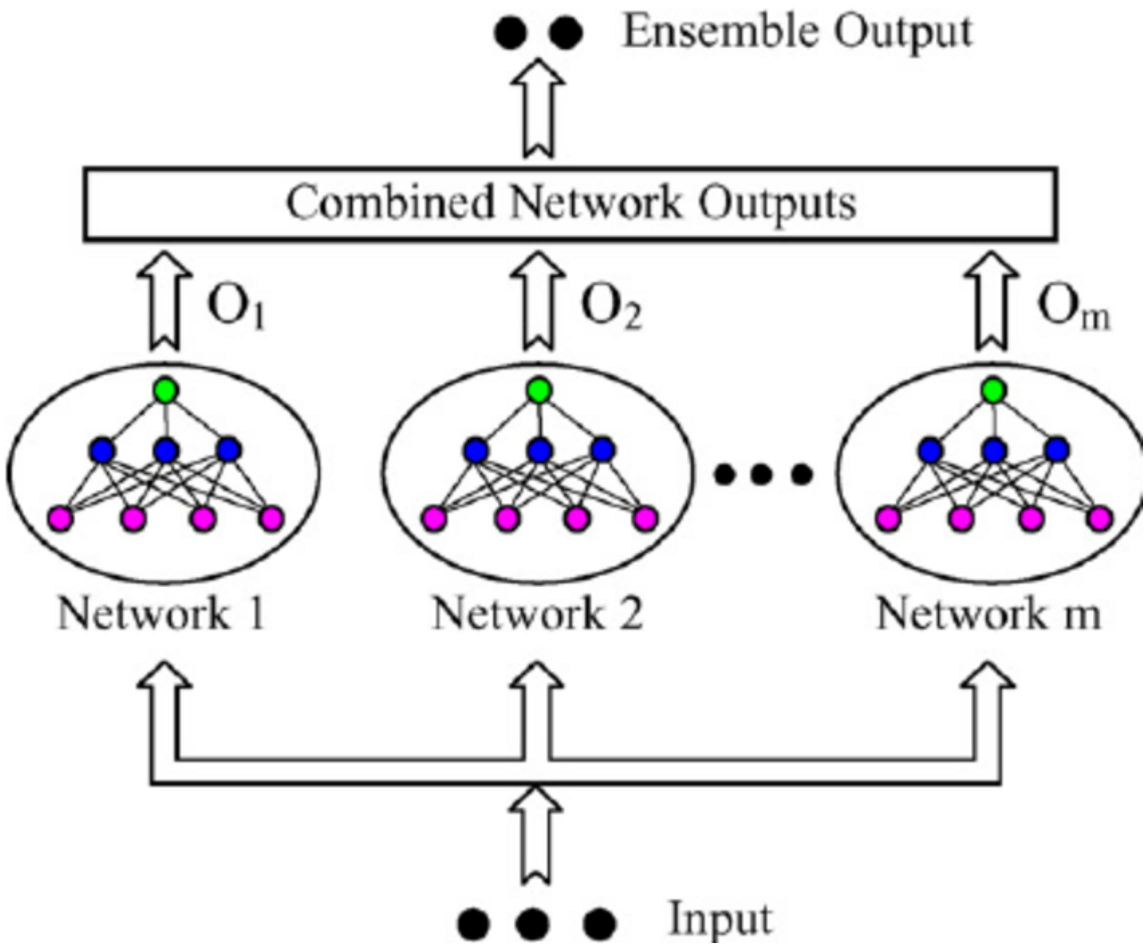
TABLE OF CONTENTS

REPORT SUMMARY

1. NEURAL NETWORK TOOL USED:

We used **Python** and **R** as tool for analyzing and building the neural network models and Neural Network ensembles. We chose these tools because it enabled us the greatest flexibility in data processing and model building. Being open-source tools, there are a myriad of packages available at our disposal.

2. DESIGN OF NEURAL NETWORK ENSEMBLES



The architecture of neural network ensemble is divided into three layers..

1. Input layer :

   Each Network, (in our case, MLFF with BP, RBF , GRNN,), is provided with the same set of inputs. After preprocessing, we split the dataset for evaluation and validation.

2. Network Layer:

   Each network is is trained individually with their own efficiency (could have been parallel also), as there were no dependencies on each individual.

3. Combined Output Layer:

   Output from each individuals are combined with popular majority voting , the technique we used for the Diabetes dataset was voting with bagging as it is a binary classification problem and bagging was an effective solution for it.

$$P_{ens} = \sum_{k=(T/2)+1}^{T} \binom{T}{k} p^k (1-p)^{T-k}$$

   where *T* is number of class , in out case it was 2 for diabetes dataset, and *p* is the probability of making the decision correctly.

   For the Wine dataset, we used Weighted Voting, as some of the networks were providing better results than others, so we provided more weights to one which had a smaller mean squared error rate.

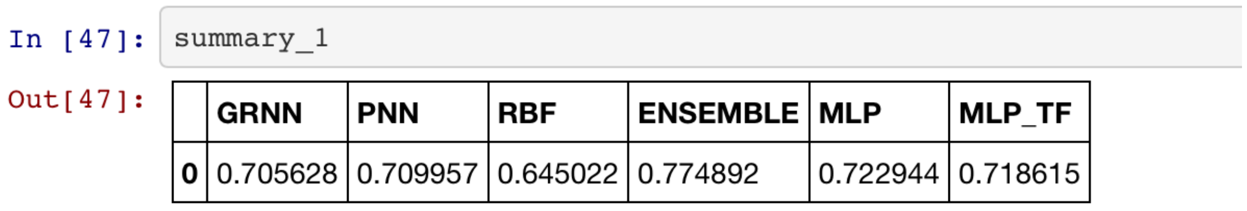
$$\sum_{t=1}^{T} w_t d_{t,J}(\mathbf{x}) = \max_{j=1,\cdots,C} \sum_{t=1}^{T} w_t d_{t,j}$$

   The optimal weight here is calculated for each network based on their probability of providing correct classification.

PERFORMACE OF NN ENSEMBLE

For Diabetes Dataset,

a. **Python →**



```
In [47]: summary_1
Out[47]:
```

	GRNN	PNN	RBF	ENSEMBLE	MLP	MLP_TF
0	0.705628	0.709957	0.645022	0.774892	0.722944	0.718615

70.5%	70.9%	64.5%	**77.4%**	72.3%	71.8%

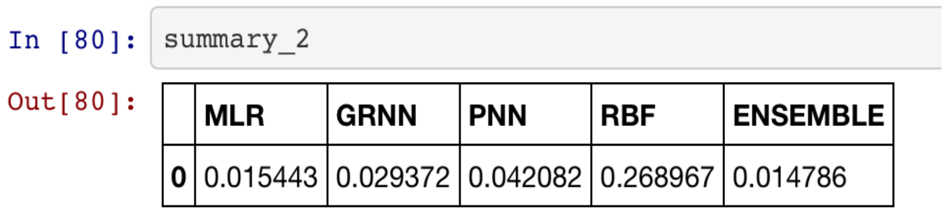
b. **R →**

MLFF	NN with PCANN	ENSEMBLE
65.2%	**65.2%**	**77.3%**

With both R and Python and with different algorithms, accuracy of ensemble was more than the individual networks.

For Wine Dataset

a. **Python →**



```
In [80]: summary_2
Out[80]:
```

	MLR	GRNN	PNN	RBF	ENSEMBLE
0	0.015443	0.029372	0.042082	0.268967	0.014786

b. **R →**

SVM with RF kernel	Stacked DNN	Ensemble
`RMSE ## 0.16278898`	`RMSE ## 0.1178194`	`RMSE ##  0.1178194`

As can be seen with Python, the ensemble model provided a smaller mean squared error rate and better probabilities of output.

3. UNDERSTANDING AND FINDINGS

With the Ensemble learning, the output from the different models are combined to have stronger prediction of correct classification or regression values.

As seen in both the Diabetes and Wine ensembles, the result of the combined networks was better than the individual networks. For the Wine dataset with Python, it was almost 7% more than others.

APPENDIX AND DETAILED REPORT

1. WITH R

NEURAL NETWORK ENSEMBLE FOR DIABETES DATASET

We will be using R to construct our Neural Network (NN) ensemble for the Diabetes dataset. In this dataset, we are attempting to predict the class variable, which we named as *positive.test*, to determine if a patient is positive for diabetes based on a given set of 8 continuous attributes.

```
##  no.of.times.preg plasma.glucose.conc diastolic.pressure
##  Min.   : 0.000   Min.   :  0.0       Min.   :  0.00    
##  1st Qu.: 1.000   1st Qu.: 99.0       1st Qu.: 62.00    
##  Median : 3.000   Median :117.0       Median : 72.00    
##  Mean   : 3.845   Mean   :120.9       Mean   : 69.11    
##  3rd Qu.: 6.000   3rd Qu.:140.2       3rd Qu.: 80.00    
##  Max.   :17.000   Max.   :199.0       Max.   :122.00    
##  triceps.skin.fold.thick serum.insulin        bmi       
##  Min.   : 0.00           Min.   :  0.0   Min.   : 0.00  
##  1st Qu.: 0.00           1st Qu.:  0.0   1st Qu.:27.30  
##  Median :23.00           Median : 30.5   Median :32.00  
##  Mean   :20.54           Mean   : 79.8   Mean   :31.99  
##  3rd Qu.:32.00           3rd Qu.:127.2   3rd Qu.:36.60  
##  Max.   :99.00           Max.   :846.0   Max.   :67.10  
##  diab.pedigree.func      age         positive.test
##  Min.   :0.0780     Min.   :21.00   0:500        
##  1st Qu.:0.2437     1st Qu.:24.00   1:268        
##  Median :0.3725     Median :29.00                
##  Mean   :0.4719     Mean   :33.24                
##  3rd Qu.:0.6262     3rd Qu.:41.00                
##  Max.   :2.4200     Max.   :81.00                
```

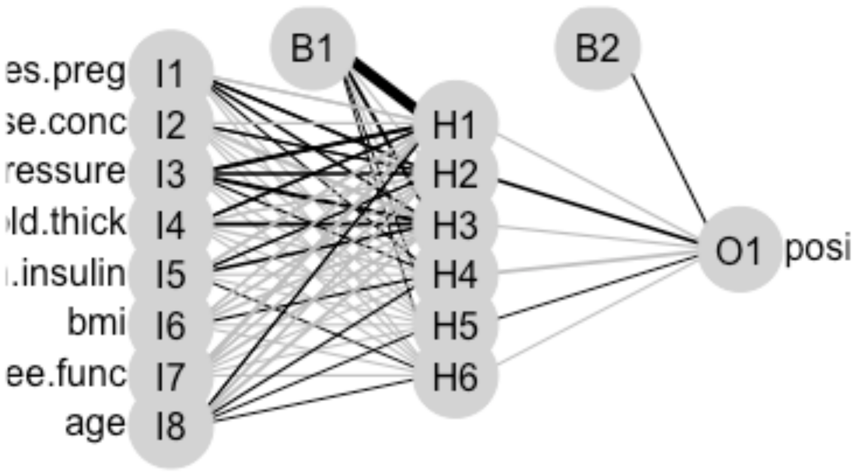
We have factorized our class variable and ascertained that there are no missing values in the dataset. We split our dataset into training and testing sets using the ratio 70:30. We will run 2 neural networks on the training set.

SINGLE-LAYER NEURAL NETWORK

Using a single-layer NN, we used 6 neurons for the hidden layer, set the maximum number of iterations at 10,000 and a learning rate of 0.0001. We keep the learning rate small in order not to exceed the local minimum.

```
## a 8-6-1 network with 61 weights
## inputs: no.of.times.preg plasma.glucose.conc diastolic.pressure triceps.sk
```

```
in.fold.thick serum.insulin bmi diab.pedigree.func age
## output(s): positive.test
## options were - entropy fitting  decay=1e-04
```



6 neurons are used in the hidden layer as this configuration produced the best accuracy, and also because it is optimally between the number of independent variables (8) and output node (1).

```
## Confusion Matrix and Statistics
##
##     predicted
## true   0   1
##    0 141   9
##    1  71   9
##
##                Accuracy : 0.6522
##                  95% CI : (0.5868, 0.7136)
##     No Information Rate : 0.9217
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0641
##  Mcnemar's Test P-Value : 9.104e-12
```

```
##
##             Sensitivity : 0.6651
##             Specificity : 0.5000
##          Pos Pred Value : 0.9400
##          Neg Pred Value : 0.1125
##              Prevalence : 0.9217
##          Detection Rate : 0.6130
##    Detection Prevalence : 0.6522
##       Balanced Accuracy : 0.5825
##
##        'Positive' Class : 0
##
```

Accuracy of **65.2%** achieved on test set with single-layer neural network.

NEURAL NETWORKS WITH FEATURE EXTRACTION

We will apply principal component analysis (PCA) to the training set before applying a single-layer NN to it. We will also keep the number of neurons, maxit and decay values the same as before.

```
## Neural Network Model with PCA Pre-Processing
##
## Created from 538 samples and 8 variables
## PCA needed 8 components to capture 99 percent of the variance
##
## a 8-6-2 network with 68 weights
## options were - decay=1e-04

## Confusion Matrix and Statistics
##
##     predicted
## true   0   1
##    0 130  20
##    1  39  41
##
##                Accuracy : 0.7435
##                  95% CI : (0.6819, 0.7986)
##     No Information Rate : 0.7348
##     P-Value [Acc > NIR] : 0.41574
##
##                   Kappa : 0.4014
##  Mcnemar's Test P-Value : 0.01911
##
```

```
##             Sensitivity : 0.7692
##             Specificity : 0.6721
##          Pos Pred Value : 0.8667
##          Neg Pred Value : 0.5125
##              Prevalence : 0.7348
##          Detection Rate : 0.5652
##    Detection Prevalence : 0.6522
##       Balanced Accuracy : 0.7207
##
##        'Positive' Class : 0
##
```

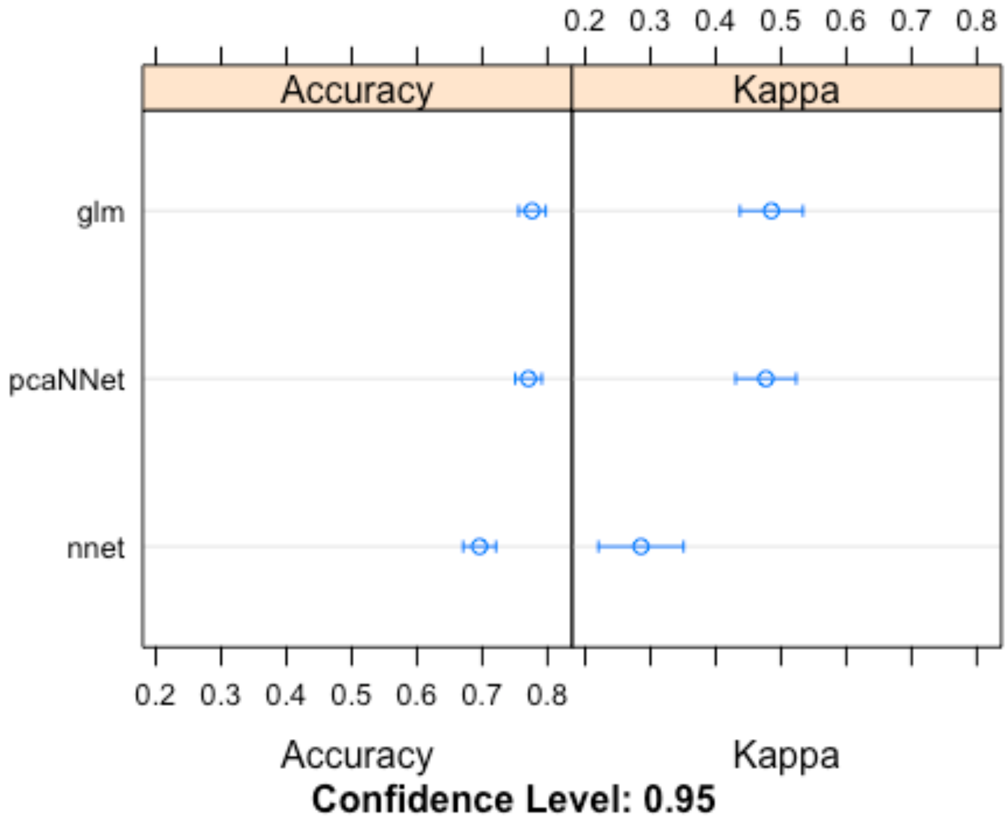
A higher accuracy rate of **74.4%** was achieved on the test set using a single-layer NN with a PCA step (PCANN).

ENSEMBLE LEARNING

Let's see if the accuracy rates can be improved with a NN ensemble with the models. The type of ensemble which we are using is of the stacking type. We train the ensemble for combining the predictions of the individual learning algorithms to achieve a higher accuracy.

The train control which we will use in training is a repeated 10-fold cross-validation iterated 3 times. We train the individual models first.

```
## Call:
## summary.resamples(object = result1)
##
## Models: glm, nnet, pcaNNet
## Number of resamples: 30
##
## Accuracy
##           Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## glm     0.6481  0.7593 0.7778 0.7764  0.8148 0.8704    0
## nnet    0.5741  0.6478 0.6852 0.6958  0.7170 0.8519    0
## pcaNNet 0.6481  0.7407 0.7778 0.7708  0.8113 0.8704    0
##
## Kappa
##             Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## glm      0.19970  0.4352 0.5068 0.4853  0.5735 0.7123    0
## nnet    -0.04722  0.1693 0.2838 0.2855  0.3316 0.6752    0
## pcaNNet  0.21920  0.4191 0.5008 0.4768  0.5554 0.6897    0
```



```
## A glm ensemble of 2 base models: glm, nnet, pcaNNet
##
## Ensemble results:
## Generalized Linear Model
##
## 1614 samples
##    3 predictor
##    2 classes: 'X0', 'X1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1453, 1452, 1453, 1452, 1453, 1453, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7722184  0.4732185
```

The glm-stacked ensemble mode performs at an accuracy of **77.2%**, which is higher than the individual models. As stacking works well by combining multiple models, it will be simpler and easy to implement by using a single stacking script for more models.

NEURAL NETWORK ENSEMBLE FOR WINE QUALITY DATASET

For the Wine dataset, we are attempting to predict the *quality* variable, which ranges from 0 to 10. This will be a regression exercise using 11 continuous predictor variables, which describe several properties of the wine content.

```
##  fixed.acidity    volatile.acidity  citric.acid     residual.sugar
##  Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600
##  1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700
##  Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.200
##  Mean   : 6.855   Mean   :0.2782   Mean   :0.3342   Mean   : 6.391
##  3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9.900
##  Max.   :14.200   Max.   :1.1000   Max.   :1.6600   Max.   :65.800
##    chlorides        free.sulfur.dioxide total.sulfur.dioxide
##  Min.   :0.00900   Min.   :  2.00      Min.   :  9.0
##  1st Qu.:0.03600   1st Qu.: 23.00      1st Qu.:108.0
##  Median :0.04300   Median : 34.00      Median :134.0
##  Mean   :0.04577   Mean   : 35.31      Mean   :138.4
##  3rd Qu.:0.05000   3rd Qu.: 46.00      3rd Qu.:167.0
##  Max.   :0.34600   Max.   :289.00      Max.   :440.0
##     density             pH          sulphates        alcohol
##  Min.   :0.9871   Min.   :2.720   Min.   :0.2200   Min.   : 8.00
##  1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50
##  Median :0.9937   Median :3.180   Median :0.4700   Median :10.40
##  Mean   :0.9940   Mean   :3.188   Mean   :0.4898   Mean   :10.51
##  3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40
##  Max.   :1.0390   Max.   :3.820   Max.   :1.0800   Max.   :14.20
##     quality
##  Min.   :3.000
##  1st Qu.:5.000
##  Median :6.000
##  Mean   :5.878
##  3rd Qu.:6.000
##  Max.   :9.000
```

Similarly, the dataset will be split into training and testing sets using the same ratio as before. We conduct an additional step of scaling the predictor variables using the minimum and maximum values of each column.

SUPPORT VECTOR MACHINE (SVM) WITH RADIAL BASIS FUNCTION (RBF) KERNEL

We will first build a SVM with a RBF kernel, but we leave the determination of the hyperparameters up to the kernel. This is done by setting *kpar* parameter to "automatic".

```
## Support Vector Machine object of class "ksvm"
##
## SV type: eps-svr  (regression)
##  parameter : epsilon = 0.1  cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0774829127912879
##
## Number of Support Vectors : 2986
##
## Objective Function Value : -1589.486
## Training error : 0.5218
```

The kernel recommends a sigma hyperparameter of 0.077. In the case of an epsilon regression, the parameters recommended are epsilon = 0.1 and cost = 1.

```
##       RMSE   Rsquared
## 0.16278898 0.06573323
```

We obtain a root mean squared error (RMSE) of **0.16** from the SVM with RBF kernel.

STACKED AUTOENCODER DEEP NEURAL NETWORK

Next we will use an autoencoder deep neural network for training, however we will only be using one hidden layer with 8 neurons. The purpose of the autoencoder in this case is to perform feature extraction on the training set. It tries to learn aspects of the input in the hidden layer so that it is able to reconstruct the output based on those representations of the input.

Learning rate will be set to a small figure to avoid missing the local minima.

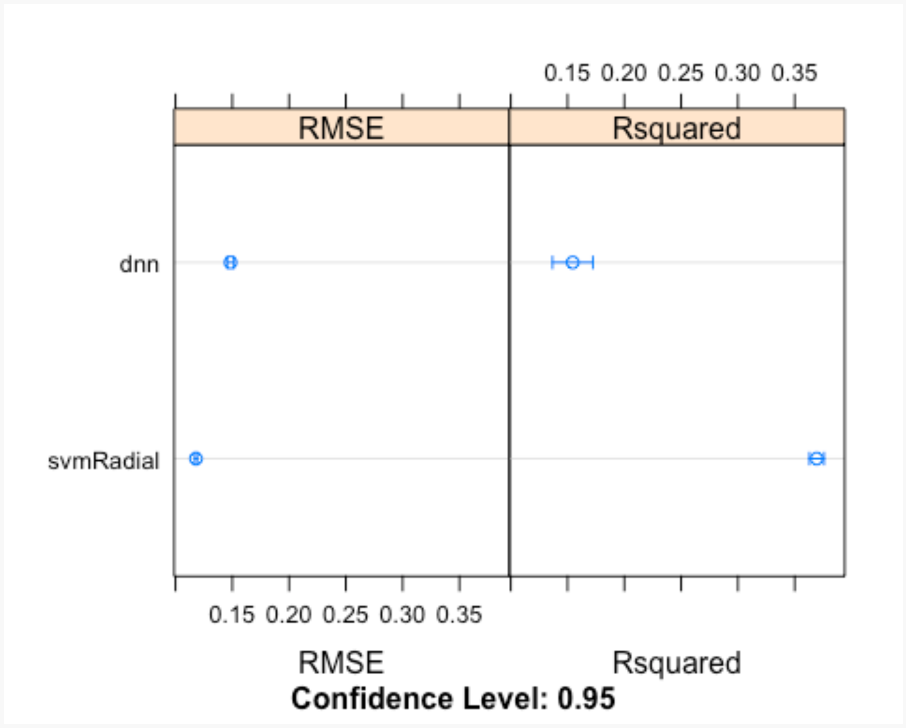
```
##                    Length Class  Mode
## input_dim            1    -none- numeric
## output_dim           1    -none- numeric
## hidden               1    -none- numeric
## size                 3    -none- numeric
## activationfun        1    -none- character
## learningrate         1    -none- numeric
## momentum             1    -none- numeric
## learningrate_scale   1    -none- numeric
## hidden_dropout       1    -none- numeric
## visible_dropout      1    -none- numeric
## output               1    -none- character
## W                    2    -none- list
## vW                   2    -none- list
## B                    2    -none- list
## vB                   2    -none- list
## post                      3    -none- list
## pre                       3    -none- list
## e                        30    -none- numeric
## L                       350    -none- numeric

##      RMSE  Rsquared
## 0.1481129 0.0352256
```

We obtain a slightly improved RMSE of **0.14** in this case.

ENSEMBLE LEARNING

We will stack the models in this ensemble to determine if we can obtain an even lower RMSE. 25 bootstrapped samples will be obtained from the training set to train the models and this is reflected in the train control parameter.

```
## Call:
## summary.resamples(object = result2)
##
## Models: svmRadial, dnn
## Number of resamples: 25
##
## RMSE
##             Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## svmRadial 0.1131  0.1160 0.1190 0.1180  0.1203 0.1218    0
## dnn       0.1398  0.1446 0.1464 0.1484  0.1504 0.1683    0
```



```
## A glm ensemble of 2 base models: svmRadial, dnn
##
## Ensemble results:
## Generalized Linear Model
##
## 31641 samples
##     2 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 31641, 31641, 31641, 31641, 31641, 31641, ...
## Resampling results:
##
##   RMSE       Rsquared
##   0.1178194  0.3674986
```

The glm-stacked ensemble model has performed slightly better at a RMSE of **0.12**, as compared to the individual models.

The full R code may be found with our R script submission.

2. WITH PYTHON

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 70% as training data and 30% 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. [Diabetes 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/uti
ls/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 - [Diabetes 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, verb
ose=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 - [Diabetes 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 - [Diabetes 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 - [DIABETES 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_laye
r_sizes=(5, 2), random_state=1, activation='relu')
mv_clf = VotingClassifier(estimators=[('pnn_nw', pnn_nw), ('clf2', clf2), ('c
lf3', 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 - [Diabetes Problem]

```
from sklearn.neural_network import MLPClassifier
mlp_nw = MLPClassifier(solver='lbfgs', alpha=0.01,max_iter=2000,  hidden_laye
r_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 - [Diabetes 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 # diabetes 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, la
bels=y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate_1).minimize(co
st)
```

```
# 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_trai
n_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/uti
ls/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, verb
ose=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_en
s_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.