```r
# function to ensure the all packages are loaded
EnsurePackage<-function(x)
{ # EnsurePackage(x) - Installs and loads a package
  # if necessary
  x <- as.character(x)
  if (!require(x, character.only=TRUE))
  {
    install.packages(pkgs=x,
                     repos="http://cran.r-project.org")
  }
  require(x, character.only=TRUE)

}

#Installs and loads all necessary packages
#
Prepare.Packages<-function(){

  EnsurePackage("neuralnet")
  EnsurePackage("Metrics")
  EnsurePackage("mosaic")
  EnsurePackage("tidyr")
  EnsurePackage("dplyr")
  EnsurePackage("nnet")
  EnsurePackage("devtools")
}

Prepare.Packages()
source_url('https://gist.githubusercontent.com/fawda123/7471137/raw/466c1474d0a505ff044412703516c34f1a4684a5/nnet_plot_update.r')

# loading the dataset
concrete <- read.csv('/home/rahul/programming/independantStudy/datasets/concrete.csv')
str(concrete)
head(concrete)

# normalizing function for dataset
normFun <- function(x) ((x - min(x))/(max(x) - min(x)))

# normalized concrete data
concreteNorm <- as.data.frame(lapply(concrete, normFun))
summary(concreteNorm)

# splitting data to trainig and testing
concrete_train <- concreteNorm[1:773,]
concrete_test <- concreteNorm[774:1030,]

#######################################################################################

# Trying out a pure neural network approach

# creating the model
formula_1 <- strength ~ cement+slag+ash+water+superplastic+coarseagg+fineagg+age

#model <- neuralnet(formula_1, data = concrete_train, hidden = 4)
model <- nnet(formula_1,concrete_train, size = 4 )
plot.nnet(model)

# prediction
#model_results <- compute(model, concrete_test[1:8])
nrow(concrete_train)
model_results <- predict(model,concrete_test[1:8] )
strength_predicted <-  model_results
real_pred <- cbind(concrete_test[9],strength_predicted )
names(real_pred)

# squared error
squared_error <- se(real_pred$strength, real_pred$strength_predicted)
squared_error
# combined real, predicted and squared error into one data frame
range <- seq(1, length(squared_error), 1)
real_pred <- as.data.frame(cbind(concrete_test$strength, strength_predicted,squared_error, range))
colnames(real_pred) <- c("real_values", "predicted_values","squared_error", "range")
real_pred_gather <- real_pred %>%  gather("attr", "value", 1:3)

# results

# finding the correlation between predicted and real value
correlation <- cor(strength_predicted,concrete_test$strength)
correlation
# mean squared error between observed and predicted values
MSE <- mse(strength_predicted,concrete_test$strength)

# ploting the results
xyplot (value ~ range, group=attr, data =real_pred_gather,type = c("p", "smooth"), lwd = 4, auto.key = list(columns = 
nlevels(real_pred_gather$attr)), main = "Neural network results")

#####################################################################################

# Trying out a fuzzy inference system for the same

library('frbs')
method.type <- "WM"
range.data<-matrix(apply(concrete_train, 2, range), nrow = 2)
range.data
control <- list(num.labels = 7, max.iter = 10, step.size = 0.01, type.tnorm = "MIN", type.snorm = "MAX",type.mf = "GAUSSIAN", 
type.defuz = "WAM", type.implication.func = "ZADEH", name = "concrete")
```

```
# generating the fuzzy inference rules from the data set
object.WM <- frbs.learn(concrete_train,range.data, method.type, control)
str(object.WM)

# predicting based on the generated fuzzy inference system
pred.WM <- predict(object.WM, concrete_test[1:8])


real_pred_fuzzy <- as.data.frame(cbind(concrete_test[,9], pred.WM))

# calculating the squared errorof real and predicted values
f_se <- se( real_pred_fuzzy$V1,  real_pred_fuzzy$V2)

f_range <- range <- seq(1, length(f_se), 1)


# combining everything into a dataframe
real_pred_fuzzy <- as.data.frame( cbind(real_pred_fuzzy, f_se, range))
colnames(real_pred_fuzzy) <- c("real.val", "pred.val", "squared.error", "range")


# merging columns real.val, pred.val and squared.error into 2 columns
real_pred_fuzzy_gather <- real_pred_fuzzy %>% gather("attr", "value", 1:3)

# ploting the results
xyplot (value ~ range, group=attr, data =real_pred_fuzzy_gather,type = c("p", "smooth"), lwd = 4, auto.key = list(columns =
nlevels(real_pred_fuzzy_gather$attr)), main = "Fuzzy inference system results")


# mean squared error of the fuzzy logic value
f_mse <- mse( real_pred_fuzzy$real.val,  real_pred_fuzzy$pred.val)

# correlation between real and predicted strength of concrete using fuzzy inference
f_correlation <-  cor(real_pred_fuzzy$real.val,real_pred_fuzzy$pred.val )

# MSE matrix for neuro and fuzzy systems
MSE_for_both <- cbind("MSE.nnet" = MSE, "MSE.fuzzy" = f_mse )
correlation_for_both <- cbind("corr.nnet" = correlation, "corr.fuzzy" = f_correlation)

##########################################################################################

#connecting fuzzy inference with a neural net

# inputs: concrete[1:8] + trained fuzzy inference system output
# setting seed for reproducing the shuffle
set.seed(123)
ind <- sample(2, nrow(concrete), replace = TRUE, prob = c(0.7, 0.3))
concrete.train_2 <- concrete[ind ==1,]
concrete.test_2 <- concrete[ind ==2,]

#concreteShuffled <- concrete[sample(nrow(concrete)),]
# normalizing function for dataset
normFun <- function(x) ((x - min(x))/(max(x) - min(x)))
# normalized concrete data
#concreteShuffledNorm <- as.data.frame(lapply(concreteShuffled, normFun))
#summary(concreteShuffledNorm)

concrete.train_2 <- as.data.frame(lapply(concrete.train_2, normFun))
concrete.test_2 <- as.data.frame(lapply(concrete.test_2, normFun))
concrete.train_2.shuffled <- concrete.train_2[sample(nrow(concrete.train_2)),]

# creating the model
control <- list(num.labels = 7, max.iter = 10, step.size = 0.01, type.tnorm = "MIN", type.snorm = "MAX",type.mf = "GAUSSIAN",
type.defuz = "WAM", type.implication.func = "ZADEH", name = "concrete_2")

range.data<-matrix(apply(concrete.train_2, 2, range), nrow = 2)
range.data


# generating the fuzzy inference rules from the data set
object.WM <- frbs.learn(concrete.train_2,range.data, method.type, control)
pred.WM_in <- predict(object.WM, concrete.train_2[1:8])
pred.WM.in.test <- predict (object.WM, concrete.test_2[1:8])

pred.WM_in <- as.numeric(pred.WM_in)
pred.WM.in.test  <- as.numeric(pred.WM.in.test)

cor(pred.WM_in, concrete.train_2[9]) # fuzzy predicted and observed value - training data

concrete.fuzzy.train <- as.data.frame(cbind (concrete.train_2, "fuzzy" = pred.WM_in ))
names(concrete.fuzzy.train[10])

ncol(concrete.fuzzy.train)
nrow(concrete.fuzzy.train)

concrete_fuzzy_test <- as.data.frame(cbind (concrete.test_2, "fuzzy" = pred.WM.in.test))

names(concrete_fuzzy_test)
ncol (concrete_fuzzy_test)
nrow (concrete_fuzzy_test)

str(concrete_fuzzy_test)

# creating a nnet model - something wrong here with the setup
formula_2 <- strength ~ cement+slag+ash+water+superplastic+coarseagg+fineagg+age+fuzzy
model2 <- nnet(formula_2,concrete.fuzzy.train, size = 4 )
names(concrete.fuzzy.train)
plot.nnet(model2)
```

```
model_results_final <- predict(model2,concrete_fuzzy_test)

# squared error
squared_error <- se(concrete_fuzzy_test[,9], model_results_final)

# combined real, predicted and squared error into one data frame
range <- seq(1, length(squared_error), 1)
real_pred <- as.data.frame(cbind(concrete_fuzzy_test$strength, model_results_final, squared_error, range))
colnames(real_pred) <- c("real_values", "predicted_values","squared_error", "range")
real_pred_gather <- real_pred %>%  gather("attr", "value", 1:3)
head(real_pred_gather)
# results

# finding the correlation between predicted and real value
correlation <- cor(model_results_final,concrete_fuzzy_test$strength)
correlation
# mean squared error between observed and predicted values
MSE <- mse(model_results_final,concrete_fuzzy_test$strength)
MSE

# MSE  and correlation matrix for neuro, fuzzy  and fuzzy-neuro systems

MSE_for_all <- cbind(MSE_for_both, "mse.neuro_fuzzy" = MSE)
MSE_for_all <- MSE_for_all[,1:3]
correlation_for_all <- cbind(correlation_for_both, "corr_neuro_fuzzy" = correlation)
colnames(correlation_for_all) <- c("corr.neuro", "corr.fuzzy", "corr.neuro.fuzzy")
# output
MSE_for_all
correlation_for_all


# ploting the results
xyplot (value ~ range, group=attr, data =real_pred_gather,type = c("p", "smooth"), lwd = 4, auto.key = list(columns =
nlevels(real_pred_gather$attr)), main = "Neuro-fuzzy system results")


##############################################################################
              #### deep learning using h2o library####
```