R Exercise: Artificial Neural Networks

Pilsung Kang

School of Industrial Management Engineering

Korea University

R Exercise: Classification

- Dataset: Cardiotocography Data Set
  - ✓ https://archive.ics.uci.edu/ml/datasets/Cardiotocography

About Citation Policy Donate a Data Set Contact

Search

Repository Web

UCI Machine Learning Repository

Center for Machine Learning and Intelligent Systems

View ALL Data Sets

Cardiotocography Data Set

Download: Data Folder, Data Set Description

Abstract: The dataset consists of measurements of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms classified by expert obstetricians.

Data Set Characteristics:	Multivariate	**Number of Instances:**	2126	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	23	**Date Donated**	2010-09-07
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	96768

Source:

Marques de SÃ¡, J.P., jpmdesa '@' gmail.com, Biomedical Engineering Institute, Porto, Portugal.
Bernardes, J., joaobern '@' med.up.pt, Faculty of Medicine, University of Porto, Portugal.
Ayres de Campos, D., sisporto '@' med.up.pt, Faculty of Medicine, University of Porto, Portugal.

Data Set Information:

2126 fetal cardiotocograms (CTGs) were automatically processed and the respective diagnostic features measured. The CTGs were also classified by three expert obstetricians and a consensus classification label assigned to each of them. Classification was both with respect to a morphologic pattern (A, B, C. ...) and to a fetal state (N, S, P). Therefore the dataset can be used either for 10-class or 3-class experiments.

R Exercise: Classification

- Dataset: Cardiotocography Data Set

Input variables	LB - FHR baseline (beats per minute)
	AC - # of accelerations per second
	FM - # of fetal movements per second
	UC - # of uterine contractions per second
	DL - # of light decelerations per second
	DS - # of severe decelerations per second
	DP - # of prolongued decelerations per second
	ASTV - percentage of time with abnormal short term variability
	MSTV - mean value of short term variability
	ALTV - percentage of time with abnormal long term variability
	MLTV - mean value of long term variability
	Width - width of FHR histogram
	Min - minimum of FHR histogram
	Max - Maximum of FHR histogram
	Nmax - # of histogram peaks
	Nzeros - # of histogram zeros
	Mode - histogram mode
	Mean - histogram mean
	Median - histogram median
	Variance - histogram variance
	Tendency - histogram tendency
	CLASS - FHR pattern class code (1 to 10)
Target variable	NSP - fetal state class code (N=normal; S=suspect; P=pathologic)

R Exercise: Classification

- Performance evaluation function

```
# Part 1: Multi-class classification with ANN & Multinimial logistic regression
# Performance evaluation function for multi-class classification ----------
perf_eval_multi <- function(cm){
    # Simple Accuracy
    ACC = sum(diag(cm))/sum(cm)
    # Balanced Correction Rate
    BCR = 1
    for (i in 1:dim(cm)[1]){
        BCR = BCR*(cm[i,i]/sum(cm[i,]))
    }
    BCR = BCR^(1/dim(cm)[1])
    return(c(ACC, BCR))
}
```

✓ perf_eval_multi

- Argument: confusion matrix (cm)
- Outputs: simple accuracy (ACC), balanced correction rate (BCR)

R Exercise: Classification

- Install package

```r
# Install nnet package and prepare it
install.packages("nnet")
library(nnet)
```

  ✓ "nnet" package

    - Many packages provide neural network module
    - "nnet" is one of the most widely used packages

R Exercise: Classification

- Data loading and preprocessing

```
# Multi-class classification (ctgs dataset)
ctgs_data <- read.csv("ctgs.csv")
n_instance <- dim(ctgs_data)[1]
n_var <- dim(ctgs_data)[2]

# Conduct normalization
ctgs_input <- ctgs_data[,-n_var]
ctgs_target <- ctgs_data[,n_var]
ctgs_input <- scale(ctgs_input, center = TRUE, scale = TRUE)
ctgs_target <- as.factor(ctgs_target)
ctgs_data_normalized <- data.frame(ctgs_input, Class = ctgs_target)
```

✓ Data loading

- read.csv( ) function, use header = TRUE option because the first row is the variable name
- Use dim( ) function to check the number of rows and columns in the dataset

✓ Data normalization

- Make the average and standard deviation of each variable to 0 and 1, respectively
- Covert the target variable type to "factor"

R Exercise: Classification

- Data loading and preprocessing

```
# Initialize performance matrix
perf_summary <- matrix(0, nrow = 2, ncol = 2)
colnames(perf_summary) <- c("ACC", "BCR")
rownames(perf_summary) <- c("Multi_Logit", "ANN")

# Split the data into the training/validation sets
set.seed(12345)
trn_idx <- sample(1:n_instance, round(0.8*n_instance))
ctgs_trn <- ctgs_data_normalized[trn_idx,]
ctgs_tst <- ctgs_data_normalized[-trn_idx,]
```

✓ Initialize the performance summary matrix

✓ Data partition

  - set.seed( ): set the seed of random initialization
  - Use 80% for training and 20% for test

R Exercise: Classification

- Training the Multinomial Logistic Regression

```
# Multinomial logistic regression -----------------------------------------
# Train multinomial logistic regression
ml_logit <- multinom(Class ~ ., data = ctgs_trn)

# Check the coefficients
summary(ml_logit)
t(summary(ml_logit)$coefficients)
```

✓ Estimated regression coefficients

```
> t(summary(ml_logit)$coefficients)
                       2            3
(Intercept) -4.694977008 -11.58608545     MLTV        -0.056153529    0.68602348
LB          -1.136169974   2.94827810     Width       -0.007595327    0.13962712
AC          -3.747793932  -3.44072166     Min          0.327512978    0.55153110
FM           0.469688181   1.07087196     Max          0.523037010    1.21168222
UC          -0.907147801  -1.07874916     Nmax         0.328949181   -0.92033096
DL           0.028822895   0.15122342     Nzeros      -0.154133544    0.36098681
DS          -0.433749154   0.23945220     Mode        -1.041006093   -0.05063308
DP           1.479317937   1.36253728     Mean         4.336477367   -1.34224223
ASTV         1.376230801   3.46304432     Median      -0.604921319   -4.03591917
MSTV        -0.269653587  -1.37476523     Variance     1.178814582    1.99001476
ALTV         0.413875887   1.48408278     Tendency     0.117217134    0.16513548
```

R Exercise: Classification

- Classify the Test Examples using the Multinomial Logistic Regression

```
# Predict the class label
ml_logit_prey <- predict(ml_logit, newdata = ctgs_tst)
cfmatrix <- table(ctgs_tst$Class, ml_logit_prey)
cfmatrix

perf_summary[1,] <- perf_eval_multi(cfmatrix)
perf_summary
```

✓ Simple accuracy: 0.8941, Balanced correction rate: 0.7468

R Exercise: Classification

- Data transformation for MLR

```
# Artificial Neural Network ----------------------------------------------
# Train ANN
ann_trn_input <- ctgs_trn[,-n_var]
ann_trn_target <- class.ind(ctgs_trn[,n_var])
```

✓ For multi-class classification, the number of output nodes is the same as the number of classes

✓ Use class.ind( ) function to convert the target variable (factor type) to one-hot (1-of-C coding) vector

Class
1
2
3
1
2

C_1	C_2	C_3
1	0	0
0	1	0
0	0	1
1	0	0
0	1	0

R Exercise: Classification

- Search the best number of hidden nodes

```
# Find the best number of hidden nodes in terms of BCR
# Candidate hidden nodes
nH <- seq(from=5, to=30, by=5)
# 5-fold cross validation index
val_idx <- sample(c(1:5), dim(ann_trn_input)[1], replace = TRUE, prob = rep(0.2,5))
val_perf <- matrix(0, length(nH), 3)
```

  ✓ The best number of hidden node depends on the dataset

  ✓ Perform 5-fold cross validation

    - The range of hidden nodes: 5 ~ 30 (step by 5)
    - Initialize validation index
    - val_perf: store the performance for each number of hidden node

R Exercise: Classification

- Search the best number of hidden nodes

```
for (i in 1:length(nH)) {
    cat("Training ANN: the number of hidden nodes:", nH[i], "\n")
    eval_fold <- c()
    for (j in c(1:5)) {
        # Training with the data in (k-1) folds
        tmp_trn_input <- ann_trn_input[which(val_idx != j),]
        tmp_trn_target <- ann_trn_target[which(val_idx != j),]
        tmp_nnet <- nnet(tmp_trn_input, tmp_trn_target, size = nH[i],
                         decay = 5e-4, maxit = 500)
```

✓ Repeat the process for all candidate number of hidden node (i) and five folds (j)

  ▪ If the val_idx is j, use the example for validation, otherwise use the example for training

✓ nnet( ): function for training the neural network

  ▪ Arg 1: input (X) of training data

  ▪ Arg 2: target (y) of training data

  ▪ Arg 3: number of hidden nodes

  ▪ Arg 4 & 5: weight decay threshold, maximum number of epochs

R Exercise: Classification

- Search the best number of hidden nodes

```
# Evaluate the model with the remaining 1 fold
tmp_val_input <- ann_trn_input[which(val_idx == j),]
tmp_val_target <- ann_trn_target[which(val_idx == j),]
eval_fold <- rbind(eval_fold, cbind(max.col(tmp_val_target),
         max.col(predict(tmp_nnet, tmp_val_input))))

# Confusion matrix
cfm <- table(eval_fold[,1], eval_fold[,2])
# nH
val_perf[i,1] <-nH[i]
# Record the validation performance
val_perf[i,2:3] <- t(perf_eval_multi(cfm))
```

✓ Combine the prediction results for 5 validation sets using rbind( ) and evaluate it together

R Exercise: Classification

- Search the best number of hidden nodes

```
ordered_val_perf <- val_perf[order(val_perf[,3], decreasing = TRUE),]
colnames(ordered_val_perf) <- c("nH", "ACC", "BCR")
ordered_val_perf
# Find the best number of hidden node
best_nH <- ordered_val_perf[1,1]
```

✓ Find the best hidden node in terms of BCR

- Different results can be obtained for different trials

```
> ordered_val_perf
     nH       ACC       BCR
[1,] 30 0.9112287 0.8260219
[2,] 25 0.9118166 0.8226941
[3,] 20 0.9135802 0.8094907
[4,] 15 0.9012346 0.8008131
[5,] 10 0.8924162 0.7813366
[6,]  5 0.8847737 0.7753846
```

R Exercise: Classification

- Model training with the best number of hidden nodes

```
# Test the ANN
ann_tst_input = ctgs_tst[,-n_var]
ann_tst_target = class.ind(ctgs_tst[,n_var])
ctgs_nnet <- nnet(ann_trn_input, ann_trn_target, size = best_nH,
                  decay = 5e-4, maxit = 500)
```

✓ Use the best number of hidden nodes determined by the 5-fold cross validation to train the entire training dataset

R Exercise: Classification

- Evaluate the MLR and compare the results with the multinomial logistic regression

```
# Performance evaluation
prey <- predict(ctgs_nnet, ann_tst_input)
tst_cm <- table(max.col(ann_tst_target), max.col(prey))
tst_cm perf_summary[2,] <- perf_eval_multi(tst_cm)
perf_summary
```

```
> cfmatrix
 ml_logit_prey
     1   2   3
  1 318  14   0
  2  18  41   1
  3   4   8  21
```

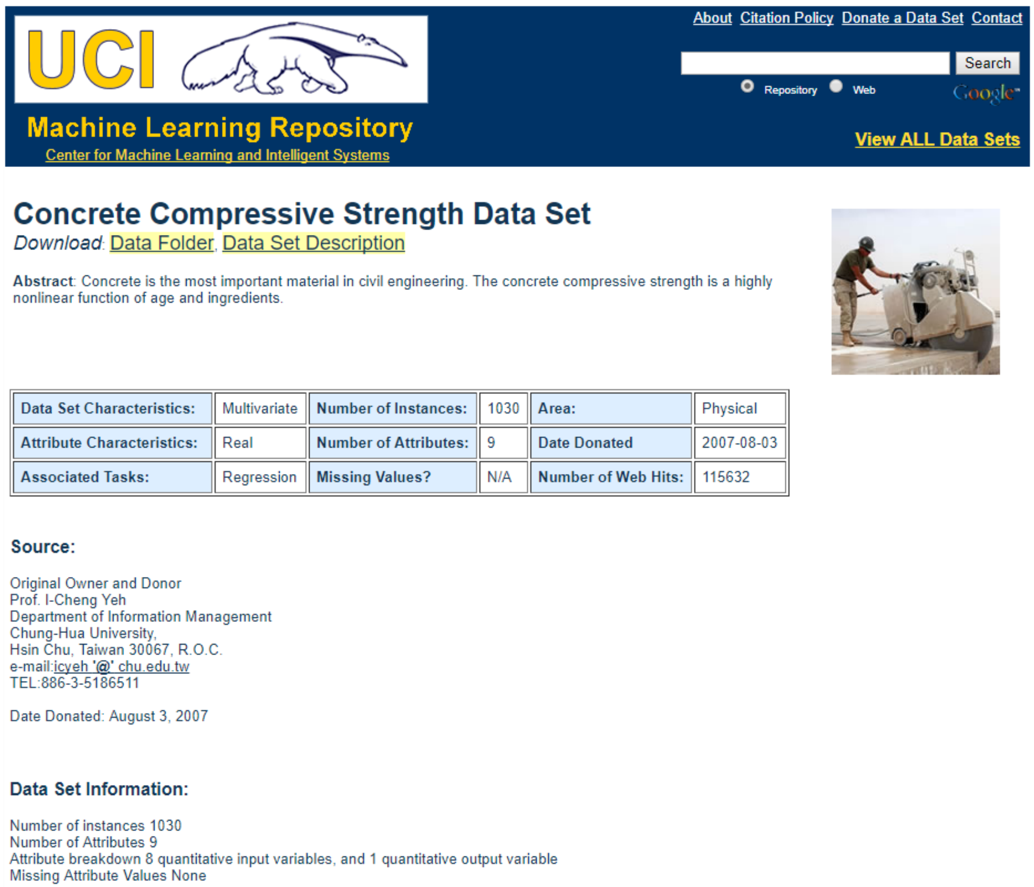
```
> tst_cm

      1   2   3
  1 321  11   0
  2  20  40   0
  3   4   5  24
```

```
> perf_summary
                  ACC       BCR
Multi_Logit 0.8941176 0.7468081
ANN         0.9058824 0.7768270
```

R Exercise: Regression

- Dataset: Concrete Compressive Strength Data Set
  - ✓ Predict the strength of concrete with different proportions of ingredients
  - ✓ https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength



About Citation Policy Donate a Data Set Contact

UCI Machine Learning Repository
Center for Machine Learning and Intelligent Systems

Search
Repository Web
View ALL Data Sets

Concrete Compressive Strength Data Set

Download: Data Folder, Data Set Description

Abstract: Concrete is the most important material in civil engineering. The concrete compressive strength is a highly nonlinear function of age and ingredients.

Data Set Characteristics:	Multivariate	Number of Instances:	1030	Area:	Physical
Attribute Characteristics:	Real	Number of Attributes:	9	Date Donated	2007-08-03
Associated Tasks:	Regression	Missing Values?	N/A	Number of Web Hits:	115632

Source:

Original Owner and Donor
Prof. I-Cheng Yeh
Department of Information Management
Chung-Hua University,
Hsin Chu, Taiwan 30067, R.O.C.
e-mail:icyeh '@' chu.edu.tw
TEL:886-3-5186511

Date Donated: August 3, 2007

Data Set Information:

Number of instances 1030
Number of Attributes 9
Attribute breakdown 8 quantitative input variables, and 1 quantitative output variable
Missing Attribute Values None

R Exercise: Regression

- Dataset: Concrete Compressive Strength Data Set
  - ✓ Input and target variables

Cement (component 1) -- quantitative -- kg in a m3 mixture -- Input Variable
Blast Furnace Slag (component 2) -- quantitative -- kg in a m3 mixture -- Input Variable
Fly Ash (component 3) -- quantitative -- kg in a m3 mixture -- Input Variable
Water (component 4) -- quantitative -- kg in a m3 mixture -- Input Variable
Superplasticizer (component 5) -- quantitative -- kg in a m3 mixture -- Input Variable
Coarse Aggregate (component 6) -- quantitative -- kg in a m3 mixture -- Input Variable
Fine Aggregate (component 7) -- quantitative -- kg in a m3 mixture -- Input Variable
Age -- quantitative -- Day (1~365) -- Input Variable
Concrete compressive strength -- quantitative -- MPa -- Output Variable

R Exercise: Regression

- Compare MLR, k-NN, & ANN
  - ✓ Performance evaluation function

```
# Part 2: Regression with MLR, k-NN, and ANN
# Performance evaluation function for regression --------------------------
perf_eval_reg <- function(tgt_y, pre_y){
    # RMSE
    rmse <- sqrt(mean((tgt_y - pre_y)^2))
    # MAE
    mae <- mean(abs(tgt_y - pre_y))
    # MAPE
    mape <- 100*mean(abs((tgt_y - pre_y)/tgt_y))
    return(c(rmse, mae, mape))
}
```

  - ✓ Args: target value, predicted value
  - ✓ Outputs: RMSE, MAE, MAPE

R Exercise: Regression

- Data loading and preprocessing

```
# Concrete strength data
concrete <- read.csv("concrete.csv", header = FALSE)
n_instance <- dim(concrete)[1]
n_var <- dim(concrete)[2]
RegX <- concrete[,-n_var]
RegY <- concrete[,n_var]
# Data Normalization
RegX <- scale(RegX, center = TRUE, scale = TRUE)
# Combine X and Y
RegData <- as.data.frame(cbind(RegX, RegY))
```

✓ Use read.csv( ) function

  - Use header = FALSE option because the first row is not the name of variable

✓ Store the number of instances and variables

✓ Perform normalization for input variables

✓ Combine the normalized input variables and target variable for modeling

R Exercise: Regression

- Data loading and preprocessing

```
# Split the data into the training/test sets
set.seed(12345)
trn_idx <- sample(1:n_instance, round(0.7*n_instance))
trn_data <- RegData[trn_idx,]
tst_data <- RegData[-trn_idx,]
perf_summary_reg <- matrix(0,3,3)
rownames(perf_summary_reg) <- c("MLR", "k-NN", "ANN")
colnames(perf_summary_reg) <- c("RMSE", "MAE", "MAPE")
```

✓ Data partitioning: 70% for training 30% for test

✓ Initialize the performance summary table

- Algorithms: MLR, k-NN, ANN
- Metrics: RMSE, MAE, MAPE

R Exercise: Regression

- Training and Evaluating MLR

```
# Multiple linear regression
full_model <- lm(RegY ~ ., data = trn_data)
mlr_prey <- predict(full_model, newdata = tst_data)
perf_summary_reg[1,] <- perf_eval_reg(tst_data$RegY, mlr_prey)
perf_summary_reg
```

✓ Train the MLR with all variables

```
> perf_summary_reg
          RMSE      MAE     MAPE
MLR   10.44926 8.166065 30.85209
k-NN   0.00000 0.000000  0.00000
ANN    0.00000 0.000000  0.00000
```

R Exercise: Regression

- Training and Evaluating the k-NN

```
# Evaluate the k-NN with the test data
# k-Nearest Neighbor Learning (Regression) --------------------------------
install.packages("FNN", dependencies = TRUE)
library(FNN)

knn_reg <- knn.reg(trn_data[,-n_var], test = tst_data[,-n_var], trn_data$RegY, k=3)
knn_prey <- knn_reg$pred
perf_summary_reg[2,] <- perf_eval_reg(tst_data$RegY, knn_prey)
perf_summary_reg
```

✓ MAE is decreased by 1.7%p, MAPE is decreased by 7.5%p compared to MLR

```
> perf_summary_reg
          RMSE      MAE     MAPE
MLR  10.449259 8.166065 30.85209
k-NN  8.452958 6.462481 23.30499
ANN   0.000000 0.000000  0.00000
```

R Exercise: Regression

- Search the best number of hidden nodes

```
# Find the best number of hidden nodes in terms of BCR
# Candidate hidden nodes
nH <- seq(from=2, to=20, by=2)
# 5-fold cross validation index
val_idx <- sample(c(1:5), length(trn_idx), replace = TRUE, prob = rep(0.2,5))
val_perf <- matrix(0, length(nH), 4)
...
for (i in 1:length(nH)) {
    ...
    for (j in c(1:5)) {
        tmp_nnet <- nnet(RegY ~ ., data = tmp_trn_data, size = nH[i],
                         linout = TRUE, decay = 5e-4, maxit = 500)
...
```

✓ Perform the 5-fold cross validation by varying the number of hidden nodes from 2 to 20 with the step size of 2

✓ (Note) linout = TRUE option must be set to solve a regression problem

R Exercise: Regression

- Search the best number of hidden nodes

```
ordered_val_perf <- val_perf[order(val_perf[,3], decreasing = FALSE),]
colnames(ordered_val_perf) <- c("nH", "RMSE", "MAE", "MAPE")
ordered_val_perf

# Find the best number of hidden node
best_nH <- ordered_val_perf[1,1]
```

```
> ordered_val_perf
      nH     RMSE      MAE     MAPE
 [1,] 14 7.258657 5.028841 17.65126
 [2,] 20 6.933016 5.090388 17.76599
 [3,] 12 6.952557 5.093377 18.49242
 [4,] 16 6.975732 5.152586 18.37121
 [5,] 10 6.839858 5.161052 18.68690
 [6,]  8 7.298235 5.253384 18.67945
 [7,] 18 7.966798 5.494715 20.98212
 [8,]  4 7.290364 5.613038 19.26955
 [9,]  6 7.325283 5.643885 19.42178
[10,]  2 9.277461 6.734610 22.96352
```

R Exercise: Regression

- Training and Evaluating ANN

```
# Test the model and compare the performance
ann_prey <- predict(best_nnet, tst_data[,-n_var])
perf_summary_reg[3,] <- perf_eval_reg(tst_data$RegY, ann_prey)
perf_summary_reg
```

✓ ANN resulted in the lowest error rate among the three regression algorithms

```
> perf_summary_reg
          RMSE      MAE     MAPE
MLR  10.449259 8.166065 30.85209
k-NN  8.452958 6.462481 23.30499
ANN   7.056971 4.950896 16.08717
```


ANY
questions ?