HW5 - Neural Network

Olivia Samples

7/13/2020

1 Homework

For the homework, we will use a dataset from https://www.kaggle.com/c/titanic/data. Using this dataset from the Titanic shipwreck you will predict who will survive and who will not. The data set is available on canvas with the name -train.csv" and -test.csv“.

2.1 Dataset description

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships. One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class. In this challenge, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply neural network to predict which passengers survived the tragedy.

VARIABLE DESCRIPTIONS:

1. Survival Survival (0 = No; 1 = Yes)
2. pclass Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
3. name Name
4. sex Sex
5. age Age
6. sibsp Number of Siblings/Spouses Aboard
7. parch Number of Parents/Children Aboard
8. ticket Ticket Number
9. fare Passenger Fare
10. cabin Cabin
11. embarked Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

SPECIAL NOTES:

Pclass is a proxy for socio-economic status (SES) 1st = Upper; 2nd = Middle; 3rd = Lower Age is in Years; Fractional if Age less than One (1) If the Age is Estimated, it is in the form xx.5 With respect to the family relation variables (i.e. sibsp and parch) some relations were ignored. The following are the definitions used for sibsp and parch.

Sibling: Brother, Sister, Stepbrother, or Stepsister of Passenger Aboard Titanic Spouse: Husband or Wife of Passenger Aboard Titanic (Mistresses and FiancesvIgnored) Parent: Mother or Father of Passenger Aboard Titanic Child: Son, Daughter, Stepson, or Stepdaughter of Passenger Aboard Titanic Other family relatives excluded from this study include cousins, nephews/nieces, aunts/uncles, and in-laws. Some children travelled only with a nanny, therefore parch=0 for them. As well, some travelled with very close friends or neighbors in a village, however, the de_nitions do not support such relations.

1.2 Your tasks

The training and the test data will be provided in canvas, look for tit-train.csv and tit-test.csv Start by cleaning the data (everything should be number,except the target variable. Also check to see if there is any missing value, if yes, and the missing is less than 60%, imputation for missing value (20 points). Split tit-train.csv data into two sets (training and testing) Using the training split, train and tweak a neural network to produce the best accuracy you can achieve (would your accuracy change if the data is scaled and centered?). Make sure you search for the best size and the best decay. (20 points) Use the testing split to create a confusion matrix (10 points) Once you are satisfied, test your neural network with the tit-test.csv (10 points) How many will survive? How many will not survive? Give a final count (20 points) Submit your work using RMarkdown including commends and output, as well as your statement. (20 points)

2 Sources

https://cran.r-project.org/web/packages/nnet/nnet.pdf https://cran.r-project.org/web/packages/caret/caret.pdf https://www.kaggle.com/

3 Solution

```
rm(list=ls())
library(nnet)
```

Read the csv files

```
train <- read.csv(file='tit-train (1).csv', head=TRUE, sep=",")
test <- read.csv(file='tit-test.csv', head=TRUE, sep=",")
```

Now we will clean our data.

```
coltype = function(data){
     sapply(data, class)
}
coltype(train)
```

```
## PassengerId    Survived      Pclass        Name         Sex         Age
##   "integer"   "integer"   "integer"    "factor"    "factor"   "numeric"
##       SibSp       Parch      Ticket        Fare       Cabin    Embarked
##   "integer"   "integer"    "factor"   "numeric"    "factor"    "factor"
```

```
cleandata = function(data){
        cleandata = data[, c(2, 5:7, 9)]
        cleandata = lapply(cleandata, as.numeric)
        numEmbarked = model.matrix(~Embarked, data = data)
        ##for test data to be the same
        if (colnames(numEmbarked)[2] == "EmbarkedC"){
                numEmbarked = numEmbarked[,c(1,3,4)]
```

```
                }
        numSex = model.matrix(~Sex, data = data)
        cleandata = data.frame(cleandata, numEmbarked, numSex)
        cleandata = cleandata[, c(1:5, 7:8, 10)]
        return(cleandata)
}

trainwotarget = train[,-c(2)]
Target = as.factor(train$Survived)
trainclean = data.frame(cleandata(trainwotarget), Target)
coltype(trainclean)
```

```
##    Pclass       Age     SibSp     Parch      Fare EmbarkedQ EmbarkedS   Sexmale
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
##    Target
##  "factor"
```

So we have all numeric data along with our target variable, "Survived". We removed "PassengerId", "Name", "Ticket", and "Cabin" variables because they were mostly all unique. We converted the originally categorical "Embarked" and "Sex" variable to four different numeric columns.

Now we will determine the number of null variables.

```
colna = function(data){
        null = is.na.data.frame(data)
        colnames(null)
        for (i in 1:length(data)){
                print(sum(null[,i]))
        }
        percentage = sum(null[,2])/NROW(null[,2])
        return(percentage)
}

colna(trainclean)
```

```
## [1] 0
## [1] 177
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
```

```
## [1] 0.1986532
```

As you can see, the "Age" category has 177 null values. This is 20% of the dataset, so we will imputate for missing values.

```
library(mice)
```

```
##
## Attaching package: 'mice'
```

```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
imputation = function(data, m=5, method = "pmm", maxit = 50, seed = 500){
        imputed = mice(data = data, m = m, method = method, maxit = maxit, seed = seed)
        mydata = as.data.frame(complete(imputed, 1))
        return(mydata)
}
imptrain = imputation(trainclean)
```

Here, m refers to number of data sets imputed, maxit refers to number of iterations to impute missing values, and method = "pmm" refers to the Predictive Mean Matching method for numeric variables.

```
colna(imptrain)
```

```
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
```

```
## [1] 0
```

```
coltype(imptrain)
```

```
##    Pclass       Age     SibSp     Parch      Fare EmbarkedQ EmbarkedS   Sexmale
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
##    Target
##  "factor"
```

As you can see, there are no longer any null values and all of the data is still numeric except for the target variable. Hence, our data is clean.

We will do this same process for the test data.

```
coltype(test)
```

```
## PassengerId      Pclass        Name         Sex         Age       SibSp
##   "integer"   "integer"    "factor"    "factor"   "numeric"   "integer"
##       Parch      Ticket        Fare       Cabin    Embarked
##   "integer"    "factor"   "numeric"    "factor"    "factor"
```

```
coltype(cleandata(test))
```

```
##    Pclass       Age     SibSp     Parch      Fare EmbarkedQ EmbarkedS   Sexmale
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
```

```
colna(cleandata(test))
```

```
## [1] 0
## [1] 86
## [1] 0
## [1] 0
## [1] 1
## [1] 0
## [1] 0
## [1] 0
```

```
## [1] 0.2057416
```

```
imptest = imputation(cleandata(test))
```

```
colna(imptest)
```

```
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0
```

```
## [1] 0
```

```
coltype(imptest)
```

```
##    Pclass       Age     SibSp     Parch      Fare EmbarkedQ EmbarkedS   Sexmale
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
```

Now we have to split the training data into two sets of its own: the training data and the test data.

```
set.seed(101)
t1 <- sample(1:nrow(imptrain), nrow(imptrain)*0.8)
train.xy <- imptrain[t1,]
test.x <- subset(imptrain[-t1,], select = -Target)
test.y <- imptrain[-t1,]$Target
```

Using the training split, we will train and tweak a neural network to produce the best accuracy we can achieve.

We will search for the best size and the best decay.

Now we are ready to start training our neural network

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
grid <- expand.grid(size=seq(from = 5, to = 17, by = 1), decay=c(0,0.01,0.1,1))
model.nn<-train(Target ~., data=train.xy, method="nnet", tuneGrid=grid,skip=FALSE,linout=FALSE)
model.nn
```

(Would your accuracy change if the data is scaled and centered?). From the mathematical point of view, the pre-processing of data for a neural network is not needed. However, for optimization purposes, it can be useful, so there is a chance the tuning process for the best size and decay would be improved.

Now we will use the testing split to create a confusion matrix.

To evaluate our neural network we need to test it with data test to see the predicted classes

```
#Here, we are using our optimized size and decay to obtain the below accuracy with our model.
model.nn$bestTune
```

```
##   size decay
## 3    5   0.1
```

```
test.pred <- predict(model.nn$finalModel, test.x, type="class")
cm <- confusionMatrix(as.factor(test.pred), test.y)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 104  16
##          1  14  45
##
##                Accuracy : 0.8324
##                  95% CI : (0.7695, 0.884)
##     No Information Rate : 0.6592
##     P-Value [Acc > NIR] : 1.816e-07
##
##                   Kappa : 0.624
##
##  Mcnemar's Test P-Value : 0.8551
##
##             Sensitivity : 0.8814
##             Specificity : 0.7377
##          Pos Pred Value : 0.8667
##          Neg Pred Value : 0.7627
##              Prevalence : 0.6592
##          Detection Rate : 0.5810
##    Detection Prevalence : 0.6704
##       Balanced Accuracy : 0.8095
##
##        'Positive' Class : 0
##
```

Now, we will test our neural network with the tit-test.csv

```
test.pred2 <- predict(model.nn$finalModel, imptest, type="class")
#survival percentage of tit-test.csv model prediction data
sum(as.numeric(test.pred2))/NROW(test.pred2)
```

```
## [1] 0.3779904
```

```
#survival percentage of tit_train.csv model prediction data
sum(as.numeric(test.pred))/NROW(test.pred)
```

```
## [1] 0.3296089
```

```
#survival percentage of tit_train.csv testing data
sum(as.numeric(as.character(test.y)))/NROW(test.y)
```

```
## [1] 0.3407821
```

Here you can see that our survival percentage on the tit-test.csv model prediciton data is in between the other two percentages we have available. Hence, we can concur that the model is relatively accurate.

How many will survive? How many will not survive?

```
#As for our trained model, we are 95% confident that our model will accurately predict the survivors ac
cm$overall[1]
```

```
##  Accuracy
```

```
## 0.8324022
```

```
#Our model can accurately predict the number of survivors accurately
cm$byClass[1]
```

```
## Sensitivity
##   0.8813559
```

```
# of the time and can accurately predict the number of non-survivors accurately
cm$byClass[2]
```

```
## Specificity
##   0.7377049
```

```
# of the time.
```

As for the passengers in the tit-test data, the survival count is below.

```
#survived
surv = sum(as.numeric(test.pred2))
surv
```

```
## [1] 158
```

```
#did not survive
nsurv = NROW(test.pred2) - surv
nsurv
```

```
## [1] 260
```