

Practicum 3

Jordan Lian

4/20/2021

```
library(tidyverse)
```

Problem 1 (60 pts)

1. (0 pts) Download the data set on customer credit data (<https://www.apispreadsheets.com/datasets>). The description of each column can be found in the data set explanation below.

```
origin_credit <- read.csv('german_credit_data_dataset.csv')
credit <- origin_credit
head(credit)
```

```
##   checking_account_status duration credit_history purpose credit_amount savings
## 1                      A11         6          A34    A43         1169      A65
## 2                      A12        48          A32    A43         5951      A61
## 3                      A14        12          A34    A46         2096      A61
## 4                      A11        42          A32    A42         7882      A61
## 5                      A11        24          A33    A40         4870      A61
## 6                      A14        36          A32    A46         9055      A65
##   present_employment installment_rate personal other_debtors present_residence
## 1                      A75              4          A93          A101              4
## 2                      A73              2          A92          A101              2
## 3                      A74              2          A93          A101              3
## 4                      A74              2          A93          A103              4
## 5                      A73              3          A93          A101              4
## 6                      A73              2          A93          A101              4
##   property age other_installment_plans housing existing_credits job dependents
## 1    A121  67                      A143  A152              2 A173              1
## 2    A121  22                      A143  A152              1 A173              1
## 3    A121  49                      A143  A152              1 A172              2
## 4    A122  45                      A143  A153              1 A173              2
## 5    A124  53                      A143  A153              2 A173              2
## 6    A124  35                      A143  A153              1 A172              2
##   telephone foreign_worker customer_type
## 1    A192          A201              1
## 2    A191          A201              2
## 3    A191          A201              1
## 4    A191          A201              1
## 5    A191          A201              2
## 6    A192          A201              1
```

2. (0 pts) Build an R Notebook named DA5030.P3-1.LastName.Rmd, where LastName is your last name.
3. (0 pts) Explore the data set as you see fit and that allows you to get a sense of the data and get comfortable with it.

I decided to change the customer_type variables, where 1 was good, and 2 was bad. I changed to 0 and 1, where 0 was bad, and 1 was good (FALSE vs TRUE). I felt that this was a more logical thing to do for the data. Additionally, it would be easy to classify, where any prediction greater than 0.5 would round up to 1, and vice versa for 0.

```
# Replace customer_type == 2 with 0
credit$customer_type[credit$customer_type == 2] <- 0

# Check Unique Values
unique(credit$customer_type)
```

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

4. (10 pts) Encode the categorical variables using one-hot encoding. You must do this manually and may not rely on model functions. You may choose a subset of variables. You may simplify the data set by eliminating up to four categorical features. You may also simplify the category levels for checking_account_status and present_employment to Boolean. Others you may reduce the number of levels.

Data elimination and simplification

I decided to eliminate purpose, other_debtors, property, and other_installment_plans, because I felt that those factors were the least relevant to predicting customer_type. I also simplified checking_account_status and present_employment to Boolean.

```
# Eliminate 4 categorical features
credit = subset(credit, select = -c(purpose, other_debtors, property,
                                   other_installment_plans))

# Simplify checking_account status to Boolean
credit$checking_account_status[credit$checking_account_status == 'A14'] <- F
credit$checking_account_status[credit$checking_account_status != F] <- T

# present_employment to Boolean
credit$present_employment[credit$present_employment == 'A71'] <- F
credit$present_employment[credit$present_employment != F] <- T
```

One-hot encoding

I found the unique values for each column, and then created columns for each unique value. I then removed the original column values itself. I divided the coding chunks for each column, so I could run them one at a time. Lastly, I changed the type of variables to factors for the prediction model.

1. Checking Account Status

```
# Unique Values  
unique(credit$checking_account_status)
```

```
## [1] "TRUE" "FALSE"
```

```
# True  
credit$check_account_T[credit$checking_account_status == T] <- 1  
credit$check_account_T[credit$checking_account_status == F] <- 0  
  
# False  
credit$check_account_F[credit$checking_account_status == F] <- 1  
credit$check_account_F[credit$checking_account_status == T] <- 0  
  
# Remove Column  
credit = subset(credit, select = -checking_account_status)
```

2. Credit History

```
# Unique Values  
unique(credit$credit_history)
```

```
## [1] "A34" "A32" "A33" "A30" "A31"
```

```
# A30  
credit$history_A30[credit$credit_history == 'A30'] <- 1  
credit$history_A30[credit$credit_history != 'A30'] <- 0  
  
# A31  
credit$history_A31[credit$credit_history == 'A31'] <- 1  
credit$history_A31[credit$credit_history != 'A31'] <- 0  
  
# A32  
credit$history_A32[credit$credit_history == 'A32'] <- 1  
credit$history_A32[credit$credit_history != 'A32'] <- 0  
  
# A33  
credit$history_A33[credit$credit_history == 'A33'] <- 1  
credit$history_A33[credit$credit_history != 'A33'] <- 0  
  
# A34  
credit$history_A34[credit$credit_history == 'A34'] <- 1  
credit$history_A34[credit$credit_history != 'A34'] <- 0  
  
# Remove  
credit = subset(credit, select = -credit_history)
```

3. Savings

```
# Unique Values  
unique(credit$savings)
```

```
## [1] "A65" "A61" "A63" "A64" "A62"
```

```
# A61
credit$savings_A61[credit$savings == 'A61'] <- 1
credit$savings_A61[credit$savings != 'A61'] <- 0

# A62
credit$savings_A62[credit$savings == 'A62'] <- 1
credit$savings_A62[credit$savings != 'A62'] <- 0

# A63
credit$savings_A63[credit$savings == 'A63'] <- 1
credit$savings_A63[credit$savings != 'A63'] <- 0

# A64
credit$savings_A64[credit$savings == 'A64'] <- 1
credit$savings_A64[credit$savings != 'A64'] <- 0

# A65
credit$savings_A65[credit$savings == 'A65'] <- 1
credit$savings_A65[credit$savings != 'A65'] <- 0

# Remove
credit = subset(credit, select = -savings)
```

4. Present Employment Status

```
# Unique Values
unique(credit$present_employment)
```

```
## [1] "TRUE" "FALSE"
```

```
# A71
credit$employment_A71[credit$present_employment == 'A71'] <- 1
credit$employment_A71[credit$present_employment != 'A71'] <- 0

# A72
credit$employment_A72[credit$present_employment == 'A72'] <- 1
credit$employment_A72[credit$present_employment != 'A72'] <- 0

# A73
credit$employment_A73[credit$present_employment == 'A73'] <- 1
credit$employment_A73[credit$present_employment != 'A73'] <- 0

# A74
credit$employment_A74[credit$present_employment == 'A74'] <- 1
credit$employment_A74[credit$present_employment != 'A74'] <- 0

# A75
credit$employment_A75[credit$present_employment == 'A75'] <- 1
credit$employment_A75[credit$present_employment != 'A75'] <- 0
```

```
# Remove
credit = subset(credit, select = -present_employment)
```

5. Personal

```
# Unique Values
unique(credit$personal)
```

```
## [1] "A93" "A92" "A91" "A94"
```

```
# A91
credit$personal_A91[credit$personal == 'A91'] <- 1
credit$personal_A91[credit$personal != 'A91'] <- 0

# A92
credit$personal_A92[credit$personal == 'A92'] <- 1
credit$personal_A92[credit$personal != 'A92'] <- 0

# A93
credit$personal_A93[credit$personal == 'A93'] <- 1
credit$personal_A93[credit$personal != 'A93'] <- 0

# A94
credit$personal_A94[credit$personal == 'A94'] <- 1
credit$personal_A94[credit$personal != 'A94'] <- 0

# Remove
credit = subset(credit, select = -personal)
```

6. Housing

```
# Unique Values
unique(credit$housing)
```

```
## [1] "A152" "A153" "A151"
```

```
# A151
credit$housing_A151[credit$housing == 'A151'] <- 1
credit$housing_A151[credit$housing != 'A151'] <- 0

# A152
credit$housing_A152[credit$housing == 'A152'] <- 1
credit$housing_A152[credit$housing != 'A152'] <- 0

# A153
credit$housing_A153[credit$housing == 'A153'] <- 1
credit$housing_A153[credit$housing != 'A153'] <- 0

# Remove
credit = subset(credit, select = -housing)
```

7. Job

```
# Unique Values  
unique(credit$job)
```

```
## [1] "A173" "A172" "A174" "A171"
```

```
# A171  
credit$job_A171[credit$job == 'A171'] <- 1  
credit$job_A171[credit$job != 'A171'] <- 0  
  
# A172  
credit$job_A172[credit$job == 'A172'] <- 1  
credit$job_A172[credit$job != 'A172'] <- 0  
  
# A173  
credit$job_A173[credit$job == 'A173'] <- 1  
credit$job_A173[credit$job != 'A173'] <- 0  
  
# A174  
credit$job_A174[credit$job == 'A174'] <- 1  
credit$job_A174[credit$job != 'A174'] <- 0  
  
# Remove  
credit = subset(credit, select = -job)
```

8. Telephone

```
# Unique Values  
unique(credit$telephone)
```

```
## [1] "A192" "A191"
```

```
# A191  
credit$phone_A191[credit$telephone == 'A191'] <- 1  
credit$phone_A191[credit$telephone != 'A191'] <- 0  
  
# A192  
credit$phone_A192[credit$telephone == 'A192'] <- 1  
credit$phone_A192[credit$telephone != 'A192'] <- 0  
  
# Remove  
credit = subset(credit, select = -telephone)
```

9. Foreign Worker

```
# Unique Values  
unique(credit$foreign_worker)
```

```
## [1] "A201" "A202"
```

```

# A201
credit$foreign_A201[credit$foreign_worker == 'A201'] <- 1
credit$foreign_A201[credit$foreign_worker != 'A201'] <- 0

# A202
credit$foreign_A202[credit$foreign_worker == 'A202'] <- 1
credit$foreign_A202[credit$foreign_worker != 'A202'] <- 0

# Remove
credit = subset(credit, select = -foreign_worker)

```

5. (20 pts) Build a classification model using a neural networks that predicts if a customer has a good or bad credit risk (column customer_type). Now build a support vector machines classifier and compare your results. You may choose the package for the ANN and SVM implementation.

For this model, I used the neuralnet library/function for this model rather than the nnet library for ANN implementation. I found this easier to work with. First I split up the data into training/validation data, and then I went forward with creating the model. I used some trial and error to determine the hidden nodes.

Training/Validation Dataset

```

# Random seed
set.seed(1234)

# Sample dataset
sample <- sample.int(n = nrow(credit), size = floor(0.75*nrow(credit)), replace = F)
training <- credit[sample, ]
validation <- credit[-sample, ]

```

Create and visualize model

```

# Create model
library(neuralnet)

```

```
## Warning: package 'neuralnet' was built under R version 4.0.4
```

```
##
```

```
## Attaching package: 'neuralnet'
```

```
## The following object is masked from 'package:dplyr':
```

```
##
```

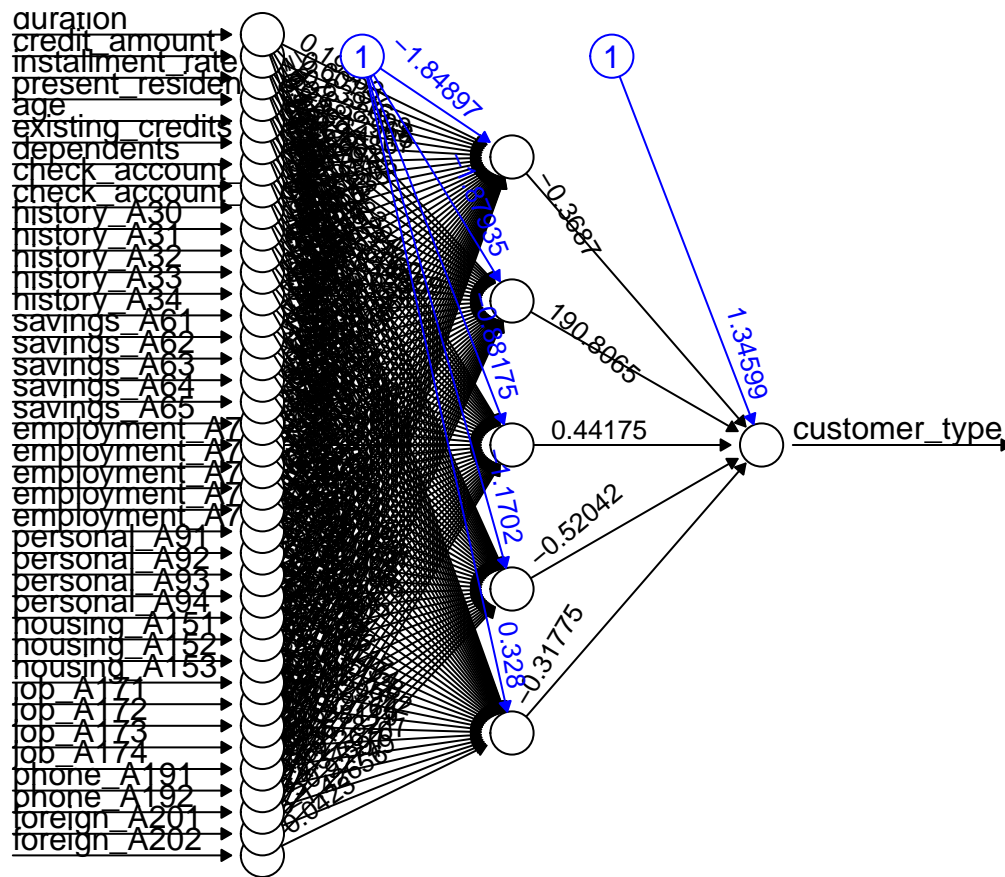
```
## compute
```

```
credit_model <- neuralnet(customer_type ~., hidden=5, data = training, linear.output = FALSE)
```

```

# Visualize network topology
plot(credit_model, rep="best")

```



Store Prediction in data frame

I stored the prediction as a new column in the data frame so I could evaluate the model's accuracy, precision, and recall (better explained in Question 7)

```
# Get predictions
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.3
```

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

```
##
```

```
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
```

```
##
```

```
## lift
```

```
prediction <- predict(credit_model, validation)
```

```
# Convert to 0 and 1
```

```
prediction[prediction >= 0.5, ] <- 1
```

```
prediction[prediction < 0.5, ] <- 0

# Store in data frame
prediction <- as.integer(prediction)
validation$cus_prediction <- prediction
```

Results

Overall these results weren't bad. A little under 70% accuracy and precision along with a 100% recall. Accuracy and precision needs to be better though in my opinion.

```
# True Positive
True_Pos <- count(validation[validation$customer_type == 1 &
                             validation$cus_prediction == 1, ])

# True Negative
True_Neg <- count(validation[validation$customer_type == 0 &
                             validation$cus_prediction == 0, ])

# False Positive
False_Pos <- count(validation[validation$customer_type == 0 &
                              validation$cus_prediction == 1, ])

# False Negative
False_Neg <- count(validation[validation$customer_type == 1 &
                              validation$cus_prediction == 0, ])

# Total
Total <- count(validation)

# Accuracy
(True_Pos + True_Neg) / Total
```

```
##      n
## 1 0.692
```

```
# Precision
True_Pos / (True_Pos + False_Pos)
```

```
##      n
## 1 0.692
```

```
# Recall
True_Pos / (True_Pos + False_Neg)
```

```
##      n
## 1 1
```

Create support vector machine (SVM) classifier

I used the e1071 library this time, where I made the kernel radial. I also used some trial and error to determine the cost.

```
# SVM Classifier  
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.0.3
```

```
svmfit = svm(customer_type ~ ., data = training, kernel = "radial", cost = 10,  
             scale = FALSE)  
print(svmfit)
```

```
##  
## Call:  
## svm(formula = customer_type ~ ., data = training, kernel = "radial",  
##      cost = 10, scale = FALSE)  
##  
##  
## Parameters:  
##   SVM-Type:  eps-regression  
## SVM-Kernel:  radial  
##      cost:   10  
##      gamma:  0.02564103  
##      epsilon: 0.1  
##  
##  
## Number of Support Vectors: 750
```

```
svm_pred <- predict(svmfit, validation)  
  
# Convert to 0 and 1  
svm_pred[svm_pred >= 0.5] <- 1  
svm_pred[svm_pred < 0.5] <- 0  
  
# Store in data frame  
svm_pred <- as.integer(svm_pred)  
validation$SVM_prediction <- svm_pred
```

SVM Results

Overall these results were similar to the neural network model. A little under 70% accuracy and precision along with a 97% recall. Accuracy and precision needs to be better though in my opinion just like the neural network model. Recall though has been good so far.

```
# True Positive  
True_Pos <- count(validation[validation$customer_type == 1 &  
                           validation$SVM_prediction == 1, ])  
  
# True Negative
```

```

True_Neg <- count(validation[validation$customer_type == 0 &
                             validation$SVM_prediction == 0, ])

# False Positive
False_Pos <- count(validation[validation$customer_type == 0 &
                              validation$SVM_prediction == 1, ])

# False Negative
False_Neg <- count(validation[validation$customer_type == 1 &
                              validation$SVM_prediction == 0, ])

# Total
Total <- count(validation)

# Accuracy
(True_Pos + True_Neg) / Total

```

```

##           n
## 1 0.684

```

```

# Precision
True_Pos / (True_Pos + False_Pos)

```

```

##           n
## 1 0.6942149

```

```

# Recall
True_Pos / (True_Pos + False_Neg)

```

```

##           n
## 1 0.9710983

```

6. (20 pts) Build another classification model using ANN that predicts if a bank customer have more than 500 DM in their savings using the other features. Again, compare the results with SVM.

First I had to encode the customer type and decode the savings account columns from the previous data frame. Then I was able to split the dataset and create the two models.

Encode Customer Type

```

# New data frame
savings <- credit

# Unique Values
unique(savings$customer_type)

```

```

## [1] 1 0

```

```

# Bad Customer
savings$customer_bad[savings$customer_type == 0] <- 1
savings$customer_bad[savings$customer_type != 0] <- 0

# Good Customer
savings$customer_good[savings$customer_type == 1] <- 1
savings$customer_good[savings$customer_type != 1] <- 0

# Remove
savings = subset(savings, select = -customer_type)

```

Decode savings back into 1 column

```

# Remove rows with A65
savings <- savings[savings$savings_A65 != 1, ]

# Initialize (A61, A62)
savings$savings_status <- 0

# A63, A64: > 500 DM
savings$savings_status[savings$savings_A63 == 1 | savings$savings_A64 == 1] <- 1

# Remove Columns
savings = subset(savings, select = -c(savings_A61, savings_A62, savings_A63, savings_A64,
savings_A65))

```

Split Dataset

```

# Random seed
set.seed(1234)

# Sample dataset
sample <- sample.int(n = nrow(savings), size = floor(0.75*nrow(savings)), replace = F)
sav_training <- savings[sample, ]
sav_validation <- savings[-sample, ]

```

Savings ANN Model

I used the nnet library and function to create this model. Based on trial and error, I determined the best size, decay, and maximum iterations.

```

# ANN Classifier
library(nnet)
sav_training$savings_status <- as.factor(sav_training$savings_status)
ANN <- nnet(savings_status ~., data = sav_training, size=5, decay=1.0e-5, maxit=500)

## # weights: 191
## initial value 552.590262

```

```
## final value 227.647912
## converged
```

```
ANN
```

```
## a 36-5-1 network with 191 weights
## inputs: duration credit_amount installment_rate present_residence age existing_credits dependents ch
## output(s): savings_status
## options were - entropy fitting decay=1e-05
```

```
# Prediction
ANN_pred <- predict(ANN, sav_validation, type="class")

# Store in data frame
ANN_pred <- as.integer(ANN_pred)
sav_validation$ANN_pred <- ANN_pred
```

Savings ANN Results

Accuracy is good at 82%, but there is no precision value (undefined) and a recall value of 0. That indicates that there was no positive predictions at all. I'm not sure what to think of that. I will say most of the validation dataset then probably had negative diagnoses given the high accuracy ratings.

```
# True Positive
True_Pos <- count(sav_validation[sav_validation$savings_status == 1 &
                               sav_validation$ANN_pred == 1, ])

# True Negative
True_Neg <- count(sav_validation[sav_validation$savings_status == 0 &
                               sav_validation$ANN_pred == 0, ])

# False Positive
False_Pos <- count(sav_validation[sav_validation$savings_status == 0 &
                               sav_validation$ANN_pred == 1, ])

# False Negative
False_Neg <- count(sav_validation[sav_validation$savings_status == 1 &
                               sav_validation$ANN_pred == 0, ])

# Total
Total <- count(sav_validation)

# Accuracy
(True_Pos + True_Neg) / Total
```

```
##           n
## 1 0.8243902
```

```
# Precision
True_Pos / (True_Pos + False_Pos)
```

```
##      n
## 1 NaN
```

```
# Recall
True_Pos / (True_Pos + False_Neg)
```

```
##      n
## 1 0
```

Savings SVM Model

I used the same packages used in Question 5 to create the SVM model.

```
# SVM Classifier
sav_svm = svm(savings_status ~ ., data = sav_training, kernel = "radial", cost = 10,
              scale = FALSE)

print(sav_svm)
```

```
##
## Call:
## svm(formula = savings_status ~ ., data = sav_training, kernel = "radial",
##      cost = 10, scale = FALSE)
##
##
## Parameters:
##   SVM-Type:  C-classification
##   SVM-Kernel: radial
##         cost: 10
##
## Number of Support Vectors: 612
```

```
sav_svm_pred <- predict(sav_svm, sav_validation, type = "class")

# Store in data frame
sav_svm_pred <- as.integer(sav_svm_pred)
sav_validation$SVM_prediction <- sav_svm_pred
```

Savings SVM Results

This definitely has the worst results, with an accuracy and precision of 17%. Recall however was 100%. That must mean that there were no false negatives and a ton of false positives. However the accuracy speaks for itself.

```
# True Positive
True_Pos <- count(sav_validation[sav_validation$savings_status == 1 &
                                sav_validation$SVM_prediction == 1, ])

# True Negative
True_Neg <- count(sav_validation[sav_validation$savings_status == 0 &
                                sav_validation$SVM_prediction == 0, ])
```

```

# False Positive
False_Pos <- count(sav_validation[sav_validation$savings_status == 0 &
                                sav_validation$SVM_prediction == 1, ])

# False Negative
False_Neg <- count(sav_validation[sav_validation$savings_status == 1 &
                                sav_validation$SVM_prediction == 0, ])

# Total
Total <- count(sav_validation)

# Accuracy
(True_Pos + True_Neg) / Total

```

```

##           n
## 1 0.1756098

```

```

# Precision
True_Pos / (True_Pos + False_Pos)

```

```

##           n
## 1 0.1756098

```

```

# Recall
True_Pos / (True_Pos + False_Neg)

```

```

##           n
## 1 1

```

7. (10 pts) Calculate accuracy, precision, and recall for both models in part 5 and 6. See this article (<https://medium.com/@shrutisaxena0617/precision-vs-recall-386cf9f89488>) to understand how to calculate these metrics or consult chapter 10 in the text book.

I calculated all three metrics in the sections above using the following formulas:

$$Accuracy = \frac{TruePositive + TrueNegative}{Total}$$

$$Precision = \frac{TruePositive}{ActualResults}$$

$$Recall = \frac{TruePositive}{PredictedResults}$$

$$ActualResults = TruePositive + FalsePositive$$

$$PredictedResults = TruePositive + FalseNegative$$

Given these formulas, I got all of the individual values using the count() functions with a condition, and then I calculated all 3 metrics.