Code

DA5030: Practicum 3

Problem 1

1. Download the data set on customer credit data (german_credit_data_dataset.csv). The description of each column can be found in the data set explanation below.

CSV file is downloaded manually due to privacy concern on Northeastern University website

2. Build an R Notebook named DA5030.P3.LastName.Rmd, where LastName is your last name

3. Explore the data set as you see fit and that allows you to get a sense of the data and get comfortable with it.

Hide

```
# Read CSV file from Downloads
German_CreditDF<-read.csv("german_credit_data_dataset.csv")

# Check dimension and properties
str(German_CreditDF)
```

```
'data.frame':	1000 obs. of  21 variables:
 $ checking_account_status: Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1 4 4 2
4 2 ...
 $ duration               : int  6 48 12 42 24 36 24 36 12 30 ...
 $ credit_history         : Factor w/ 5 levels "A30","A31","A32",..: 5 3 5 3 4 3 3 3
3 5 ...
 $ purpose                : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4 1 8 4
2 5 1 ...
 $ credit_amount          : num  1169 5951 2096 7882 4870 ...
 $ savings                : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1 5 3 1
4 1 ...
 $ present_employment     : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3 3 5 3
4 1 ...
 $ installment_rate       : num  4 2 2 2 3 2 3 2 2 4 ...
 $ personal               : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3 3 3 3
1 4 ...
 $ other_debtors          : Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1 1 1 1
...
 $ present_residence      : num  4 2 3 4 4 4 4 2 4 2 ...
 $ property               : Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2 3 1 3
...
 $ age                    : num  67 22 49 45 53 35 53 35 61 28 ...
 $ other_installment_plans: Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3 3 3 3
...
 $ housing                : Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2 1 2 2
...
 $ existing_credits       : num  2 1 1 1 2 1 1 1 1 2 ...
 $ job                    : Factor w/ 4 levels "A171","A172",..: 3 3 2 3 3 2 3 4 2 4
...
 $ dependents             : int  1 1 2 2 2 2 1 1 1 1 ...
 $ telephone              : Factor w/ 2 levels "A191","A192": 2 1 1 1 1 2 1 2 1 1 ...
 $ foreign_worker         : Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1 1 1 ...
 $ customer_type          : int  1 2 1 1 2 1 1 1 1 2 ...
```

4. 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.

Categorized variables that will be converted into one-hot coding:

checking_account_status, credit_history, purpose, savings, present_employment, personal, other_debtors, property, other_installment_plans

Additional variables that might be used in the model:

exisiting_credits, job

APPLY ONE-HOT CODING FOR checking_account_status column:

Hide

```
# Simplify column checking_account_status category levels

# Check levels of checking_account_status
levels(German_CreditDF$checking_account_status)
```

```
[1] "A11" "A12" "A13" "A14"
```

Combine A11, A12 and A14 into 1 level. This is because the 3 status of checking account cover from 0 to less than 200 DM. This will be easier for us to determine whether a checking account contains 0 to 200 DM or more than 200 DM.

Thus this information will be changed as shown below:

A12 : 0 <= x < 200 DM

A13 : >= 200 DM

Hide

```
# Combine "A11", "A12" and "A14" to "A12"
levels(German_CreditDF$checking_account_status)<-c("A12", "A12", "A13", "A12")
```

Apply one-hot encoding to checking_account_status column to 0 and 1. Here, we determine 0 as status of checking account is 0 to 200DM and 1 as more than 200 DM:

Hide

```
# Apply one-hot encoding to checking_account_status
German_CreditDF$checking_account_status<-ifelse(German_CreditDF$checking_account_stat
us == "A12", 0, 1)
```

APPLY ONE-HOT CODING FOR present_employment column:

Hide

```
# Simplify column present_employment category levels

# Check levels of present_employment
levels(German_CreditDF$present_employment)
```

```
[1] "A71" "A72" "A73" "A74" "A75"
```

In present_employment column, we combine “A71” “A72” “A73” into “A74”. This means A74 contains employment range from unemployed to less than 7 years of employment.

Hide

```
# Combine "A71" "A72" "A73" into "A74"
levels(German_CreditDF$present_employment)<-c("A74", "A74", "A74", "A74", "A75")
```

Apply one-hot encoding to present_employment column to 0 and 1. Here, we determine 0 as present employment range is unemployed to < 7 years and 1 as more than 7 years of employment:

Hide

```
# Apply one-hot encoding to checking_account_status
German_CreditDF$present_employment<-ifelse(German_CreditDF$present_employment == "A74
", 0, 1)
```

APPLY ONE-HOT CODING FOR credit_history column:

Hide

```
# Simplify column credit_history category levels

# Check levels of credit_history
levels(German_CreditDF$credit_history)
```

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

A30 to A32: Represent no credit taken and credits paid back duly. Here we can combine them into a single level within A32

A33 to A34: Represent credit is in critical account and contain delay in paying off. Here we can combine them into single level within A34

Hide

```
# Combine A30 to A32 and A33 to A34
levels(German_CreditDF$credit_history)<-c("A32", "A32", "A32", "A34", "A34")
```

Hide

```
# Apply one-hot encoding to credit_history: A32 as 0, A34 as 1
German_CreditDF$credit_history<-ifelse(German_CreditDF$credit_history == "A32", 0, 1)
```

APPLY ONE-HOT CODING FOR purpose column:

Hide

```
# Simplify column purpose category levels

# Check levels of purpose
levels(German_CreditDF$purpose)
```

```
 [1] "A40"  "A41"  "A410" "A42"  "A43"  "A44"  "A45"  "A46"  "A48"  "A49"
```

A40, A41, A42, A43, A44, A45, A410: These levels represent common appliances thus we put them within 1 level. This also includes “others”, which could be groceries etc. We put this in one level as A44

A46, A48, A49: These levels represent common investment needs. This includes business, education, and retraining. Thus, we will put this in one level as A49

Hide

```
# Combine levels into 2 levels
levels(German_CreditDF$purpose)<-c("A44", "A44",  "A44", "A44",  "A44",  "A44",  "A44
",  "A49",  "A49",  "A49")
```

Hide

```
# Apply one-hot encoding to purpose: A44 as 0, A49 as 1
German_CreditDF$purpose<-ifelse(German_CreditDF$purpose == "A44", 0, 1)
```

APPLY ONE-HOT CODING FOR savings column:

Hide

```
# Simplify column savings category levels

# Check levels of savings
levels(German_CreditDF$savings)
```

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

A65, A61, A62 : Savings contain 0 to 500 DM (as one level with A62) A63, A64 : Savings contain >500 (as one level with A64)

Hide

```
# Combine levels into 2 levels
levels(German_CreditDF$savings)<-c("A62", "A62", "A64", "A64", "A62")
```

Hide

```
# Apply one-hot encoding to purpose: A62 as 0, A64 as 1
German_CreditDF$savings<-ifelse(German_CreditDF$savings == "A62", 0, 1)
```

APPLY ONE-HOT CODING FOR personal column:

Hide

```
# Simplify column personal category levels

# Check levels of personal
levels(German_CreditDF$personal)
```

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

A91, A93, A94 : All 3 variables contain male regardless of their marriage status (put into single level as A91)
A92 : Variable contains female and marriage status

Hide

```
# Combine levels into 2 levels: Female and Male
levels(German_CreditDF$personal)<-c("A91", "A92", "A91", "A91")
```

Hide

```
# Apply one-hot encoding to purpose: A91 as 0, A92 as 1
German_CreditDF$personal<-ifelse(German_CreditDF$personal == "A91", 0, 1)
```

APPLY ONE-HOT CODING FOR other_debtors column:

Hide

```
# Simplify column other_debtors category levels

# Check levels of other_debtors
levels(German_CreditDF$other_debtors)
```

```
[1] "A101" "A102" "A103"
```

A101: None A102, A103: Contain guarantor and co-applicant. Here, we combine as a single level as A102

Hide

```
# Combine levels into 2 levels: None and guarantor/co-applicant
levels(German_CreditDF$other_debtors)<-c("A101", "A102", "A102")
```

Hide

```
# Apply one-hot encoding to other_debtors: A101 as 0, A102 as 1
German_CreditDF$other_debtors<-ifelse(German_CreditDF$other_debtors == "A101", 0, 1)
```

APPLY ONE-HOT CODING FOR property column:

Hide

```
# Simplify column property category levels

# Check levels of property
levels(German_CreditDF$property)
```

```
[1] "A121" "A122" "A123" "A124"
```

A121, A122, A123 : Contain known properties (put as single level A123) A124 : No property

Hide

```
# Combine levels into 2 levels: None and guarantor/co-applicant
levels(German_CreditDF$property)<-c("A123", "A123", "A123", "A124")
```

Hide

```
# Apply one-hot encoding to property: A124 as 0, A123 as 1
German_CreditDF$property<-ifelse(German_CreditDF$property == "A124", 0, 1)
```

APPLY ONE-HOT CODING FOR other_installment_plans column:

Hide

```
# Simplify column other_installment_plans category levels

# Check levels of other_installment_plans
levels(German_CreditDF$other_installment_plans)
```

```
[1] "A141" "A142" "A143"
```

A141, A142 : Installment plans available “store” and “bank” (as one level with 141) A143 : None

Hide

```
# Combine levels into 2 levels: None and available plans
levels(German_CreditDF$other_installment_plans)<-c("A141", "A141", "A143")
```

Hide

```
# Apply one-hot encoding to other_installment_plans: A143 as 0, A141 as 1
German_CreditDF$other_installment_plans<-ifelse(German_CreditDF$other_installment_pla
ns == "A143", 0, 1)
```

APPLY ONE-HOT CODING FOR job column:

Hide

```
# Simplify column job category levels

# Check levels of job
levels(German_CreditDF$job)
```

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

A171, A172 : Contain both unemployed/unskilled/non-resident and unskilled/resident (as single level A172)
A173, A174 : Contain both skilled employees / highly qualified employees (as a single level A174)

Hide

```
# Combine levels into 2 levels: low skilled and highly skilled
levels(German_CreditDF$job)<-c("A172", "A172", "A174", "A174")
```

Hide

```
# Apply one-hot encoding to job: A172 as 0, A174 as 1
German_CreditDF$job<-ifelse(German_CreditDF$job== "A172", 0, 1)
```

Create a dataset containing only the variables that have been modified including some numerics for building model:

Hide

```
library(dplyr)
```

```


Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union
```

Hide

```
# Exclude 4 variables
GermanCredit<-German_CreditDF %>%
  select(-c(foreign_worker, telephone, dependents, housing))

# Check the dataset: All variables should be numerics/integers
str(GermanCredit)
```

```
'data.frame':	1000 obs. of  17 variables:
 $ checking_account_status: num  0 0 0 0 0 0 0 0 0 0 ...
 $ duration               : int  6 48 12 42 24 36 24 36 12 30 ...
 $ credit_history         : num  1 0 1 0 1 0 0 0 0 1 ...
 $ purpose                : num  0 0 1 0 0 1 0 0 0 0 ...
 $ credit_amount          : num  1169 5951 2096 7882 4870 ...
 $ savings                : num  0 0 0 0 0 0 1 0 1 0 ...
 $ present_employment     : num  1 0 0 0 0 0 1 0 0 0 ...
 $ installment_rate       : num  4 2 2 2 3 2 3 2 2 4 ...
 $ personal               : num  0 1 0 0 0 0 0 0 0 0 ...
 $ other_debtors          : num  0 0 0 1 0 0 0 0 0 0 ...
 $ present_residence      : num  4 2 3 4 4 4 4 2 4 2 ...
 $ property               : num  1 1 1 1 0 0 1 1 1 1 ...
 $ age                    : num  67 22 49 45 53 35 53 35 61 28 ...
 $ other_installment_plans: num  0 0 0 0 0 0 0 0 0 0 ...
 $ existing_credits       : num  2 1 1 1 2 1 1 1 1 2 ...
 $ job                    : num  1 1 0 1 1 0 1 1 0 1 ...
 $ customer_type          : int  1 2 1 1 2 1 1 1 1 2 ...
```

5. Build a classification model using an artificial neural networks (ANN) that predicts if a customer has a good or bad credit risk (column customer_type). Use one hidden layer and try to optimize number of hidden neurons in your ANN. Now build a support vector machines (SVM) classifier and compare your results. You may choose the package for the ANN and SVM implementation.

PRE-PROCESS DATA:

Hide

```
# Create Normalization Function
normalize <- function(x) {
    return((x - min(x)) / (max(x) - min(x)))
  }
```

Hide

```
# Apply GermanCredit with normalization
GermanCredit_norm<-as.data.frame(lapply(GermanCredit, normalize))

# Check Summary: Min should be 0 and Max. should be 0 (if successful)
summary(GermanCredit_norm$customer_type)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0     0.0     0.0     0.3     1.0     1.0
```

RANDOMLY ORDER THE DATASET & SPLIT INTO TRAINING AND TESTING

Hide

```
set.seed(123)

RandomRows<- sort(sample(nrow(GermanCredit_norm), nrow(GermanCredit_norm)*.75))
GermanCredit_train<-GermanCredit_norm[RandomRows,]
GermanCredit_test<-GermanCredit_norm[-RandomRows,]
```

IMPLEMENT ANN

Hide

```
#install.packages("neuralnet")
library(neuralnet)
```

```


Attaching package: 'neuralnet'

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

    compute
```

TRAIN SIMPLEST MULTILAYER FEEDFORWARD NETWORK WITH SINGLE HIDDEN NODE:

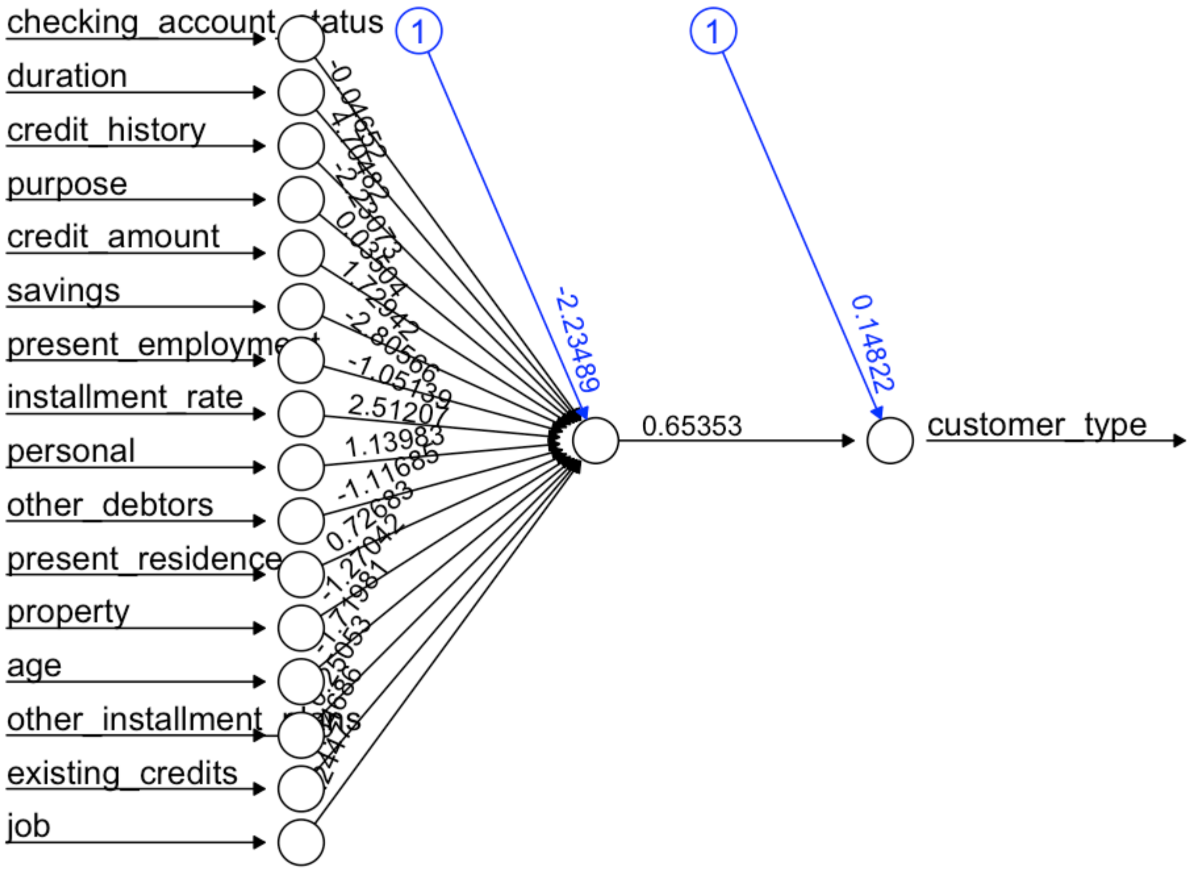
Hide

```
GermanCredit_model<-neuralnet(customer_type ~., data = GermanCredit_train)
```

VISUALIZE NETWORK TOPOLOGY

Hide

```
plot(GermanCredit_model)
```



Check the SSE of the model:

Hide

```
GermanCredit_model$result.matrix
```

```
                                              [,1]
error                                 6.790975e+01
reached.threshold                     9.715598e-03
steps                                 1.005000e+03
Intercept.to.1layhid1                -2.234890e+00
checking_account_status.to.1layhid1 -4.652378e-02
duration.to.1layhid1                  4.704820e+00
credit_history.to.1layhid1           -2.230735e+00
purpose.to.1layhid1                   3.503633e-02
credit_amount.to.1layhid1             1.729418e+00
savings.to.1layhid1                  -2.805660e+00
present_employment.to.1layhid1       -1.051388e+00
installment_rate.to.1layhid1          2.512074e+00
personal.to.1layhid1                  1.139830e+00
other_debtors.to.1layhid1            -1.116852e+00
present_residence.to.1layhid1         7.268254e-01
property.to.1layhid1                 -1.270420e+00
age.to.1layhid1                      -1.719811e+00
other_installment_plans.to.1layhid1  2.505311e-01
existing_credits.to.1layhid1          1.546863e+00
job.to.1layhid1                      -1.244699e+00
Intercept.to.customer_type            1.482247e-01
1layhid1.to.customer_type             6.535256e-01
```

Error Result : 67.9 Steps : 1005.000

Based on the result shown above, the error result is quite high scoring 67.9. To obtain better model, we must obtain lower error score and higher steps.

EVALUATE MODEL PERFORMANCE:

Hide

```
# Generate predictions on test dataset using compute ()
model_result<-compute(GermanCredit_model, GermanCredit_test[1:16])
```

Hide

```
# Apply net.result to check predicted values
predicted_customer_type<-model_result$net.result
```

PERFORM CONFUSION MATRIX: Classification Problem

Hide

```
library(caret)
```

```
Loading required package: lattice
Loading required package: ggplot2

Attaching package: ‘ggplot2’

The following objects are masked from ‘package:psych’:

    %+%, alpha

Registered S3 method overwritten by 'data.table':
  method           from
  print.data.table
```

CONVERT PROBABILITIES INTO BINARY CLASS:

Hide

```
scorePredicted_customer_type<-ifelse(predicted_customer_type >=.5, 1, 0)
scorePredicted_customer_type<-as.factor(scorePredicted_customer_type)
```

Hide

```
confusionMatrix(scorePredicted_customer_type, as.factor(GermanCredit_test$customer_type))
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 156  55
         1  13  26

               Accuracy : 0.728
                 95% CI : (0.6683, 0.7822)
    No Information Rate : 0.676
    P-Value [Acc > NIR] : 0.04402

                  Kappa : 0.2822

 Mcnemar's Test P-Value : 6.627e-07

            Sensitivity : 0.9231
            Specificity : 0.3210
         Pos Pred Value : 0.7393
         Neg Pred Value : 0.6667
             Prevalence : 0.6760
         Detection Rate : 0.6240
   Detection Prevalence : 0.8440
      Balanced Accuracy : 0.6220

       'Positive' Class : 0
```

Based on the result obtained from ANN, the model was able to obtain a prediction with an accuracy of 72.8% through Confusion Matrix comparison with the actual values.

IMPLEMENT SVM

TRAIN MODEL ON DATA

Apply kernlab package to build SVM model

Hide

```
install.packages("kernlab")
```

```
trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/kernlab_0.9-29
.tgz'
Content type 'application/x-gzip' length 2407821 bytes (2.3 MB)
==================================================
downloaded 2.3 MB
```

```
The downloaded binary packages are in
    /var/folders/y3/cl_f_r9542nddh_lgzdwb1kr0000gn/T//RtmptT4rHi/downloaded_packages
```

Hide

```
library(kernlab)
```

```
Attaching package: ‘kernlab’

The following object is masked from ‘package:ggplot2’:

    alpha

The following object is masked from ‘package:psych’:

    alpha
```

Hide

```
GermanCredit_Classifier<-ksvm(as.factor(customer_type) ~., data = GermanCredit_train,
kernel = "vanilladot")
```

```
 Setting default kernel parameters
```

Hide

```
GermanCredit_Classifier
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 507

Objective Function Value : -438
Training error : 0.292
```

EVALUATING MODEL SVM MODEL PERFORMANCE

Hide

```
GermanCredit_predictions<-predict(GermanCredit_Classifier, GermanCredit_test)
```

Hide

```
head(GermanCredit_predictions)
```

```
[1] 0 0 0 0 0 0
Levels: 0 1
```

COMPARE PREDICTED VALUES TO TRUE VALUES IN TESTING DATASET:

Hide

```
table(GermanCredit_predictions, as.factor(GermanCredit_test$customer_type))
```

```

GermanCredit_predictions   0   1
                       0 169  81
                       1   0   0
```

TRUE VALUE: 0 with 169 correctly predicted MISIDENTIFIED CASES: 1 with 81 incorrectly predicted

Hide

```
agreement <- GermanCredit_predictions == as.factor(GermanCredit_test$customer_type)
```

Hide

```
table(agreement)
```

```
agreement
FALSE  TRUE
   81   169
```

Hide

```
prop.table(table(agreement))
```

```
agreement
FALSE  TRUE
0.324 0.676
```

From the result shown above we can see that there are 81 false cases and 169 true cases predicted. This gives us the SVM model to be 67% accurate. By comparison, ANN model was able to predict more accurate with 72.8%.

6. 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 (please make sure to use accuracy, precision, and recall for comparing the models in each of the part 5 and 6. See this article (Links to an external site.) to understand how to calculate these metrics or consult chapter 10 in the text book).

IMPLEMENT ANN

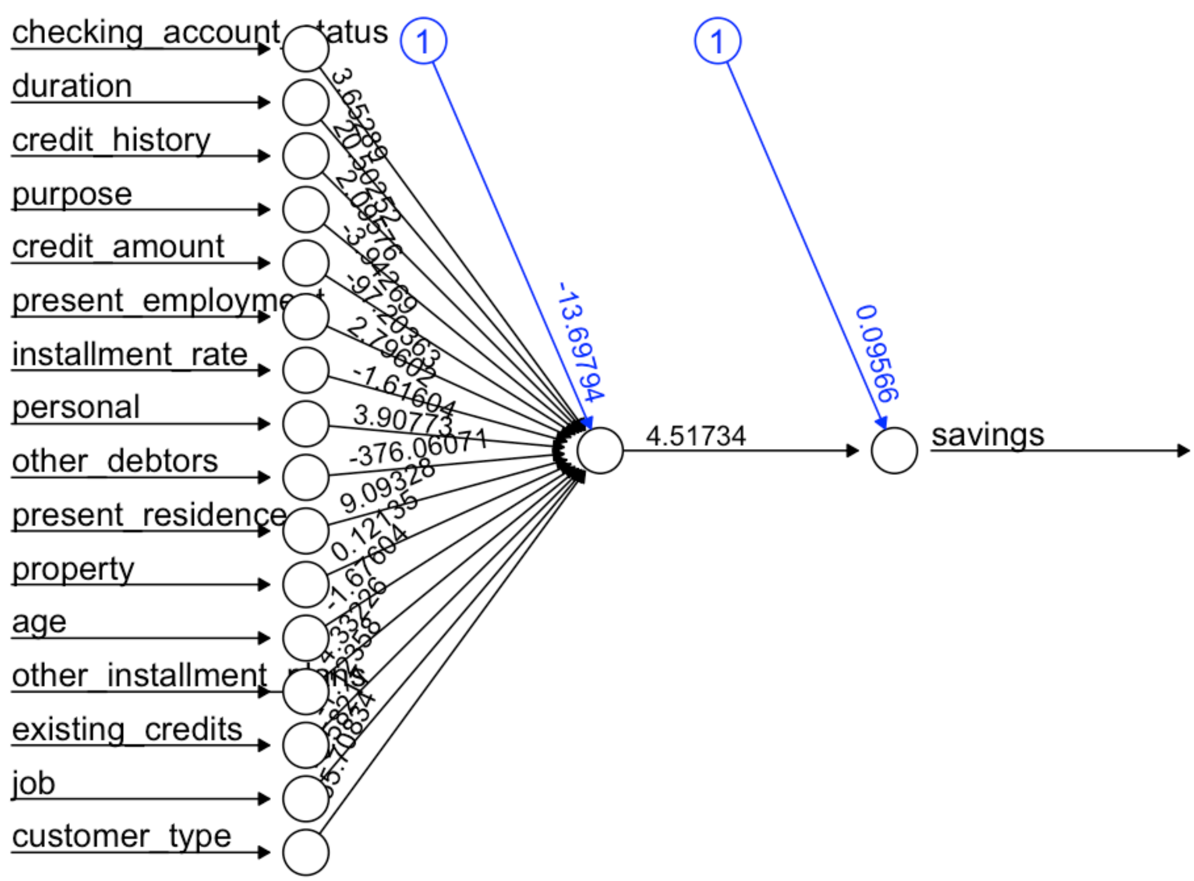
Hide

```
# Build model based on savings more than 500 DM
GermanSavings_model<-neuralnet(savings ~., data = GermanCredit_train)
```

VISUALIZE NETWORK TOPOLOGY

Hide

```
plot(GermanSavings_model)
```



Check SSE of the model:

Hide

```
GermanSavings_model$result.matrix
```

```
                                              [,1]
error                                 3.314060e+01
reached.threshold                     9.954521e-03
steps                                 9.930000e+03
Intercept.to.1layhid1                -1.369794e+01
checking_account_status.to.1layhid1   3.652888e+00
duration.to.1layhid1                  2.050252e+01
credit_history.to.1layhid1            2.095763e+00
purpose.to.1layhid1                  -3.942692e+00
credit_amount.to.1layhid1            -9.720363e+01
present_employment.to.1layhid1        2.796025e+00
installment_rate.to.1layhid1         -1.616043e+00
personal.to.1layhid1                  3.907729e+00
other_debtors.to.1layhid1            -3.760607e+02
present_residence.to.1layhid1         9.093285e+00
property.to.1layhid1                  1.213526e-01
age.to.1layhid1                      -1.676038e+00
other_installment_plans.to.1layhid1   4.332263e+00
existing_credits.to.1layhid1         -1.123580e+00
job.to.1layhid1                      -3.582166e+00
customer_type.to.1layhid1            -3.557083e+02
Intercept.to.savings                  9.566195e-02
1layhid1.to.savings                   4.517343e+00
```

Error Result: 33.14 Steps : 9930.000

Based on the result shown above, the error result for this model is moderately fair 33.14 with a quite moderate high steps of 9930.000

EVALUATE MODEL PERFORMANCE:

Hide

```
# Generate predictions on test dataset using compute ()
savingmodel_result<-compute(GermanSavings_model, GermanCredit_test[-c(6)])
```

Hide

```
# Apply net.result to check predicted values
predicted_savings<-savingmodel_result$net.result
```

PERFORM CONFUSION MATRIX: Classification Problem

Hide

```
# Convert Probabilities into Binary Class
scorePredicted_savings<-ifelse(predicted_savings >=.5, 1, 0)
scorePredicted_savings<-as.factor(scorePredicted_savings)
```

Hide

```
confusionMatrix(scorePredicted_savings, as.factor(GermanCredit_test$savings))
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 218  29
         1   3   0

               Accuracy : 0.872
                 95% CI : (0.8241, 0.9108)
    No Information Rate : 0.884
    P-Value [Acc > NIR] : 0.7594

                  Kappa : -0.0222

 Mcnemar's Test P-Value : 9.897e-06

            Sensitivity : 0.9864
            Specificity : 0.0000
         Pos Pred Value : 0.8826
         Neg Pred Value : 0.0000
             Prevalence : 0.8840
         Detection Rate : 0.8720
   Detection Prevalence : 0.9880
      Balanced Accuracy : 0.4932

       'Positive' Class : 0
```

Based on the result of ANN model, we were able to obtain accuracy score of 87.2% for model that focuses on savings. Some concern that the author would like to raise is how to distinguish the factor level of 0 and 1 since savings itself has a numeric/factor levels of 0 and 1 originally.

IMPLEMENT SVM

Build SVM model

Hide

```
GermanSavings_Classifier<-ksvm(as.factor(savings) ~., data = GermanCredit_train, kern
el = "vanilladot")
```

```
 Setting default kernel parameters
```

Hide

```
GermanSavings_Classifier
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 261

Objective Function Value : -164
Training error : 0.109333
```

EVALUATING MODEL PERFORMANCE

Hide

```
GermanSavings_predictions<-predict(GermanSavings_Classifier, GermanCredit_test)
```

Hide

```
head(GermanSavings_predictions)
```

```
[1] 0 0 0 0 0 0
Levels: 0 1
```

COMPARE PREDICTED VALUES TO TRUE VALUES IN TESTING DATASET:

Hide

```
table(GermanSavings_predictions, as.factor(GermanCredit_test$savings))
```

```
GermanSavings_predictions   0   1
                        0 221  29
                        1   0   0
```

Hide

```
prop.table(table(GermanSavings_predictions == as.factor(GermanCredit_test$savings)))
```

```
FALSE  TRUE
0.116 0.884
```

Based on the result shown above, there are 221 correctly predicted cases and 29 falsely predicted. This gives us SVM model to be 88.4 accurate. By comparison to the ANN model, SVM model was able to predict higher accuracy.

7. What are some of the insights that you learned after completing part 5 and 6? which target variable (customer_type or savings) was easier to predict for each of the algorithms? which algorithm was faster to train?

As we trained 2 models from customer_type and savings, we found that savings are way faster and more accurate when it comes to predicting. This could be because when the dataset was converted into one-hot coding, some levels are more dispersed (easier to classify) compared to customer_type.

Based on the 2 results, we also found that for customer_type, ANN was able to predict more accurate than SVM, while for savings, SVM was able to perform better. This could be due to how well the data is distributed. Main concern that the author would like to raise is how to make sure that we are converting the right binary when applying to Confusion Matrix. Since the references focus more on numerical predictions, correlation score is used. However, classification such as in this case represent binary outcome “0” or “1”.

8. Optional: Use a decision tree ensemble algorithm of your choice (e.g. bagging, boosting or random forest) to predict the customer_type. Compare the result with the ANN and SVM results in part 5 (you may refer to chapter 11 of your text book, Improving Model Performance, for more info on ensemble learning).

INSTALL RANDOM FOREST:

Hide

```
install.packages("randomForest")
```

```
trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/randomForest_4
.6-14.tgz'
Content type 'application/x-gzip' length 253893 bytes (247 KB)
==================================================
downloaded 247 KB
```

```


The downloaded binary packages are in
    /var/folders/y3/cl_f_r9542nddh_lgzdwb1kr0000gn/T//RtmptT4rHi/downloaded_packages
```

Hide

```
library(randomForest)
```

```
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.

Attaching package: 'randomForest'

The following object is masked from 'package:ggplot2':

    margin

The following object is masked from 'package:psych':

    outlier

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

    combine
```

FIT DECISION TREE TO TRAINING SET

Hide

```
# Build Random Forest Model: customer_type as target variable
GermanCredit_RFClassifier<-randomForest(x = GermanCredit_train[1:16], y = as.factor(G
ermanCredit_train$customer_type), ntree = 500, random_state = 0)
```

PREDICT WITH TEST SET

Hide

```
RFPredict_customer_type<-predict(GermanCredit_RFClassifier, newdata = GermanCredit_te
st[1:16])
```

Check the predicted result

Hide

```
RFPredict_customer_type
```

```
  1   3   7   9  12  15  18  21  22  25  27  28  32  35  42  43  44  47  60  62  63 
66  70  73  75  82 
  0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0 
0   1   0   0   0 
 86  92  97 101 102 103 107 109 126 133 140 142 144 145 146 147 149 150 154 156 157 1
74 176 182 183 192 
  0   0   0   0   1   0   0   0   0   0   0   1   0   0   1   0   1   0   0   0   0 
0   0   0   0   1 
194 198 202 208 213 214 215 216 233 245 247 249 253 254 257 269 272 283 285 288 293 2
96 300 307 313 314 
  0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0 
0   0   0   0   0 
321 325 329 335 345 350 353 354 356 359 360 366 367 368 369 370 375 380 383 385 387 4
08 410 411 416 423 
  0   0   0   0   0   0   0   0   1   1   1   0   0   0   0   0   1   0   1   0   1 
0   0   0   0   0 
425 432 436 439 449 453 454 460 467 472 474 482 484 485 486 489 491 495 496 497 502 5
06 511 514 515 517 
  0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 
0   0   0   0   0 
518 520 521 525 531 542 543 546 556 563 565 568 572 576 579 580 583 584 586 587 592 5
94 599 607 616 622 
  0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0 
1   0   0   1   0 
628 631 641 642 643 653 656 669 674 675 683 684 689 693 699 701 708 713 715 728 730 7
31 735 736 737 740 
  0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   0 
0   0   0   1   0 
743 748 749 756 758 759 763 772 773 776 786 787 791 793 795 796 799 806 825 826 827 8
28 829 830 833 839 
  0   0   0   1   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0 
0   0   1   1   0 
848 849 850 855 856 866 868 874 875 879 884 892 896 907 909 914 919 921 924 929 936 9
39 945 946 950 952 
  0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 
1   0   0   0   0 
963 964 967 970 971 972 973 978 983 984 985 989 993 995 997 998 
  0   0   0   0   0   0   1   0   0   0   0   0   1   0   0   0 
Levels: 0 1
```

COMPARE WITH CONFUSION MATRIX

Hide

```
confusionMatrix(RFPredict_customer_type, as.factor(GermanCredit_test$customer_type))
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 155  62
         1  14  19

               Accuracy : 0.696
                 95% CI : (0.6349, 0.7524)
    No Information Rate : 0.676
    P-Value [Acc > NIR] : 0.2733

                  Kappa : 0.1794

 Mcnemar's Test P-Value : 6.996e-08

            Sensitivity : 0.9172
            Specificity : 0.2346
         Pos Pred Value : 0.7143
         Neg Pred Value : 0.5758
             Prevalence : 0.6760
         Detection Rate : 0.6200
   Detection Prevalence : 0.8680
      Balanced Accuracy : 0.5759

       'Positive' Class : 0
```

Based on the result shown above from the Confusion Matrix, it shows that Random Forest predicted the value with lower accuracy compared to ANN when it comes to customer_type. Random Forest Model obtains 69.6% accuracy, ANN with 72.8% and SVM with 67.6%. In conclusion, SVM model is the weakest among the 3 in predicting whether customer is good or bad from customer_type.

Problem 2

1. Download this data set on Whole Sale Customers

Hide

```
# Load RCurl to obtain CSV file from website address
library(RCurl)
```

Hide

```
URL<-"https://archive.ics.uci.edu/ml/machine-learning-databases/00292/Wholesale%20cus
tomers%20data.csv"
WholeSaleCustomers<-read.csv(URL)
```

Hide

```
# Check Whole Sale Customers Dataset
str(WholeSaleCustomers)
```

```
'data.frame':	440 obs. of  8 variables:
 $ Channel         : int  2 2 2 1 2 2 2 2 1 2 ...
 $ Region          : int  3 3 3 3 3 3 3 3 3 3 ...
 $ Fresh           : int  12669 7057 6353 13265 22615 9413 12126 7579 5963 6006 ...
 $ Milk            : int  9656 9810 8808 1196 5410 8259 3199 4956 3648 11093 ...
 $ Grocery         : int  7561 9568 7684 4221 7198 5126 6975 9426 6192 18881 ...
 $ Frozen          : int  214 1762 2405 6404 3915 666 480 1669 425 1159 ...
 $ Detergents_Paper: int  2674 3293 3516 507 1777 1795 3140 3321 1716 7425 ...
 $ Delicassen      : int  1338 1776 7844 1788 5185 1451 545 2566 750 2098 ...
```

Based on the dataset shown above, we can see that there are 440 observations with 8 variables of full numerics/integers.

CHECK WHETHER DATASET CONTAIN ANY MISSING VALUE

Hide

```
anyNA(WholeSaleCustomers)
```

```
[1] FALSE
```

FALSE means there is no missing value detected in the overall dataset.

CHECK THE SUMMARY OF EACH VARIABLES:

Hide

```
summary(WholeSaleCustomers)
```

```
    Channel          Region           Fresh             Milk           Grocery        
Frozen       
 Min.   :1.000   Min.   :1.000   Min.   :     3   Min.   :   55   Min.   :    3   Min
.   :   25.0  
 1st Qu.:1.000   1st Qu.:2.000   1st Qu.:  3128   1st Qu.: 1533   1st Qu.: 2153   1st 
Qu.:  742.2  
 Median :1.000   Median :3.000   Median :  8504   Median : 3627   Median : 4756   Med
ian : 1526.0  
 Mean   :1.323   Mean   :2.543   Mean   : 12000   Mean   : 5796   Mean   : 7951   Mea
n   : 3071.9  
 3rd Qu.:2.000   3rd Qu.:3.000   3rd Qu.: 16934   3rd Qu.: 7190   3rd Qu.:10656   3rd 
Qu.: 3554.2  
 Max.   :2.000   Max.   :3.000   Max.   :112151   Max.   :73498   Max.   :92780   Max
.   :60869.0  
 Detergents_Paper    Delicassen     
 Min.   :    3.0   Min.   :    3.0  
 1st Qu.:  256.8   1st Qu.:  408.2  
 Median :  816.5   Median :  965.5  
 Mean   : 2881.5   Mean   : 1524.9  
 3rd Qu.: 3922.0   3rd Qu.: 1820.2  
 Max.   :40827.0   Max.   :47943.0  
```

2. Using an implementation of your choice of the k-means algorithm, determine clusters that may exist. Define 3, 4, and 5 clusters. What K do you think would result in the best clusters? What are some of the characteristics of the determined clusters? How would you label them?

TRAINING MODEL ON THE DATA

Since all of the variables in the dataset are all numerics, we can then proceed to build the k-means model

Hide

```
# Apply k-means algorithm from stats package
library(stats)
```

Hide

```
features<-WholeSaleCustomers[1:8] # Consider all features
```

PRE-PROCESS DATA: Z-Score Standardization

In this section, we apply Z-Score standardization to normalize the features. This is essential as common practice employed prior to any analysis using distance calculations for Classification Using Nearest Neighbors.

Hide

```
features_z<-as.data.frame(lapply(features, scale))
```

CHECK THE SUMMARY

Hide

```
# Original features without normalized
summary(features)
```

```
    Channel          Region          Fresh             Milk          Grocery     
Frozen
 Min.   :1.000   Min.   :1.000   Min.   :     3   Min.   :   55   Min.   :    3   Min
.   :   25.0
 1st Qu.:1.000   1st Qu.:2.000   1st Qu.:  3128   1st Qu.: 1533   1st Qu.: 2153   1st
Qu.:  742.2
 Median :1.000   Median :3.000   Median :  8504   Median : 3627   Median : 4756   Med
ian : 1526.0
 Mean   :1.323   Mean   :2.543   Mean   : 12000   Mean   : 5796   Mean   : 7951   Mea
n   : 3071.9
 3rd Qu.:2.000   3rd Qu.:3.000   3rd Qu.: 16934   3rd Qu.: 7190   3rd Qu.:10656   3rd
Qu.: 3554.2
 Max.   :2.000   Max.   :3.000   Max.   :112151   Max.   :73498   Max.   :92780   Max
.   :60869.0
 Detergents_Paper    Delicassen     
 Min.   :    3.0   Min.   :    3.0  
 1st Qu.:  256.8   1st Qu.:  408.2  
 Median :  816.5   Median :  965.5  
 Mean   : 2881.5   Mean   : 1524.9  
 3rd Qu.: 3922.0   3rd Qu.: 1820.2  
 Max.   :40827.0   Max.   :47943.0  
```

Hide

```
# Features after normalized with standardization
summary(features_z)
```

```
    Channel             Region              Fresh               Milk               Grocery
 Min.   :-0.6895   Min.   :-1.9931   Min.   :-0.9486   Min.   :-0.7779   Min.   :-0.8
364
 1st Qu.:-0.6895   1st Qu.:-0.7015   1st Qu.:-0.7015   1st Qu.:-0.5776   1st Qu.:-0.6
101
 Median :-0.6895   Median : 0.5900   Median :-0.2764   Median :-0.2939   Median :-0.3
363
 Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0
000
 3rd Qu.: 1.4470   3rd Qu.: 0.5900   3rd Qu.: 0.3901   3rd Qu.: 0.1889   3rd Qu.: 0.2
846
 Max.   : 1.4470   Max.   : 0.5900   Max.   : 7.9187   Max.   : 9.1732   Max.   : 8.9
264
     Frozen           Detergents_Paper    Delicassen
 Min.   :-0.62763   Min.   :-0.6037   Min.   :-0.5396
 1st Qu.:-0.47988   1st Qu.:-0.5505   1st Qu.:-0.3960
 Median :-0.31844   Median :-0.4331   Median :-0.1984
 Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000
 3rd Qu.: 0.09935   3rd Qu.: 0.2182   3rd Qu.: 0.1047
 Max.   :11.90545   Max.   : 7.9586   Max.   :16.4597
```

DEFINE 3, 4 and 5 NUMBERS OF CLUSTERS:

Hide

```
set.seed(123)
WholeSaleCluster3<-kmeans(features_z, 3)
WholeSaleCluster4<-kmeans(features_z, 4)
WholeSaleCulster5<-kmeans(features_z, 5)
```

EVALUATE MODEL PERFORMANCE ON K-MEANS MODEL:

Examine number of examples mentioned in each group

Hide

```
# Apply SIZE to evaluate model performance: if size too small or too large, they migh
t not be useful

# Cluster 3
WholeSaleCluster3$size
```

```
[1]  14 290 136
```

Cluster 3: Smallest is 14 and largest is 290

Hide

```
# Cluster 4
WholeSaleCluster4$size
```

```
[1] 209   9 131  91
```

Cluster 4: Smallest is 9 and largest is 209

Hide

```
# Cluster 5
WholeSaleCulster5$size
```

```
[1] 126  91   9  10 204
```

Cluster 5: Smallest is 9 and largest is 204

OBTAIN IN-DEPTH LOOK FOR EACH CLUSTERS:

Hide

```
# In-depth cluster 3
WholeSaleCluster3$centers
```

```
     Channel      Region        Fresh       Milk    Grocery       Frozen Detergents_P
aper  Delicassen
1 -0.5369039  0.31323918  2.607675597  0.9959545  0.3738463  3.283683566       -0.286
3957  2.62854395
2 -0.6526757 -0.06913154  0.004304482 -0.3669671 -0.4503362 -0.004713004       -0.443
9111 -0.14568593
3  1.4470045  0.11516763 -0.277615869  0.6799786  0.9217916 -0.327976462        0.976
0571  0.04006842
```

In this section, we can see the list of averages accordingly to our requested numbers of clusters. Above, we can see that within WholeSaleCluster3, cluster 3 has the highest average feature within Channel, and cluster 1 has the highest average feature within Fresh compared to the rest of 2 clusters mentioned.

Hide

```
# In-depth cluster 4
WholeSaleCluster4$centers
```

```
     Channel     Region       Fresh       Milk    Grocery       Frozen Detergents_Pap
er  Delicassen
1 -0.6383994  0.5899967  0.07453955 -0.3550081 -0.4523998  0.007431551       -0.44484
92 -0.14327048
2 -0.4521215  0.4464928  2.68193935  1.3696320  0.5802805  3.957575213       -0.16444
48  3.93358611
3  1.4470045  0.1364806 -0.31326411  0.7115062  0.9611591 -0.339029688        1.01550
37  0.05068558
4 -0.5721212 -1.5956780  0.01452064 -0.3443661 -0.4020087  0.079577121       -0.42392
85 -0.13295117
```

```
# In-depth cluster 5
WholeSaleCulster5$centers
```

```
     Channel      Region       Fresh       Milk    Grocery       Frozen Detergents_Pa
per   Delicassen
1  1.4470045  0.16973529 -0.30601450  0.4170255  0.6511383 -0.356863482        0.6760
784  0.006324548
2 -0.5721212 -1.59567800  0.01452064 -0.3443661 -0.4020087  0.079577121       -0.4239
285 -0.132951172
3 -0.4521215  0.44649280  2.68193935  1.3696320  0.5802805  3.957575213       -0.1644
448  3.933586111
4  1.4470045 -0.05577083  0.31347349  3.9174467  4.2707490 -0.003570131        4.6129
149  0.502793007
5 -0.6895122  0.58999669  0.04884441 -0.3564169 -0.4577973  0.010494142       -0.4473
408 -0.142786968
```

The information below shows list of highest average in the features found on each cluster for every model:

WholeSaleCluster3: Highest Average

1. Cluster 1: Region, Fresh, Milk, Frozen, Delicassen
2. Cluster 2: NaN
3. Cluster 3: Channel, Grocery, Detergents_Paper

WholeSaleCluster4: Highest Average

1. Cluster 1: Region
2. Cluster 2 : Fresh, Milk, Frozen, Delicassen
3. Cluster 3 : Channel, Grocery, Detergents_Paper
4. Cluster 4 : NaN

WholeSaleCluster5: Highest Average

1. Cluster 1 : Channel
2. Cluster 2 : NaN
3. Cluster 3 : Fresh, Frozen, Delicassen
4. Cluster 4 : Channel, Milk, Grocery, Detergents_Paper
5. Cluster 5 : Region

CHECK SUMMARY FOR EACH MODEL:

Hide

```
# Model 1: WholeSaleCluster3
summary(WholeSaleCluster3)
```

```
             Length Class  Mode
cluster      440    -none- numeric
centers       24    -none- numeric
totss          1    -none- numeric
withinss       3    -none- numeric
tot.withinss   1    -none- numeric
betweenss      1    -none- numeric
size           3    -none- numeric
iter           1    -none- numeric
ifault         1    -none- numeric
```

Hide

```
# Model 2: WholeSaleCluster4
summary(WholeSaleCluster4)
```

```
             Length Class  Mode
cluster      440    -none- numeric
centers       32    -none- numeric
totss          1    -none- numeric
withinss       4    -none- numeric
tot.withinss   1    -none- numeric
betweenss      1    -none- numeric
size           4    -none- numeric
iter           1    -none- numeric
ifault         1    -none- numeric
```

Hide

```
# Model 3: WholeSaleCluster5
summary(WholeSaleCulster5)
```

```
             Length Class  Mode   
cluster      440    -none- numeric
centers       40    -none- numeric
totss          1    -none- numeric
withinss       5    -none- numeric
tot.withinss   1    -none- numeric
betweenss      1    -none- numeric
size           5    -none- numeric
iter           1    -none- numeric
ifault         1    -none- numeric
```

VISUALIZE EACH MODEL

To visualize each model with its clusters, we apply fviz_cluster from factoextra package. Here, we can depict how each of the model presents subgroups of observations based on the number of clusters we requested (eg., 3, 4 and 5). Notice that when visualizing, we put the normalized features instead of the original ones.

Hide

```
library(ggplot2)
install.packages("useful")
```

```
trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/useful_1.2.6.t
gz'
Content type 'application/x-gzip' length 166348 bytes (162 KB)
==================================================
downloaded 162 KB
```

```


The downloaded binary packages are in
    /var/folders/y3/cl_f_r9542nddh_lgzdwb1kr0000gn/T//RtmptT4rHi/downloaded_packages
```

Hide

```
library(useful)
```

Hide

```
install.packages("factoextra")
```

```
also installing the dependencies ‘crosstalk’, ‘DT’, ‘ellipse’, ‘flashClust’, ‘leaps’,
```

```
‘dendextend’, ‘FactoMineR’

trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/crosstalk_1.1.
1.tgz'
Content type 'application/x-gzip' length 782735 bytes (764 KB)
==================================================
downloaded 764 KB

trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/DT_0.18.tgz'
Content type 'application/x-gzip' length 1678727 bytes (1.6 MB)
==================================================
downloaded 1.6 MB

trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/ellipse_0.4.2.
tgz'
Content type 'application/x-gzip' length 70534 bytes (68 KB)
==================================================
downloaded 68 KB

trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/flashClust_1.0
1-2.tgz'
Content type 'application/x-gzip' length 24171 bytes (23 KB)
==================================================
downloaded 23 KB

trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/leaps_3.1.tgz'
Content type 'application/x-gzip' length 100461 bytes (98 KB)
==================================================
downloaded 98 KB

trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/dendextend_1.1
5.1.tgz'
Content type 'application/x-gzip' length 3887411 bytes (3.7 MB)
==================================================
downloaded 3.7 MB

trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/FactoMineR_2.4
.tgz'
Content type 'application/x-gzip' length 3764532 bytes (3.6 MB)
==================================================
downloaded 3.6 MB

trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.6/factoextra_1.0
.7.tgz'
Content type 'application/x-gzip' length 416329 bytes (406 KB)
==================================================
downloaded 406 KB
```

```
The downloaded binary packages are in
    /var/folders/y3/cl_f_r9542nddh_lgzdwb1kr0000gn/T//RtmptT4rHi/downloaded_packages
```

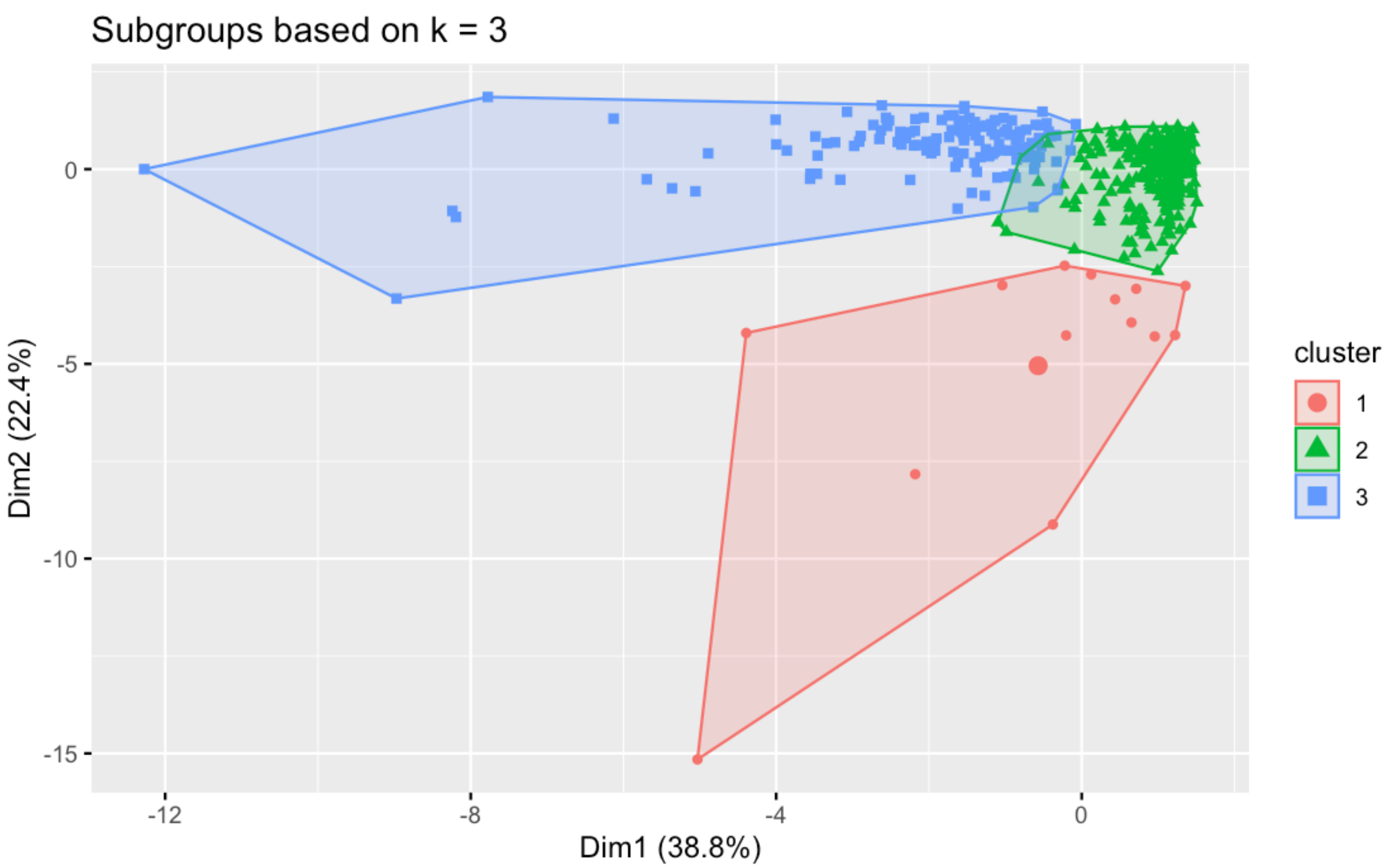
Hide

```
library(factoextra)
```

```
Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WB
a
```

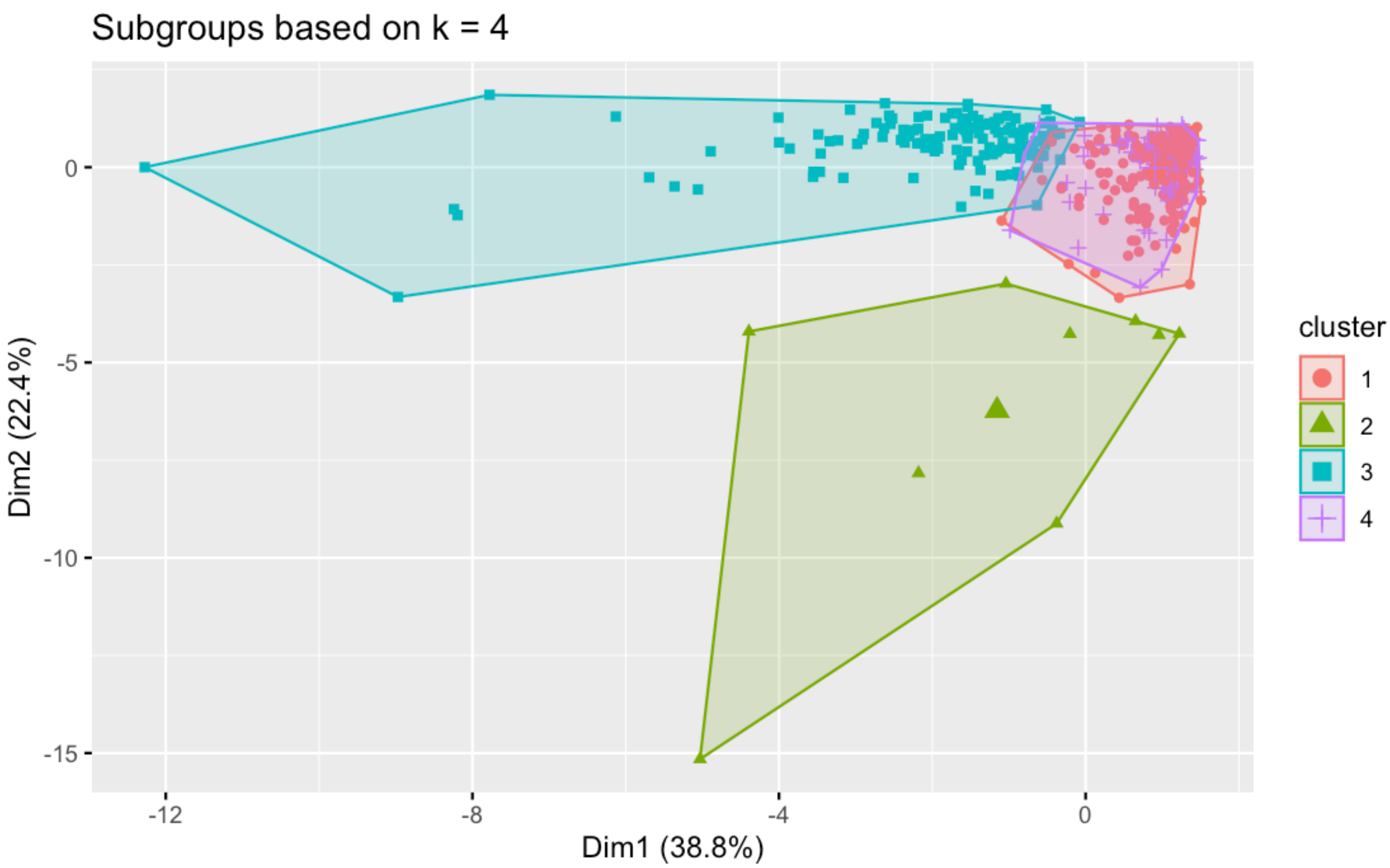
Hide

```
# Model 1: WholeSaleCluster3
fviz_cluster(WholeSaleCluster3, geom = "point", data = features_z, main = "Subgroups
based on k = 3")
```



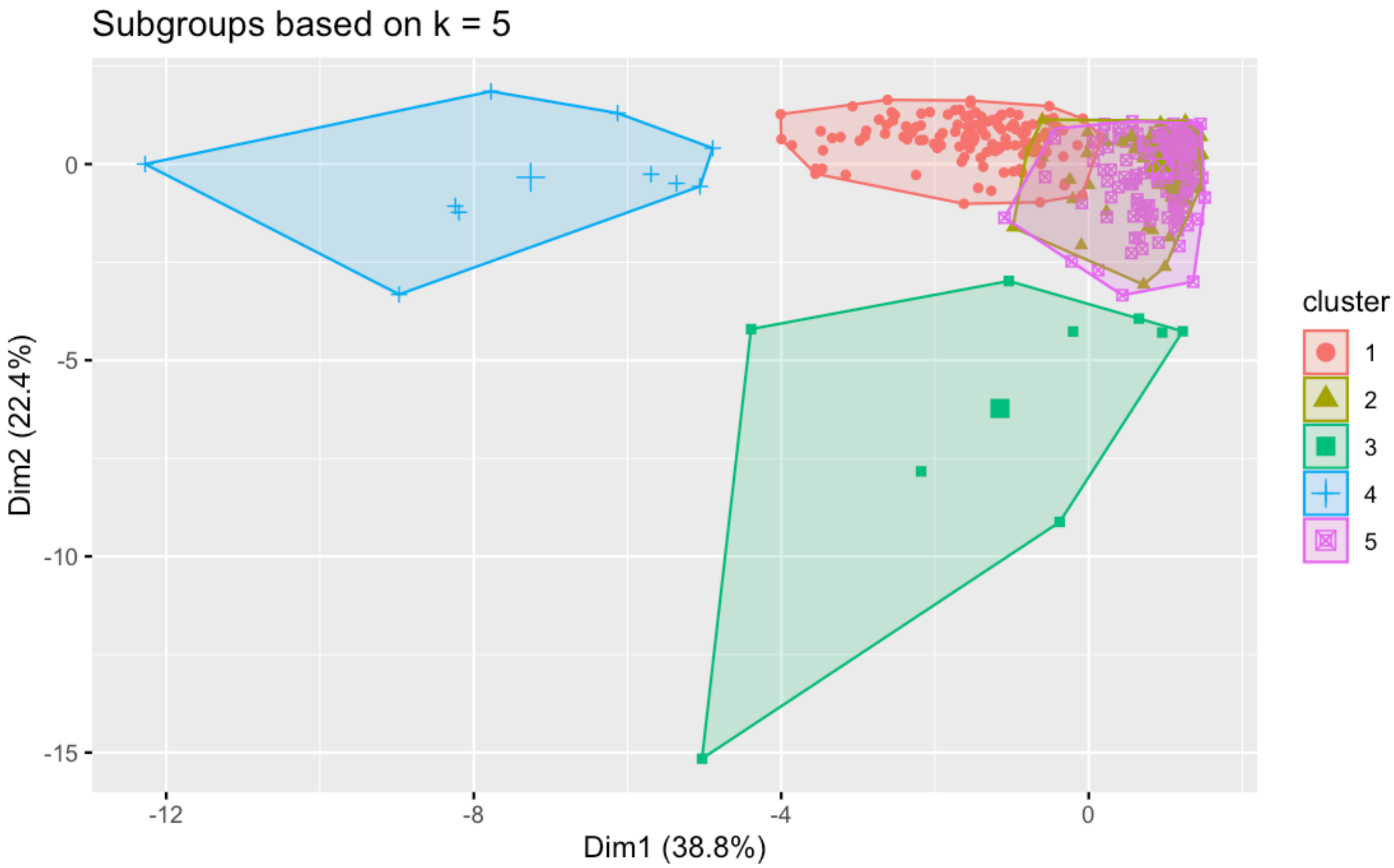
Hide

```
# Model 2: WholeSaleCluster4
fviz_cluster(WholeSaleCluster4, geom = "point", data = features_z, main = "Subgroups
based on k = 4")
```



Hide

```
# Model 3: WholeSaleCluster5
fviz_cluster(WholeSaleCulster5, geom = "point", data = features_z, main = "Subgroups
based on k = 5")
```



Based on the result of the visualization, we were able to obtain how the model subgroup the features accordingly to the number of clusters given.

As we notice, Model 2 and 3 show many subgroups within such close proximity, making the subroups overlap with one another. In Model 2, we can see cluster 1 and 4 cover almost the same area. Similar condition with Model 3 where less than 1/2 of the features of cluster 1 merge with cluster 5 and cluster 2. This unfortunate condition could create less-classifiable features in Model 2 and 3.

In conclusion, Model 1 known as “WholeSaleCluster3” seems to have better model among the 2 models due to its more classifiable subgroups.

VISUALIZE THE NORMALIZED DATASET WITH ELBOW METHOD VISUALIZATION:

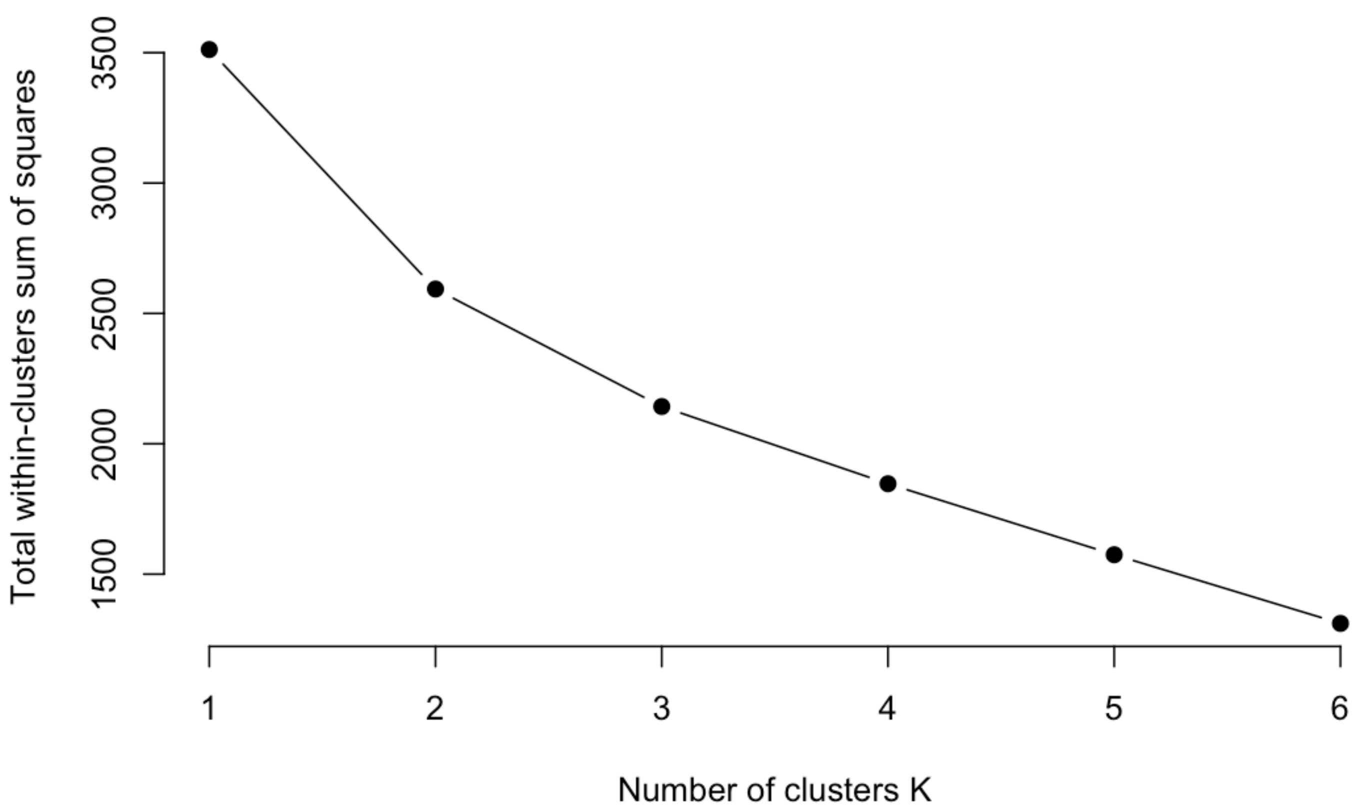
Here we visualize the normalized dataset and check the potential optimal k-means. Notice that we do not include the models here instead the normalized original dataset.

Hide

```
library(purrr)
```

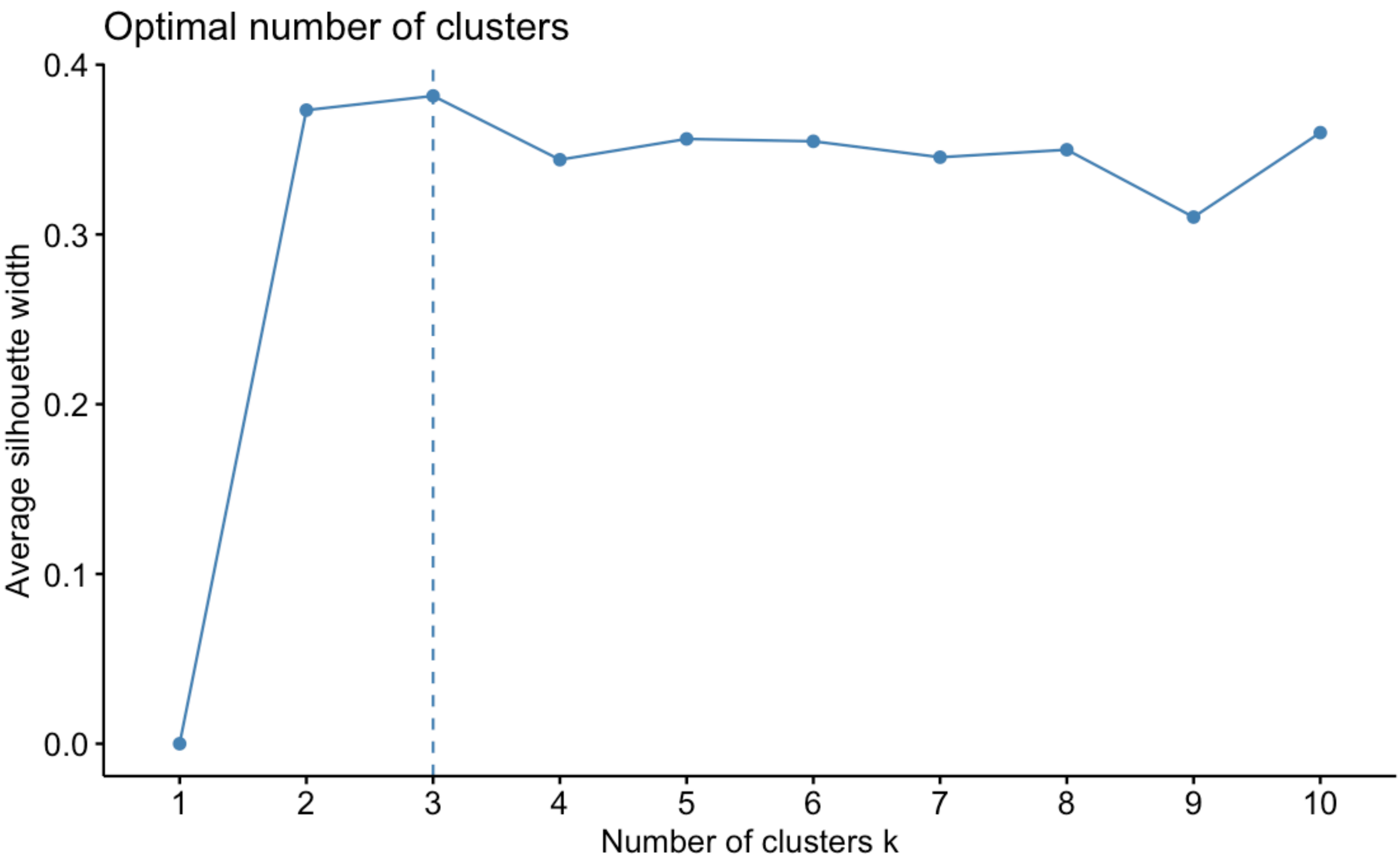
Hide

```
set.seed(123)

# function to compute total within-cluster sum of square
wss <- function(k) {
  kmeans(features_z, k, nstart = 5 )$tot.withinss
}

# Compute and plot wss for k = 1 to k = 6
k.values <- 1:6

# extract wss for 2-15 clusters
wss_values <- map_dbl(k.values, wss)

plot(k.values, wss_values,
       type="b", pch = 19, frame = FALSE,
       xlab="Number of clusters K",
       ylab="Total within-clusters sum of squares")
```



Hide

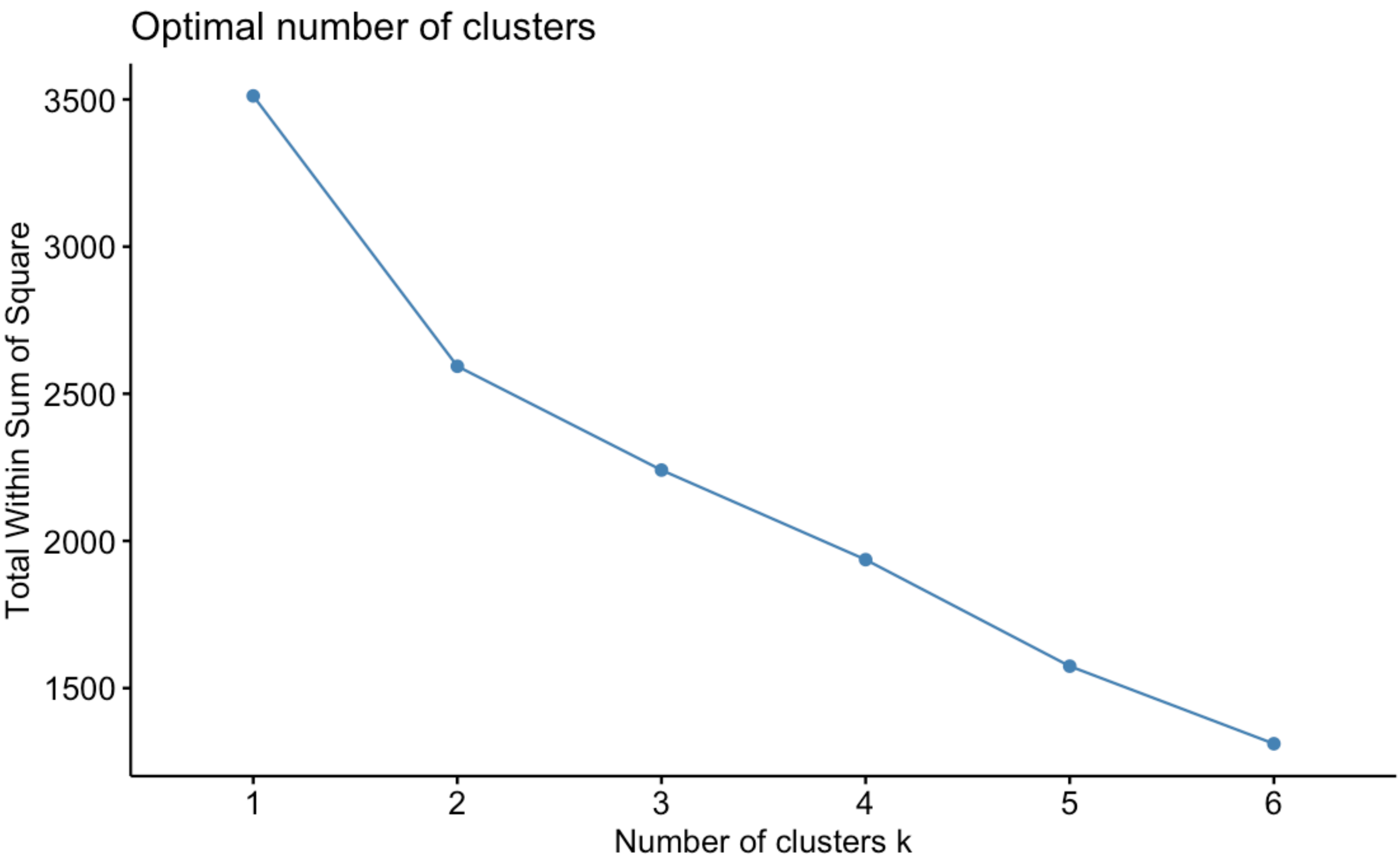
```
# Obtain optimal kmeans
fviz_nbclust(features_z, kmeans, method = "silhouette")
```



We also implemented the Elbow Curve Method where optimal kmeans is obtained from the “elbow” side of the line. In this case, the graph shows us k-means of 3 is the optimal number of clusters for normalized dataset.

Hide

```
fviz_nbclust(features_z, kmeans, method = "wss", k.max = 6)
```


Optimal number of clusters
3500
3000
2500
2000
1500
Total Within Sum of Square
1
2
3
4
5
6
Number of clusters k