Final Exam

Hari Aravind

3/17/2021

```
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
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.4
```

```
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(rpart)
library(e1071)
library(randomForest)
```

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

```
## 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:dplyr':
##
##     combine
```

```
library(gbm)
```

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

```
## Loaded gbm 2.1.8
```

```
library("readxl")
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
```

1. Model comparison

We will analyze CAR.DAT.

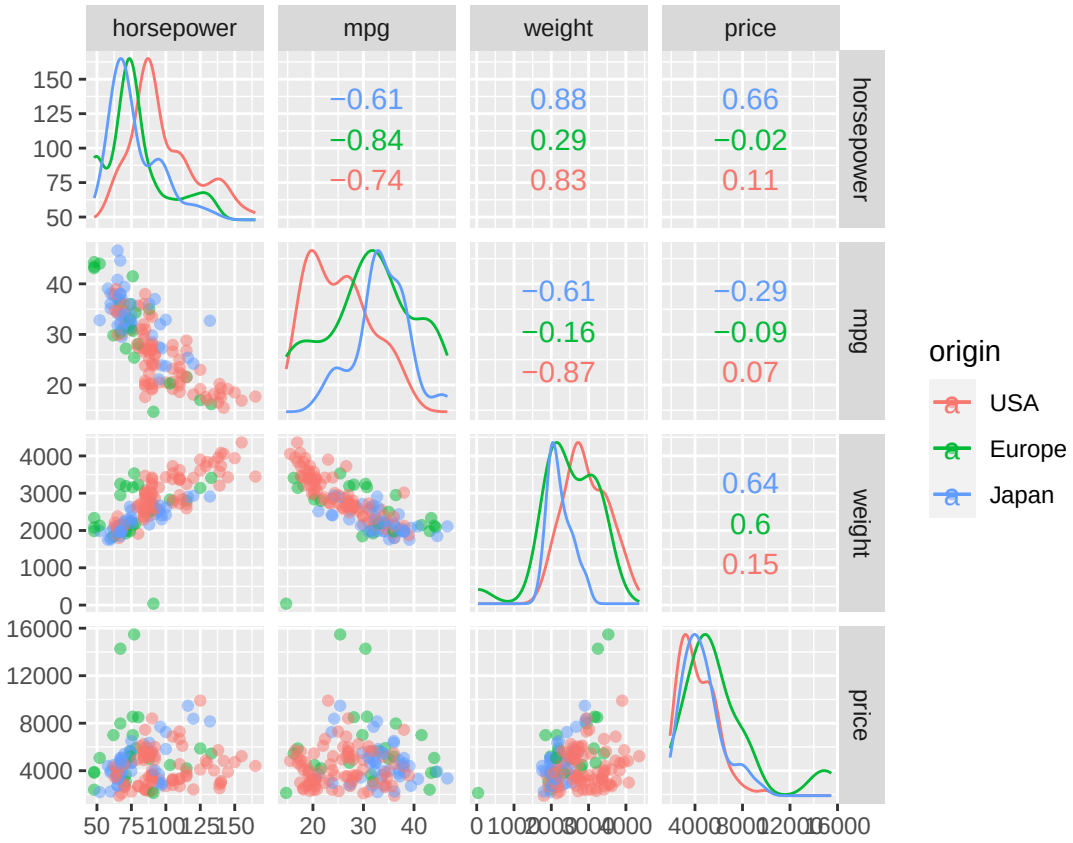
Let's select columns: *horsepower*, *mpg*, *weight*, *price*, *origin*. And transform *origin*.

We will classify cars by *horsepower*, *mpg*, *weight*, *price*.

```
data=read.delim("C:/Users/Jarvis/Documents/UWT MSBA/Data Mining/DMBA-R-datasets/CAR.DAT",sep="")

data = data %>% select(horsepower, mpg, weight, price, origin)
data = data[-62,]

cols<-c("horsepower","mpg", "weight", "price")

origins <- c('USA', 'Europe', 'Japan')
data$origin <- factor(data$origin, labels = origins)

head(data)
```

```
##   horsepower  mpg weight price origin
## 1         48 43.1   1985  2400 Europe
## 2         66 36.1   1800  1900    USA
## 3         52 32.8   1985  2200  Japan
## 4         70 39.4   2070  2725  Japan
## 5         60 36.1   1800  2250  Japan
## 6        110 19.9   3365  3300    USA
```

```
ggscatmat(data, columns = cols, color = "origin" , alpha=0.5)
```



1.A

Let's calculate number of data for each origin.

```
data %>% group_by(origin) %>% summarise(no_rows = length(origin)) %>% ungroup()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 2
##   origin no_rows
##   <fct>    <int>
## 1 USA         83
## 2 Europe      23
## 3 Japan       43
```

We see, that there are only 23 cars from Europe. Let's balance this data: we will take 23 cars from each origin. Also lets shuffle the data.

```
set.seed(123)
data = data %>% group_by(origin) %>% mutate(
  ind = sample(1:length(origin))
) %>% arrange(ind) %>% filter(ind<=23) %>% select(-ind) %>% ungroup()

data %>% group_by(origin) %>% summarise(no_rows = length(origin))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 2
##   origin no_rows
```

```
##   <fct>    <int>
## 1 USA         23
## 2 Europe      23
## 3 Japan       23
```

1.B

Now we have 63 rows. Let's consider the whole data (63 rows) and a part of the data (30 rows, balanced).

```
data63 = data %>% group_by(origin) %>% mutate(
  ind = 1:length(origin)
) %>% ungroup()

data30 = data %>% group_by(origin) %>% mutate(
  ind = 1:length(origin)
) %>% filter(ind<=10) %>% ungroup()
```

Let's make test and train (balanced).

```
smp_size63 <- floor(0.7 * nrow(data63)/3)
train_ind63 <- 1:smp_size63
train63 <- data63 %>% filter(ind %in% train_ind63)
test63 <- data63 %>% filter(!ind %in% train_ind63)

smp_size30 <- floor(0.7 * nrow(data30)/3)
train_ind30 <- 1:smp_size30
train30 <- data30 %>% filter(ind %in% train_ind30)
test30 <- data30 %>% filter(!ind %in% train_ind30)
```

1.C

Let's see on decision tree ans SVM

```
accuracy = function(test, predict){
  return(sum(test==predict)/length(test))
}

dt63 <- rpart(origin~., data = train63, method = 'class')
dt63_train = accuracy(train63$origin, predict(dt63, train63, type = 'class'))
dt63_test = accuracy(test63$origin, predict(dt63, test63, type = 'class'))

dt30 <- rpart(origin~., data = train30, method = 'class')
dt30_train = accuracy(train30$origin, predict(dt30, train30, type = 'class'))
dt30_test = accuracy(test30$origin, predict(dt30, test30, type = 'class'))



svm63 <- svm(origin ~ ., data = train63)
svm63_train = accuracy(train63$origin, predict(svm63, train63, type = 'class'))
svm63_test = accuracy(test63$origin, predict(svm63, test63, type = 'class'))

svm30 <- svm(origin ~ ., data = train30)
svm30_train = accuracy(train30$origin, predict(svm30, train30, type = 'class'))
svm30_test = accuracy(test30$origin, predict(svm30, test30, type = 'class'))

Table = data.frame(classifier = rep(c("Decision tree", "SVM"), each=4),
```

```
                  size = rep(c("63","63", "30", "30"), 2),
                  train_test = rep(c("train", "test"), 4),
                  acc = round(c(dt63_train, dt63_test, dt30_train, dt30_test,
                                svm63_train, svm63_test, svm30_train, svm30_test),3))

Table
```

```
##      classifier size train_test   acc
## 1 Decision tree   63      train 0.625
## 2 Decision tree   63       test 0.524
## 3 Decision tree   30      train 0.524
## 4 Decision tree   30       test 0.444
## 5           SVM   63      train 0.688
## 6           SVM   63       test 0.429
## 7           SVM   30      train 0.810
## 8           SVM   30       test 0.556
```

1.D

We see that both classifiers work better on train sets. In general, for bigger data size classifiers should work better on train, but here we have small total data size, so we don't see that.

SVM has better results on test, also the classifier doen't tend to overfit compared with decision tree. So I would use SVM.

1.E

We will usee a bagging algorithm – random forest and gradient boosting as a boosting algorithm.

```
rf63 <- randomForest(origin~., data = train63, method = 'class')
rf63_train = accuracy(train63$origin, predict(rf63, train63, type = 'class'))
rf63_test = accuracy(test63$origin, predict(rf63, test63, type = 'class'))

rf30 <- randomForest(origin~., data = train30, method = 'class')
rf30_train = accuracy(train30$origin, predict(rf30, train30, type = 'class'))
rf30_test = accuracy(test30$origin, predict(rf30, test30, type = 'class'))



gb63 <- gbm(origin ~ ., data = train63, n.trees = 100)
```

```
## Distribution not specified, assuming multinomial ...
```

```
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
```

```
gb63_train = accuracy(as.numeric(train63$origin), apply(predict(gb63, train63, n.trees = 100), 1, which
gb63_test = accuracy(as.numeric(test63$origin), apply(predict(gb63, test63, n.trees = 100), 1, which.ma

gb30 <- gbm(origin ~ ., data = train30, n.trees = 100, n.minobsinnode = 0)
```

```
## Distribution not specified, assuming multinomial ...
```

```
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
```

```
gb30_train = accuracy(as.numeric(train30$origin), apply(predict(gb30, train30, n.trees = 100), 1, which
gb30_test = accuracy(as.numeric(test30$origin), apply(predict(gb30, test30, n.trees = 100), 1, which.ma

Table = data.frame(classifier = rep(c("Random forest", "Gradient boosting"), each=4),
                   size = rep(c("63","63", "30", "30"), 2),
                   train_test = rep(c("train", "test"), 4),
                   acc = round(c(rf63_train, rf63_test, rf30_train, rf30_test,
                                 gb63_train, gb63_test, gb30_train, gb30_test),3))

Table
```

```
##          classifier size train_test   acc
## 1     Random forest   63      train 1.000
## 2     Random forest   63       test 0.667
## 3     Random forest   30      train 1.000
## 4     Random forest   30       test 0.556
## 5 Gradient boosting   63      train 0.771
## 6 Gradient boosting   63       test 0.524
## 7 Gradient boosting   30      train 1.000
## 8 Gradient boosting   30       test 0.444
```

1.F

Bagging and boosting work much better on train. We have very good results on test for Random forest on data63. For our data Random forest works better.

2. Exploring

2.A

To predict the churn we need to collect the following data for each school:

1) School coordinates / districts – some schools can have similar churn and parameters, because the are closely located. The data we can take from the map.
2) Income in the school region – different income affects ability to pay the rent => affects the churn. We can find the data in city database.
3) Non-white – we see that it can affect the churn from the map.
4) Average rent price – the logic the same as in 2)

2.B

Using the variables (X1, X2, X3, X4) from 2.A we can built our model

Churn_Value = M(X1, X2, X3, X4).

For some model we need to transform our variables: centering, scaling, one-hot encoding and so on. But for some models we do not need it. So we will choose random forest.

2.C

Let's create an example and fit the model.

```
num = 200
```

```r
data_churn = data.frame(
  Region = sample(c("A", "B", "C"), num, replace = TRUE),
  Income = rep(0, num),
  Non_white = sample(c("Yes", "No"), num, replace = TRUE),
  Rent = rep(0, num)
) %>% mutate(
  Income = case_when(
    Region == "A" ~ runif(num)*1500,
    Region == "B" ~ runif(num)*2000,
    Region == "C" ~ runif(num)*2500
  ) + case_when(
    Non_white == "Yes" ~ runif(num)*1200,
    Non_white == "No" ~ runif(num)*1000
  ),
  Rent = case_when(
    Region == "A" ~ runif(num)*500,
    Region == "B" ~ runif(num)*700,
    Region == "C" ~ runif(num)*1000
  ),
  Churn = Income - Rent + runif(num, min=-1)*1000,
  Churn = as.factor(Churn < mean(Churn))
)

head(data_churn)
```

```
##   Region    Income Non_white      Rent Churn
## 1      B 1157.6550       Yes 270.28339 FALSE
## 2      A 1506.5900       Yes 296.39683 FALSE
## 3      A 1294.4367        No 112.38015 FALSE
## 4      C 3293.5322       Yes 613.32302 FALSE
## 5      A  586.6019        No 326.07772  TRUE
## 6      C 2160.3100        No  45.83126 FALSE
```

```r
train_size <- floor(0.7 * nrow(data_churn))
train_ind <- 1:train_size
train_churn <- data_churn[train_ind,]
test_churn <- data_churn[-train_ind,]

rf <- randomForest(Churn~., data = train_churn, method = 'class')
rf_train = accuracy(train_churn$Churn, predict(rf, train_churn %>% select(-Churn), type = 'class'))
rf_test = accuracy(test_churn$Churn, predict(rf, test_churn %>% select(-Churn), type = 'class'))

print(sprintf("Accuracy for train: %s", rf_train))
```

```
## [1] "Accuracy for train: 1"
```

```r
print(sprintf("Accuracy for test: %s", rf_test))
```

```
## [1] "Accuracy for test: 0.7"
```

We have got an interesting example.

3. Understanding measures

3.A

```
accuracy_fun = function(data){
  return((data[1,1]+data[2,2])/sum(data))
}

precision_fun = function(data){
  return(data[1,1]/sum(data[1:2,1]))
}

recall_fun = function(data){
  return(data[1,1]/sum(data[1,1:2]))
}

f1_fun = function(data){
  return(2 * precision_fun(data) * recall_fun(data) /(precision_fun(data) + recall_fun(data)))
}

Model1 = data.frame(Predicted1 = c(512, 11), Predicted2 = c(488, 899))
Model2 = data.frame(Predicted1 = c(495, 1203), Predicted2 = c(505, 98797))

print(Model1)
```

```
##   Predicted1 Predicted2
## 1        512        488
## 2         11        899
```

```
print(Model2)
```

```
##   Predicted1 Predicted2
## 1        495        505
## 2       1203      98797
```

```
results = data.frame(
  Scores = c("accuracy","precision","recall","F-Score"),
  Model1 = round(c(accuracy_fun(Model1),precision_fun(Model1),recall_fun(Model1),f1_fun(Model1)),3),
  Model2 = round(c(accuracy_fun(Model2),precision_fun(Model2),recall_fun(Model2),f1_fun(Model2)),3)
)
results
```

```
##      Scores Model1 Model2
## 1  accuracy  0.739  0.983
## 2 precision  0.979  0.292
## 3    recall  0.512  0.495
## 4   F-Score  0.672  0.367
```

Now we have all scores for both models.

3.B

Discussion:

Accuracy: tells us how many predictions were correct, but gives no information about mistakes. We can't understand which class is easier to classify. For the first model we have lower accuracy, but in the first model

we have balanced classes, so accuracy is more reliable for the first class. We use the score when we want to make as much as possible correct answers.

Precision: tells how many predicted 1 classes are actually – 1. In the first model we have a lot of in data[1,1] and few in data[2,1]. In the second the opposite situation. We see it in the precision scores. The score is better to use when we do not want to make false positive mistakes.

Recall: tells us how many of true positive results (1 class) were predicted correctly. These models have close Recall In the tables we see, that in the fists rows the numbers are close, using these numbers we get Recall. The score is better to use when we do not want to make false negative mistakes.

F-Score: is combined version of precision and recall. The score is better in general situation.

The first model have bigger F-score and we don't know anything about the data. So it is better to choose F-score

4. Logistic regression

4.A

Logistic regression coefficient for Dose is 0.674 and its CI is much bigger than zero, it means we have significant coefficient (also p-value is very low, around zero). We can say, that the dose affects the insects deaths. Also we see that Odds ratio is bigger than 1 (and CI biger than 1), it means that Dose coefficient makes difference in deaths. If we didn't have it, we would have OR=1=exp(0).

4.B

```
insect_data=read_excel("C:/Users/Jarvis/Documents/UWT MSBA/Data Mining/DMBA-R-datasets/LRTEST.xls")
insect_data$Death = ifelse(insect_data$Death=="YES", 1, 0)

log_reg <- glm(Death ~ Dose, data = insect_data, family = "binomial")
print(summary(log_reg))
```

```
##
## Call:
## glm(formula = Death ~ Dose, family = "binomial", data = insect_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8004  -0.9272  -0.5111   0.8883   2.0495
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.64367    0.15611  -16.93   <2e-16 ***
## Dose         0.67399    0.03911   17.23   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2061.9  on 1499  degrees of freedom
## Residual deviance: 1680.3  on 1498  degrees of freedom
## AIC: 1684.3
##
## Number of Fisher Scoring iterations: 4
```

```r
exp(coef(log_reg))
```

```
## (Intercept)        Dose
##   0.0710995   1.9620557
```

We have the same coefficient and the same Odds Ratio.

4.C

```r
insect_data %>% group_by(Dose) %>% summarise(
  Prob = mean(Death)
)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 6 x 2
##    Dose  Prob
##   <dbl> <dbl>
## 1     1 0.112
## 2     2 0.212
## 3     3 0.372
## 4     4 0.504
## 5     5 0.688
## 6     6 0.788
```

We see the observed probabilities.

4.D

```r
probabilities <- log_reg %>% predict(insect_data, type = "response")
unique(probabilities)
```

```
## [1] 0.1224230 0.2148914 0.3493957 0.5130710 0.6739903 0.8022286
```

Now we have probabilities for different doses using log regression.

5. Association rules

Association rules help to find relations between variables in a dataset. The logic is simple: using a set of variables we can understand and describe relations (associations) with another set of variables. For simplicity let's consider the next example: we have variables X1, X2, X3 with values X1={1, 0 ,1}, X2={1, 1 ,0}, X3={1, 0 ,0}, then we can say (from the first values of each variable), that X1 and X2 are associated with X3.

Association rules can bring wrong assocoations, when we have a large dataset. But we can control wrong associations by significant-level. There are a lot ofalgorithms for generating association rules.

Association rules are employed today in many application areas including market basket analysis, Web usage mining, intrusion detection, continuous production, and bioinformatics.

6. Neural network

6.A

Let's predict car prices by variables *displace*, *horsepower*, *mpg*, *weight* using NN.

```r
set.seed(12)
data_nn=read.delim("C:/Users/Jarvis/Documents/UWT MSBA/Data Mining/DMBA-R-datasets/CAR.DAT",sep="")

data_nn = data_nn %>% select(displace, horsepower, mpg, weight, price)
data_nn = data_nn[-62,]

#scailing data
max = apply(data_nn  , 2 , max)
min = apply(data_nn , 2 , min)
data_nn = as.data.frame(scale(data_nn , center = min, scale = max - min))

inds_train = sample(1:nrow(data_nn), floor(nrow(data_nn)*0.7))
train_nn <- data_nn[inds_train,]
test_nn <- data_nn[-inds_train,]
```

```r
NN_3_lr0.1 = neuralnet(price ~ displace + horsepower + mpg + weight, train_nn, hidden = 3 , learningrat

NN_5_lr0.1 = neuralnet(price ~ displace + horsepower + mpg + weight, train_nn, hidden = 5 , learningrat

NN_3_3_lr0.1 = neuralnet(price ~ displace + horsepower + mpg + weight, train_nn, hidden = c(3, 3) , lea

NN_5_5_lr0.1 = neuralnet(price ~ displace + horsepower + mpg + weight, train_nn, hidden = c(5, 5) , lea

NN_3_lr1 = neuralnet(price ~ displace + horsepower + mpg + weight, train_nn, hidden = 3 , learningrate =

NN_5_lr1 = neuralnet(price ~ displace + horsepower + mpg + weight, train_nn, hidden = 5 , learningrate =

NN_3_3_lr1 = neuralnet(price ~ displace + horsepower + mpg + weight, train_nn, hidden = c(3, 3) , learni

NN_5_5_lr1 = neuralnet(price ~ displace + horsepower + mpg + weight, train_nn, hidden = c(5, 5) , learni
```

```r
MSE = function(val, pred){
  return(round((sum((val - pred)^2) / length(val)) ^ 0.5, 3))
}

NNs = list(NN_3_lr0.1, NN_5_lr0.1, NN_3_3_lr0.1, NN_5_5_lr0.1,
        NN_3_lr1, NN_5_lr1, NN_3_3_lr1, NN_5_5_lr1)

k=0
train_mse = rep(NA, 8)
for(NN in NNs){
  k = k+1
  train_mse[k] = MSE(compute(NN, train_nn %>% select(-price))$net.result, train_nn$price)
}

k=0
test_mse = rep(NA, 8)
for(NN in NNs){
  k = k+1
  test_mse[k] = MSE(compute(NN, test_nn %>% select(-price))$net.result, test_nn$price)
}

table_nn = data.frame(
  Hidden = c("3","5","3, 3","5, 5","3","5","3, 3","5, 5"),
```

```
  LR = c("0.1","0.1","0.1","0.1","1","1","1","1"),
  MSE_train = train_mse,
  MSE_test = test_mse
)
table_nn
```

```
##   Hidden  LR MSE_train MSE_test
## 1      3 0.1     0.120    0.169
## 2      5 0.1     0.101    0.142
## 3   3, 3 0.1     0.100    0.116
## 4   5, 5 0.1     0.067    0.210
## 5      3   1     0.119    0.146
## 6      5   1     0.119    0.163
## 7   3, 3   1     0.095    0.097
## 8   5, 5   1     0.075    0.194
```

6.B

We see that Learning Rate = 1 works worse on train a bit, but on test learning rates look similar.

Neural networks with 2 hidden layers work better on train. But the best result (0.109) on test we have with LR=0.1 and 1 hidden layers with 5 nodes.

7. K-means

Make dataset.

```
data_kmeans = data.frame(
  Customer = 1:20,
  Cluster = 0,
  A = c(0,0,1,1,1,0,1,1,1,0,0,1,1,0,0,0,1,0,0,0),
  B = c(0,1,1,1,0,0,0,1,0,0,0,1,0,1,1,0,0,0,1,1),
  C = c(1,0,0,0,0,1,1,0,0,1,1,0,1,0,0,1,0,0,1,0),
  D = c(1,1,0,1,0,0,1,0,0,1,1,0,0,0,1,1,0,1,1,1)
)
data_kmeans
```

```
##    Customer Cluster A B C D
## 1         1       0 0 0 1 1
## 2         2       0 0 1 0 1
## 3         3       0 1 1 0 0
## 4         4       0 1 1 0 1
## 5         5       0 1 0 0 0
## 6         6       0 0 0 1 0
## 7         7       0 1 0 1 1
## 8         8       0 1 1 0 0
## 9         9       0 1 0 0 0
## 10       10       0 0 0 1 1
## 11       11       0 0 0 1 1
## 12       12       0 1 1 0 0
## 13       13       0 1 0 1 0
## 14       14       0 0 1 0 0
## 15       15       0 0 1 0 1
## 16       16       0 0 0 1 1
## 17       17       0 1 0 0 0
```

```
## 18       18       0 0 0 0 1
## 19       19       0 0 1 1 1
## 20       20       0 0 1 0 1
```

Set initial centroid.

```
euc.dist = function(x1, x2) sqrt(sum((x1 - x2) ^ 2))

centroid = data.frame(
  Cluster = 1:3,
  A = c(1,0,0),
  B = c(1,1,1),
  C = c(0,1,0),
  D = c(1,1,1)
)
centroid
```

```
##   Cluster A B C D
## 1       1 1 1 0 1
## 2       2 0 1 1 1
## 3       3 0 1 0 1
```

For each customer calculate 3 distances to clusters, then choose the closest cluster.

Update centroids.

Repeat this 3 times.

```
for(r in 1:3){
  for(k in 1:20){
    custumer = as.numeric(data_kmeans[k, 3:6])
    cur_cluster = which.min(sapply(1:3, function(x) euc.dist(as.numeric(centroid[x, 2:5]),custumer)))
    data_kmeans$Cluster[k] = cur_cluster
  }
  centroid[1, 2:5] = apply(data_kmeans[data_kmeans$Cluster==1,3:6], 2, mean)
  centroid[2, 2:5] = apply(data_kmeans[data_kmeans$Cluster==2,3:6], 2, mean)
  centroid[3, 2:5] = apply(data_kmeans[data_kmeans$Cluster==3,3:6], 2, mean)

  print(sprintf("After the %s iteration:", r))
  print(centroid)
  print("")
}
```

```
## [1] "After the 1 iteration:"
##   Cluster A         B         C         D
## 1       1 1 0.4444444 0.2222222 0.2222222
## 2       2 0 0.1666667 1.0000000 0.8333333
## 3       3 0 0.8000000 0.0000000 0.8000000
## [1] ""
## [1] "After the 2 iteration:"
##   Cluster         A         B     C         D
## 1       1 1.0000000 0.5000000 0.125 0.1250000
## 2       2 0.1428571 0.1428571 1.000 0.8571429
## 3       3 0.0000000 0.8000000 0.000 0.8000000
## [1] ""
## [1] "After the 3 iteration:"
##   Cluster         A         B     C         D
## 1       1 1.0000000 0.5000000 0.125 0.1250000
```

```
## 2       2 0.1428571 0.1428571 1.000 0.8571429
## 3       3 0.0000000 0.8000000 0.000 0.8000000
## [1] ""
```

Now we have all customers clustered:

```
data_kmeans
```

```
##    Customer Cluster A B C D
## 1         1       2 0 0 1 1
## 2         2       3 0 1 0 1
## 3         3       1 1 1 0 0
## 4         4       1 1 1 0 1
## 5         5       1 1 0 0 0
## 6         6       2 0 0 1 0
## 7         7       2 1 0 1 1
## 8         8       1 1 1 0 0
## 9         9       1 1 0 0 0
## 10       10       2 0 0 1 1
## 11       11       2 0 0 1 1
## 12       12       1 1 1 0 0
## 13       13       1 1 0 1 0
## 14       14       3 0 1 0 0
## 15       15       3 0 1 0 1
## 16       16       2 0 0 1 1
## 17       17       1 1 0 0 0
## 18       18       3 0 0 0 1
## 19       19       2 0 1 1 1
## 20       20       3 0 1 0 1
```