Georgia Institute of Technology

Karim Akram

January 2025

Cross-validation on a KNN model

Data splitting on a SVM model

K-means clustering model

CV and Clustering

Step (1): calling all the needed libraries

```
# KKNN, KSVM and ggplot2 are already installed
library(ggplot2)
library(kknn)
library(kernlab)

##
## Attaching package: 'kernlab'

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

find("trainControl")

## character(0)
```

Step (2): Loading the data set into our environment

```
data <- read.table("/Users/karimakram/Downloads/hw1/data
2.2/credit_card_data-headers.txt", header = TRUE)

head(data) #to make sure the data is correctly loaded into the environment

##   A1    A2    A3   A8 A9 A10 A11 A12 A14 A15 R1
## 1  1 30.83 0.000 1.25  1   0   1   1 202   0  1
## 2  0 58.67 4.460 3.04  1   0   6   1  43 560  1
## 3  0 24.50 0.500 1.50  1   1   0   1 280 824  1
## 4  1 27.83 1.540 3.75  1   0   5   0 100   3  1
## 5  1 20.17 5.625 1.71  1   1   0   1 120   0  1
## 6  1 32.08 4.000 2.50  1   1   0   0 360   0  1
```

Step(3): Pre-processing layer (best practice)

```
sum(is.na(data))

## [1] 0

as.factor(data[,11])

##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [112] 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

CV and Clustering

```
1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 1
## [260] 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 0 0 0 0 0 0
0 0 0
## [297] 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 1 1
## [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [519] 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [556] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 0 0
## [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
## [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1

#Standardizing the data for better performance""

standardizing <- function(x) { (x-min(x))/(max(x)-min(x))}
data[, 1:(ncol(data) - 1)] <- as.data.frame(lapply(data[, 1:(ncol(data) -
1)], standardizing))


summary(data)

##        A1                A2                A3                A8
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.:0.1328   1st Qu.:0.03714   1st Qu.:0.00579
##  Median :1.0000   Median :0.2212   Median :0.10196   Median :0.03509
##  Mean   :0.6896   Mean   :0.2681   Mean   :0.17252   Mean   :0.07866
##  3rd Qu.:1.0000   3rd Qu.:0.3684   3rd Qu.:0.26562   3rd Qu.:0.09175
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.00000
##        A9               A10               A11               A12
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.0000
##  Median :1.0000   Median :1.0000   Median :0.00000   Median :1.0000
##  Mean   :0.5352   Mean   :0.5612   Mean   :0.03729   Mean   :0.5382
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:0.04478   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000
##       A14               A15                R1
```

```
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.0000
##  1st Qu.:0.03537   1st Qu.:0.00000   1st Qu.:0.0000
##  Median :0.08000   Median :0.00005   Median :0.0000
##  Mean   :0.09004   Mean   :0.01013   Mean   :0.4526
##  3rd Qu.:0.13550   3rd Qu.:0.00399   3rd Qu.:1.0000
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.0000
```

Step(4): Cross-Validation on a KNN model

The code below will do a 4-folds cross-validation with 10 different K (KNN) values to see which K value is best.

```
library(caret)
```

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

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

```
## The following object is masked from 'package:kknn':
##
##     contr.dummy
```

```
#K-fold = 4
training_control <- trainControl(method = "cv", number= 4)

Columns <- colnames(data)

set.seed(123) #to ensure consistency
knn_model <- train(R1~ ., data=data,method ="kknn", trControl =
training_control, tuneLength = 5)
```

```
## Warning in train.default(x, y, weights = w, ...): You are trying to do
## regression and your outcome only has two possible values Are you trying to
do
## classification? If so, use a 2 level factor as your outcome column.
```

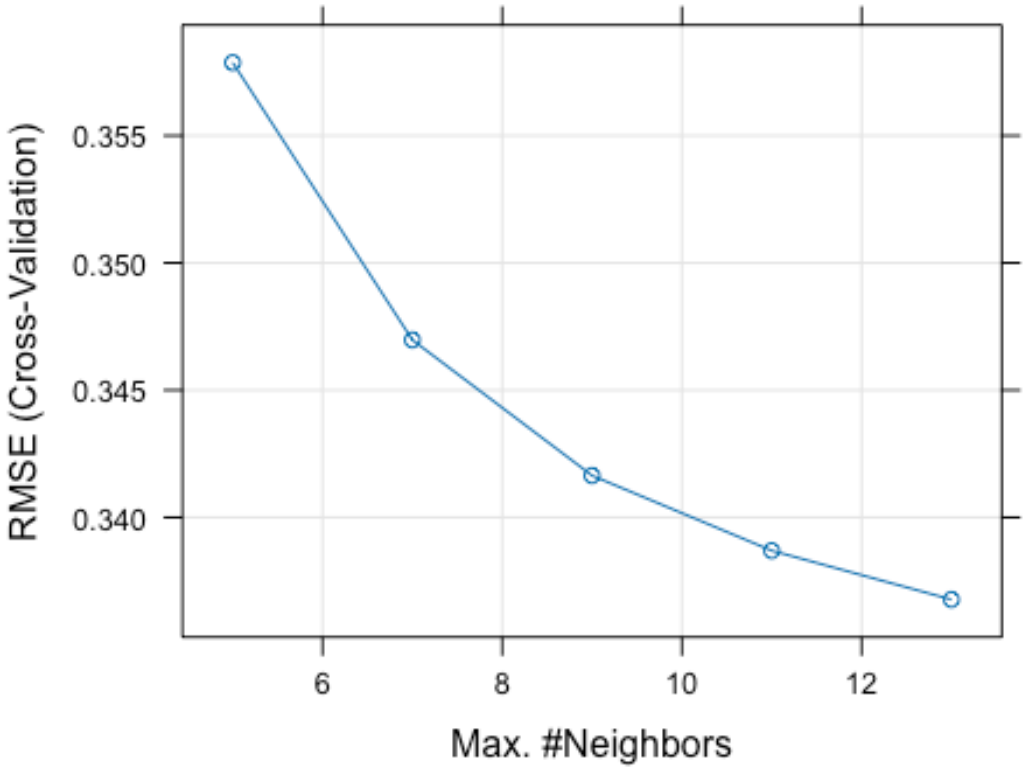
```
print(knn_model)
```

```
## k-Nearest Neighbors
##
## 654 samples
##  10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 491, 491, 490, 490
## Resampling results across tuning parameters:
##
##   kmax  RMSE       Rsquared   MAE
##    5    0.3578703  0.4967512  0.2030142
##    7    0.3469732  0.5194060  0.2050442
```

CV and Clustering

```
##    9    0.3416431  0.5310102  0.2070264
##   11    0.3386920  0.5374214  0.2090453
##   13    0.3367718  0.5416479  0.2110343
##
## Tuning parameter 'distance' was held constant at a value of 2
## Tuning
##  parameter 'kernel' was held constant at a value of optimal
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were kmax = 13, distance = 2 and
kernel
##  = optimal.
```

```
plot(knn_model)
```



Step(5): Splitting the data into training, validation and testing data sets, manually and not using the caret library

```
library(kernlab)

# Set seed for reproducibility
set.seed(123)

n <- nrow(data)
```

CV and Clustering

```
# Manual indices to split the rows of the data set into 3 stages
tr_indx <- sample(1:n, size = round(0.7*n))
remaining_indx <- setdiff(1:n, tr_indx)
valid_indx <- sample(remaining_indx, size = round(0.2*n))
test_indx <- setdiff(remaining_indx, valid_indx)

# Split the data
train_data <- data[tr_indx, ]
valid_data <- data[valid_indx, ]
test_data <- data[test_indx, ]

train_data$R1 <- as.factor(train_data$R1)
valid_data$R1 <- as.factor(valid_data$R1)
test_data$R1  <- as.factor(test_data$R1)
svm_model <- ksvm(R1 ~ ., data= train_data, kernel= "rbfdot", C=1)

valid_prediction <- predict(svm_model, valid_data)
test_prediction <- predict(svm_model, test_data)

valid_accuracy <- sum(valid_prediction == valid_data$R1) /
length(valid_prediction)
test_accuracy <- sum(test_prediction == test_data$R1) /
length(test_prediction)

cat("Validaction accuracy is equal to:", valid_accuracy, "\n")

## Validaction accuracy is equal to: 0.8396947

cat("testing accuracy is equal to:", test_accuracy, "\n")

## testing accuracy is equal to: 0.8461538
```

```
================================================================================
=========================================================
```

```
data(iris)
head(iris)

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa

# I executed this code to check the predictor of columns/predictors we need
to use to check the number of clusters needed, and also load the data :)
```

CV and Clustering

Step(1): Preparing the data

```
data_iris <- iris[,1:4] # using the numerical data only as predictor so we
can cluster them efficiently
iris_scaled <- scale(data_iris) # best practice, for best performance
```

Step(2): Finding the the best number of cluster

I know for a fact that the best number of cluster is probably going to be 3 because the number of species are 3, but we shouldn't assume that the number of classes = the number of clusters. So after a thorough search , I came across the "total within-cluster sum of squares" (Elbow method) to evaluate the clusters.

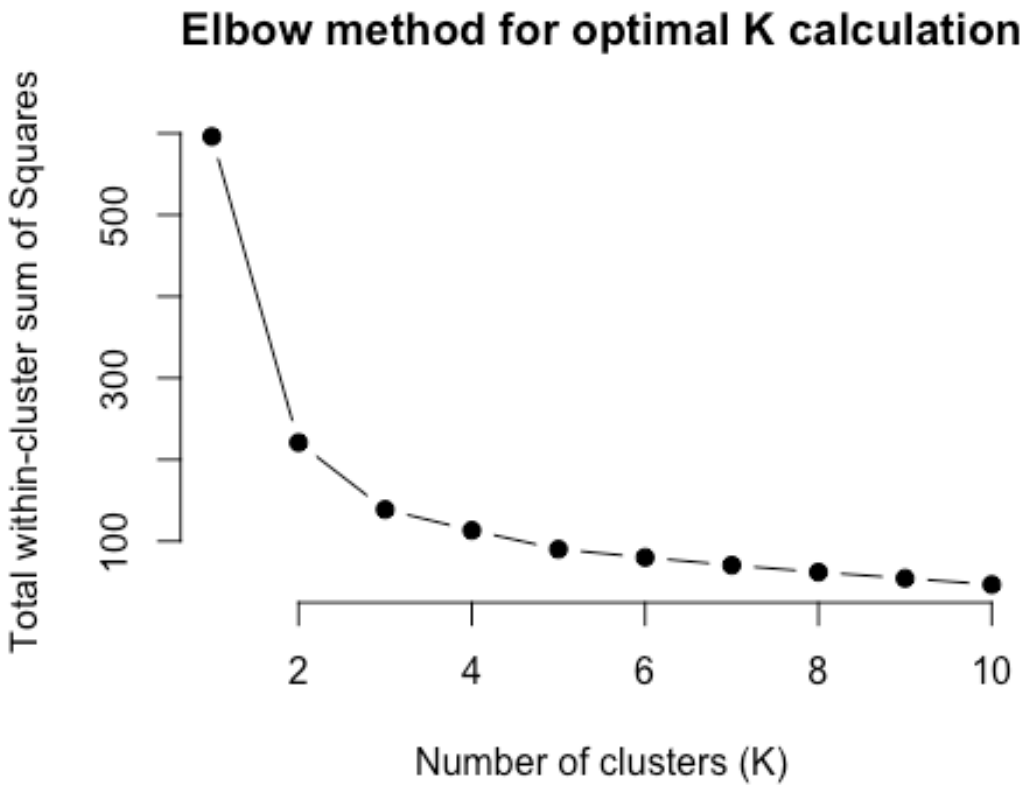
This code is partially generated by Claude/chatgpt

```
wss <- numeric(10)

for (k in 1:10) {
  wss[k] <- kmeans(iris_scaled, centers = k, nstart=25)$tot.withinss
}

plot(1:10, wss, type = "b", pch = 19, frame= FALSE, xlab= "Number of clusters
(K)", ylab= "Total within-cluster sum of Squares", main= "Elbow method for
optimal K calculation")
```

CV and Clustering



So based on the plot above, the elbow point should be = 150 (the sweet spot), and we notice that the WSS reaches 150 when k=3

Step (3): Applying k-means clustering

```
set.seed(123)

kmeans_calc <- kmeans(iris_scaled, centers= 3, nstart=25)
```

Step(5): Evaluating the performance of the clusters

```
table(kmeans_calc$cluster, iris$Species)
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         39        14
##   3      0         11        36
```

Results interpretation

For the Setosa species the accuracy of the clustering is 100% accurate which means that we can distinct it from the others, not the same for Versicolor and Virginica.

CV and Clustering

- After checking the data set, I noticed that the Versicolor and Virginica have almost the same Sepal features. So, this time we're going to use a different feature (Petal Featurs) to test the clusters' effectiveness.

```
iris_Petal <- iris[,3:4]
SCALED_iris_petal <- scale(iris_Petal)

kmeans_calc_2 <- kmeans(SCALED_iris_petal, centers = 3, nstart = 25)

table(kmeans_calc_2$cluster, iris$Species)

##
##     setosa versicolor virginica
##   1      0          2        46
##   2     50          0         0
##   3      0         48         4
```

So after this re-evaluation, it is is obvious the best combination is the Petal features in the iris data set.

- Setosa = 100% accurate
- Versicolor = 96% accurate
- Virginica = 92% Accurate