k-Fold Cross Validation with KNN

Data Divas (Group A)

2024-04-06

KNN Homework

Importing required libraries

```
library(caret)
library(ggplot2)
library(dplyr)
library(factoextra)
library(FNN)
library(reshape2)
```

Exploratory Data Analysis (EDA)

In this section we perform some data exploration activities, such as:

- Quick view of the data summary
- Check for categorical variables and any missing values

Import dataset

```
data = read.csv("wineq_train.csv")
summary(data)
```

```
##  fixed.acidity   volatile.acidity  citric.acid     residual.sugar
##  Min.   : 4.20   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600
##  1st Qu.: 6.30   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.800
##  Median : 6.80   Median :0.2600   Median :0.3200   Median : 5.200
##  Mean   : 6.86   Mean   :0.2791   Mean   :0.3348   Mean   : 6.441
##  3rd Qu.: 7.30   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.:10.000
##  Max.   :14.20   Max.   :1.1000   Max.   :1.0000   Max.   :65.800
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :0.01200   Min.   :  2.00      Min.   :  9.0        Min.   :0.9871
##  1st Qu.:0.03600   1st Qu.: 23.00      1st Qu.:108.0        1st Qu.:0.9917
##  Median :0.04300   Median : 34.00      Median :134.0        Median :0.9938
##  Mean   :0.04562   Mean   : 35.51      Mean   :138.6        Mean   :0.9940
##  3rd Qu.:0.05000   3rd Qu.: 46.00      3rd Qu.:168.0        3rd Qu.:0.9962
##  Max.   :0.29000   Max.   :289.00      Max.   :440.0        Max.   :1.0390
##        pH          sulphates         alcohol         quality
##  Min.   :2.720   Min.   :0.2200   Min.   : 8.40   Min.   :3.000
##  1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.40   1st Qu.:5.000
##  Median :3.180   Median :0.4700   Median :10.40   Median :6.000
##  Mean   :3.186   Mean   :0.4899   Mean   :10.52   Mean   :5.879
```

```
##  3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40   3rd Qu.:6.000
##  Max.   :3.820   Max.   :1.0800   Max.   :14.20   Max.   :9.000
```

We can see that variables like "free.sulfur.dioxide" and "total.sulfur.dioxide" have much higher values than others variables such as "citric.acid" and "volatile.acidity", so it would be a good practice to scale the data.

Checking for categorical predictors and null values

```
str(data)
```

```
## 'data.frame':    3698 obs. of  12 variables:
##  $ fixed.acidity       : num  7 6.3 8.1 7.2 7.2 8.1 7 8.1 8.1 8.6 ...
##  $ volatile.acidity    : num  0.27 0.3 0.28 0.23 0.23 0.28 0.27 0.22 0.27 0.23 ...
##  $ citric.acid         : num  0.36 0.34 0.4 0.32 0.32 0.4 0.36 0.43 0.41 0.4 ...
##  $ residual.sugar      : num  20.7 1.6 6.9 8.5 8.5 6.9 20.7 1.5 1.45 4.2 ...
##  $ chlorides           : num  0.045 0.049 0.05 0.058 0.058 0.05 0.045 0.044 0.033 0.035 ...
##  $ free.sulfur.dioxide : num  45 14 30 47 47 30 45 28 11 17 ...
##  $ total.sulfur.dioxide: num  170 132 97 186 186 97 170 129 63 109 ...
##  $ density             : num  1.001 0.994 0.995 0.996 0.996 ...
##  $ pH                  : num  3 3.3 3.26 3.19 3.19 3.26 3 3.22 2.99 3.14 ...
##  $ sulphates           : num  0.45 0.49 0.44 0.4 0.4 0.44 0.45 0.45 0.56 0.53 ...
##  $ alcohol             : num  8.8 9.5 10.1 9.9 9.9 10.1 8.8 11 12 9.7 ...
##  $ quality             : int  6 6 6 6 6 6 6 6 5 5 ...
```

```
data[is.na(data)]
```

```
## numeric(0)
```

We can see that there are neither categorical predictors nor missing values in the dataset.

Fit KNN on training set and k-fold cross-validation

In this section we perform KNN regression, trying different values of K, first on the whole training data, and then using k-fold cross validation. The procedure is done with respect to the following experimental settings:

- without applying any pre-processing or feature engineering techniques
- applying pre-processing and feature engineering techniques

For each experiment, we compare the plots of the training and k-fold cross validation errors, and we show the value of K (K of KNN) which got the lowest CV error.

Helper functions

The following chunks of code contain the main functions used to perform the experiments

```
#Computes the RMSE for the predicted y's
RMSE <- function(y_true, y_pred){
  return(sqrt(mean((y_true - y_pred)^2)))
}
```

```
# Returns the top_n features with least correlation with
# the response variable "quality"
get_least_corr_vars <- function(X, top_n){
  # Calculate correlation between features and "quality"
  # except for the pair "quality-quality"
  quality_correlation <- cor(X$quality, X[, -which(names(X) == "quality")])
  least_correlated_indexes <- order(abs(quality_correlation))
```

```
  least_correlated_features <- names(X)[least_correlated_indexes[1:top_n]]
  return(least_correlated_features)
}
```

```
# Removes from X the specified features
rm_feature <- function(X,omitted_features){
  X <- X[,!(names(X) %in% omitted_features) ]
  return(X)
}
```

```
#Fits a KNN regression model with the specified K
fit_knn <- function(X_train, y_train, X_test, k_val){
  y_hat = knn.reg(train = X_train, test=X_test, y=y_train, k = k_val)
  y_hat = y_hat$pred
  return(y_hat)
}
```

```
# Fits several KNN regressors, trying different values of K (1-50), on
# the whole training set X. It returns a dataframe which associates to
# each value of K its corresponding RMSE
fit_knn_training_set <- function(X, y, k_range, perform_feature_sel){
  train_rmse <- c()
  if (perform_feature_sel){
    X <- as.data.frame(scale(X))
    omitted_features <- get_least_corr_vars(X, 2)
    X <- rm_feature(X, omitted_features)
  }
  X <- rm_feature(X, c("quality"))

  for (k in k_range){
    y_hat = fit_knn(X, y, X, k)
    train_rmse <- append(train_rmse, RMSE(y, y_hat))
  }

  results_train_rmse <- data.frame(k = k_range, train_rmse=train_rmse)
  return(results_train_rmse)
}
```

```
# Performs k-fold-cross-validation trying several values of
# K (K of KNN)
k_fold_cv <- function(X, y, train_rmse, k_range, perform_feature_sel){
  set.seed(42)
  # we perform 5-fold cross-validation
  folds <- createFolds(y, k = 5, list = TRUE)
  cv_rmse <- c()

  for (k in k_range){
    cv_rmse_fold <- c()
    #iterates through every fold
    for (i in 1:length(folds)) {
      train_indexes <- unlist(folds[-i])
      test_indexes <- unlist(folds[i])

      # splits data using the i-th fold as validation set, and the
      # others folds as training set
```

```
      X_train <- X[train_indexes, ]
      y_train <- y[train_indexes]
      X_test <- X[test_indexes, ]
      y_test <- y[test_indexes]


      if (perform_feature_sel){
        # fits a scaler only on the training set and then applies it
        # on validation set
        preproc <- preProcess(X_train, method = c("center", "scale"))
        X_train <- predict(preproc, X_train)
        X_test <- predict(preproc,X_test)

        X_train <- as.data.frame(X_train)
        X_test <- as.data.frame(X_test)

        # gets the 2 least correlated features with "quality" using ONLY
        # the training set
        omitted_features <- get_least_corr_vars(X_train, 2)
        # remove omitted features from X_train
        X_train <- rm_feature(X_train, omitted_features)
        # remove features also from validation set
        X_test <- rm_feature(X_test, omitted_features)
      }
      X_train <- rm_feature(X_train, c("quality"))
      X_test <- rm_feature(X_test, c("quality"))

      y_hat <- fit_knn(X_train, y_train, X_test, k)
      # accumulates cv RMSE over the folds
      cv_rmse_fold <- append(cv_rmse_fold, RMSE(y_test, y_hat))
    }
    # append to the cv_rmse array the mean error,
    # for the current K (K of KNN), computed over the folds
    cv_rmse <- append(cv_rmse, mean(cv_rmse_fold))
  }
  results_kfold <- data.frame(k = k_range, train_rmse, cv_rmse = cv_rmse)
  return(results_kfold)

}
```

An important comment for the "k_fold_cv" function defined above, which is the heart of the homework, is that any pre-processing and feature engineering on the data should be done ONLY on the training set inside the k-fold cross validation loop. Otherwise, we would face a data leakage problem.

In order to deal with this problem, we introduced a Boolean parameter called "perform_feature_sel", which determines if the function should perform some pre-processing and feature engineering. In our case, if "perform_feature_sel" is TRUE, we scale the data, we detect the 2 features with the least correlation with the response variable "quality", using ONLY the training set X_train, and we remove them both from training and validation set. Then, we fit the KNN model. Note that also the scaling of data is done by first fitting a scaler only on X_train and then applying it to the hold-out set X_test.

The important part is that every data preparation is done only with respect to X_train, without considering X_test, in order to avoid data leakage. Then, the transformation is also applied to X_test.

```r
# Produces the plot of RMSE based on train data and k-fold
# cross-validation as a function of 1/K (K of KNN).
# note that we used log(1/K) as x labels to have a better
# visualization
plot_training_cv_rmse <- function(results, train_rmse, cv_rmse){
  ggplot(results, aes(x = log(1/k), y = train_rmse, color = "Train RMSE")) +
  geom_line() +
  geom_point() +
  geom_line(aes(x = log(1/k), y = cv_rmse, color = "CV RMSE")) +
  geom_point(aes(x = log(1/k), y = cv_rmse, color = "CV RMSE")) +
  scale_x_continuous(labels = scales::scientific_format()) +
  labs(x = "log (1/K) (K of KNN)", y = "RMSE", color = "Data") +
  theme_minimal()
}
```

```r
# Finds the value of K (K of KNN) for which the model has the
# lowest cv error
find_best_k <- function(cv_rmse){
  minimum <- min(cv_rmse)
  print(paste('min cv_rmse:',minimum))
  print(paste('          k:', match( minimum , cv_rmse)))
}
```

KNN and CV without preprocessing and feature engineering

In this experiment we perform k-fold cross validation without any pre-processing and feature engineering.

```r
X <- data
y <- data$quality
k_range = seq(1, 50, by = 1) #We try 50 values for K
```

First, we fit several KNN regressors, trying different values of K (1-50), on the whole training set. We expect to see a RMSE which decreases as the model becomes more flexible.

```r
results_train_rmse <- fit_knn_training_set(X, y, k_range,
                                           perform_feature_sel=FALSE)
train_rmse <- results_train_rmse$train_rmse
head(results_train_rmse,10)
```

```
##     k train_rmse
## 1   1  0.0000000
## 2   2  0.4493701
## 3   3  0.5602351
## 4   4  0.6162105
## 5   5  0.6509800
## 6   6  0.6761983
## 7   7  0.6952180
## 8   8  0.7086050
## 9   9  0.7158486
## 10 10  0.7229473
```

Then, we perform 5-fold cross validation, using the same range (1-50) as before for the parameter K (K of KNN). We expect the CV RMSE to increase as the model becomes more flexible.

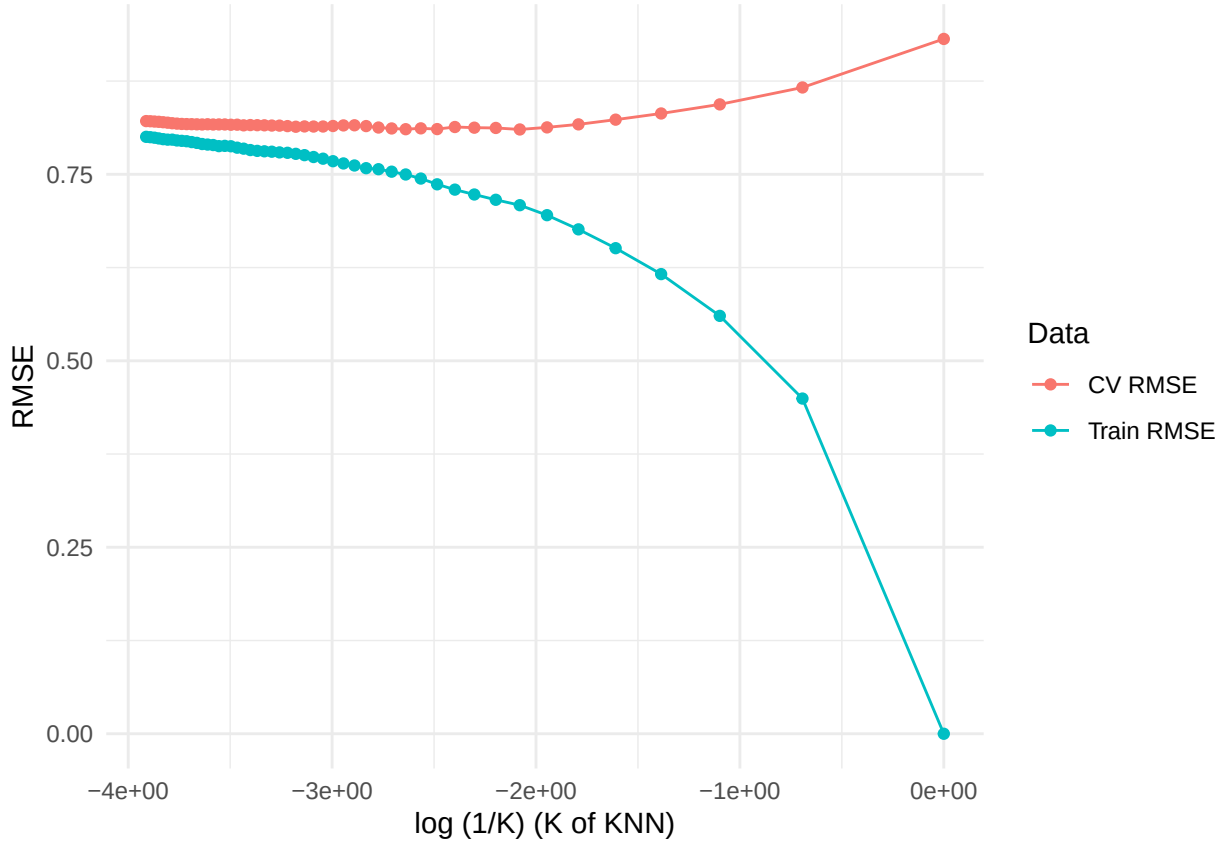
```r
results_kfold <- k_fold_cv(X, y, train_rmse, k_range,
                           perform_feature_sel = FALSE)
```

```
cv_rmse = results_kfold$cv_rmse
head(results_kfold,10)
```

```
##     k train_rmse   cv_rmse
## 1   1  0.0000000 0.9314889
## 2   2  0.4493701 0.8664259
## 3   3  0.5602351 0.8437320
## 4   4  0.6162105 0.8315595
## 5   5  0.6509800 0.8232803
## 6   6  0.6761983 0.8169545
## 7   7  0.6952180 0.8129140
## 8   8  0.7086050 0.8101380
## 9   9  0.7158486 0.8121605
## 10 10  0.7229473 0.8125212
```

Plot

```
plot_training_cv_rmse(results_kfold, train_rmse, cv_rmse)
```



From the plot above, we can see that as the model becomes more flexible, the training error decreases. However, the 5-fold cross validation error increases even if the training error is low. Therefore, in order to avoid overfitting, we should select the model with the lowest CV error, not the one with the lowest training error

```
#It finds K (K of KNN) which got the lowest CV error
find_best_k(cv_rmse)
```

```
## [1] "min cv_rmse: 0.81013795258334"
## [1] "          k: 8"
```

K-fold Cross Validation with pre-processing and feature engineering

In this experiment, we perform 5-fold cross validation with some data pre-processing and feature engineering. The 5-fold cv loop proceeds as follows, for each value of K (K of KNN):

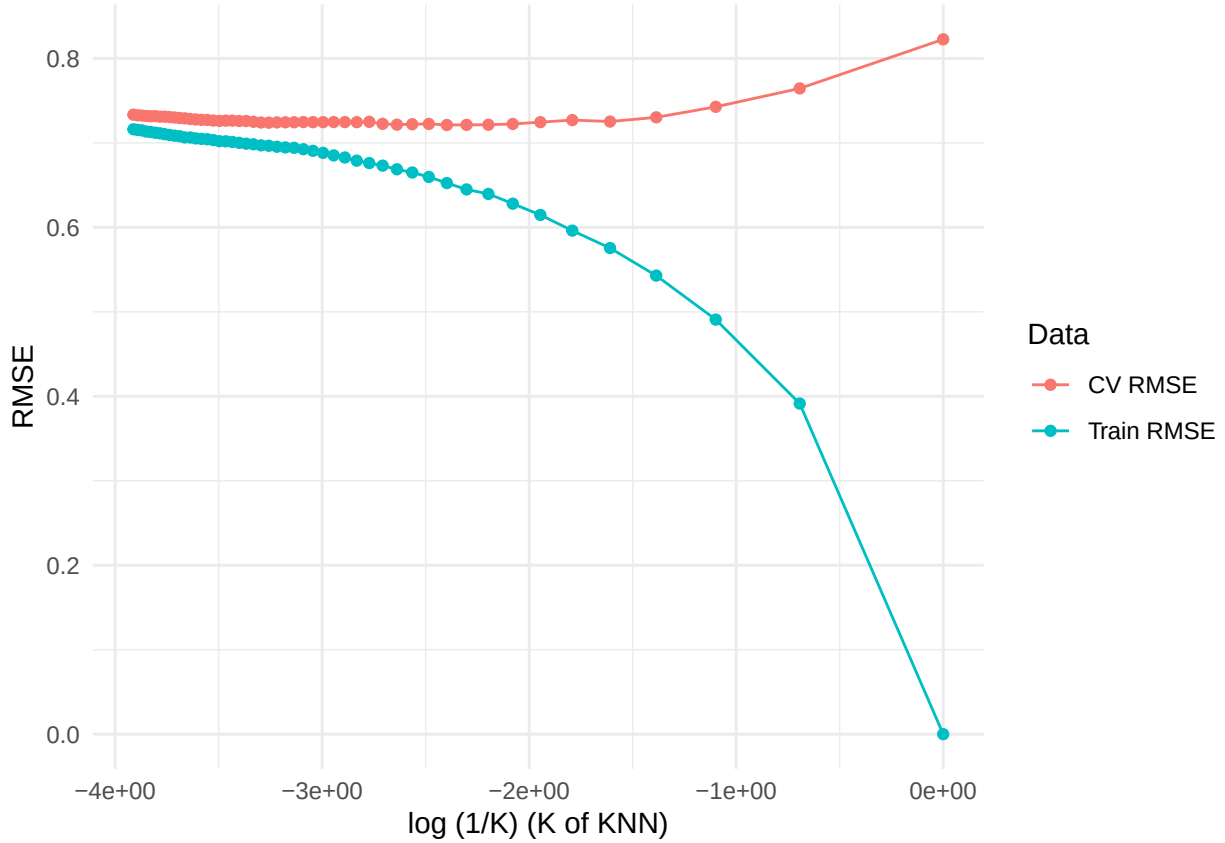
- it creates 5 folds by randomly splitting the dataset
- it splits data using the i-th fold as validation set (X_test), and the others folds as training set (X_train)
- since the parameter "perform_feature_sel" is TRUE, it performs feature selection. In particular, as we said before, it scales the data and detects the 2 features with the least correlation with the response variable "quality", using ONLY the training set (X_train), and it remove them both from training and validation set (X_test). Note that these computations are done using only X_train in order to avoid the data leakage problem, that is, knowledge of the hold-out set leaks into the dataset used to train the model.
- it fits the KNN model on X_train and tests it against X_test
- it accumulates the RMSE error for the current fold

The final RMSE error (for the current K) is computed by averaging the RMSE errors computed over the 5 folds.

```
#Fitting knn on training set
results_train_rmse_fe <- fit_knn_training_set(X, y, k_range,
                                              perform_feature_sel = TRUE)
train_rmse_fe = results_train_rmse_fe$train_rmse

#Fitting knn with k-fold cross validation
results_kfold_fe <- k_fold_cv(X, y, train_rmse_fe, k_range,
                              perform_feature_sel = TRUE)
cv_rmse_fe <- results_kfold_fe$cv_rmse

plot_training_cv_rmse(results_kfold_fe, train_rmse_fe, cv_rmse_fe)
```



We can see that the training and k-fold CV errors show the same behavior as before.

```
# Find best k
find_best_k(cv_rmse_fe)
```

```
## [1] "min cv_rmse: 0.721189712540844"
## [1] "          k: 11"
```

However, the estimated test error is lower in this case.

Conclusions

In this homework we compared the behaviors of the training and k-fold cross validation errors when performing KNN regression with several flexibility degrees. The results show that, as the flexibility of the model increases (or equivalently as the number of neighbors decreases), the training RMSE decreases, while the CV error follows a kind of U-shaped function, with very high values for too flexible models. Note that the shape of the CV error does not strictly match the nice U-function shown in the book. We believe that this is because we only tried 50 values of K (for computational reasons), but if we were to increase the range of K, the CV error would probably increase a lot even for models with low flexibility.

We noticed that scaling the data to have 0 mean and 1 standard deviation, and removing the 2 least correlated features with "quality" improved a lot the model's performances. Note that we performed these pre-processing and feature engineering computations only on the training set inside the k-fold cross validation loop, without looking at the validation set. This avoids the data leakage problem, which could lead to incorrect estimate of the test error.