K-Nearest Neighbours for Predicting Credit Card Eligibility

Michaella

R Markdown

Finding a good classifier using K-Nearest Neighbors

The KNN Algorithm (k-nearest Neighbours) is used in classification – a type of supervised machine learning model.

The purpose of this notebook is to use K-Nearest Neighbours and K-means clustering to make a classification model of those eligible for credit cards. The dataset used contains predictor variables with their corresponding column titles redacted.

Data will be split into training, validation, and test data sets. Then, we will use K-fold cross validation to assess the performance and generalization ability of a predictive model.

```
# ---------------------- SETUP ---------
rm(list = ls())
set.seed(123)
library(kknn)
library(caret)
```

```
## Loading required package: ggplot2
```

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

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

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

```
# Preview of data
data <- read.table("/Users/michaella/Downloads/data_folder/data/cc_data-headers.txt", stringsAsFactors =

head(data)
```

```
##   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
```

```r
# Sample Data: random sampling, partition into 60%. Returns positions
data_pt1 <- createDataPartition (y= data$R1, p=.60, list=F)
str(data_pt1)
```

```
##  int [1:393, 1] 4 5 7 8 10 11 12 13 14 16 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "Resample1"
```

```r
# Training Data : reference positions
train_data <- data[data_pt1,]
dim(train_data)
```

```
## [1] 393  11
```

Our training set has 393 data points, which is 60% of the data.

```r
# Split remaining 40% into validate and test
remaining <- data[-data_pt1,]
data_pt2 <- createDataPartition(y=remaining$R1, p=0.40, list=F)

#Validation data set
validation_set <- remaining[data_pt2,]
dim(validation_set)
```

```
## [1] 105  11
```

Our validation data has 105 observations, which is ~16% of our total data.

```r
# Test set
test_set <- remaining[-data_pt2, ]
dim(test_set)
```

```
## [1] 156  11
```

The test set contains 156 observations, which is ~24% of our total data.

KNN

Now, we can use K-nearest neighbours. - The code performs k-NN classification with different values of `k` (ranging from 1 to 50) on the training and validation sets. - For each `k`, it computes the accuracy of the k-NN model on the validation set and stores the results in `knn_results`. - It identifies the `k` that gives the highest accuracy on the validation set. - The code then evaluates the final k-NN model on the test set and computes the accuracy.

```r
# Model
knn_prediction <- c()
k_list <- c(seq(from=1 ,to=50))
```

```r
for (k in k_list){
  knn_model <- kknn(R1~. ,
                    train_data,
                    validation_set,
                    k= k,
                    kernel = "rectangular",
                    distance = 2,
                    scale = TRUE)
  knn_predict <- as.integer(fitted(knn_model)+0.5)
  knn_prediction[k] <- sum(knn_predict == validation_set$R1)/nrow(validation_set)
}

#Output
knn_results <- do.call(rbind, Map(data.frame, K= k_list, knn_prediction = knn_prediction))

knn_results
```

```
##     K knn_prediction
## 1   1      0.8095238
## 2   2      0.8190476
## 3   3      0.8571429
## 4   4      0.8380952
## 5   5      0.8380952
## 6   6      0.8190476
## 7   7      0.8285714
## 8   8      0.8190476
## 9   9      0.8285714
## 10 10      0.8190476
## 11 11      0.8190476
## 12 12      0.8380952
## 13 13      0.8190476
## 14 14      0.8380952
## 15 15      0.8380952
## 16 16      0.8380952
## 17 17      0.8476190
## 18 18      0.8476190
## 19 19      0.8476190
## 20 20      0.8476190
## 21 21      0.8571429
## 22 22      0.8476190
## 23 23      0.8666667
## 24 24      0.8476190
## 25 25      0.8571429
## 26 26      0.8476190
## 27 27      0.8571429
## 28 28      0.8476190
## 29 29      0.8761905
## 30 30      0.8380952
## 31 31      0.8571429
## 32 32      0.8285714
## 33 33      0.8476190
## 34 34      0.8380952
## 35 35      0.8571429
```

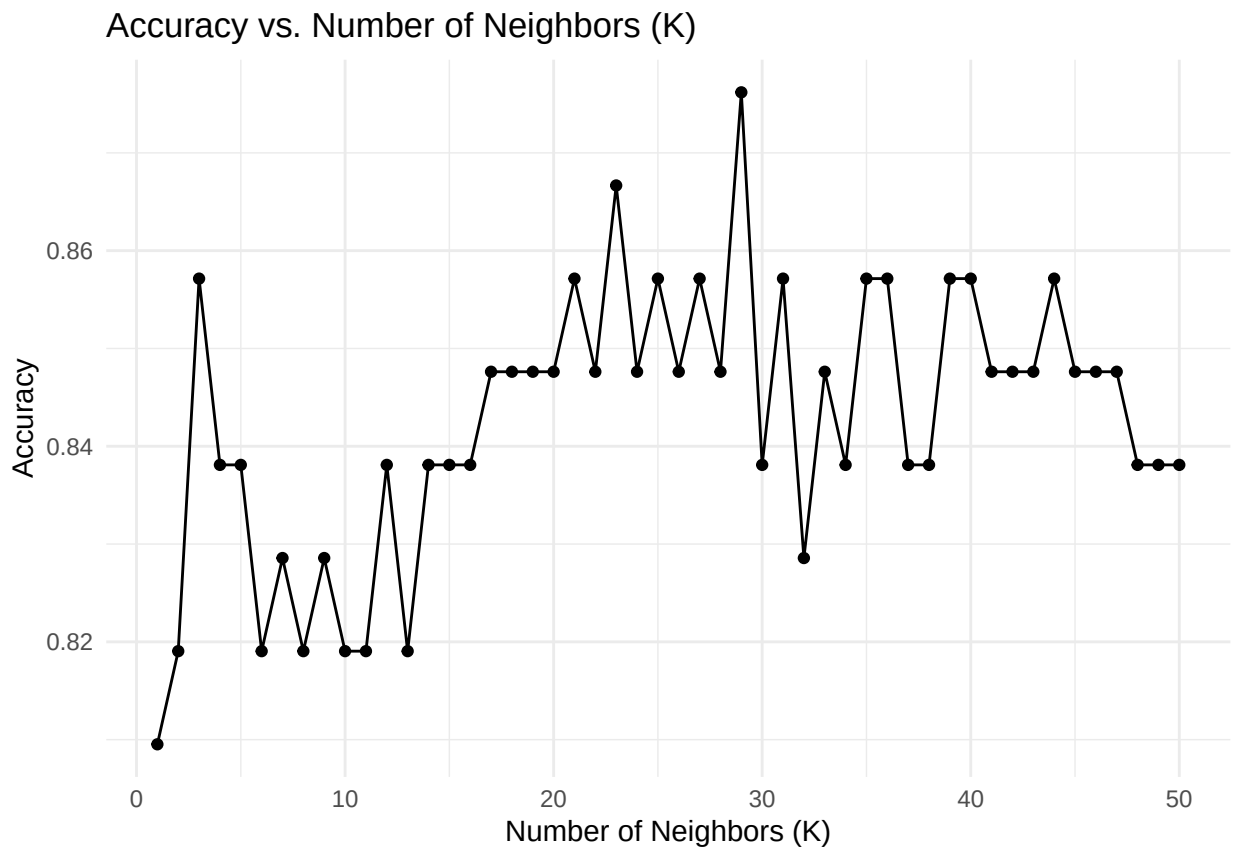
```
## 36 36      0.8571429
## 37 37      0.8380952
## 38 38      0.8380952
## 39 39      0.8571429
## 40 40      0.8571429
## 41 41      0.8476190
## 42 42      0.8476190
## 43 43      0.8476190
## 44 44      0.8571429
## 45 45      0.8476190
## 46 46      0.8476190
## 47 47      0.8476190
## 48 48      0.8380952
## 49 49      0.8380952
## 50 50      0.8380952
```

Let's take a look at the knn_results and their respective performances.

```r
# Plot model performance
library(ggplot2)

# Create a data frame for the results
results_df <- data.frame(K = k_list, Accuracy = knn_results$knn_prediction)

# Create the plot
ggplot(results_df, aes(x = K, y = Accuracy)) +
  geom_line() +
  geom_point() +
  labs(x = "Number of Neighbors (K)", y = "Accuracy") +
  ggtitle("Accuracy vs. Number of Neighbors (K)") +
  theme_minimal()
```



From the plot, k = 29 performed the best.

```
# Max k
max_index <- which.max(knn_results$knn_prediction)
max_k <- knn_results$K[max_index]

max_pred <- knn_results$knn_prediction[max_index]
cat(max_k, max_pred)
```

```
## 29 0.8761905
```

```
# Estimate Model Quality : Accuracy
knn_model_test <- kknn(R1~. ,
                       train_data,
                       test_set,
                       k = max_k,
                       kernel = "rectangular",
                       distance = 2,
                       scale = TRUE)
knn_model_predict <- round(fitted(knn_model_test))


knn_predict_test <- sum(knn_model_predict == test_set$R1) / nrow(test_set)
knn_predict_test
```

```
## [1] 0.8397436
```

```
cat("Mean Accuracy after ", max_k ,"fold cross-validation: ", knn_predict_test, "\n")
```

```
## Mean Accuracy after  29 fold cross-validation:  0.8397436
```

Our best performing k was at K=29 which achieved an accuracy of 83.97%. We will use that number moving forward to cross-validation.

Cross Validation

The standard value of k in k-fold cross validation is 10, so we will do the same.This approach helps assess the model's performance under different data splits, providing a more robust evaluation than using a single train-test split.

Here's a breakdown of the steps:

- The variable `k` is set to 10, indicating 10-fold cross-validation will be performed.
- The data is split into `k` folds using the `createFolds` function.
- A loop runs `k` times, each time using one fold as the test set and the remaining folds as the training set.
- The k-NN model is trained on the training data using the previously determined optimal `k` value (`max_k`).
- Predictions are made on the test set using the trained model.
- Accuracy is calculated for each fold, measuring the proportion of correct predictions.
- The mean accuracy across all folds is computed and displayed at the end.

```
k <- 10
folds <- createFolds(data$R1, k = k)
performance_metrics <- numeric(k)

for (i in 1:k) {

  #Split data into train and test based on folds
  train_index <- unlist(folds[-i])
  test_index <- unlist(folds[i])

  cv_train_data <- data[train_index, ]
  cv_test_data <- data[test_index, ]

  #Train model using our k=29 value
  cv_knn_model <- kknn(R1~.,
                       cv_train_data,
                       cv_test_data,
                       k=max_k,
                       kernel = "rectangular",
                       distance = 2,
                       scale = TRUE)

  # Predict
  cv_knn_predict <- round(fitted(cv_knn_model) )

  # Performance check
  cv_accuracy <- sum(cv_knn_predict == cv_test_data$R1) /nrow(cv_test_data)
```

```
  performance_metrics[i] <- cv_accuracy
}

mean_performance <- mean(performance_metrics)
cat("Mean Accuracy after k=10 fold cross-validation: ", mean_performance, "\n")
```

```
## Mean Accuracy after k=10 fold cross-validation:  0.8379021
```

Our model produced a reasonably good predicting performance, resulting in a Mean Accuracy of 0.8379021. This suggests that, on average, the model correctly predicted the target variable for approximately 83.79% of the instances during the 10-fold cross-validation process.