DTS301TC_32_1931097

Jiayuan Zhu

10/13/2022

T3

T3-1

The data set was first divided into a training set and a test set with a 4:1 ratio. The KNN model was trained on the training set, and then predictions were made on the test set. The output presents the evaluation of the predicted results with the true values.

```
library(caret)
```

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

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

```
library(readr)

# Load data set
data <- read_csv("cleaned_ford.csv")
```

```
## Rows: 11859 Columns: 29
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (29): model_B-MAX, model_C-MAX, model_EcoSport, model_Edge, model_Fiesta...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Split into Train set and Test set
set.seed(1234)
sample <- sample(c(TRUE, FALSE), nrow(data), replace=TRUE, prob=c(0.8, 0.2))
train <- data[sample, ]
test <- data[!sample, ]

# Train
model <- train( price ~ .,
                data = train,
                method = 'knn' )

# Predict
knn_output <- predict( model, test )

# Evaluate
postResample(knn_output, test$price)
```

```
##         RMSE     Rsquared          MAE
## 1430.7412472    0.8789603   916.0035292
```

T3-2

K-Fold was set for validation. And the output shows the evaluation.

```
# Using K fold with k = 5
train_control <- trainControl( method = 'cv', number = 5 )

# Train
model <- train( price ~ .,
                data = train,
                method = 'knn',
                trControl = train_control )

# Predict
knn_output <- predict( model, test )

# Evaluate
postResample(knn_output, test$price)
```

```
##        RMSE    Rsquared         MAE
## 1373.242984    0.888446  911.249765
```

T3-3

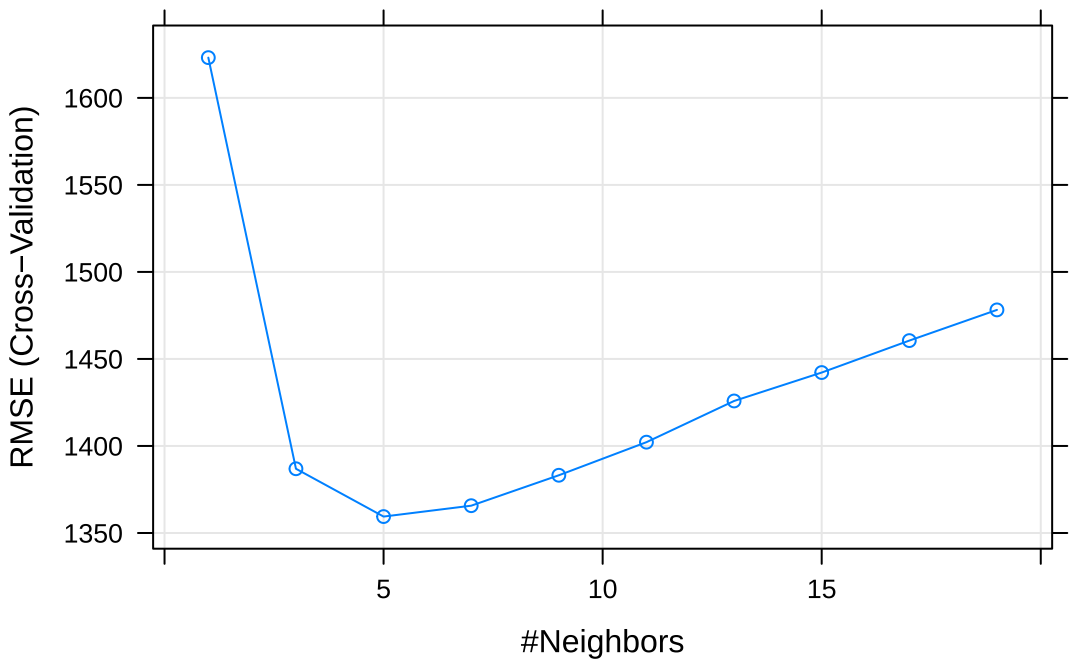
We set Gird Search for tuning parameters for KNN model and K-fold for validation. The first output demonstrates the performance of different parameters on the validation set. And the second output plots the tuning parameters and the corresponding RMSE. It shows that the KNN model has the best performance with k = 7.

```
# Using K fold with k = 5
train_control <- trainControl( method = 'cv', number = 5 )

# Set grid for grid search for tuning parameters
grid <- expand.grid( k = seq( from = 1, to = 20, by = 2 ) )

# Train
model <- train( price ~ .,
                data = train,
                method = 'knn',
                metric = "RMSE",
                trControl = train_control,
                tuneGrid = grid )

# Print training results with different parameter settings
model
```

```
## k-Nearest Neighbors
##
## 9490 samples
##   28 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
```

```
## Summary of sample sizes: 7592, 7592, 7591, 7592, 7593
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    1  1623.125  0.8541437  1114.4359
##    3  1386.904  0.8900029   968.2366
##    5  1359.435  0.8936579   947.8888
##    7  1365.685  0.8924441   944.8194
##    9  1383.165  0.8896415   953.8416
##   11  1402.185  0.8866741   963.5791
##   13  1425.864  0.8829011   978.1860
##   15  1442.195  0.8803278   990.1155
##   17  1460.549  0.8773690  1001.6929
##   19  1478.189  0.8745418  1013.3875
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 5.
```

```
# Plot the performance with different parameter settings
plot(model)
```



T3-4

The first output shows the performance of model on the test set. And the prediction took 0.3806589 secs.

```
# Using K fold with k = 5
train_control <- trainControl( method = 'cv', number = 5 )

grid <- expand.grid( k = 7 )

# Train
S_Train_time <- proc.time()
model <- train( price ~ .,
                data = train,
```

```
                method = 'knn',
                metric = "RMSE",
                trControl = train_control,
                tuneGrid = grid )
E_Train_time <- proc.time()
# Training Time
E_Train_time - S_Train_time
```

```
##    user  system elapsed
##   1.438   0.007   1.447
```

```
# Predict
S_Pred_time <- proc.time()
knn_output <- predict( model, test )
E_Pred_time <- proc.time()
# Predicting Time
E_Pred_time - S_Pred_time
```

```
##    user  system elapsed
##   0.385   0.007   0.384
```

```
# Evaluation predictions
postResample(knn_output, test$price)
```

```
##         RMSE     Rsquared          MAE
## 1382.7269013    0.8869048  905.0055542
```

T4

T4-1

The KNN model first finds the neighbours of the test set in the feature space to predict. Then the mean of the neighbours' labels is calculated as the predicted values for the test set.

For ease of illustrating and visualize how the KNN model is used for the task, we select 3 features, namely 'mileage', 'mpg', and 'year', as examples. Also, for ease of illustration of the predictions, only one row of data was left for testing.

```
library(dplyr, warn.conflicts = F)

sample_train <- train %>% select(price, mileage, mpg, year)

sample_test <- test %>% select(price, mileage, mpg, year)
sample_test <- sample_test[1,]
```

The scatter plot of features on the training set was plotted in Figure 1.

```
# Plot training data feature distribution
library(plotly, warn.conflicts = FALSE)

fig <- plot_ly(sample_train,
               x = sample_train$mileage,
               y = sample_train$mpg,
               z = sample_train$year,
               marker = list(size = 0.3),
               color = I("blue"),
               type="scatter3d",
               mode = "markers",
               name = "Training Data")
fig
```

The feature of the data to be predicted was plotted in red, as shown in Figure.2.

```
# Plot training data and test feature distribution
fig <- fig %>% add_trace(sample_test,
                         x = sample_test$mileage,
                         y = sample_test$mpg,
                         z = sample_test$year,
                         marker = list(size = 2),
                         color = I("red"),
                         mode = "markers",
                         type="scatter3d",
                         name = "Test Data")
fig
```

In order to predict the value of the data, the KNN model needs to find the k nearest neighbours on the feature map to the point to be predicted. Here, we set $k = 7$ as example.

```
# K nearest neighbours
k <- 7
# Compute Distance
train_distance <- sample_train %>% mutate( distance = (sample_test$mileage - c(mileage)) ^ 2 +
                                           (sample_test$mpg - c(mpg)) ^ 2 +
                                           (sample_test$year - c(year)) ^ 2 ) %>%
  arrange(distance)
```

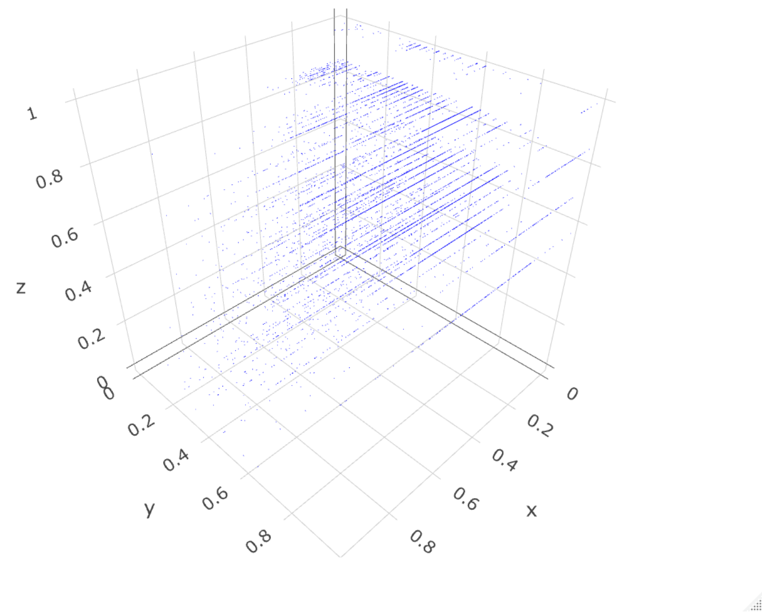


Figure 1: Training Data Feature istribution

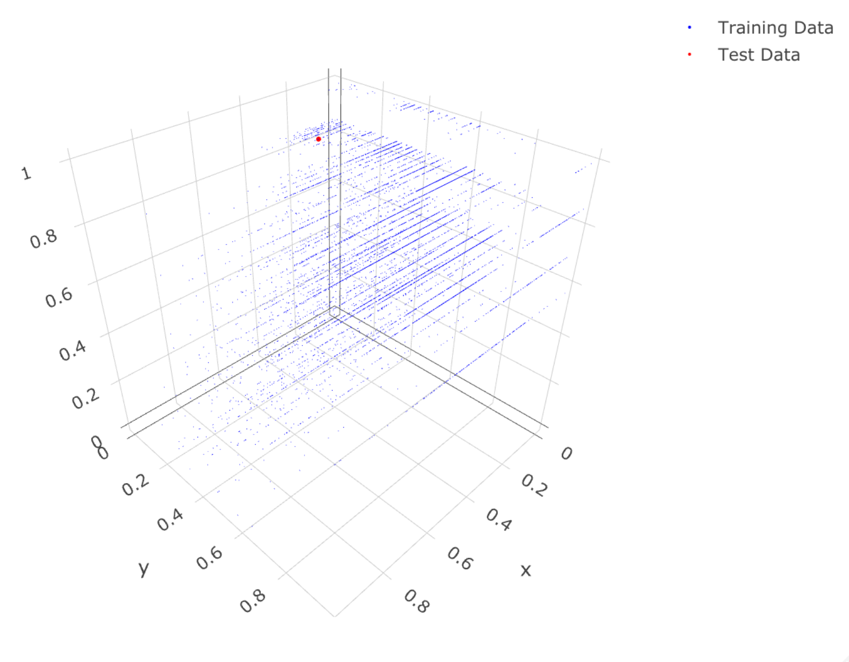


Figure 2: Training and Test Data Feature istribution

```r
# Find K nearest
neighbours <- train_distance[c(1:k),]
```

Then, we plot the neighbours on the figure, as shown in Figure 3. We can see the neighbours are around the test data.

```r
# Plot K nearest neighbours
fig <- fig %>% add_trace(neighbours,
                         x = neighbours$mileage,
                         y = neighbours$mpg,
                         z = neighbours$year,
                         marker = list(size = 1.5),
                         color = I("green"),
                         mode = "markers",
                         type="scatter3d",
                         name = "Neighbours")
fig
```

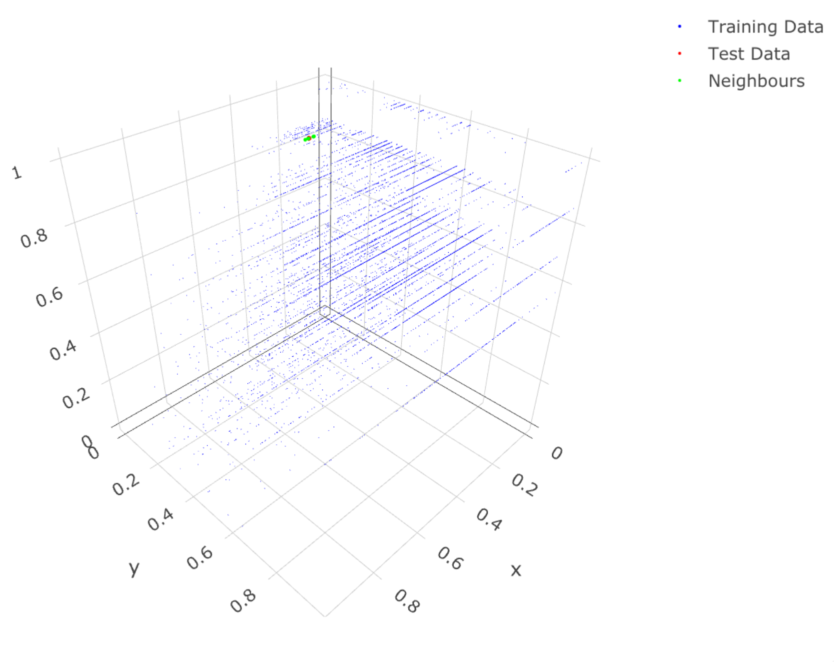


Figure 3: Neighbours Feature Distribution

Finally, predicting the test set data by calculating the mean value of the neighbours and evaluate with the true values. That's how KNN model is used for regression tasks.

```r
pred = mean(neighbours$price)
pred
```

```
## [1] 15465.29
```

```r
postResample(pred, sample_test$price)
```

```
##     RMSE Rsquared      MAE
## 1034.714       NA 1034.714
```

T4-2

	RMSE	Rsquared	MAE	Training Time
KNN	1382.727	0.887	905.006	1.35
SVM	1728.017	0.834	1271.253	25.15
Random-Forest	1258.108	0.902	694.289	87.30

The performance of the three models is written in the table above. We can find that RMSE, Rsquared, and MAE all show the highest accuracy for RF and the lowest for SVM. In terms of running time, KNN is significantly faster than the other two models, and SVM is significantly better than RF.

Since KNN requires essentially no training, it takes very little time to train. And KNN model is more suitable for large data sets with few features, because large data sets can have a more dense feature space distribution to find closer neighbours and thus get more accurate prediction results. And this dataset meets these requirements, so KNN is highly accurate on it.

T4-3

Advantages

1. Since KNN requires essentially no training, it is much faster than other models.
2. As the KNN algorithm does not require training, new data can be added seamlessly and this does not affect the accuracy of the algorithm.
3. KNN is very easy to implement.

Disadvantages

1. KNN is very sensitive to noise in the dataset, since it has not been able to handle noise.
2. KNNs are not able to handle datasets with many features because the error in the prediction results increases as the dimensionality increases.