mobile-price-prediction

Henry Xia

2025-04-29

1) Executive summary

The original data set is sourced from the website source: https://www.kaggle.com/datasets/vinothkannaece/mobiles-and-laptop-sales-data, which is a zip format.

The original dataste has been unzipped, transformed as a Rdata file, and included at a github repository, https://github.com/Henryisagoodguy/EDX-Capstone-Project-Mobile-Price-prediction/blob/main/dat_raw.Rdata, please download the file to current working directory as a file named dat_raw.

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://c
ran.us.r-project.org")

## 载入需要的程序包：tidyverse

## ── Attaching core tidyverse packages ──────────────────────── tidyve
rse 2.0.0 ──
## ✔ dplyr     1.1.4     ✔ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.5.1     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.1
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_co
nflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to fo
rce all conflicts to become errors

if(!require(caret)) install.packages("caret", repos = "http://cran.us.r
-project.org")

## 载入需要的程序包：caret
## 载入需要的程序包：lattice
##
## 载入程序包：'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift

if(!require(gam)) install.packages("gam", repos = "http://cran.us.r-pro
ject.org")
```

```
## 载入需要的程序包：gam
## 载入需要的程序包：splines
## 载入需要的程序包：foreach
##
## 载入程序包：'foreach'
##
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
##
## Loaded gam 1.22-5
```

```
if(!require(rpart)) install.packages("rpart", repos = "http://cran.us.r
-project.org")
```

```
## 载入需要的程序包：rpart
```

```
if(!require(randomForest)) install.packages("randomForest", repos = "ht
tp://cran.us.r-project.org")
```

```
## 载入需要的程序包：randomForest
## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
##
## 载入程序包：'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(tidyverse)
library(caret)
library(gam)
library(rpart)
library(randomForest)


load("dat_raw.Rdata")
dim(dat_raw)
```

```
## [1] 50000    16
```

```
head(dat_raw)
```

```
## # A tibble: 6 × 16
##   Product      Brand  `Product Code` Product Specificatio…¹  Price `
Inward Date`
```

```
##   <chr>        <chr>  <chr>          <chr>                      <dbl> <
date>
## 1 Mobile Phone Motor… 88EB4558       Site candidate activi…   78570 2
023-08-02
## 2 Laptop       Oppo   416DFEEB       Beat put care fight a…   44613 2
023-10-03
## 3 Mobile Phone Samsu… 9F975B08       Energy special low se… 159826 2
025-03-19
## 4 Laptop       Sony   73D2A7CC       Friend record hard co…   20911 2
024-02-06
## 5 Laptop       Micro… CCE0B80D       Program recently feel…   69832 2
023-08-10
## 6 Laptop       HP     10A5C53D       Recognize happen midd… 190474 2
025-03-19
## # ℹ abbreviated name: ¹ `Product Specification`
## # ℹ 10 more variables: `Dispatch Date` <date>, `Quantity Sold` <db
l>,
## #   `Customer Name` <chr>, `Customer Location` <chr>, Region <chr>,
## #   `Core Specification` <chr>, `Processor Specification` <chr>, RAM
 <chr>,
## #   ROM <chr>, SSD <chr>
```

This is a data set for laptop and mobile sales with 50000 rows and 16 columns, I use some portion of this data set to train several algorithms to predict mobile price with features of brand, RAM and ROM, and select the best algorithm base on rmse score

```
RMSE <- function(true_price, predicted_price){
  sqrt(mean((true_price - predicted_price)^2))
}
```

2) methods/analysis

Preprocessing: clean data to remove unuseful data, keep data for mobile and features of Brand, RAM, and ROM.

```
dat <- dat_raw %>% filter(Product == "Mobile Phone") %>% select(Brand,
Price, RAM, ROM)
head(dat)

## # A tibble: 6 × 4
##   Brand     Price RAM   ROM
##   <chr>     <dbl> <chr> <chr>
## 1 Motorola  78570 12GB  128GB
## 2 Samsung  159826 8GB   256GB
## 3 Dell      11670 16GB  256GB
## 4 Motorola 174698 6GB   64GB
## 5 Apple     51251 4GB   256GB
## 6 Toshiba  179710 6GB   512GB

dim(dat)
```

```
## [1] 24983     4
```

Generate data sets for train and test

```
library(caret)
set.seed(1)
test_index <- createDataPartition(dat$Price, times = 1, p = 0.2, list =
 FALSE)
temp <- dat[test_index, ]
train_set <- dat[-test_index, ]
test_set <- temp %>% semi_join(train_set, by = "Brand") %>% semi_join(t
rain_set, by ="RAM") %>% semi_join(train_set, by = "ROM")


removed <- anti_join(temp, test_set)

## Joining with `by = join_by(Brand, Price, RAM, ROM)`

train_set <- rbind(train_set, removed)
```

2.1) Use group average price as prediction. I calculate average price for each group of combinations of Brand, ROM and RAM at train set, and use the average prices as predicted price for each groups in test set.

```
mu <- train_set %>% group_by(Brand, RAM, ROM) %>% summarize(mu = mean(P
rice))

## `summarise()` has grouped output by 'Brand', 'RAM'. You can override
 using the
## `.groups` argument.

predicted_price_mu <- test_set %>% left_join(mu, by=c("Brand", "RAM", "
ROM")) %>% pull(mu)


mu_rmse <- RMSE(test_set$Price, predicted_price_mu)


mu_rmse

## [1] 57618.94

mean(test_set$Price)

## [1] 102488.3
```

mu_rmse is 57618.94, the average price of test_set is 102488.3.

2.2) Use KNN model to predict price.

```
train_knn <- train(Price ~ ., method = "knn", data = train_set)


predicted_price_knn <- predict(train_knn, test_set, type = "raw")
```

```
knn_rmse <- RMSE(test_set$Price, predicted_price_knn)


knn_rmse

## [1] 57618.94
```

knn_rmse is 57618.94, sames as mu_rmse.

2.3) fine tune KNN model, I have tested several span of data frame to select the best K value, it turned out the best K value shall fall at somewhere between 50 and 70:

```
data.frame(k = seq(50, 70, 5))

##    k
## 1 50
## 2 55
## 3 60
## 4 65
## 5 70

train_knn_tunegrid <- train(Price ~ ., method = "knn", data = train_set,
 tuneGrid = data.frame(k = seq(50, 70, 5)))

## Warning: predictions failed for Resample02: k=50 Error in knnregTrai
n(train = structure(c(0, 0, 0, 0, 0, 0, 0, 1, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample02: k=55 Error in knnregTrai
n(train = structure(c(0, 0, 0, 0, 0, 0, 0, 1, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample02: k=60 Error in knnregTrai
n(train = structure(c(0, 0, 0, 0, 0, 0, 0, 1, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample02: k=65 Error in knnregTrai
n(train = structure(c(0, 0, 0, 0, 0, 0, 0, 1, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample02: k=70 Error in knnregTrai
n(train = structure(c(0, 0, 0, 0, 0, 0, 0, 1, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample05: k=50 Error in knnregTrai
n(train = structure(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  :
##   too many ties in knn
```

```
## Warning: predictions failed for Resample05: k=55 Error in knnregTrai
n(train = structure(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample05: k=60 Error in knnregTrai
n(train = structure(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample05: k=65 Error in knnregTrai
n(train = structure(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample05: k=70 Error in knnregTrai
n(train = structure(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample16: k=50 Error in knnregTrai
n(train = structure(c(0, 0, 1, 1, 0, 0, 0, 0, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample16: k=55 Error in knnregTrai
n(train = structure(c(0, 0, 1, 1, 0, 0, 0, 0, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample16: k=60 Error in knnregTrai
n(train = structure(c(0, 0, 1, 1, 0, 0, 0, 0, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample16: k=65 Error in knnregTrai
n(train = structure(c(0, 0, 1, 1, 0, 0, 0, 0, 0, 0,  :
##   too many ties in knn

## Warning: predictions failed for Resample16: k=70 Error in knnregTrai
n(train = structure(c(0, 0, 1, 1, 0, 0, 0, 0, 0, 0,  :
##   too many ties in knn

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
trainInfo,
## : There were missing values in resampled performance measures.
```

```
predicted_price_knntunegird <- predict(train_knn_tunegrid, test_set, ty
pe = "raw")


knntunegrid_rmse <- RMSE(test_set$Price, predicted_price_knntunegird)


knntunegrid_rmse
```

```
## [1] 56782.35
```

```
train_knn_tunegrid$bestTune
```

```
##    k
## 4 65

train_knn_tunegrid$finalModel

## 65-nearest neighbor regression model
```

After tuning grid, the knntunegrid_rmse is 56782.4 less than 57618.94, the best K value is 65.

2.4) Adjust cross validation parameters to speed up computation.

```
control <- trainControl(method = "cv", number = 10, p = .9)


train_knn_cv <- train(Price ~ ., method = "knn", data = train_set, tune
Grid = data.frame(k = seq(50, 80, 10)), trControl = control)


predicted_price_cv <- predict(train_knn_cv, test_set, type = "raw")


knncv_rmse <- RMSE(test_set$Price, predicted_price_cv)


knncv_rmse

## [1] 56782.35
```

This knnCV rmse score is same as the rmse of knn tunegrid model, but it took significant less time to run.

2.5) Use Loess model to predict price

```
library(gam)
grid <- expand.grid(span = seq(0.15, 0.65, len = 10), degree = 1)


train_loess <- train(Price ~ ., method = "gamLoess", tuneGrid = grid, d
ata = train_set)


predicted_price_loess <- predict(train_loess, test_set, type = "raw")


loess_rmse <- RMSE(test_set$Price, predicted_price_loess)


loess_rmse

## [1] 56600.13
```

The rmse score is further down to 56600.13 with Loess model.

2.6) Use decision tree model to predict price

```
library(rpart)

train_dt <- train(Price ~ ., method = "rpart", tuneGrid = data.frame(cp
 = seq(0.1, 1, len = 25)), data = train_set)

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
trainInfo,
## : There were missing values in resampled performance measures.

predicted_price_dt <- predict(train_dt, test_set, type = "raw")

dt_rmse <- RMSE(test_set$Price, predicted_price_dt)

dt_rmse

## [1] 56556.19
```

rmse score with decision tree model is 56556.19, very close to Loess model (56600.13), lower than knn models.

2.7) predict prices with random forest model

```
library(randomForest)

set.seed(1)

train_rf <- randomForest(Price ~ ., data = train_set)

predicted_price_rf <- predict(train_rf, test_set)

rf_rmse <- RMSE(test_set$Price, predicted_price_rf)

rf_rmse

## [1] 56679.02
```

The rmse score with random forest model is 56679.02, close to score of Loess model (56600.13) and decision tress model(56556.19), but this code Chunk can be run in seconds., the fastest model.

2.8) use cross validation to choose parameter to optimize random forest model.

```
set.seed(1)

train_rf_2 <- train(Price ~ ., method = "Rborist", tuneGrid = data.fram
e(predFixed = 2, minNode = c(3, 50)), data = train_set)
```

```r
predicted_price_rf_2 <- predict(train_rf_2, test_set)

rf_rmse_2 <- RMSE(test_set$Price, predicted_price_rf_2)

rf_rmse_2
## [1] 56685.96
```

We get the score of 56685.96 with random forest cross validation model.

3) Result:

I tried 8 models (group average, KNN, KNN tunegrid, KNN cross validation, Loess, decision tree, random forest, random forest with tuned parameter) to make prediction, and use rmse score from dataset of test set to compare performance.

The decision tress model performs best among all models with rmse score of 56556, Loess is the 2nd best model with rmse score of 56600..

4) Conclusion

The original data set is a spreadsheet with 16 columns and 50000 rows, I used portion of it to train a model to predict prices of mobile with features of brand, RAM and ROM, so use total 3 features to predict price. RMSE from dataset of test set is the measurement for evaluating performance of models, I run 8 algorithms, it turned out the a decision tree algorithm produce best score, and Loess model perform 2nd best, a confusing issue is that the tune parameter at random forest model that generate best rmse score at train_set dataset may not generate best rmse score at test_set dataset.

5) reference

EDX HarvardX PH125 Data Science