KNN Modelling and Decision Trees: The Case of NANSE Sales Data

2023-12-05

KNN Modeling

Initial loading of data, packages, and functions

```
my_confusion_matrix <- function(cf_table) {
  true_positive <- cf_table[4]
  true_negative <- cf_table[1]
  false_positive <- cf_table[2]
  false_negative <- cf_table[3]
  accuracy <- (true_positive + true_negative) / (true_positive + true_negative + false_positive + false_negative)
  sensitivity_recall <- true_positive / (true_positive + false_negative)
  specificity_selectivity <- true_negative / (true_negative + false_positive)
  precision <- true_positive / (true_positive + false_positive)
  neg_pred_value <- true_negative/(true_negative + false_negative)
  print(cf_table)
  my_list <- list(sprintf("%1.0f = True Positive (TP), Hit", true_positive),
                  sprintf("%1.0f = True Negative (TN), Rejection", true_negative),
                  sprintf("%1.0f = False Positive (FP), Type 1 Error", false_positive),
                  sprintf("%1.0f = False Negative (FN), Type 2 Error", false_negative),
                  sprintf("%1.4f = Accuracy (TP+TN/(TP+TN+FP+FN))", accuracy),
                  sprintf("%1.4f = Sensitivity, Recall, Hit Rate, True Positive Rate (How many positives did the
model get right? TP/(TP+FN))", sensitivity_recall),
                  sprintf("%1.4f = Specificity, Selectivity, True Negative Rate (How many negatives did the model
get right? TN/(TN+FP))", specificity_selectivity),
                  sprintf("%1.4f = Precision, Positive Predictive Value (How good are the model's positive predic
tions? TP/(TP+FP))", precision),
                  sprintf("%1.4f = Negative Predictive Value (How good are the model's negative predictions? TN/
(TN+FN)", neg_pred_value)
  )
  return(my_list)
}

library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.2     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.4.4     ✔ tibble    3.2.1
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
df <- read_rds(r"(mod6HE_logit.rds)")
```

Preprocessing data for KNN

```
ColumnsNotUsedKNN <- df %>%
  select(store, week, region, high_med_rev, high_med_gp, high_med_gpm)

knn1 <- df %>%
  mutate(high_med_units = factor(if_else(high_med_units==1, 'high', 'low'), levels=c('low', 'high')))

knn1 <- knn1 %>%
  select(high_med_units, size, region,
         promo_units_per,
         altbev_units_per, confect_units_per, salty_units_per,
         velocityA_units_per, velocityB_units_per, velocityC_units_per, velocityD_units_per, velocityNEW_units_pe
r)

library(fastDummies)
```

```
## Thank you for using fastDummies!
```

```
## To acknowledge our work, please cite the package:
```

```
## Kaplan, J. & Schlegel, B. (2023). fastDummies: Fast Creation of Dummy (Binary) Columns and Rows from Categoric
al Variables. Version 1.7.1. URL: https://github.com/jacobkap/fastDummies, https://jacobkap.github.io/fastDummie
s/.
```

```
knn1 <- fastDummies::dummy_cols(knn1, select_columns = c("region"), remove_selected_columns=T)
```

Partitioning the data

```
library(caret)
```

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

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

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
set.seed(42)
partition <- caret::createDataPartition(y=knn1$high_med_units, p=.75, list=FALSE)
data_train <- knn1[partition, ]
data_test <- knn1[-partition, ]

X_train <- data_train %>%
  select(-high_med_units)

X_test <-  data_test %>%
  select(-high_med_units)

y_train <- data_train$high_med_units

y_test <- data_test$high_med_units
```

Standardizing variables using z-score standardization

```
X_train <- scale(X_train)
X_test <- scale(X_test)
```

Running the analysis to make the prediction

```
library(class)

knn_prediction = class::knn(train=X_train, test=X_test, cl=y_train, k=43)
```

Checking Accuracy by Confusion matrix

```
table2 <- table(knn_prediction, y_test)
my_confusion_matrix(table2)
```

```
##               y_test
## knn_prediction  low high
##           low   966  222
##           high  294 1035
```

```
## [[1]]
## [1] "1035 = True Positive (TP), Hit"
##
## [[2]]
## [1] "966 = True Negative (TN), Rejection"
##
## [[3]]
## [1] "294 = False Positive (FP), Type 1 Error"
##
## [[4]]
## [1] "222 = False Negative (FN), Type 2 Error"
##
## [[5]]
## [1] "0.7950 = Accuracy (TP+TN/(TP+TN+FP+FN))"
##
## [[6]]
## [1] "0.8234 = Sensitivity, Recall, Hit Rate, True Positive Rate (How many positives did the model get right? T
P/(TP+FN))"
##
## [[7]]
## [1] "0.7667 = Specificity, Selectivity, True Negative Rate (How many negatives did the model get right? TN/(TN
+FP))"
##
## [[8]]
## [1] "0.7788 = Precision, Positive Predictive Value (How good are the model's positive predictions? TP/(TP+F
P))"
##
## [[9]]
## [1] "0.8131 = Negative Predictive Value (How good are the model's negative predictions? TN/(TN+FN)"
```

Putting the data back together for future use

```
data_test$knn <- knn_prediction

data_test <- data_test %>%
  mutate(correct_knn = if_else(knn == high_med_units, 'correct', 'WRONG!'))

temp1 <- ColumnsNotUsedKNN[-partition, ]
full_test_knn <- bind_cols(temp1, data_test)

full_test_knn <- full_test_knn %>%
  select(store, week, high_med_units, knn, correct_knn, size, region, promo_units_per, salty_units_per)
slice_sample(full_test_knn, n=10)
```

```
## # A tibble: 10 × 9
##    store  week high_med_units knn   correct_knn  size region  promo_units_per
##    <fct> <dbl> <fct>          <fct> <chr>       <int> <fct>             <dbl>
##  1 92576    26 high           high  correct      1015 WEST              0.390
##  2 14141    52 low            high  WRONG!        896 QUEBEC            0.302
##  3 2519     30 low            low   correct       917 WEST              0.402
##  4 35284    30 low            low   correct       904 ONTARIO           0.445
##  5 399       6 low            low   correct       948 ONTARIO           0.447
##  6 35277    15 low            high  WRONG!        955 ONTARIO           0.299
##  7 69874    21 high           high  correct       943 WEST              0.316
##  8 35326    14 high           high  correct      1031 ONTARIO           0.364
##  9 35277    44 low            high  WRONG!        955 ONTARIO           0.132
## 10 36780    52 low            high  WRONG!        964 WEST              0.347
## # ℹ 1 more variable: salty_units_per <dbl>
```

Decision Trees

Preprocessing data

```
ColumnsNotUsedTREE <- df %>%
  select(store, week, high_med_rev, high_med_gp, high_med_gpm)

tree1 <- df %>%
  mutate(high_med_units = factor(if_else(high_med_units==1, 'high', 'low'), levels=c('low', 'high')),
                     region = factor(region))

tree1 <- tree1 %>%
  select(high_med_units, size, region,
         promo_units_per,
         altbev_units_per, confect_units_per, salty_units_per,
         velocityA_units_per, velocityB_units_per, velocityC_units_per, velocityD_units_per, velocityNEW_units_pe
r)
```

Using the `caret` package to split the data, 75% training and 25% testing

```
library(caret)
set.seed(42)
partition <- caret::createDataPartition(y=tree1$high_med_units, p=.75, list=FALSE)
data_train <- tree1[partition, ]
data_test <- tree1[-partition, ]
```

Using the `rpart()` function from the `rpart` package to train the

model

```
library(rpart)
library(rpart.plot)

model_tree <- rpart::rpart(high_med_units ~ ., data_train)
```

Using the trained model to predict whether `high_med_units` is high or low

```
predict_tree <- predict(model_tree, data_test, type='class')
```

Using the confusion matrix code above to examine the accuracy of this model

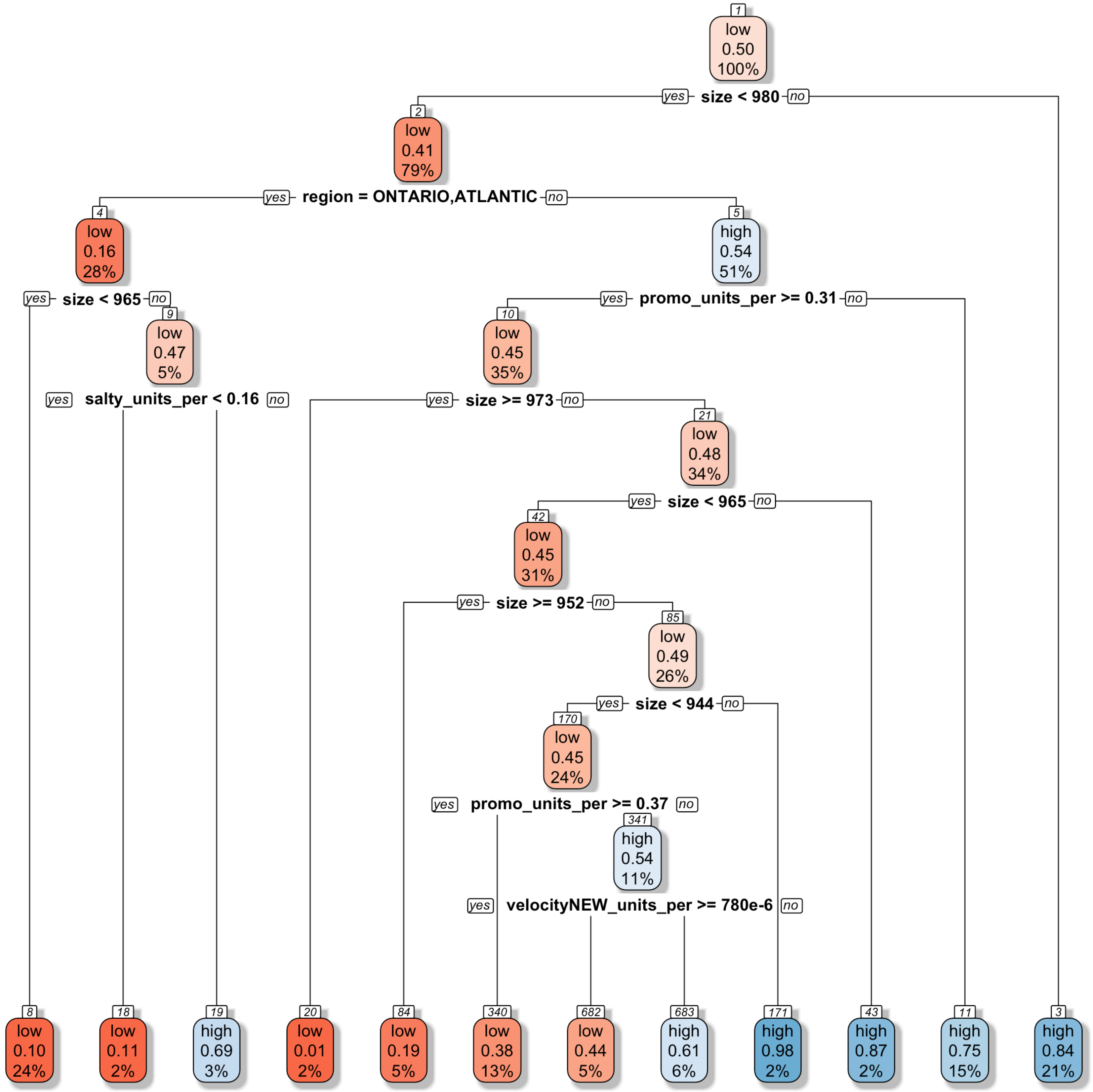
```
table1 <- table(predict_tree, data_test$high_med_units)
my_confusion_matrix(table1)
```

```
##
## predict_tree low high
##         low  990  291
##         high 270  966
```

```
## [[1]]
## [1] "966 = True Positive (TP), Hit"
##
## [[2]]
## [1] "990 = True Negative (TN), Rejection"
##
## [[3]]
## [1] "270 = False Positive (FP), Type 1 Error"
##
## [[4]]
## [1] "291 = False Negative (FN), Type 2 Error"
##
## [[5]]
## [1] "0.7771 = Accuracy (TP+TN/(TP+TN+FP+FN))"
##
## [[6]]
## [1] "0.7685 = Sensitivity, Recall, Hit Rate, True Positive Rate (How many positives did the model get right? T
P/(TP+FN))"
##
## [[7]]
## [1] "0.7857 = Specificity, Selectivity, True Negative Rate (How many negatives did the model get right? TN/(TN
+FP))"
##
## [[8]]
## [1] "0.7816 = Precision, Positive Predictive Value (How good are the model's positive predictions? TP/(TP+F
P))"
##
## [[9]]
## [1] "0.7728 = Negative Predictive Value (How good are the model's negative predictions? TN/(TN+FN)"
```

Using the `plot()` function draw a labeled picture of the tree model.

```
rpart.plot::rpart.plot(model_tree, box.palette = 'RdBu', shadow.col = 'gray', nn=TRUE, yesno=2)
```


1
low
0.50
100%
yes size < 980 no
2
low
0.41
79%
yes region = ONTARIO,ATLANTIC no
4
low
0.16
28%
5
high
0.54
51%
yes size < 965 no
9
low
0.47
5%
yes salty_units_per < 0.16 no
yes promo_units_per >= 0.31 no
10
low
0.45
35%
yes size >= 973 no
21
low
0.48
34%
yes size < 965 no
42
low
0.45
31%
yes size >= 952 no
85
low
0.49
26%
yes size < 944 no
170
low
0.45
24%
yes promo_units_per >= 0.37 no
341
high
0.54
11%
yes velocityNEW_units_per >= 780e-6 no
8
low
0.10
24%
18
low
0.11
2%
19
high
0.69
3%
20
low
0.01
2%
84
low
0.19
5%
340
low
0.38
13%
682
low
0.44
5%
683
high
0.61
6%
171
high
0.98
2%
43
high
0.87
2%
11
high
0.75
15%
3
high
0.84
21%

Putting the data back together for future use

```
data_test$tree <- predict_tree

data_test <- data_test %>%
  mutate(correct_tree = if_else(tree == high_med_units, 'correct', 'WRONG!'))

temp1 <- ColumnsNotUsedTREE[-partition, ]
full_test_tree <- bind_cols(temp1, data_test)

full_test_tree <- full_test_tree %>%
  select(store, week, high_med_units, tree, correct_tree, size, region, promo_units_per, salty_units_per)
slice_sample(full_test_tree, n=10)
```

```
## # A tibble: 10 × 9
##    store  week high_med_units tree  correct_tree  size region  promo_units_per
##    <fct> <dbl> <fct>          <fct> <chr>        <int> <fct>             <dbl>
##  1 74485    23 low            high  WRONG!         910 WEST              0.248
##  2 2591     43 high           high  correct       1038 WEST              0.250
##  3 38893    14 high           high  correct        890 WEST              0.337
##  4 67800     1 low            low   correct        907 ONTARIO           0.242
##  5 14229    41 low            low   correct        906 QUEBEC            0.380
##  6 68295    12 low            low   correct        899 WEST              0.404
##  7 92576    26 high           high  correct       1015 WEST              0.390
##  8 84325    27 high           high  correct        900 WEST              0.345
##  9 1446     18 low            low   correct        944 ONTARIO           0.410
## 10 813      30 low            low   correct        906 ONTARIO           0.379
## # ℹ 1 more variable: salty_units_per <dbl>
```

Comparing Models

Putting both predictions together

```
full_test <- bind_cols(full_test_knn %>%
                         select(store, week, high_med_units, knn, correct_knn),
                       full_test_tree %>%
                         select(-store, -week, -high_med_units))

slice_sample(full_test, n=10)
```

```
## # A tibble: 10 × 11
##    store  week high_med_units knn   correct_knn tree  correct_tree  size region
##    <fct> <dbl> <fct>          <fct> <chr>       <fct> <chr>        <int> <fct>
##  1 85964    32 high           high  correct     high  correct        946 WEST
##  2 14212    36 high           high  correct     high  correct       1015 ONTARIO
##  3 59212    50 high           low   WRONG!      high  correct        985 ONTARIO
##  4 36718    30 high           high  correct     high  correct       1119 WEST
##  5 35284     7 low            low   correct     low   correct        904 ONTARIO
##  6 68295     9 low            low   correct     low   correct        899 WEST
##  7 53002    42 low            low   correct     low   correct        922 ONTARIO
##  8 13808    45 low            low   correct     low   correct        895 QUEBEC
##  9 14229    41 low            low   correct     low   correct        906 QUEBEC
## 10 77809    31 high           high  correct     high  correct        982 WEST
## # ℹ 2 more variables: promo_units_per <dbl>, salty_units_per <dbl>
```

Findings

Error higher for the KNN algorithm

Type 1 error (294) is higher than Type 2 error (222).

Aspect of the accuracy of the KNN algorithm that better

Sensitivity (0.8234) is better than specificity (0.7667).

Sensitivity is calculated as TP/(TP+FN) = 1035/(1035+222). Specificity is calculated as TN/(TN+FP) = 966/(966 + 294). True positives were higher than True negatives, and False negatives were lower than False positives. Therefore, the higher true positive and the lower false negative rates led to a greater sensitivity when compared to specificity.

KNN Algorithm Business Implication

The algorithm aligns well with the business's goals to target stores that have higher than median units sold. The sensitivity is significantly higher than specificity.

Other measures of accuracy that can be used in the future

Logarithmic Loss (Log Loss) - uses assigned probabilities on the sample classes, then penalizes classifications that are false. The closer Log Loss is to zero, the higher accuracy of the model.

Source: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234 (https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234)

Area Under the Curve - aggregate measure that determines if the model ranks a random positive example more highly than a random negative example. This is derived from the receiver operating characteristic curve (ROC curve) that plots both the true positive rate and the false positive rate at different classification thresholds. AUC values range from 0 to 1, with values closer to 1 suggesting higher rate of accurate predictability.

Source: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc (https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc)

Most important factor in decision tree

The most important factor in determining above median units sold is the size of the store.

Regions are more likely to lead to above median units sold - for smaller store (less than 980 products for sale)

West and Quebec (said another way, the regions that are not Ontario or Atlantic)

Best model based on highest precision (PPV)

Problem statement: "We would like to build a company-wide dashboard next year that tells us at the end of each week which stores sold enough units to be in the top half of units sold for that year, even though the year is not over. Can you use the data from the year that just ended to create a predictive model that, with a high degree of accuracy, tells us which of our stores in a given week is likely to sell above median units?"

Results:

Logistic Regression PPV: 0.7500

KNN PPV: 0.7788

Decision Tree PPV: 0.7816

Therefore, the decision tree model would be the best model to use as it has the highest value for Precision (PPV).

Best model for understanding relationships

Problem statement: "In addition to this dashboard, we would like to use last year's data to understand which variables help our stores have successful weeks. Can you use that data to tell us which factors are most important in helping our stores have above median units sold in a given week?"

Logistic regression would be the best model that can help assess the factors that are the most important in helping stores have above median units sold (positive correlation). In this model, we are able to look at the various coefficients and P values for the factors to see which is statistically significant.

Based on logistic regression, salty_units_per, velocityC_units_per, and velocityA_units_per are the top 3 most important factors in helping our stores have above-median units sold in a given week, with positive coefficients of 26.0703631, 14.4423422, and 11.4092559, respectively. Also, all these factors are statistically significant, with a p-value less than 0.05.

Listing this algorithm

Naive Bayes Classifier

Pros and cons of Naive Bayes classifier

Advantages: simple and quick to implement, works well for categorical values (yes/no), scaleable to include multiple variables

Disadvantages: assumes independence between the feature variables which may not always be the case. Also assumes that the feature variables make an equal contribution to the outcome

The fact that Naive Bayes assumes independence and equality among the variables is probably its biggest disadvantage when compared to the three algorithms we have studied thus far. In real world business problems, it is not uncommon to have different factors have larger or smaller impact on outcomes (as we saw in the logistic regression model).

Two relevant lines of R code that are used to run this algorithm

```
# training the model on training set from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X_train, y_train)

# making predictions on the testing set

y_pred = gnb.predict(X_test)
```

Source: https://www.geeksforgeeks.org/naive-bayes-classifiers/ (https://www.geeksforgeeks.org/naive-bayes-classifiers/)