ST563Project

ST 563 Project

Kevin Kronk

Goal:

The purpose of this project is to create a report where you read in data, fit models using concepts from the class, and choose between several different models.

To Do:

Create a document that goes through your process of reading the data, any basic cleaning / transformations, splitting the data, and fitting and choosing a final model.

Read in the Data

- Give a brief introduction to the data and the source of the data.
- State your goal for the project (modeling something). Be specific on why you want to model the variable.
- Read the data in via a URL (or locally but then you must submit the data with your submission)

Online Shoppers Purchasing Intention Dataset - UCI Machine Learning Repository. https://archive.ics.uci.edu/dataset/468/online+shoppers+purchasing+intention+dataset

This data set represents information about an individuals online session and whether or not the session ended with shopping. It is from the UC Irvine Machine Learning Repository and can be used under the Creative Commons Attribution 4.0 International license that allows it to be shared and adapted for any purpose. The creators of the data set are C. Sakar and Yomi Kastro. An additional note is that the data set was made so that each session belongs to a different user in a 1 year period to avoid any tendency to a specific campaign, special day, user profile, or period.

The goal is to be able to predict whether or not a user ended their internet session with shopping based on many features about their session. We want the model not only with the highest accuracy, but also ideally the highest sensitivity. So the model that has the highest true positive predictions out of the total number of positive values, in this case that the person did shop. One of the reasons to model this variable is to figure out which features are important for determining if a user shopped. Therefore, a model that is interpretable may be best. Additionally, theoretically a company could use this data to determine which users are more likely to purchase during their session. Perhaps they could show more ads to those users, or in some way try to direct those users to places they can purchase goods.

```
# Import libraries

suppressWarnings(library(tidyverse))
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```
suppressWarnings(library(caret))
```

```
Loading required package: lattice

Attaching package: 'caret'

The following object is masked from 'package:purrr':

    lift
```

```
suppressWarnings(library(gam))
```

```
Loading required package: splines
Loading required package: foreach
```

```
Attaching package: 'foreach'

The following objects are masked from 'package:purrr':

    accumulate, when

Loaded gam 1.22-5
```

```
suppressWarnings(library(gbm))
```

```
Loaded gbm 2.2.2
This version of gbm is no longer under development. Consider transitioning to gbm3, https://g
```

```
suppressWarnings(library(knitr))
suppressWarnings(library(splines))
suppressWarnings(library(glmnet))
```

```
Loading required package: Matrix

Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack

Loaded glmnet 4.1-8
```

```
suppressWarnings(library(tree))
suppressWarnings(library(rpart))
suppressWarnings(library(rpart.plot))
suppressWarnings(library(e1071))
```

```
# Read in the data from a csv file

shoppers <- read_csv('online_shoppers_intention.csv', show_col_types=FALSE)

head(shoppers)
```

```
# A tibble: 6 x 18
  Administrative Administrative_Duration Informational Informational_Duration
```

```
          <dbl>                    <dbl>         <dbl>                    <dbl>
1             0                        0             0                        0
2             0                        0             0                        0
3             0                        0             0                        0
4             0                        0             0                        0
5             0                        0             0                        0
6             0                        0             0                        0
# i 14 more variables: ProductRelated <dbl>, ProductRelated_Duration <dbl>,
#   BounceRates <dbl>, ExitRates <dbl>, PageValues <dbl>, SpecialDay <dbl>,
#   Month <chr>, OperatingSystems <dbl>, Browser <dbl>, Region <dbl>,
#   TrafficType <dbl>, VisitorType <chr>, Weekend <lgl>, Revenue <lgl>
```

Data Cleaning & Transformations

Go through a brief process where you make sure that each variable is read in correctly (correct data type), check for missing values, check for valid data values (perhaps by summarizing each column and seeing if you have reasonable values), removing observations where needed (give an explanation as to why), and doing any data transformations you deem necessary.

```
# Get the rows and columns of the data frame
dim(shoppers)
```

```
[1] 12330    18
```

There are 12330 observations, 17 features, and 1 response variable, as was described on the data set webpage.

```
# Get the class type of each feature
data.frame(t(sapply(shoppers, class)))
```

```
  Administrative Administrative_Duration Informational Informational_Duration
1        numeric                 numeric       numeric                numeric
  ProductRelated ProductRelated_Duration BounceRates ExitRates PageValues
1        numeric                 numeric     numeric   numeric    numeric
  SpecialDay     Month OperatingSystems Browser  Region TrafficType VisitorType
1    numeric character          numeric numeric numeric     numeric   character
  Weekend Revenue
1 logical logical
```

Numeric Features: Administrative, Administrative_Duration, Informational, Informational_Duration, ProductRelated, ProductRelated_Duration, BounceRates, ExitRates, PageValues, SpecialDay, OperatingSystem, Browser, Region, TrafficType

Character Features: Month, VisitorType

Logical Features: Weekend, Revenue

Extra Information

- Administrative, Administrative_Duration, Informational, Informational_Duration, ProductRelated, ProductRelated_Duration all represent the different types of pages visited in that session and the total time spent in each.
- BounceRate refers to the percentage of visitors who enter the site from that page and then leave without triggering any other requests to the analytics server during that session.
- ExitRate is calculated as for all pageviews to the page, the percentage that were the last in the session.
- PageValues represents the average value for a web page that a user visited before completing an e-commerce transaction.
- SpecialDay indicates the closeness of the site visiting time to a specific special day in which the sessions are more likely to be finalized with transaction.

Revenue - Binary TRUE or FALSE. This is the response variable that tracks whether or not an internet session ended with shopping. Therefore this is a binary classification supervised learning problem.

```
# Check for missing values
data.frame(t(colSums(is.na(shoppers))))
```

```
  Administrative Administrative_Duration Informational Informational_Duration
1              0                       0             0                      0
  ProductRelated ProductRelated_Duration BounceRates ExitRates PageValues
1              0                       0           0         0          0
  SpecialDay Month OperatingSystems Browser Region TrafficType VisitorType
1          0     0                 0       0      0           0           0
  Weekend Revenue
1       0       0
```

As we can see there are no missing values from the data set.

Now we can summarize each column to see if the values look reasonable.

```r
# Summarize the numeric features
num_features <- c("Administrative", "Administrative_Duration", "Informational",
                  "Informational_Duration", "ProductRelated",
                  "ProductRelated_Duration", "BounceRates", "ExitRates",
                  "PageValues", "SpecialDay", "OperatingSystems", "Browser",
                  "Region", "TrafficType")

num_summary <- data.frame(t(sapply(shoppers[num_features], summary)))
num_summary
```

```
                         Min.      X1st.Qu.       Median         Mean
Administrative              0   0.00000000 1.000000e+00 2.315166e+00
Administrative_Duration     0   0.00000000 7.500000e+00 8.081861e+01
Informational               0   0.00000000 0.000000e+00 5.035685e-01
Informational_Duration      0   0.00000000 0.000000e+00 3.447240e+01
ProductRelated              0   7.00000000 1.800000e+01 3.173147e+01
ProductRelated_Duration     0 184.13750000 5.989369e+02 1.194746e+03
BounceRates                 0   0.00000000 3.112468e-03 2.219138e-02
ExitRates                   0   0.01428571 2.515640e-02 4.307280e-02
PageValues                  0   0.00000000 0.000000e+00 5.889258e+00
SpecialDay                  0   0.00000000 0.000000e+00 6.142741e-02
OperatingSystems            1   2.00000000 2.000000e+00 2.124006e+00
Browser                     1   2.00000000 2.000000e+00 2.357097e+00
Region                      1   1.00000000 3.000000e+00 3.147364e+00
TrafficType                 1   2.00000000 2.000000e+00 4.069586e+00
                          X3rd.Qu.       Max.
Administrative        4.000000e+00    27.0000
Administrative_Duration 9.325625e+01  3398.7500
Informational         0.000000e+00    24.0000
Informational_Duration  0.000000e+00  2549.3750
ProductRelated        3.800000e+01   705.0000
ProductRelated_Duration 1.464157e+03 63973.5222
BounceRates           1.681256e-02     0.2000
ExitRates             5.000000e-02     0.2000
PageValues            0.000000e+00   361.7637
SpecialDay            0.000000e+00     1.0000
OperatingSystems      3.000000e+00     8.0000
Browser               2.000000e+00    13.0000
Region                4.000000e+00     9.0000
TrafficType           4.000000e+00    20.0000
```

There's nothing that immediately pops out as being wrong. Many of the features have 0 values

for up to 50% or even 75% of the data set. I'm not sure if this would cause any issues, but for those features, it makes sense that many of the values would be 0. For instance, Informational tracks how many informational websites a user visited. It's entirely plausible that a user didn't visit any informational websites during their session. The scales of different features are very different though, so for models that are sensitive, scaling will need to be done.

```
# Summarize the character features
char_features <- c("Month", "VisitorType")

char_summary <- data.frame(t(sapply(shoppers[char_features], summary)))
char_summary
```

```
            Length     Class      Mode
Month        12330 character character
VisitorType  12330 character character
```

```
# Look at the unique character values for each of these features
unique(shoppers$Month)
```

```
 [1] "Feb"  "Mar"  "May"  "Oct"  "June" "Jul"  "Aug"  "Nov"  "Sep"  "Dec"
```

```
table(shoppers$Month)
```

```
 Aug  Dec  Feb  Jul June  Mar  May  Nov  Oct  Sep
 433 1727  184  432  288 1907 3364 2998  549  448
```

```
unique(shoppers$VisitorType)
```

```
[1] "Returning_Visitor" "New_Visitor"       "Other"
```

```
table(shoppers$VisitorType)
```

```
      New_Visitor             Other Returning_Visitor
             1694                85             10551
```

Both of these features will need to be transformed in a way to convert them to numeric. Month could be converted to sin and cos transformations to capture the cyclical nature, but in numeric form. In that case December would be closest to January and November. VisitorType can be turned into two dummy variables for new visitor and returning visitor, where the baseline (both features being 0) is the other type of visitor.

```
# Summarize the binary features
bin_features <- c("Weekend", "Revenue")

bin_summary <- data.frame(t(sapply(shoppers[bin_features], summary)))
bin_summary
```

```
           Mode FALSE. TRUE.
Weekend logical   9462  2868
Revenue logical  10422  1908
```

Both of these features will be converted to 0 for False and 1 for True.

```
# Convert Month column to numeric
shoppers_df <- shoppers %>%
                mutate(MonthNum = recode(Month,
                  Jan = 1,
                  Feb = 2,
                  Mar = 3,
                  Apr = 4,
                  May = 5,
                  June = 6,
                  Jul = 7,
                  Aug = 8,
                  Sep = 9,
                  Oct = 10,
                  Nov = 11,
                  Dec = 12))
```

```
summary(shoppers_df$MonthNum)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   5.000   7.000   7.652  11.000  12.000
```

```r
# Then create sin and cos versions of the Month feature
shoppers_df['MonthSin'] <- sin(shoppers_df['MonthNum'] * (2*pi/12))
shoppers_df['MonthCos'] <- cos(shoppers_df['MonthNum'] * (2*pi/12))
```

```r
# Create dummy variables, in this case the baseline class is other
shoppers_df$VisitorNew <- ifelse(shoppers_df$VisitorType == 'New_Visitor',
                                 1, 0)
shoppers_df$VisitorReturn <- ifelse(shoppers_df$VisitorType ==
                                    'Returning_Visitor',
                                    1, 0)
```

```r
# Remove extra features
shoppers_df <- within(shoppers_df, rm("VisitorType", "Month", "MonthNum"))
```

```r
# Convert binary features to 0 and 1
shoppers_df$Weekend <- ifelse(shoppers_df$Weekend == TRUE, 1, 0)
shoppers_df$Revenue <- ifelse(shoppers_df$Revenue == TRUE, 1, 0)
```

```r
head(shoppers_df)
```

```
# A tibble: 6 x 20
  Administrative Administrative_Duration Informational Informational_Duration
           <dbl>                   <dbl>         <dbl>                   <dbl>
1              0                       0             0                       0
2              0                       0             0                       0
3              0                       0             0                       0
4              0                       0             0                       0
5              0                       0             0                       0
6              0                       0             0                       0
# i 16 more variables: ProductRelated <dbl>, ProductRelated_Duration <dbl>,
#   BounceRates <dbl>, ExitRates <dbl>, PageValues <dbl>, SpecialDay <dbl>,
#   OperatingSystems <dbl>, Browser <dbl>, Region <dbl>, TrafficType <dbl>,
#   Weekend <dbl>, Revenue <dbl>, MonthSin <dbl>, MonthCos <dbl>,
#   VisitorNew <dbl>, VisitorReturn <dbl>
```

```r
# Look at the response variable
table(shoppers_df$Revenue)
```

```
    0     1
10422  1908
```

```r
# Get the proportion
table(shoppers_df$Revenue)/length(shoppers_df$Revenue)
```

```
        0         1
0.8452555 0.1547445
```

Split the Data into a Train and Test Set

Use whatever reasonable proportion you'd like.

The response is relatively imbalanced, which will require using stratified train and test splits. It turns out that according to the createDataPartition documentation, when splitting the data, the random sampling is done within the levels of the response, when the response is a factor. This attempts to balance the class distributions within the splits, so we are doing stratified sampling.

```r
# Set seed for repeatability
set.seed(123)

# Partition the data to get 80% in the training set
train_index <- createDataPartition(shoppers_df$Revenue,
                                   p=0.8, list=FALSE)

# Create Train and Test Set
shoppers_train <- shoppers_df[train_index,]
shoppers_test <- shoppers_df[-train_index,]
```

```r
# Dimensions of the train and test set
dim(shoppers_train)
```

```
[1] 9864   20
```

```r
dim(shoppers_test)
```

```
[1] 2466   20
```

```r
# Count and proportion of each response class in the train set
table(shoppers_train$Revenue)
```

```
   0    1
8332 1532
```

```
table(shoppers_train$Revenue)/length(shoppers_train$Revenue)
```

```
        0         1
0.8446878 0.1553122
```

```
# Count and proportion of each response class in the test set
table(shoppers_test$Revenue)
```

```
   0    1
2090  376
```

```
table(shoppers_test$Revenue)/length(shoppers_test$Revenue)
```

```
        0         1
0.8475264 0.1524736
```

```
# Create scaled train and test set for the models that need it
shoppers_train_s <- scale(shoppers_train[,-16],
                          center = TRUE,
                          scale = TRUE)

shoppers_test_s <- scale(shoppers_test[,-16],
                         center = attr(shoppers_train_s, "scaled:center"),
                         scale = attr(shoppers_train_s, "scaled:scale"))

shoppers_train_s <- data.frame(shoppers_train_s)
shoppers_test_s <- data.frame(shoppers_test_s)
```

```
# Add Revenue back to the scaled datasets
shoppers_train_s['Revenue'] <- shoppers_train$Revenue
shoppers_test_s['Revenue'] <- shoppers_test$Revenue
head(shoppers_train_s)
```

```
  Administrative Administrative_Duration Informational Informational_Duration
1     -0.6952122              -0.4690142    -0.3943766             -0.2435602
2     -0.6952122              -0.4690142    -0.3943766             -0.2435602
3     -0.6952122              -0.4690142    -0.3943766             -0.2435602
4     -0.6952122              -0.4690142    -0.3943766             -0.2435602
5     -0.6952122              -0.4690142    -0.3943766             -0.2435602
6     -0.6952122              -0.4690142    -0.3943766             -0.2435602
  ProductRelated ProductRelated_Duration BounceRates  ExitRates PageValues
1     -0.6994567              -0.6530937  3.71569230  3.2706681 -0.3156988
2     -0.6766231              -0.6180380 -0.45483875  1.1909276 -0.3156988
3     -0.6994567              -0.6530937  3.71569230  3.2706681 -0.3156988
4     -0.4939545              -0.3093840 -0.03778564  0.1510574 -0.3156988
5     -0.2884522              -0.5686223 -0.12558629 -0.3779994 -0.3156988
6     -0.6994567              -0.6530937  3.71569230  3.2706681 -0.3156988
  SpecialDay OperatingSystems    Browser      Region TrafficType    Weekend
1 -0.3116136       -1.2381995 -0.7892986 -0.89278718 -0.75752443 -0.5538999
2 -0.3116136       -0.1365850 -0.2192978 -0.89278718 -0.50856658 -0.5538999
3 -0.3116136        2.0666440 -0.7892986  2.44156950 -0.25960873 -0.5538999
4 -0.3116136        0.9650295  0.3507031 -0.89278718 -0.01065087  1.8051972
5 -0.3116136       -0.1365850 -0.2192978 -0.89278718 -0.25960873 -0.5538999
6  1.6740174       -0.1365850  0.9207040 -0.05919801 -0.25960873 -0.5538999
  MonthSin  MonthCos VisitorNew VisitorReturn Revenue
1 1.272829 0.5612036 -0.3945833      0.406004       0
2 1.272829 0.5612036 -0.3945833      0.406004       0
3 1.272829 0.5612036 -0.3945833      0.406004       0
4 1.272829 0.5612036 -0.3945833      0.406004       0
5 1.272829 0.5612036 -0.3945833      0.406004       0
6 1.272829 0.5612036 -0.3945833      0.406004       0
```

Training Models

In the course so far, some of the models we've looked at are:

1. kNN
2. (Regularized) Linear Regression & Logistic Regression Models
3. GAMs (using piece-wise polynomial regression, local regression, or splines)
4. Single tree models
5. Ensemble tree models
6. Support vector machines

You'll fit one model from each of these model types.

kNN

The k-nearest neighbors algorithm is non-parametric because it is not estimating parameters that define some model, but rather directly predicting the response based on the average response of the k-nearest neighbors. The one tuning parameter is k, the number of nearest neighbors to average. The lower k, the more flexible the model is, the higher it is, the more bias the model will have. Other than reporting the probability that a new data point belongs to one class or the other, we can not use kNN for inference. This model does not perform any variable selection. kNN is sensitive to the scale of the features, since it needs to calculate a distance value to determine the nearest neighbors. Large features could affect this process, so predictors need to be standardized.

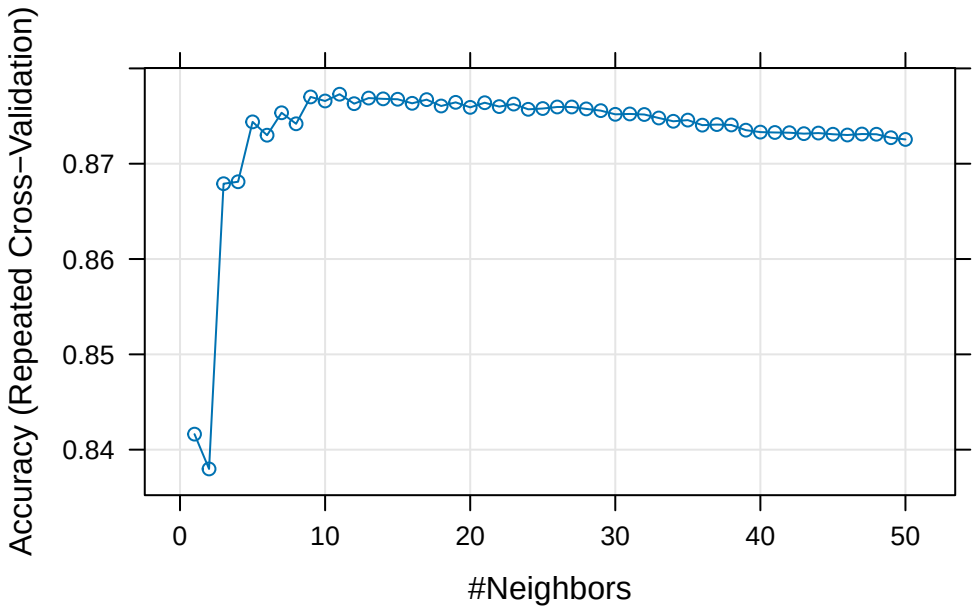
It's important to remember that our training set has roughly 0.845 false values for the response. This means the no information rate, if a classifier were to only predict false, would be 0.845, and the accuracy would be 84.5%.

```
# Set seed for repeatability
set.seed(100)

# K values for tuning
knn_grid <- expand.grid(k = c(1:50))

# Performing 5-fold CV, repeated 5 times
knn_cv <- trainControl(method = "repeatedcv",
                       number = 5,
                       repeats = 5)

# Training the knn classifier on training data
knn_fit <- train(as.factor(Revenue) ~ .,
                 data = shoppers_train_s,
                 method = 'knn',
                 tuneGrid = knn_grid,
                 trControl = knn_cv)

summary(knn_fit)
```

```
            Length Class      Mode
learn        2     -none-     list
k            1     -none-     numeric
theDots      0     -none-     list
xNames      19     -none-     character
problemType  1     -none-     character
tuneValue    1     data.frame list
```

```
obsLevels     2      -none-      character
param         0      -none-      list
```

```
plot(knn_fit)
```




```
# Getting the results of the best K model
k_opt <- knn_fit$bestTune$k
knn_fit$results[k_opt,]
```

```
    k  Accuracy    Kappa  AccuracySD    KappaSD
11 11 0.8772909 0.398987 0.004231873 0.02308912
```

```
# Refitting the model to the full training set
knn_best <- train(as.factor(Revenue) ~ .,
                  data = shoppers_train_s,
                  method = 'knn',
                  tuneGrid = expand.grid(k = knn_fit$bestTune$k),
                  trControl = trainControl(method = 'none'))

summary(knn_best)
```

```
            Length Class      Mode
learn        2     -none-     list
k            1     -none-     numeric
theDots      0     -none-     list
xNames      19     -none-     character
problemType  1     -none-     character
tuneValue    1     data.frame list
obsLevels    2     -none-     character
param        0     -none-     list
```

```r
# Test on the test set
knn_pred <- predict(knn_best, newdata=shoppers_test_s)

# Compare predicted vs actual values to get performance metrics
confusionMatrix(knn_pred, as.factor(shoppers_test_s$Revenue),
                positive="1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2037  253
         1   53  123

               Accuracy : 0.8759
                 95% CI : (0.8622, 0.8887)
    No Information Rate : 0.8475
    P-Value [Acc > NIR] : 3.249e-05

                  Kappa : 0.3859

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.32713
            Specificity : 0.97464
         Pos Pred Value : 0.69886
         Neg Pred Value : 0.88952
             Prevalence : 0.15247
         Detection Rate : 0.04988
   Detection Prevalence : 0.07137
      Balanced Accuracy : 0.65088
```

```
       'Positive' Class : 1
```

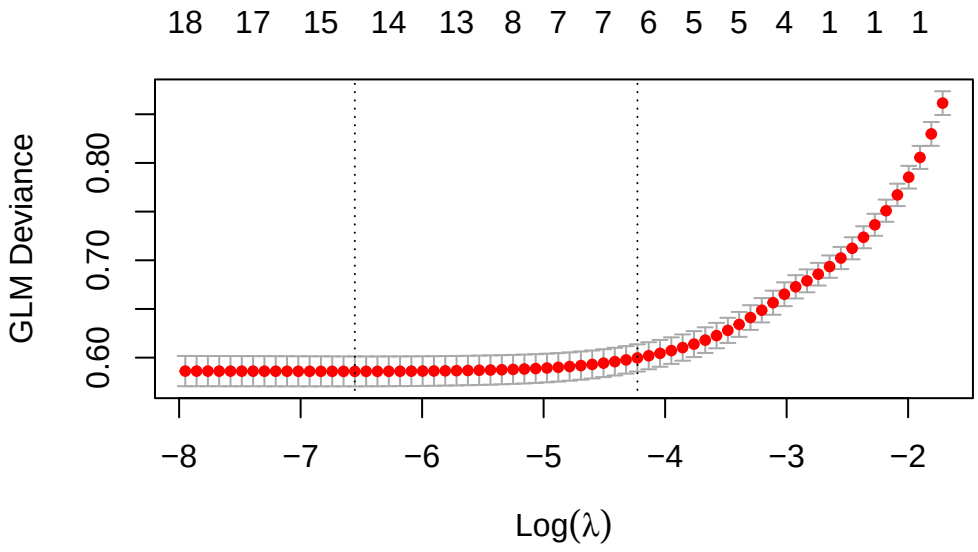
The kNN had an accuracy of 0.8759, which is above the no information rate, but more importantly a sensitivity of only 0.32713. So only about a third of the time that Revenue was true did it correctly predict it. Added to the fact that kNN doesn't allow us to perform inference means that it is likely a poor choice for this particular problem.

Regularized Logistic Regression

Lasso regression is one form of regularized logistic regression where the regularization term allows for the model to shrink some coefficients to zero. This means it can effectively perform feature selection. This model is a parametric model, using the typical linear combination of the predictors and their coefficients. This is applied to the logistic regression version of the model so that it can predict the probability of y = 1. In glmnet alpha =1 makes it so the elastic net equation reduces down to just the lasso regression. The tuning parameter is lambda which controls how large of an impact the regularization term has. The higher lambda, the more the model will shrink coefficients. This model can be used for inference as we get the coefficients of each predictor and we can get their standard errors, calculate p-values, and perform hypothesis testing and confidence intervals. Variable selection is done when the model shrinks certain coefficients all the way down to 0, so they no longer have an impact on the model. The predictors do need to be standardized and glmnet automatically does this before estimating the regression coefficients.

```
# Set seed for repeatability
set.seed(200)

# Train the glmnet, performing cv to tune the lambda value
logit_cv <- cv.glmnet(x = as.matrix(shoppers_train[-16]),
                      y = shoppers_train$Revenue,
                      family = binomial(),
                      alpha = 1)
```

```
plot(logit_cv)
```




```
# Get the 1se lambda value
logit_cv$lambda.1se
```

```
[1] 0.01458282
```

```
# Retrain the model using the 1se lambda, which is a larger lambda within 1
# standard error of the best lambda value, so the model coefficients are further
# shrunk down
logit_best <- glmnet(x = as.matrix(shoppers_train[,-16]),
                     y = shoppers_train$Revenue,
                     family = binomial(),
                     alpha = 1,
                     lambda = logit_cv$lambda.1se)

# Estimated coefs, many have been shrunk to 0
coef(logit_best)
```

```
20 x 1 sparse Matrix of class "dgCMatrix"
                                      s0
(Intercept)                -2.166034e+00
Administrative              .
Administrative_Duration     .
```

```
Informational             .
Informational_Duration    .
ProductRelated            1.293052e-03
ProductRelated_Duration   6.725327e-05
BounceRates               .
ExitRates                -7.784174e+00
PageValues                6.954657e-02
SpecialDay                .
OperatingSystems          .
Browser                   .
Region                    .
TrafficType               .
Weekend                   .
MonthSin                 -3.001253e-01
MonthCos                  5.072560e-02
VisitorNew                7.604419e-03
VisitorReturn             .
```

```r
# Test on the test set
logit_pred <- predict(logit_best, newx=as.matrix(shoppers_test[-16]),
                      s = logit_cv$lambda.1se, type='response')

# Predict 1 if probability is over 0.5, 0 otherwise
logit_pred <- ifelse(logit_pred > 0.5, 1, 0)

# Compare predicted vs actual values to get performance metrics
confusionMatrix(as.factor(logit_pred), as.factor(shoppers_test$Revenue),
                positive="1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2046  270
         1   44  106

               Accuracy : 0.8727
                 95% CI : (0.8589, 0.8856)
    No Information Rate : 0.8475
    P-Value [Acc > NIR] : 0.0002145

                  Kappa : 0.3462
```

```
Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.28191
            Specificity : 0.97895
         Pos Pred Value : 0.70667
         Neg Pred Value : 0.88342
             Prevalence : 0.15247
         Detection Rate : 0.04298
   Detection Prevalence : 0.06083
      Balanced Accuracy : 0.63043

       'Positive' Class : 1
```

Lasso logistic regression had an accuracy of 0.8727 and a sensitivity of 0.28191. So in this case it performed worse than knn. We don't have a lot of features, so it's possible that shrinking the feature space down is not only unnecessary but hurt performance. Therefore, this model is not a good choice for this task.

Smoothing Spline GAM

Smoothing splines train a model by using the maximal number of knots, while having a regularization term control the complexity of the model. The smoothing spline itself is non-parametric because it doesn't have a specific parametric form. It's effectively interpolating the data and then using a regularization term to smooth out the model fit. However, creating a generalized additive model (gam) with a smoothing spline on each feature means the model is parametric. The tuning parameter is lambda which controls the weight of the regularization term that smooths the model. A small lambda means the model is just interpolating the data, and a large lambda becomes linear regression with a straight line. This is found during computation due to smoothing splines allowing for an easy calculation of leave–one-out-cross-validation. Another tuning parameter could also be the df for each smoothing spline, which is a measure of the flexibility of the smoothing spline. The model can be used for inference like an anova for parametric and nonparametric effects for each smoothing spline feature. The model does not inherently perform variable selection, however the hypothesis testing for each feature tells us which were found to be significant or not, which could aid in manually performing variable selection. While I'm not sure if smoothing splines require standardizing the features, it certainly can't hurt.

```
# Set seed for repeatability
set.seed(300)
```

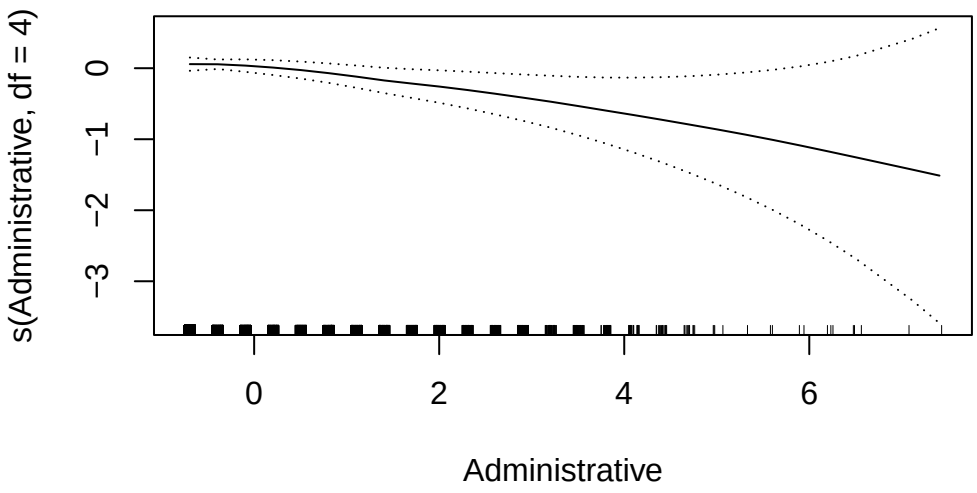
```r
# GAM model with smoothing splines on features, each 4th degree
ss_fit <- gam(as.factor(Revenue) ~ s(Administrative, df = 4) +
                s(Administrative_Duration, df = 4) +
                s(Informational, df = 4) +
                s(Informational_Duration, df = 4) +
                s(ProductRelated, df = 4) +
                s(ProductRelated_Duration, df = 4) +
                s(BounceRates, df = 4) +
                s(ExitRates, df = 4) +
                s(PageValues, df = 4) +
                s(SpecialDay, df = 4) +
                s(OperatingSystems, df = 4) +
                s(Browser, df = 4) +
                s(Region, df = 4) +
                s(TrafficType, df = 4) +
                Weekend +
                s(MonthSin, df = 4) +
                s(MonthCos, df = 4) +
                VisitorNew +
                VisitorReturn,
              data = shoppers_train_s,
              family = binomial())

summary(ss_fit)
```

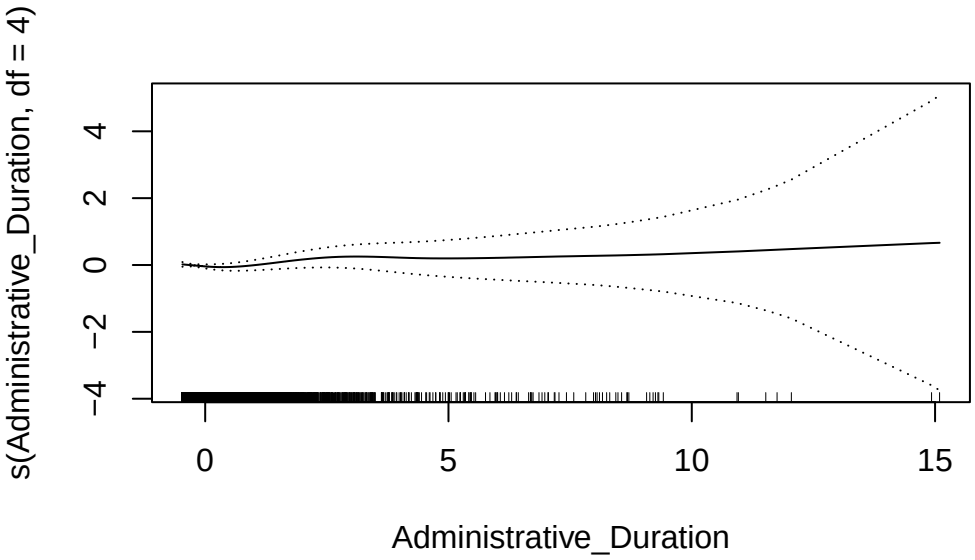
```
Call: gam(formula = as.factor(Revenue) ~ s(Administrative, df = 4) +
    s(Administrative_Duration, df = 4) + s(Informational, df = 4) +
    s(Informational_Duration, df = 4) + s(ProductRelated, df = 4) +
    s(ProductRelated_Duration, df = 4) + s(BounceRates, df = 4) +
    s(ExitRates, df = 4) + s(PageValues, df = 4) + s(SpecialDay,
    df = 4) + s(OperatingSystems, df = 4) + s(Browser, df = 4) +
    s(Region, df = 4) + s(TrafficType, df = 4) + Weekend + s(MonthSin,
    df = 4) + s(MonthCos, df = 4) + VisitorNew + VisitorReturn,
    family = binomial(), data = shoppers_train_s)
Deviance Residuals:
    Min      1Q  Median      3Q     Max
-2.5567 -0.4004 -0.2760 -0.1323  3.3783

(Dispersion Parameter for binomial family taken to be 1)

    Null Deviance: 8518.829 on 9863 degrees of freedom
```

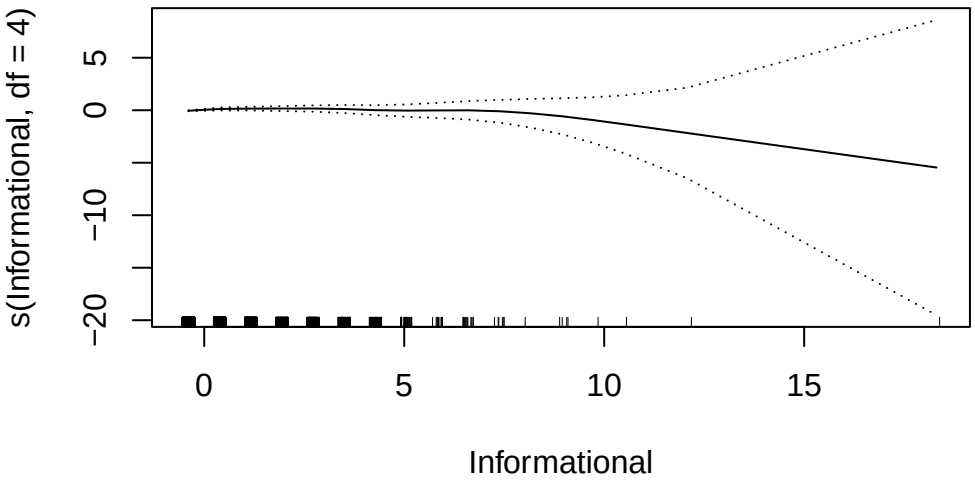
```
Residual Deviance: 4948.161 on 9796 degrees of freedom
AIC: 5084.161

Number of Local Scoring Iterations: NA

Anova for Parametric Effects
                                  Df Sum Sq Mean Sq   F value    Pr(>F)
s(Administrative, df = 4)          1   11.2   11.18   15.3202 9.136e-05 ***
s(Administrative_Duration, df = 4) 1    1.2    1.17    1.6100  0.204523
s(Informational, df = 4)           1    4.9    4.93    6.7546  0.009365 **
s(Informational_Duration, df = 4)  1    0.2    0.15    0.2115  0.645596
s(ProductRelated, df = 4)          1   32.6   32.65   44.7561 2.354e-11 ***
s(ProductRelated_Duration, df = 4) 1    1.9    1.89    2.5937  0.107324
s(BounceRates, df = 4)             1   58.2   58.24   79.8368 < 2.2e-16 ***
s(ExitRates, df = 4)               1  113.6  113.59  155.7161 < 2.2e-16 ***
s(PageValues, df = 4)              1  985.2  985.23 1350.6350 < 2.2e-16 ***
s(SpecialDay, df = 4)              1   14.0   13.98   19.1615 1.214e-05 ***
s(OperatingSystems, df = 4)        1    2.1    2.14    2.9276  0.087108 .
s(Browser, df = 4)                 1    0.3    0.33    0.4502  0.502233
s(Region, df = 4)                  1    1.9    1.86    2.5467  0.110558
s(TrafficType, df = 4)             1    0.2    0.22    0.3032  0.581917
Weekend                            1    4.5    4.46    6.1190  0.013390 *
s(MonthSin, df = 4)                1  103.9  103.86  142.3794 < 2.2e-16 ***
s(MonthCos, df = 4)                1    4.2    4.20    5.7636  0.016380 *
VisitorNew                         1   26.8   26.85   36.8060 1.353e-09 ***
VisitorReturn                      1    1.6    1.62    2.2152  0.136693
Residuals                       9796 7145.7    0.73
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Anova for Nonparametric Effects
                                  Npar Df Npar Chisq    P(Chi)
(Intercept)
s(Administrative, df = 4)                3       1.50 0.6832167
s(Administrative_Duration, df = 4)       3       4.74 0.1919579
s(Informational, df = 4)                 3       7.23 0.0650370 .
s(Informational_Duration, df = 4)        3       3.51 0.3200687
s(ProductRelated, df = 4)                3       0.72 0.8686423
s(ProductRelated_Duration, df = 4)       3      10.40 0.0154350 *
s(BounceRates, df = 4)                   3      16.59 0.0008568 ***
s(ExitRates, df = 4)                     3      21.33 9.007e-05 ***
s(PageValues, df = 4)                    3     880.56 < 2.2e-16 ***
s(SpecialDay, df = 4)                    3       2.03 0.5666146
```

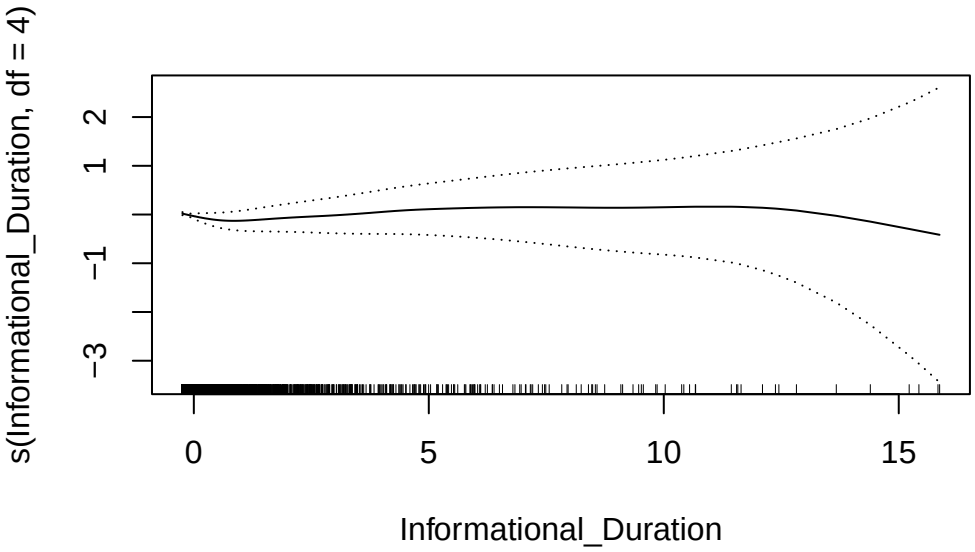
```
s(OperatingSystems, df = 4)              3       3.12 0.3740408
s(Browser, df = 4)                       3       5.14 0.1620248
s(Region, df = 4)                        3       2.10 0.5513921
s(TrafficType, df = 4)                   3      10.09 0.0178233 *
Weekend
s(MonthSin, df = 4)                      3      57.16 2.377e-12 ***
s(MonthCos, df = 4)                      3      10.52 0.0146252 *
VisitorNew
VisitorReturn
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

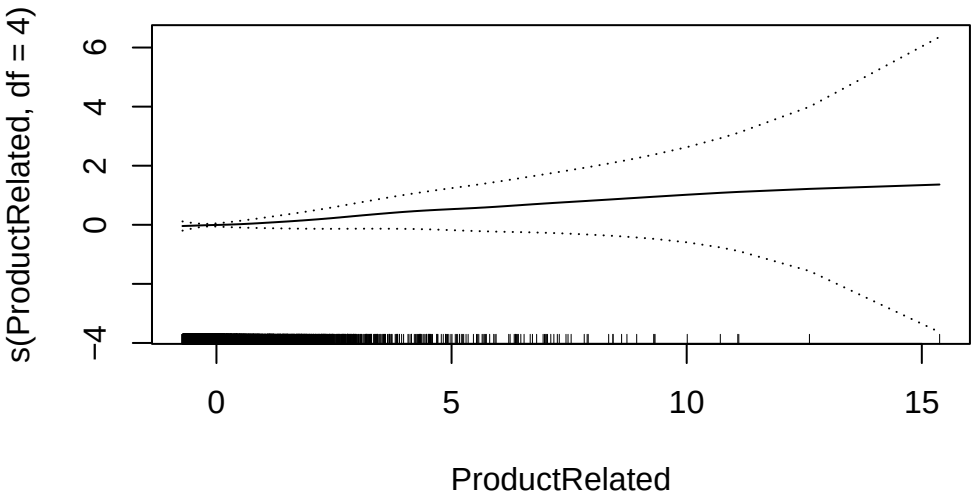
```r
plot(ss_fit, se=TRUE)
```

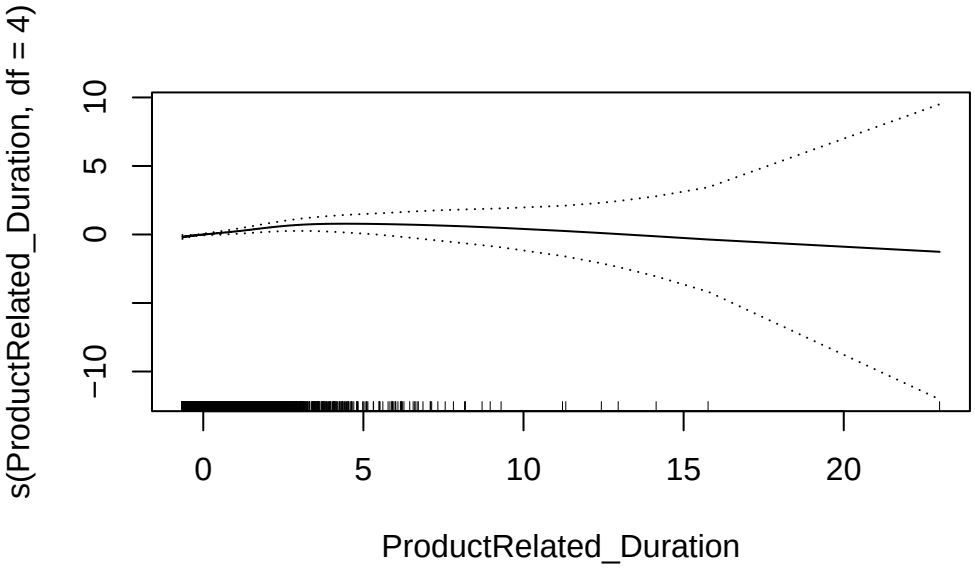


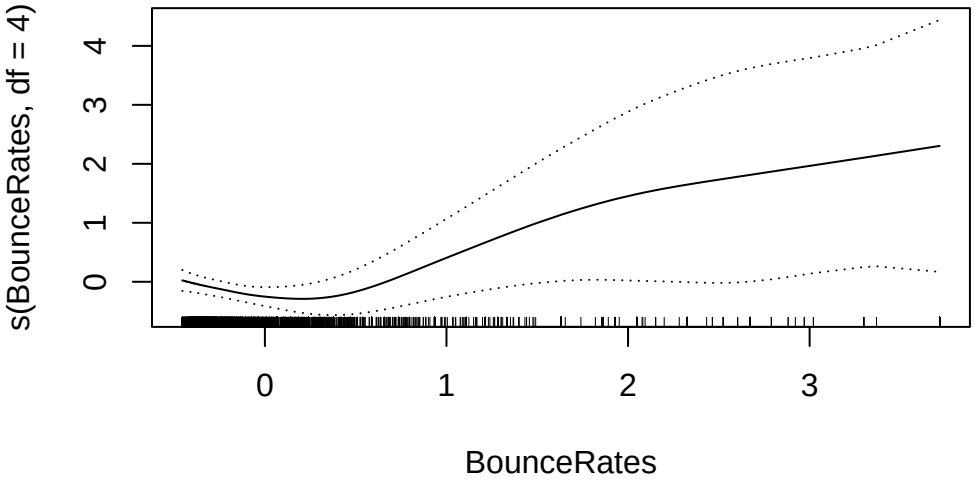

s(Administrative_Duration, df = 4)
−4
−2
0
2
4
0
5
10
15
Administrative_Duration

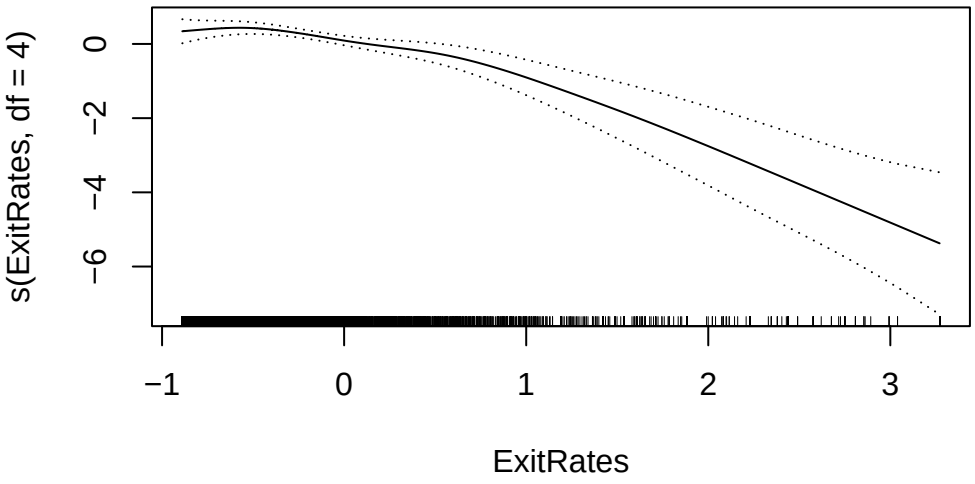

s(Informational, df = 4)
−20
−10
0
5
0
5
10
15
Informational

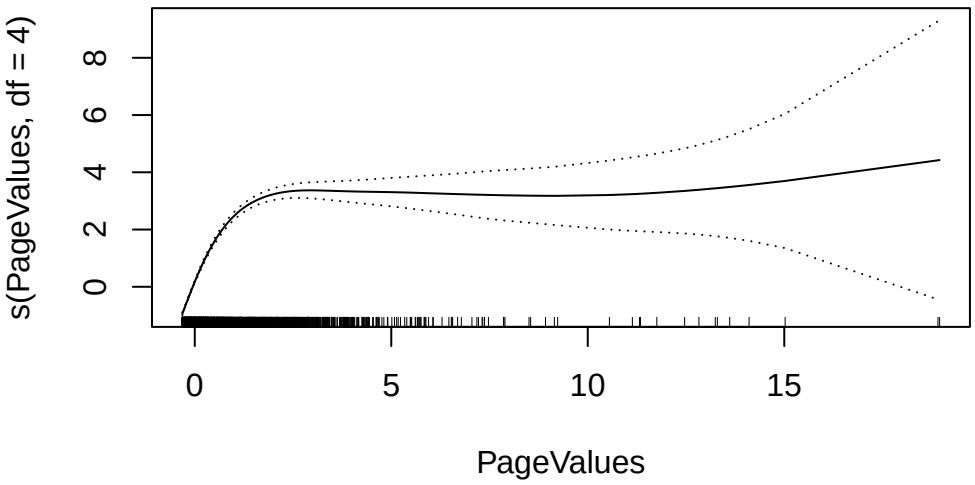

s(Informational_Duration, df = 4)
−3
−1
1
2
0
5
10
15
Informational_Duration

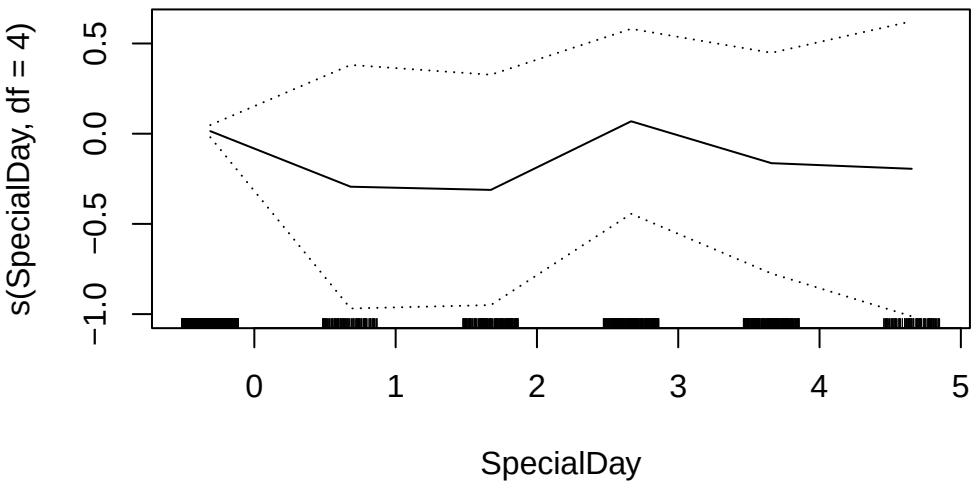

s(ProductRelated, df = 4)
−4
0
2
4
6
0
5
10
15
ProductRelated

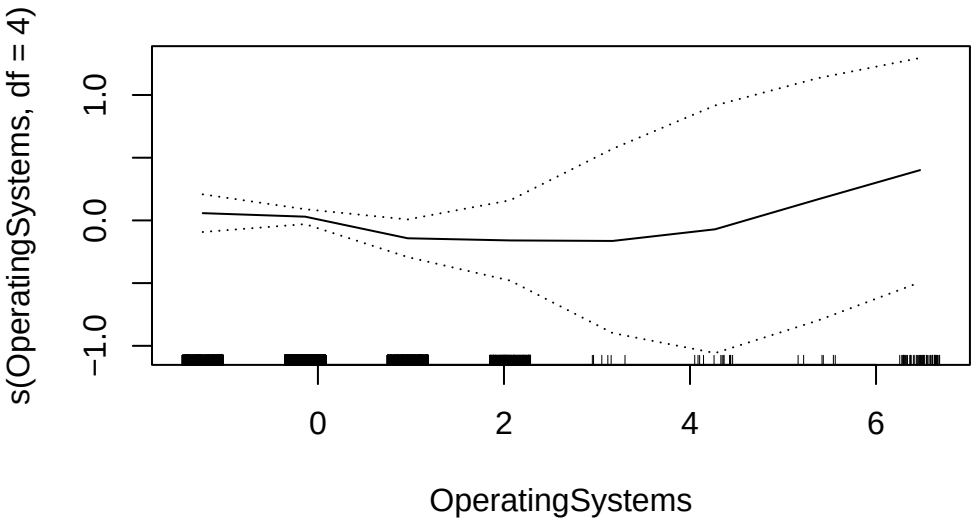

s(ProductRelated_Duration, df = 4)
10
5
0
−10
0
5
10
15
20
ProductRelated_Duration

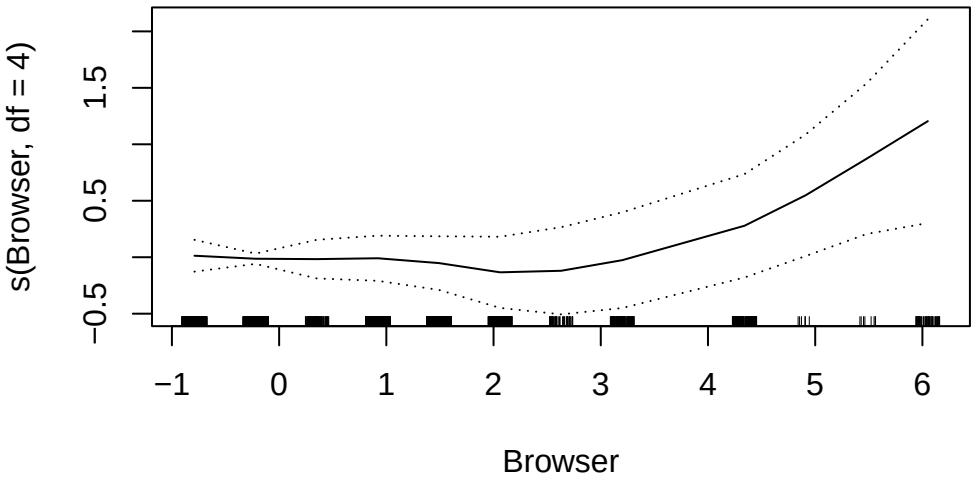

s(BounceRates, df = 4)
4
3
2
1
0
0
1
2
3
BounceRates

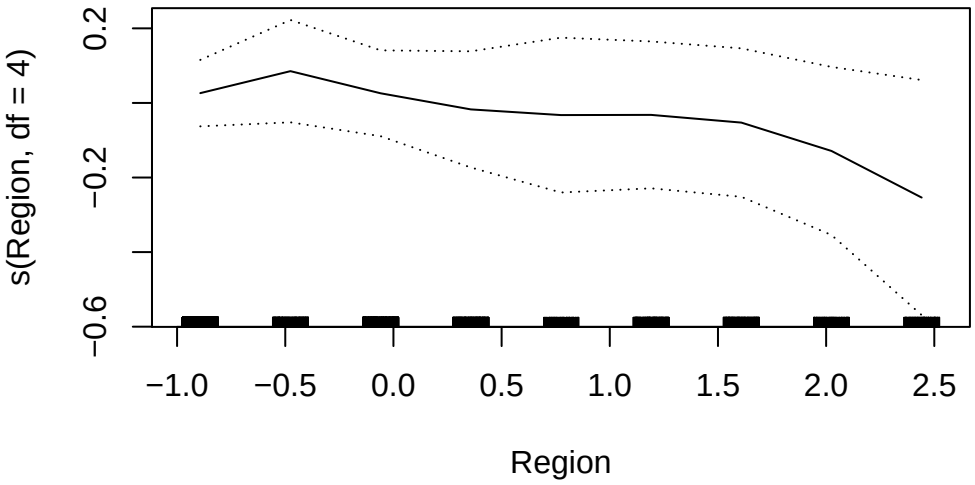

s(ExitRates, df = 4)
0
−2
−4
−6
−1
0
1
2
3
ExitRates

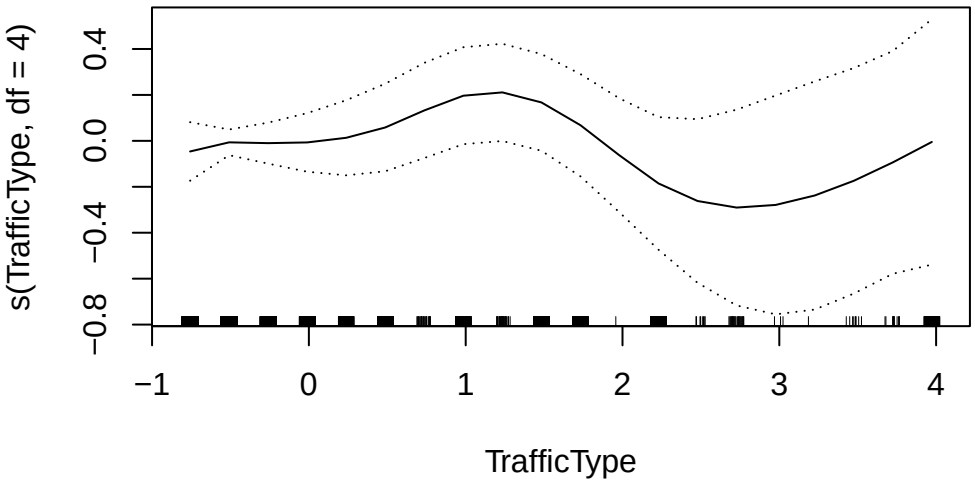

s(PageValues, df = 4)
0
2
4
6
8
0
5
10
15
PageValues

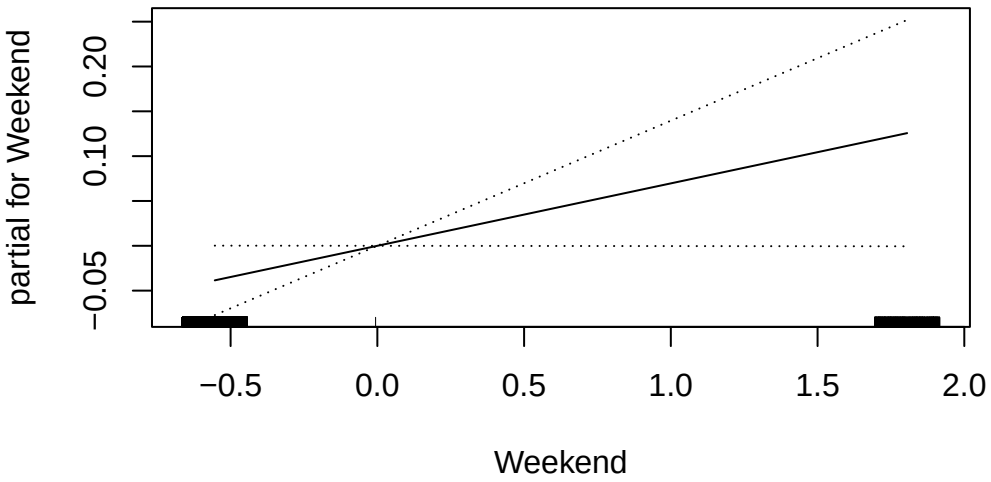

s(SpecialDay, df = 4)
−1.0
−0.5
0.0
0.5
0
1
2
3
4
5
SpecialDay

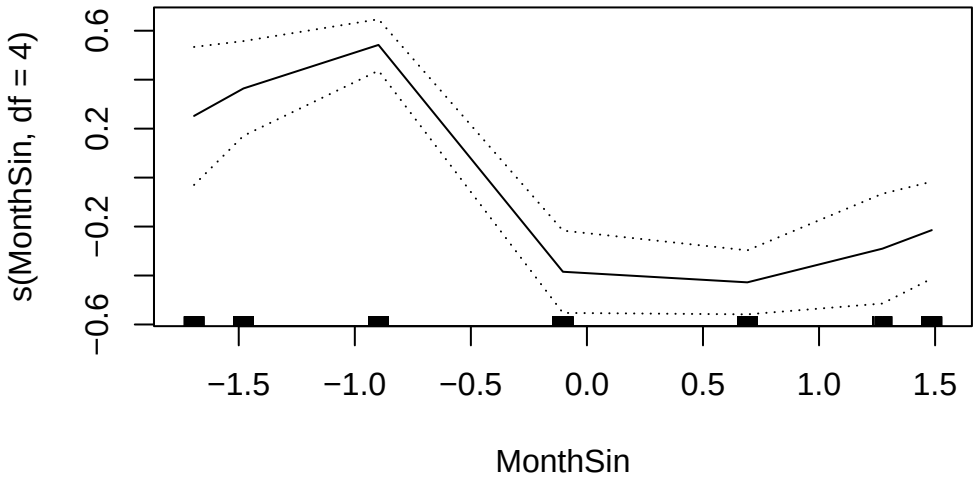

s(OperatingSystems, df = 4)
−1.0
0.0
1.0
0
2
4
6
OperatingSystems

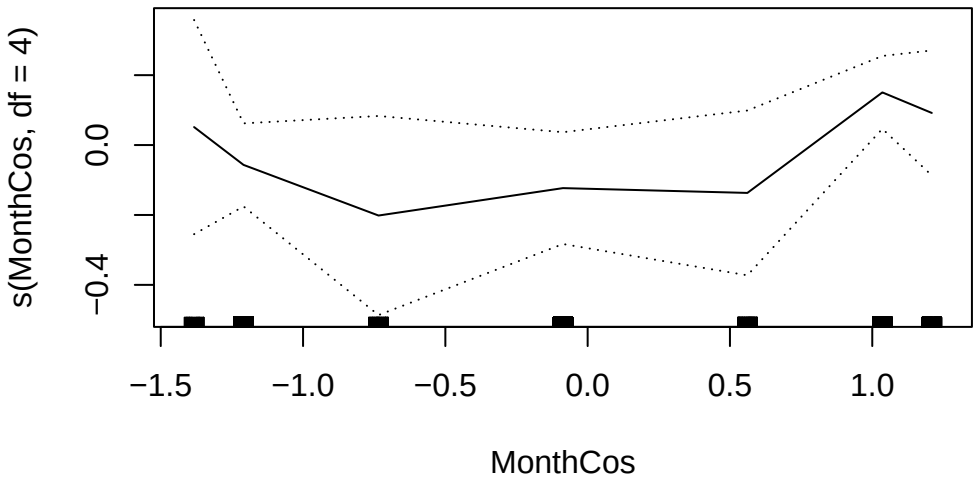

s(Browser, df = 4)
−0.5
0.5
1.5
−1
0
1
2
3
4
5
6
Browser

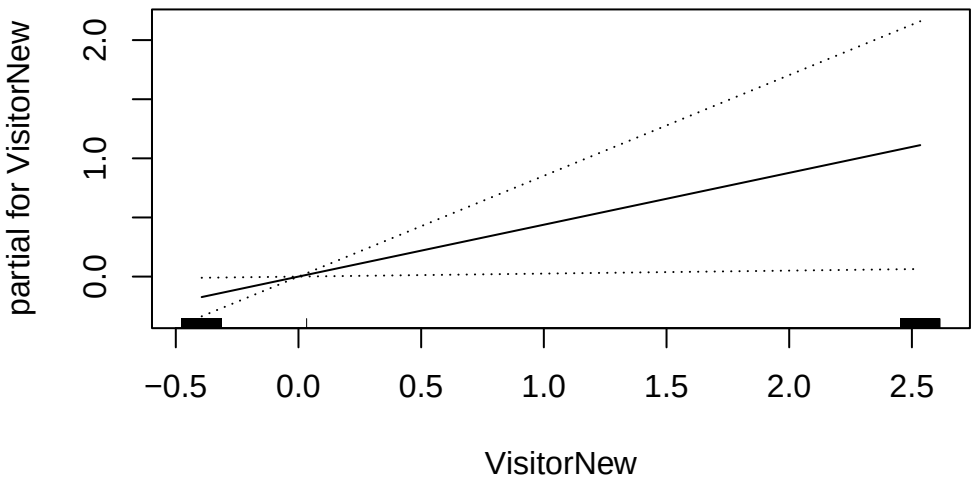

s(Region, df = 4)
−0.6
−0.2
0.2
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Region

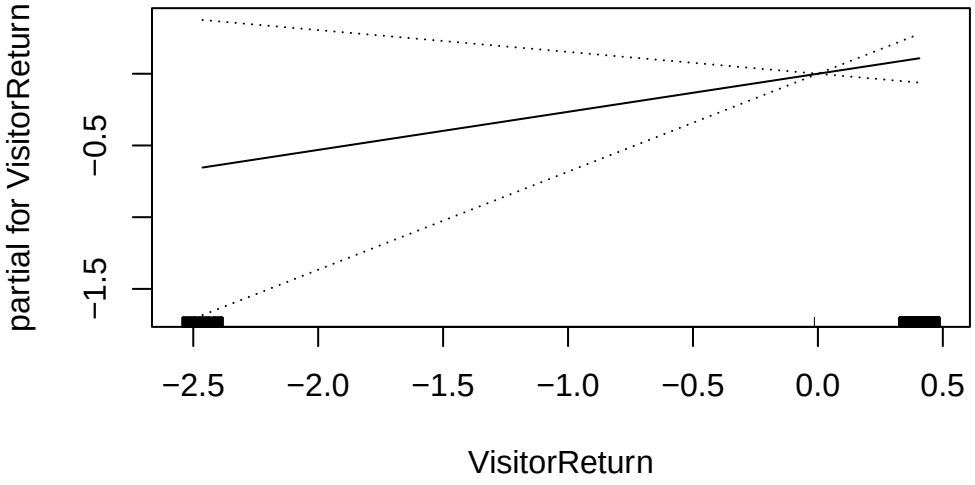

s(TrafficType, df = 4)
−0.8
−0.4
0.0
0.4
−1
0
1
2
3
4
TrafficType


partial for Weekend
−0.05
0.10
0.20
−0.5
0.0
0.5
1.0
1.5
2.0
Weekend


s(MonthSin, df = 4)
0.6
0.2
−0.2
−0.6
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
MonthSin


s(MonthCos, df = 4)
0.0
−0.4
−1.5
−1.0
−0.5
0.0
0.5
1.0
MonthCos






```
# Test on the test set
ss_pred <- predict(ss_fit, newdata=shoppers_test_s[-20],
                   type='response')
```

```r
# Predicts 1 if probability is greater than 0.5, 0 otherwise
ss_pred <- ifelse(ss_pred > 0.5, 1, 0)

# Compare predicted vs actual values to get performance metrics
confusionMatrix(as.factor(ss_pred), as.factor(shoppers_test_s$Revenue),
                positive="1")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2001  198
         1   89  178

               Accuracy : 0.8836
                 95% CI : (0.8703, 0.896)
    No Information Rate : 0.8475
    P-Value [Acc > NIR] : 1.451e-07

                  Kappa : 0.4889

 Mcnemar's Test P-Value : 1.829e-10

            Sensitivity : 0.47340
            Specificity : 0.95742
         Pos Pred Value : 0.66667
         Neg Pred Value : 0.90996
             Prevalence : 0.15247
         Detection Rate : 0.07218
   Detection Prevalence : 0.10827
      Balanced Accuracy : 0.71541

       'Positive' Class : 1
```

The smoothing spline gam had an accuracy of 0.8836 and a sensitivity of 0.47340, by far beating out the previous two methods. It also has the benefit of being able to perform inference to understand the significance of each feature. It's possible with more feature selection and tuning, in particular the df of each smoothing spline, the model would perform even better.

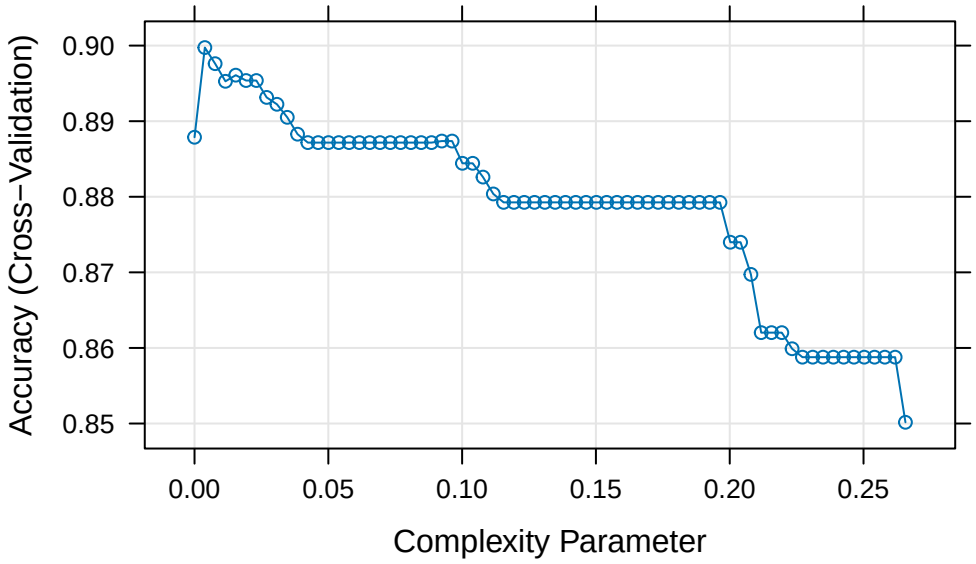
Single Tree Model

A decision tree model is non-parametric because it doesn't assume any parametric form for the predictors. The decision tree has the complexity parameter (cp), where a cp of 0 includes the 'full' tree as a possibility, and a cp of 1 results in a tree with no splits. So it can be thought of as how 'complex' we want the tree to be. Decision trees don't allow for formal inference like hypothesis testing and confidence intervals, however it does give a measure of feature importance. The model does perform variable selection because only the feature splits that provided a better model end up being included. So some of the features end up not being used at all, or at least very little. Predictors do not need to be standardized with a decision tree because the tree is based off of splits in the data at different predictors. It doesn't matter what scale those splits are decided on.

```
# Set the seed for repeatability
set.seed(400)

# Train the decision tree using cv to tune cp
st_fit <- train(as.factor(Revenue) ~ .,
                data = shoppers_train,
                method = 'rpart',
                tuneLength = 70,
                trControl = trainControl(method = 'cv',
                                         number = 10))

# Get the best performing cp value
st_fit$bestTune
```

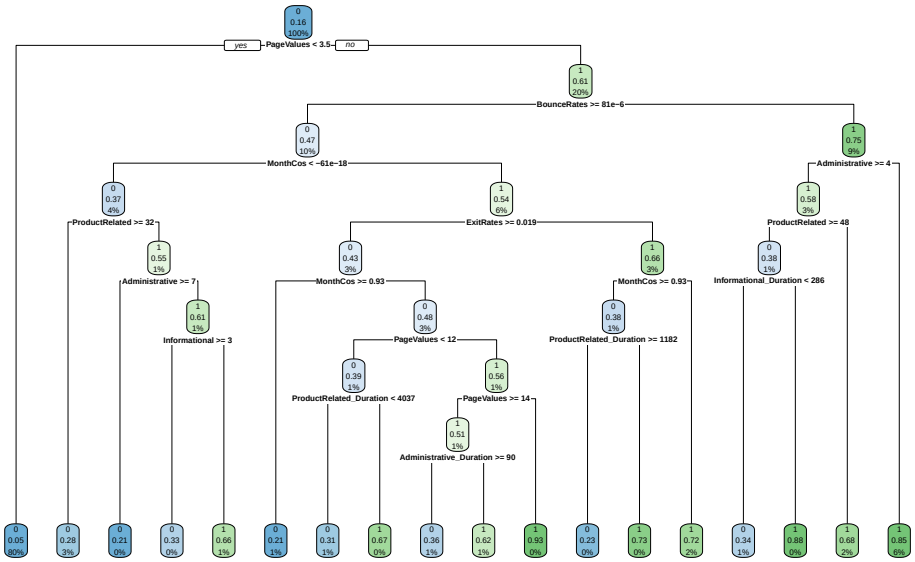
```
          cp
2 0.003850229
```

```
# Plots the cp vs model accuracy
plot(st_fit)
```

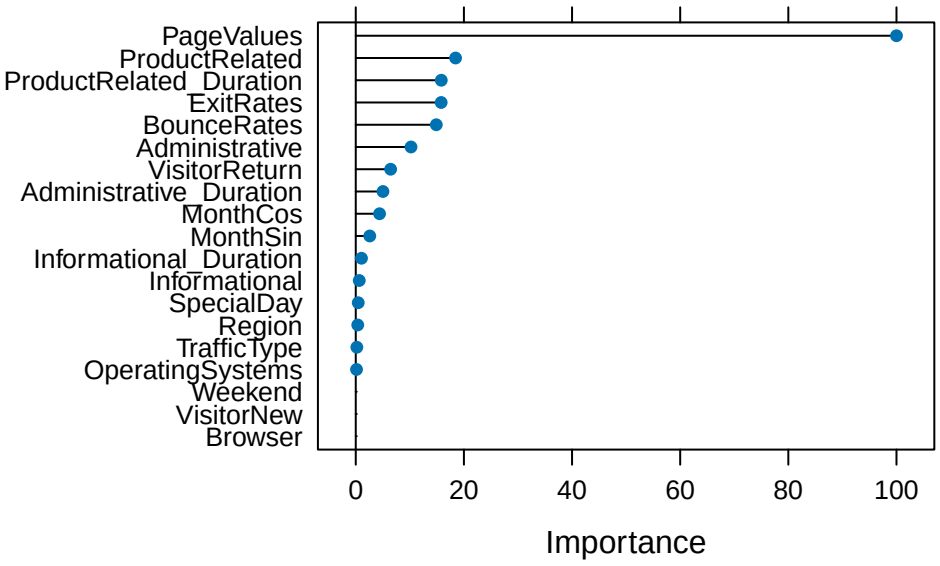



```
# Retrain the model using the best cp on the full training set
st_fit2 <- train(as.factor(Revenue) ~ .,
                 data = shoppers_train,
                 method = 'rpart',
                 tuneGrid = expand.grid(cp = st_fit$bestTune$cp))
```

```
# Perform pruning to find the best subset tree, helps reduce overfitting
st_best <- prune(st_fit2$finalModel, cp = st_fit$bestTune$cp)

# Get the tree with the decisions made at each split
rpart.plot(st_best)
```




```
# Plot the variable importances
plot(varImp(st_fit2))
```



```r
# Predict on the test set data
tree_pred <- predict(st_best, newdata = shoppers_test,
                           type = "class")

# Compare predicted vs actual values to get performance metrics
confusionMatrix(as.factor(tree_pred), as.factor(shoppers_test$Revenue),
                positive = '1')
```

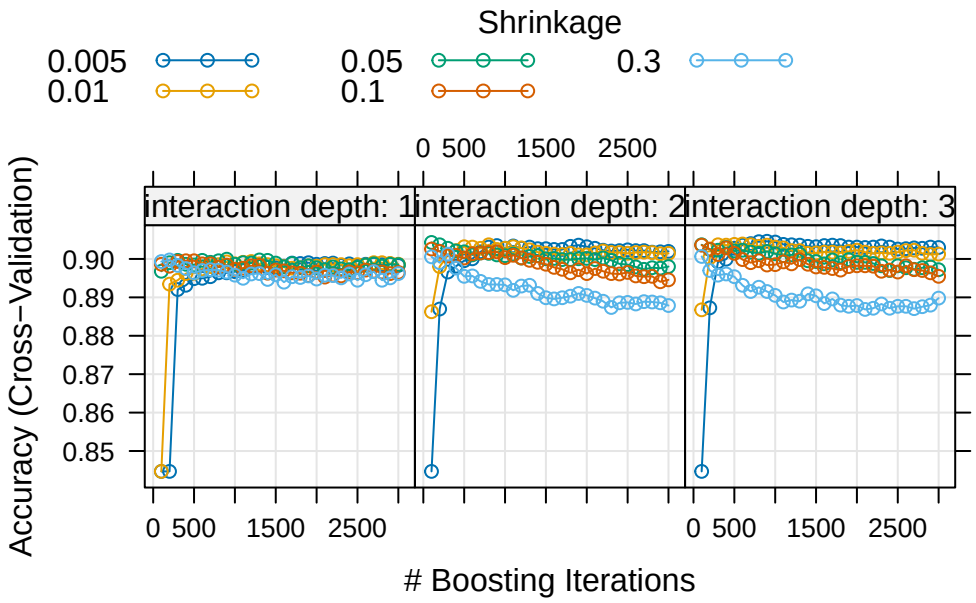
```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1983  159
         1  107  217

               Accuracy : 0.8921
                 95% CI : (0.8792, 0.9041)
    No Information Rate : 0.8475
    P-Value [Acc > NIR] : 7.296e-11

                  Kappa : 0.5575

 Mcnemar's Test P-Value : 0.001766

            Sensitivity : 0.5771
            Specificity : 0.9488
         Pos Pred Value : 0.6698
         Neg Pred Value : 0.9258
             Prevalence : 0.1525
         Detection Rate : 0.0880
   Detection Prevalence : 0.1314
      Balanced Accuracy : 0.7630

       'Positive' Class : 1
```

The decision tree had an accuracy of 0.8921 and a sensitivity of 0.5771. Thus far this is the best performing model. While the accuracy was basically the same, the model was better able to correctly predict that a user would shop. While the model doesn't offer the full inference abilities, the feature importances could still be used to learn more about which features were useful in this prediction problem.

Ensemble Tree Model

Like the decision tree the gradient boosting machines (gbm) model is also a non-parametric model. It combines many simple trees together that cast a weighted vote and learn over time to improve predictions. There are many different tuning parameters in a gbm model. Shrinkage is the learning rate lambda of the model. The interaction depth is the number of splits performed on any given tree. The number of trees is how many trees are in the total ensemble. The minobsinnode is the minimum number of observations in a node in order for the model to continue splitting. Like decision trees, ensemble tree models don't perform traditional inference, but do provide feature importances. The model performs variable selection by some features providing so little benefit to the model that they end up not being used. Finally, the predictors do not need to be standarized in a tree based model.

```
# Set seed for repeatability
set.seed(500)

# Grid for the tuning parameters
gr <- expand.grid(shrinkage = c(0.3, 0.1, 0.05, 0.01, 0.005),
                  interaction.depth = c(1, 2, 3),
                  n.trees = seq(100, 3000, by=100),
                  n.minobsinnode = 10)

# Train a gbm model using cv with the tuning grid
boost_fit <- train(as.factor(Revenue) ~ .,
                   data = shoppers_train,
                   method = 'gbm',
                   trControl = trainControl(method = 'cv',
                                            number = 5),
                   tuneGrid = gr,
                   verbose = FALSE)

# Plot the cv results
plot(boost_fit)
```

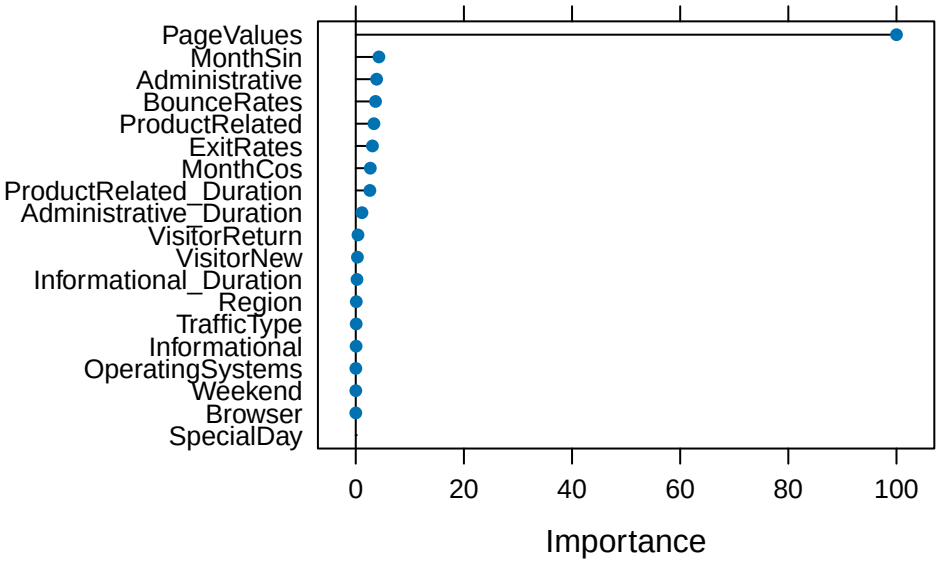



```
# Get the best tuning parameter values
boost_fit$bestTune
```

```
   n.trees interaction.depth shrinkage n.minobsinnode
69     900                 3     0.005             10
```

```
# Retrain with the best tuning parameters on the full training set
boost_best <- train(as.factor(Revenue) ~ .,
                    data = shoppers_train,
                    method = 'gbm',
                    trControl = trainControl('none'),
                    tuneGrid = expand.grid(shrinkage =boost_fit$bestTune$shrinkage,
                                           interaction.depth =
                                             boost_fit$bestTune$interaction.depth,
                                           n.trees = boost_fit$bestTune$n.trees,
                                           n.minobsinnode =
                                             boost_fit$bestTune$n.minobsinnode),
                    verbose = FALSE)
```

```
# Plot the variable importances
plot(varImp(boost_best))
```




```
# Predict on the test set
boost_pred <- predict(boost_best, newdata = shoppers_test)

# Compare predicted vs actual values to get performance metrics
confusionMatrix(as.factor(boost_pred), as.factor(shoppers_test$Revenue),
                positive = '1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1987  156
         1  103  220

               Accuracy : 0.895
                 95% CI : (0.8822, 0.9068)
    No Information Rate : 0.8475
    P-Value [Acc > NIR] : 3.895e-12

                  Kappa : 0.5687

 Mcnemar's Test P-Value : 0.001233
```

```
            Sensitivity : 0.58511
            Specificity : 0.95072
         Pos Pred Value : 0.68111
         Neg Pred Value : 0.92720
             Prevalence : 0.15247
         Detection Rate : 0.08921
   Detection Prevalence : 0.13098
      Balanced Accuracy : 0.76791

       'Positive' Class : 1
```

The gbm model had an accuracy of 0.8933 and a sensitivity of 0.58511. This is the best performance on this data set. It's also possible with more tuning and feature selection the model could have performed better. This model is only slightly better than the decision tree one. One benefit the decision tree has over the ensemble is the ability to see the final tree that the data was split on. That may or may not be an important enough factor in choosing a slightly worse model. The variable importances can also be used to get a better sense of which ones contributed most to the model. In this case, PageValues is by far the most important feature.

Support Vector Machines

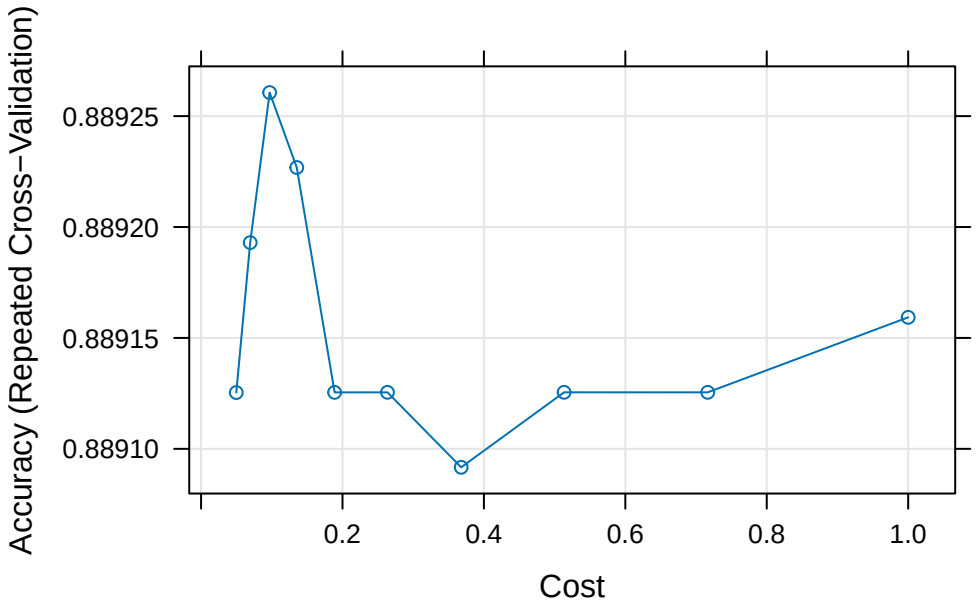
The svm model is non-parametric, using support vectors to define a margin that minimizes the number of violations. The tuning parameter is cost. When cost is small the margin will be wide and there will be more support vectors. When cost is large the margin will be narrow and there will be few support vectors. Different kernels could be used with the svm which could also count as a tuning parameter. The model can not be used for inference, and doesn't perform any variable selection. It is technically not needed to standardize for svm, but it doesn't hurt to standardize the predictors and may improve model performance.

```
# Set seed for repeatability
set.seed(600)

# Perform repeatedcv with 5-folds, 3 times each
tr <- trainControl(method = 'repeatedcv',
                   number = 5, repeats = 3)

# Values to test for tuning parameter cost
tune_grid_sv <- expand.grid(cost = exp(seq(-3, 0, len=10)))
```

```r
# Train the svm
sv_fit <- train(as.factor(Revenue) ~ .,
                data = shoppers_train_s,
                method = 'svmLinear2',
                tuneGrid = tune_grid_sv,
                trControl = tr)

summary(sv_fit)
```

```
Call:
svm.default(x = as.matrix(x), y = y, kernel = "linear", cost = param$cost,
    probability = classProbs)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.09697197

Number of Support Vectors:  2387

 ( 1201 1186 )


Number of Classes:  2

Levels:
 0 1
```

```r
# Plot the cost tuning parameter vs model accuracy
plot(sv_fit)
```




```
# Show the best cost value
sv_fit$bestTune
```

```
        cost
3 0.09697197
```

```
# Retrain the model on the full training set with the best cost
sv_best <- svm(as.factor(Revenue) ~ .,
               data = shoppers_train_s,
               type = 'C-classification',
               kernel = 'linear',
               cost = sv_fit$bestTune$cost)
```

```
# Predict on the test set
svm_pred <- predict(sv_best, newdata = shoppers_test_s)

# Compare predicted vs actual values to get performance metrics
confusionMatrix(as.factor(svm_pred), as.factor(shoppers_test_s$Revenue),
                positive = '1')
```

```
Confusion Matrix and Statistics
```

```
          Reference
Prediction    0    1
         0 2027  240
         1   63  136

               Accuracy : 0.8771
                 95% CI : (0.8635, 0.8898)
    No Information Rate : 0.8475
    P-Value [Acc > NIR] : 1.51e-05

                  Kappa : 0.4109

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.36170
            Specificity : 0.96986
         Pos Pred Value : 0.68342
         Neg Pred Value : 0.89413
             Prevalence : 0.15247
         Detection Rate : 0.05515
   Detection Prevalence : 0.08070
      Balanced Accuracy : 0.66578

       'Positive' Class : 1
```

The svm had an accuracy of 0.8771 and a sensitivity of 0.36170. This is worse than some of the other models. Additionally, svm doesn't perform variable selection or inference, so other models would be a better fit for this problem.

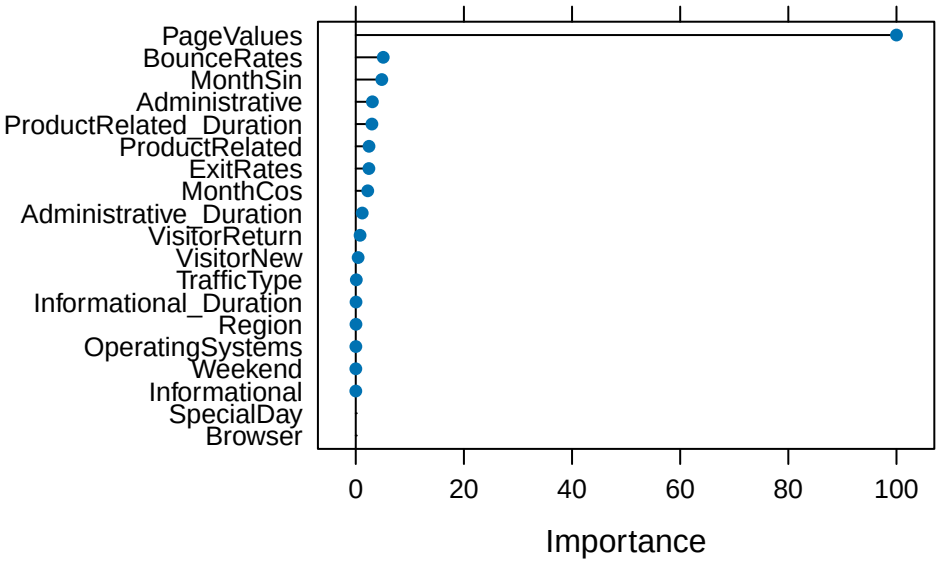
Testing Models

Fit your overall best model on the entire data set.

```
# Retrain best gbm model on the entire data set
boost_final <- train(as.factor(Revenue) ~ .,
                     data = shoppers_df,
                     method = 'gbm',
                     trControl = trainControl('none'),
                     tuneGrid = expand.grid(shrinkage=boost_fit$bestTune$shrinkage,
                                            interaction.depth =
```

```
                                                boost_fit$bestTune$interaction.depth,
                                              n.trees = boost_fit$bestTune$n.trees,
                                              n.minobsinnode =
                                                boost_fit$bestTune$n.minobsinnode),
                      verbose = FALSE)
```

```
# Plot the variable importances
plot(varImp(boost_final))
```



While the gbm model performed the best overall on accuracy and sensitivity in the training and test set, it is difficult to interpret the model. The only sort of inference that can be done is based on the variable importance measures. This is the measure of how much each variable contributed to the model's ability to make accurate predictions. It's clear that PageValues was by far the most important feature. It's a feature that is tracked directly before the user purchases something, so it makes sense that it would be especially useful in predicting if a user will shop. The features Browser and SpecialDay had no importance which means they were not used by the model. You would think SpecialDay would matter for when a user might purchase something, but the variable was tracked across many days before and after the actual special day, so that may have affected it's ability to discern. It also seems that being a Weekend and the number of OperatingSystems used weren't important features. Informational and it's duration didn't contribute much as well, so perhaps people going online to learn something don't tend to make purchases. Of course most of this is just speculation. All the feature

importances really tells us is what features were important in the model making accurate predictions.