ST563 Project

Jarrett Glass

2025-04-02

Analysis of Bike Sharing Data of Casual Riders

There has been a new-found development toward biking as an enthusiastic hobby of the author, who recently acquired an e-bike and has found using it to be an enjoyable form of exercise. He has an interest in examining how other bicyclists may approach this activity as well - namely, if there are specific times or conditions for other riders (including casual riders versus less-casual riders) that data indicate other riders would be most likely to participate.

This project will include an analysis of the Bike Sharing data set from the UCI Machine Learning Repository. It is readily available to load into R through the `ISLR2` package.

```
# Load the requisite libraries and dataframe into R
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.3.3
```

```
data(Bikeshare)
```

According to the R documentation, the columns in this data set are:

Variable name	Content	Expected type
`season`	Season of the year. Winter, Spring, Summer, Fall equals 1, 2, 3, 4, respectively.	Factor
`mnth`	Month of the year	Factor
`day`	Day of the year, 1-365	Factor

Variable name	Content	Expected type
`hr`	Hour of the day, factor from 0 to 23	Factor
`holiday`	Whether the day in question is a holiday (0 or 1 for False or True)	Factor
`weekday`	Day of the week, coded Sunday through Saturday as 0 through 6 respectively.	Factor
`workingday`	Whether this is a working day (Yes=1, No=0)	Factor
`weathersit`	The weather situation. Contains four levels: `clear`, `cloudy/misty`, `light rain/snow`, `heavy rain/snow`.	Factor
`temp`	Temperature (normalized) in Celsius, where values are derived $(t - t_{min})/(t_{max} - t_{min})$, with $t_{min} = -8$ and $t_{max} = +39$.	Numeric
`atemp`	Normalized "feeling" temperature in Celsius - same derivation as above, with $t_{min} = -16, t_{max} = +50$.	Numeric
`hum`	Normalized humidity, with values divided to 100.	Numeric
`windspeed`	Normalized wind speed, with all values divided by the max value of 67.	Numeric
`casual`	The number of casual bikers	Numeric
`registered`	The number of registered bikers	Numeric
`bikers`	Total number of bikers (`casual + registered`)	Numeric

The response variable with this analysis will be the `casual` variable, as the primary interest is analyzing the habits of other biker hobbyists.

Data Cleaning

The numbers of casual riders are not strictly needed here, instead an approximation of the magnitude is appropriate. The interest is in knowing if a specific set of conditions would have lesser, moderate, or a large amount of casual riders. The variable `casual` will be transformed into a categorical variable based on the numerical entries falling into the bottom, middle, or top third of the values.

```
# Transform `casual` into a three-level factor (levels 1, 2, 3)
Bikeshare$casual <- factor(cut(Bikeshare$casual,
                        breaks=quantile(Bikeshare$casual, c(0, 1/3, 2/3, 1)),
                        labels=1:3, include.lowest=TRUE))
```

The variable for total `bikers` can be removed, however the variable for `registered` bikers will remain - just in case the number of these less-casual bikers may have an influence on the number of casual bikers in attendance. Similarly, the variables `mnth` and `day` will be removed, since the specific day or month of the year is not of concern here.

```
Bikeshare <- Bikeshare[, !(names(Bikeshare) %in% c("bikers","day","mnth"))]
```

Most of the factor variables indicated above are presently identified as numeric variables, and these will need to be corrected.

```
# The current list of variables and their type
data.frame(Type=sapply(Bikeshare, class)) |> kable()
```

	Type
season	numeric
hr	factor
holiday	numeric
weekday	numeric
workingday	numeric
weathersit	factor
temp	numeric
atemp	numeric
hum	numeric
windspeed	numeric
casual	factor
registered	numeric

```
# Transform the indicated columns into factors
Bikeshare <- Bikeshare %>%
  mutate(across(c(season, holiday, weekday, workingday), factor))

# Confirm columns and class types
str(Bikeshare)
```

```
## 'data.frame':    8645 obs. of  12 variables:
##  $ season    : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
##  $ hr        : Factor w/ 24 levels "0","1","2","3",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ holiday   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ weekday   : Factor w/ 7 levels "0","1","2","3",..: 7 7 7 7 7 7 7 7 7 7 ...
##  $ workingday: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ weathersit: Factor w/ 4 levels "clear","cloudy/misty",..: 1 1 1 1 1 2 1 1 1 1 ...
##  $ temp      : num  0.24 0.22 0.22 0.24 0.24 0.24 0.22 0.2 0.24 0.32 ...
##  $ atemp     : num  0.288 0.273 0.273 0.288 0.288 ...
##  $ hum       : num  0.81 0.8 0.8 0.75 0.75 0.75 0.8 0.86 0.75 0.76 ...
##  $ windspeed : num  0 0 0 0 0 0.0896 0 0 0 0 ...
##  $ casual    : Factor w/ 3 levels "1","2","3": 1 2 1 1 1 1 1 1 1 2 ...
##  $ registered: num  13 32 27 10 1 1 0 2 7 6 ...
```

```
# Confirming that there are no NA values in the data set
colSums(is.na(Bikeshare))
```

```
##     season         hr    holiday    weekday workingday weathersit       temp
##          0          0          0          0          0          0          0
##      atemp        hum  windspeed     casual registered
##          0          0          0          0          0
```

Split the Data into a Train and Test Set

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.3.3

## Loading required package: lattice
```

```
datasplit <- createDataPartition(Bikeshare$casual, p=0.7, times=1, list=FALSE)
train <- Bikeshare[datasplit, ]
test <- Bikeshare[-datasplit, ]

accuracies <- as.data.frame(matrix(NA, ncol=2)) # For tracking the accuracy of each model
colnames(accuracies) <- c("Model","Accuracy")
```

Training the models

```
set.seed(0218) # For reproducibility
```

The next several sections will involve training the data on different models, and comparing to see which produces the most accurate predictions against the test set.

k-Nearest Neighbors

The k-Nearest Neighbors model is a non-parametric learning algorithm with tuning parameter `k` to denote how many "closest neighbors" to a value should be evaluated to make a prediction for a new data point. The prediction is based on the majority class or average value of these `k` neighbors. This model is typically used to make predictions and is not well-suited to inference, nor does it perform variable selection. It also does not have any specific need to standardize its predictors, but that could be performed.

```
# The library `caret` will be used for this (previously loaded)

# Set up the KNN for 5-fold cross-validation.
# This will tune to find the optimum value of `k` (evaluating possibilities 1:15)
knn <- train(casual ~ ., data=train, method="knn",
             tuneGrid=expand.grid(k=1:15),
             trControl=trainControl(method="cv", number=5))

# The optimum value of `k` is:
knn$bestTune$k
```

```
## [1] 5
```

```
# Repeat this exercise, using the best value of K provided.
knn_tuned <- train(casual ~ ., data=train, method="knn",
                   tuneGrid=knn$bestTune,
                   trControl=trainControl(method="cv", number=5))

# Make predictions using the test set, based on this tuned model.
preds <- predict(knn_tuned, newdata=test)
conf <- confusionMatrix(preds, reference=test$casual)
conf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3
##          1 757 156  11
##          2 147 418 147
##          3  24 232 700
##
## Overall Statistics
##
##                Accuracy : 0.7234
##                  95% CI : (0.7057, 0.7405)
##     No Information Rate : 0.358
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5839
##
##  Mcnemar's Test P-Value : 2.314e-05
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
```

```
## Sensitivity            0.8157   0.5186   0.8159
## Specificity            0.8996   0.8354   0.8524
## Pos Pred Value         0.8193   0.5871   0.7322
## Neg Pred Value         0.8975   0.7936   0.9034
## Prevalence             0.3580   0.3110   0.3310
## Detection Rate         0.2921   0.1613   0.2701
## Detection Prevalence   0.3565   0.2747   0.3688
## Balanced Accuracy      0.8577   0.6770   0.8341
```

```r
accuracies <- rbind(accuracies, c("K-Nearest Neighbors", conf$overall[[1]]))
```

Regularized Logistic Regression

This model, regularized logistic regression, is a *parametric* model with two tuning parameters, α and λ. The α parameter can dictate the type of regularization that is performed - in other words, if $\alpha = 0$ corresponds to Ridge Regression, $\alpha = 1$ to LASSO, and anything $0 < \alpha < 1$ to Elastic Net. The λ parameter dictates the strength of the regularization. This model can be used for inference with Ridge Regression, and LASSO enables variable selection. Standardization of the predictors is completed by the `glmnet` function.

```
# Load the necessary library
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.3.3
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
# Multinomial logistic regression with L1 regularization.
# Transform `casual` in train, test sets to (0,1,2) to work with glmnet.
train.glmnet <- train |> mutate(casual=as.numeric(casual)-1)
test.glmnet <- test |> mutate(casual=as.numeric(casual)-1)

# Set up data matrices
train.matrix.glmnet <- model.matrix(casual ~ ., data=train.glmnet)
test.matrix.glmnet <- model.matrix(casual ~ ., data=test.glmnet)

# Perform 5-fold CV to determine optimal lambda value
cv_glmnet <- cv.glmnet(x=train.matrix.glmnet,
                       y=train.glmnet$casual,
                       family="multinomial",
                       alpha=1, # For LASSO
                       nfolds=5)

# The optimal lambda was calculated to be
cv_glmnet$lambda.min
```

```
## [1] 0.0002255372
```

```
# The training model, using optimized lambda, is
logist <- glmnet(x=train.matrix.glmnet,
                 y=train.glmnet$casual,
                 family="multinomial",
                 alpha=1, # LASSO
                 lambda=cv_glmnet$lambda.min)

# Make predictions on test set using this tuned model
preds <- predict(logist, newx=test.matrix.glmnet, type="class")

# Convert these predictions to be comparable to original `test` dataframe
preds <- factor(preds, levels=c(0,1,2), labels=1:3)

# Generate confusion matrix
conf <- confusionMatrix(preds, reference=test$casual)
conf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3
##          1 812 143   0
##          2 113 550 117
##          3   3 113 741
##
## Overall Statistics
##
##                Accuracy : 0.8113
##                  95% CI : (0.7957, 0.8262)
##     No Information Rate : 0.358
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.7163
##
##  Mcnemar's Test P-Value : 0.08636
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8750   0.6824   0.8636
## Specificity            0.9141   0.8712   0.9331
## Pos Pred Value         0.8503   0.7051   0.8646
## Neg Pred Value         0.9291   0.8587   0.9326
## Prevalence             0.3580   0.3110   0.3310
## Detection Rate         0.3133   0.2122   0.2859
## Detection Prevalence   0.3684   0.3009   0.3306
## Balanced Accuracy      0.8945   0.7768   0.8984
```

```
accuracies <- rbind(accuracies, c("Regularized Logistic Regression", conf$overall[[1]]))
```

General Additive Model

GAM models are *semi*-parametric, with tuning parameters that may be used to control the smoothness of the splines (but are not necessary). These models extend linear models by allowing for non-linear relationships through the use of basis functions and splines. This model can be used for inference and may be used for variable selection with appropriate basis functions or techniques like penalizing splines. The predictors in a GAM model do not need to be standardized, but some transformation techniques could greatly assist with smoothing predictions or estimations.

```
# Load the requisite library
library(gam)
```

```
## Warning: package 'gam' was built under R version 4.3.3
```

```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 4.3.3
```

```
## Loaded gam 1.22-4
```

```
# The GAM in logistic regression requires binary response variables.
# This response has 3 levels, but it's necessary to convert each to a binary response.

train.gam <- train
train.gam$casual_low <- ifelse(train$casual == 1, 1, 0)
train.gam$casual_mid <- ifelse(train$casual == 2, 1, 0)
train.gam$casual_hig <- ifelse(train$casual == 3, 1, 0)

test.gam <- test
test.gam$casual_low <- ifelse(test$casual == 1, 1, 0)
test.gam$casual_mid <- ifelse(test$casual == 2, 1, 0)
test.gam$casual_hig <- ifelse(test$casual == 3, 1, 0)

# Separate GAM models are created for each binary outcome.
gam_low <- gam(casual_low ~ s(temp) + s(atemp) + s(hum) + s(windspeed) + s(registered) +
                 season + hr + holiday + weekday + workingday + weathersit,
               data=train.gam,
               family=binomial())
gam_mid <- gam(casual_mid ~ s(temp) + s(atemp) + s(hum) + s(windspeed) + s(registered) +
                 season + hr + holiday + weekday + workingday + weathersit,
               data=train.gam,
               family=binomial())
gam_hig <- gam(casual_hig ~ s(temp) + s(atemp) + s(hum) + s(windspeed) + s(registered) +
                 season + hr + holiday + weekday + workingday + weathersit,
               data=train.gam,
               family=binomial())

# Predict the probabilities for each class, then combine into one matrix
preds <- cbind(predict(gam_low, newdata=test.gam, type="response"),
               predict(gam_mid, newdata=test.gam, type="response"),
               predict(gam_hig, newdata=test.gam, type="response"))
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases
```

```
preds <- apply(preds, 1, which.max)
preds <- as.factor(preds)

# Generate confusion matrix
conf <- confusionMatrix(preds, reference=test$casual)
conf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3
##          1 832 166   0
##          2  94 505  91
##          3   2 135 767
##
## Overall Statistics
##
##                Accuracy : 0.8117
##                  95% CI : (0.7961, 0.8266)
##     No Information Rate : 0.358
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7164
##
##  Mcnemar's Test P-Value : 1.081e-06
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8966   0.6266   0.8939
## Specificity            0.9002   0.8964   0.9210
## Pos Pred Value         0.8337   0.7319   0.8485
## Neg Pred Value         0.9398   0.8417   0.9461
## Prevalence             0.3580   0.3110   0.3310
## Detection Rate         0.3210   0.1948   0.2959
## Detection Prevalence   0.3850   0.2662   0.3488
## Balanced Accuracy      0.8984   0.7615   0.9075
```

```
accuracies <- rbind(accuracies, c("Generalized Additive Model", conf$overall[[1]]))
```

Single tree model

Single tree models are non-parametric models that partition the predictor space into rectangular regions, opposed to linear ones. These models can be used for inference and inference, and can perform variable selection during the tree-building process by identifying the variables that are most influential on the response. These can be tuned by controlling for tree depth or node size, and by pruning them down after initial growth to reduce complexity and prevent overfitting. Standardization of trees is generally not required.

```
# Use the package `rpart` for Classification Trees
library(rpart)

# Generate the tree model
tree_mod <- rpart(casual ~ ., data=train,
                  method="class",
                  parms=list(split="information"),
                  control=rpart.control(xval=10,
                                        minsplit=15,
                                        minbucket=15))

# Summary of the tree
cp_table <- printcp(tree_mod)
```

```
##
## Classification tree:
## rpart(formula = casual ~ ., data = train, method = "class", parms = list(split = "information"),
##     control = rpart.control(xval = 10, minsplit = 15, minbucket = 15))
##
## Variables actually used in tree construction:
## [1] atemp      registered temp       workingday
##
## Root node error: 3886/6053 = 0.642
##
## n= 6053
##
##         CP nsplit rel error  xerror      xstd
## 1 0.418425      0   1.00000 1.00000 0.0095983
## 2 0.066907      1   0.58157 0.58749 0.0097037
## 3 0.059701      2   0.51467 0.54169 0.0095351
## 4 0.029336      3   0.45497 0.47864 0.0092370
## 5 0.012095      5   0.39629 0.40968 0.0088145
## 6 0.010293      6   0.38420 0.40015 0.0087476
## 7 0.010000      7   0.37391 0.38626 0.0086458
```

```
cp_table
```

```
##           CP nsplit rel error    xerror        xstd
## 1 0.41842512      0 1.0000000 1.0000000 0.009598265
## 2 0.06690685      1 0.5815749 0.5874936 0.009703658
## 3 0.05970149      2 0.5146680 0.5416881 0.009535127
## 4 0.02933608      3 0.4549665 0.4786413 0.009236996
## 5 0.01209470      5 0.3962944 0.4096758 0.008814537
## 6 0.01029336      6 0.3841997 0.4001544 0.008747557
## 7 0.01000000      7 0.3739063 0.3862584 0.008645763
```

```r
# The number of splits producing the lowest cross-validation error is
min_index <- which.min(cp_table[,"xerror"])
cp_table[min_index, "nsplit"]
```

```
## [1] 7
```

```r
# The lowest number of splits that is within 1SE of this is
oneSE_threshold <- sum(cp_table[min_index, c("xerror","xstd")])
opt_cp <- cp_table[which(cp_table[,"xerror"] <= oneSE_threshold)[1], "CP"]
opt_cp
```

```
## [1] 0.01
```

```r
# Using this tuned CP, update the model with some pruning.
pruned_mod <- prune(tree_mod, cp=opt_cp)

# Generate predictions using the test set
preds <- predict(pruned_mod, newdata=test, type="class")

# Generate confusion matrix
conf <- confusionMatrix(preds, reference=test$casual)
conf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3
##          1 777 172   0
##          2 141 461 164
##          3  10 173 694
##
## Overall Statistics
##
##                Accuracy : 0.7454
##                  95% CI : (0.7281, 0.7621)
##     No Information Rate : 0.358
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6171
##
##  Mcnemar's Test P-Value : 0.004011
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8373   0.5720   0.8089
## Specificity            0.8966   0.8292   0.8945
## Pos Pred Value         0.8188   0.6018   0.7913
## Neg Pred Value         0.9081   0.8111   0.9044
## Prevalence             0.3580   0.3110   0.3310
## Detection Rate         0.2998   0.1779   0.2677
## Detection Prevalence   0.3661   0.2955   0.3383
## Balanced Accuracy      0.8670   0.7006   0.8517
```

```r
accuracies <- rbind(accuracies, c("Single Tree Model", conf$overall[[1]]))
```

Ensemble tree models

These models, including Random Forest, Gradient Boosting Machines, and others, are non-parametric models that do combine multiple trees to improve prediction accuracy. These are not well developed for inference, but can give insight into feature performance, and through this can perform variable selection. Similar to the single tree models, these can be tuned by the number of trees in the model, the number of predictors analyzed at each split, learning rate, tree depth, and other regularization factors. These are generally not sensitive to transformations and don't require standardization of the predictors.

```
# The library `caret` will be used here, first findind the best tuning value

bagged_mod <- train(casual ~ ., method="rf", data=train,
                    tuneGrid=expand.grid(mtry=1:(ncol(train)-1)),
                    trControl=trainControl(method="oob", number=3000))
bagged_mod$results |> round(4) |> kable()
```

Accuracy	Kappa	mtry
0.6491	0.4620	1
0.7988	0.6964	2
0.8226	0.7330	3
0.8288	0.7426	4
0.8272	0.7402	5
0.8236	0.7347	6
0.8260	0.7385	7
0.8242	0.7358	8
0.8226	0.7333	9
0.8227	0.7336	10
0.8274	0.7405	11

```
# Best number of predictors at each split (produces highest accuracy) is
bagged_mod$bestTune
```

```
##   mtry
## 4    4
```

```
# Regenerating this model with the tuned number of parameters,
bagged_tuned <- train(casual ~ ., method="rf", data=train,
                      tuneGrid=bagged_mod$bestTune,
                      trControl=trainControl(method="oob", number=3000))

# And making predictions based on the test set, and comparing with confusion matrix
preds <- predict(bagged_tuned, newdata=test)
conf <- confusionMatrix(preds, reference=test$casual)
conf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3
##          1 822 143   0
```

```
##          2 106 560 120
##          3   0 103 738
##
## Overall Statistics
##
##                Accuracy : 0.8179
##                  95% CI : (0.8025, 0.8326)
##     No Information Rate : 0.358
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7262
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8858   0.6948   0.8601
## Specificity            0.9141   0.8735   0.9406
## Pos Pred Value         0.8518   0.7125   0.8775
## Neg Pred Value         0.9348   0.8638   0.9315
## Prevalence             0.3580   0.3110   0.3310
## Detection Rate         0.3171   0.2160   0.2847
## Detection Prevalence   0.3723   0.3032   0.3245
## Balanced Accuracy      0.8999   0.7841   0.9004
```

```
accuracies <- rbind(accuracies, c("Ensemble Tree Model", conf$overall[[1]]))
```

Support vector machines

SVMs are non-parametric models that aim to find an optimal hyperplane to separate data into specific classes. These do not explicitly perform variable selection, but the process of generating these does implicitly highlight some variables as more 'important' to the process than others. These are primarily used for prediction, and are not generally used for inference. These models are sensitive to scaling and do require standardization of the predictors. They have two tuning parameters, *cost* (the penalty for misclassified data points) and *gamma* (influences the shape of the hyperplane, where a higher *gamma* results in more complex boundaries and more potential for overfitting).

```
# Load the requisite library
library(e1071)

# Fit an SVM model for tuning to 10-fold CV
svm_mod <- tune(svm, casual~., data=train, kernel="radial",
                ranges=list(cost=c(10, 15, 20, 25),
                            gamma=c(0.001, 0.01, 0.1)),
                tunecontrol=tune.control(cross=10))

# This tunes the cost and gamma parameters simultaneously.
opt_cost <- svm_mod$best.parameters$cost
opt_gamma <- svm_mod$best.parameters$gamma
data.frame(var=c("Optimal Cost", "Optimal Gamma"), opts=c(opt_cost, opt_gamma)) |>
  kable(col.names=NULL)
```

Optimal Cost	15.00
Optimal Gamma	0.01

```
# Use optimal cost and gamma parameters to generate tuned model
svm_tuned <- svm(casual ~ ., data=train, kernel="radial",
                 cost=opt_cost, gamma=opt_gamma)

# Generate predictions on the test set and confirm with confusion matrix
preds <- predict(svm_tuned, newdata=test)
conf <- confusionMatrix(preds, reference=test$casual)
conf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   2   3
##          1 818 140   0
##          2 108 557 106
##          3   2 109 752
##
## Overall Statistics
##
##                Accuracy : 0.8206
##                  95% CI : (0.8053, 0.8352)
##     No Information Rate : 0.358
##     P-Value [Acc > NIR] : <2e-16
##
```

```
##                   Kappa : 0.7302
##
##  Mcnemar's Test P-Value : 0.1036
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8815   0.6911   0.8765
## Specificity            0.9159   0.8802   0.9360
## Pos Pred Value         0.8539   0.7224   0.8714
## Neg Pred Value         0.9327   0.8633   0.9387
## Prevalence             0.3580   0.3110   0.3310
## Detection Rate         0.3156   0.2149   0.2901
## Detection Prevalence   0.3696   0.2975   0.3329
## Balanced Accuracy      0.8987   0.7856   0.9062
```

```r
accuracies <- rbind(accuracies, c("Support Vector Machine", conf$overall[[1]]))
```

Comparing the results

Each model has been generated, tuned, re-generated, and used to make predictions that have been compared to actual data values. The below table is a record of the recorded accuracy of each model:

```
accuracies[-1, ] |> kable(row.names=FALSE)
```

Model	Accuracy
K-Nearest Neighbors	0.72337962962963
Regularized Logistic Regression	0.811342592592593
Generalized Additive Model	0.811728395061728
Single Tree Model	0.74537037037037
Ensemble Tree Model	0.817901234567901
Support Vector Machine	0.820601851851852

The data indicate that, of all possible models analyzed here, the Support Vector Machine does produce the highest accuracy. This model will be extended to the entire `Bikeshare` data set with an eye still toward analyzing the `casual` response variable -

```
final_model <- train(casual ~ .,
                     data=Bikeshare,
                     method="svmRadial",
                     tuneGrid=data.frame(C=opt_cost, sigma=opt_gamma))
```

```
## Warning in .local(x, ...): Variable(s) '' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) '' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) '' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) '' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) '' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) '' constant. Cannot scale data.
```

```
varImp(final_model)
```

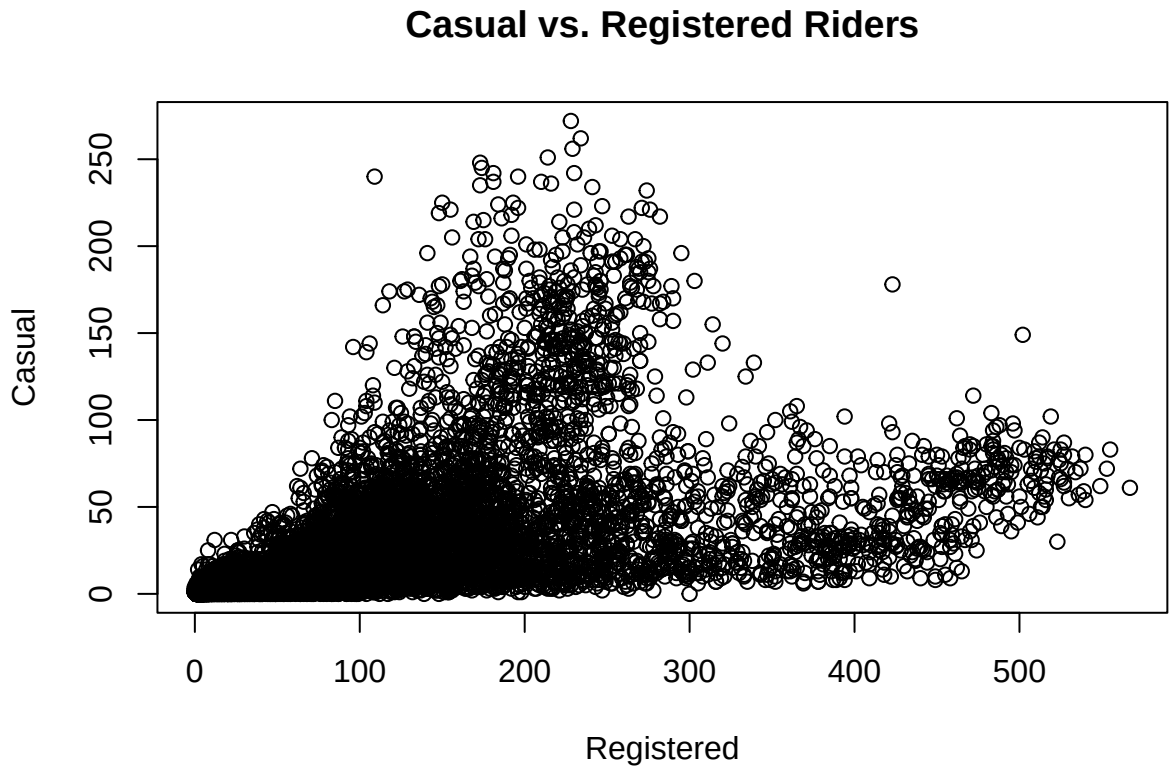
```
## ROC curve variable importance
##
##   variables are sorted by maximum importance across the classes
##                    X1     X2        X3
## registered 100.00000 76.025 100.00000
## atemp       81.26429 59.214  81.26429
## temp        80.90840 59.291  80.90840
## hr          63.33178 40.801  63.33178
## hum         50.48222 34.443  50.48222
## season      27.70915 18.873  27.70915
## weathersit  19.39292 14.675  19.39292
## workingday  15.92071 13.690  15.92071
```

```
## windspeed   15.80275  8.139  15.80275
## weekday      2.94006  2.940   1.61766
## holiday      0.05495  0.000   0.05495
```

The **Variable Importance** (varImp) function from the `{caret}` package provides scores of relative importance for each variable (by row), and its relation to each level of the response (`X1, X2, X3` correspond to levels `1, 2, 3` respectively). There is some fluctuation in the score that's given, but the scores indicate the same order of importance for the rows (excepting `atemp` and `temp` by an extremely slim amount in `X2`).

These indicate that the number of `registered` riders present at any given time does have the most influence on the number of casual riders. To look further into the relationship of these two variables,

```
# Refer to the raw numbers in the original dataset
plot(ISLR2::Bikeshare[,c("registered","casual")], xlab="Registered", ylab="Casual",
     main="Casual vs. Registered Riders")
```



The data do seem to indicate that when there are more registered riders, there are fewer casual riders; and more casual riders for fewer registered ones. It would be interesting to further compare other factors.

Moving further down this list, temperature (experienced and actual), the hour of the day, and the humidity do also play a stronger role in determining whether there are few, moderate, or many riders. The other factors - the specific season or weather situation, or whether or not it is a working day or a weekday, are not important factors in determining the number of casual riders.