ST537 Final Project

Mana Azizsoltani & Chris Comora

28 October, 2020

Contents

Introduction

In the hospitality industry, there is currently a new push towards the implementation of different sorts of analytics solutions, especially in the realm of revenue management and optimization. Hotel rooms are a limited commodity, and once a room on a certain date is booked in advance, the hotel must guarantee the room to that customer. The problem is that the hotel takes on risk when they book the room, because the customer could cancel with relatively short notice, which leaves the hotel either with a vacant room or with no other choice than to downsell the room last minute. The impact on the bottom-line can be significant; research has shown that cancellations impacted "almost 40% of on-the-books revenue" in 2018 for some hotels.

For this project we will be assessing the accuracy, sensitivity, and specificity of various machine learning techniques when faced with predicting hotel cancellations. Essentially, these machine learning models will be attempting to classify a given hotel booking as either cancelled or not cancelled. We will be running the following four binary classification models:

- Random Forest
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)
- Logistic Regression

The data sets used come from two different hotels located in Southern Portugal. The first hotel is a resort hotel located along the coast and the second one is located within a city. They have 31 variables that correspond to different booking information. The first hotel has about 40,000 records, while the second has just under 80,000. Each record corresponds to a booking from the period between July 1, 2015 and August 31, 2017. The data are in .csv format. No-shows are considered cancellations. Because of limitations on our computational power, we used the first 5,000 records of each data set.

Required Libraries

To run the code for the project, the following libraries are required:

- `caret`: to do the heavy lifting of training and tuning the models
- `tidyverse`: for all the data reading and wrangling
- `knitr`: for rmarkdown table outputs
- `rmarkdown`: for output documents
- `rpart`: fitting the basic classification tree
- `randomForest`: fitting random forest models
- `kernlab`: fitting the support vector machine
- `class`: fitting the k-nearest neighbor model

Data

Variable Descriptions

- IsCanceled: value indicating whether or not the booking was cancelled
- PreviousCancellations: number of previous bookings that were cancelled by the customer prior to the current booking
- ADR: average daily rate of the room
- AssignedRoomType: code for the type of room assigned to the booking
- CustomerType: type of booking
- DepositType: indication on if the customer made a deposit to guarantee the booking
  - Deposit can take on the values `No Deposit`, `Non Refund`, or `Refundable`
- LeadTime: number of days that elapsed between the entering date of the booking into the PMS and the arrival date
- MarketSegment: market segment designation
  - In categories, the term "TA" means "Travel Agents" and "TO" means "Tour Operators"
- RequiredCarParkingSpaces: number of car parking spaces required by the customer
- StaysInWeekNights: number of week nights (Monday to Friday) the guest stayed or booked to stay at the hotel

Reading in the Data

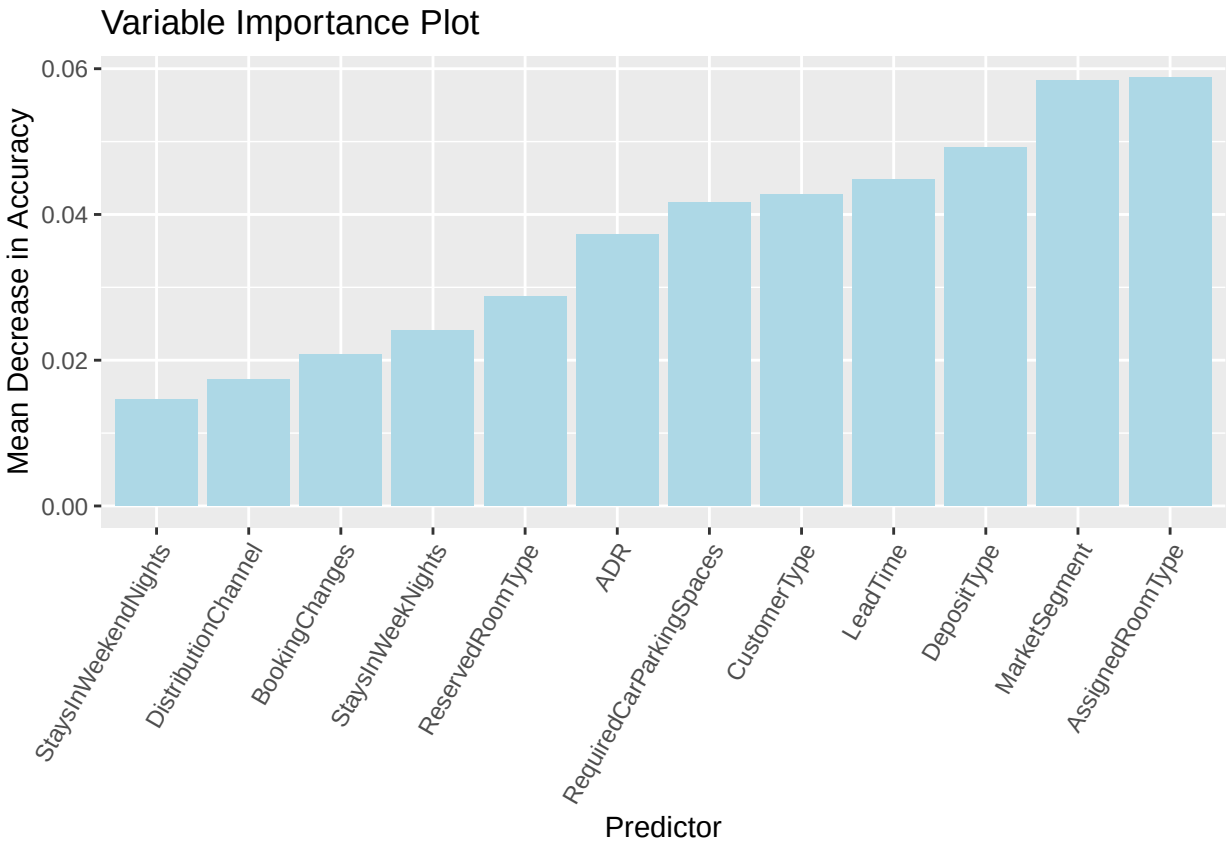
First things first, we read in the data and check to see that there are no missing values. Luckily, the two hotel data sets are clean and ready-to-go.

```
# Read in H1 dataset, take the first 5000 rows
h1 <- read_csv("H1.csv", col_names = TRUE)
h1$IsCanceled <- h1$IsCanceled %>% as.factor()
h1 <- h1[1:5000,]

# Read in H2 dataset, take the first 5000 rows
h2 <- read_csv("H1.csv", col_names = TRUE)
h2$IsCanceled <- h2$IsCanceled %>% as.factor()
h2 <- h2[1:5000,]

# Check for missing values
print(c("NAs H1" = sum(is.na(h1)), "NAs H2" = sum(is.na(h2))))
```

```
## NAs H1 NAs H2
##      0      0
```

Variable Selection

After running an initial random forest model on the entire dataset with selected predictors, we looked at the variable importance of the predictors to decide which variables we would use in our final model. Variable importance, in the context on machine learning, refers to how much a given model "uses" that variable to make accurate predictions. In other words, the more a model relies on a variable to make predictions, the more important it is for the model.



Based on this plot, we chose the 8 predictors with the highest variable importance.

Data Partitioning

Before creating the models, we must split the data into a training and test data set in order to later evaluate the model's prediction accuracy. For this project we will be using a 75/25 split, training the data on the 75% and testing the trained models on the withheld 25%.

```
# Get only necessary variables
preds <- names(h1)[c(1,2,8,26,20,28,22,14,27)]

# Create Data Partition
h1.index <- createDataPartition(h1$IsCanceled, p = .75, list = FALSE)
h2.index <- createDataPartition(h2$IsCanceled, p = .75, list = FALSE)

# Create train/test data for H1
h1.train <- h1[h1.index,preds]
h1.test <- h1[-h1.index,preds]
```

```r
# Create train/test data for H2
h2.train <- h2[h2.index,preds]
h2.test <- h2[-h2.index,preds]
```

Model Building

Before actually running the models, we went ahead and wrote the final formula of the models based on the variables we selected. Then, we specified our method of cross-validation, which will be a 5-fold cross-validation. For each model that required parameters to be tuned, we let the `caret` package tune the parameters and used the best tune of the parameters as specified in the model output.

```r
# Create a formula for the models using selected variables
form <- IsCanceled ~ ADR +
  AssignedRoomType + CustomerType +
  RequiredCarParkingSpaces + DepositType +
  MarketSegment + LeadTime + StaysInWeekNights

# Specify the cross-validation method (5-fold CV)
trctrl <- trainControl(method = "cv", number = 5)
```

Basic Classification Tree

The basic classification tree is based on partitioning the data into subgroups using simple binary splitting. Initially, all objects are considered a single group. Then, the group is binarily split into two subgroups based on the criteria of a certain variable. We then classify the observations in a specific region with majority vote. We want to grow the tree as big as we can, and then prune it back using cost-complexity pruning. This is done to not overfit the data, but pruning increases the bias. The pruning parameter needs to get tuned, which is done automatically in the `caret` package.

```r
#### HOTEL 1 ####
# Fit basic tree model for H1
h1.tree <- train(form, data = h1.train, method = "rpart",
                 trControl = trctrl)

# Obtain confusion matrix and results for H1
h1.tree.pred <- predict(h1.tree, newdata = h1.test)
h1.tree.CM <- confusionMatrix(h1.tree.pred, h1.test$IsCanceled)
h1.tree.res <- c(h1.tree.CM$overall[1], h1.tree.CM$byClass[1:2])

#### HOTEL 2 ####
# Fit basic tree model for H2
h2.tree <- train(form, data = h2.train, method = "rpart",
                 trControl = trctrl)

# Obtain confusion matrix and results for H2
h2.tree.pred <- predict(h2.tree, newdata = h2.test)
h2.tree.CM <- confusionMatrix(h2.tree.pred, h2.test$IsCanceled)
h2.tree.res <- c(h2.tree.CM$overall[1], h2.tree.CM$byClass[1:2])
```

Random Forest

The random forest model is an ensemble tree-based method, which creates multiple trees from bootstrap models and averages the results. A classification tree is fitted on each bootstrap sample with a random subset of the predictors. The final classification is based on majority vote of the bootstrap predictions.

```
#### HOTEL 1 ####
# Fit random forest model for H1
h1.rf <- train(form, data = h1.train, method = "rf",
               trControl = trctrl, importance = TRUE)

# Obtain confusion matrix and results for H1
h1.rf.pred <- predict(h1.rf, newdata = h1.test)
h1.rf.CM <- confusionMatrix(h1.rf.pred, h1.test$IsCanceled)
h1.rf.res <- c(h1.rf.CM$overall[1], h1.rf.CM$byClass[1:2])

#### HOTEL 2 ####
# Fit random forest model for H1
h2.rf <- train(form, data = h2.train, method = "rf",
               trControl = trctrl, importance = TRUE)

# Obtain confusion matrix and results for H1
h2.rf.pred <- predict(h2.rf, newdata = h2.test)
h2.rf.CM <- confusionMatrix(h2.rf.pred, h2.test$IsCanceled)
h2.rf.res <- c(h2.rf.CM$overall[1], h2.rf.CM$byClass[1:2])
```

Support Vector Machine

A support vector machine is a type of classification rule where it essentially maximizes the margin between groups by choosing the line with the widest "margin", but allowing for error. The support vectors are the points closest to the middle that carry the most weight when classifying, since they are the most prone to error and are deciding factors of where the classification line could go. We use a radial kernel function to deal with the non-linearity and higher-dimensions.

```
#### HOTEL 1 ####
# Fit support vector machine for H1
h1.svm <- train(form, data = h1.train, method = "svmRadial",
                preProcess = c("center", "scale"),
                trControl = trctrl, tuneLength = 10)

# Obtain confusion matrix and results for H1
h1.svm.pred <- predict(h1.svm, newdata = h1.test)
h1.svm.CM <- confusionMatrix(h1.svm.pred, h1.test$IsCanceled)
h1.svm.res <- c(h1.svm.CM$overall[1], h1.svm.CM$byClass[1:2])

#### HOTEL 2 ####
# Fit support vector machine for H2
h2.svm <- train(form, data = h2.train, method = "svmRadial",
                preProcess = c("center", "scale"),
                trControl = trctrl, tuneLength = 10)

# Obtain confusion matrix and results for H2
h2.svm.pred <- predict(h2.svm, newdata = h2.test)
```

```
h2.svm.CM <- confusionMatrix(h2.svm.pred, h2.test$IsCanceled)
h2.svm.res <- c(h2.svm.CM$overall[1], h2.svm.CM$byClass[1:2])
```

K-Nearest Neighbor (KNN)

K-NN is a "model free" approach, meaning that it doesn't assume any probability model on the data. Given an observation, $\boldsymbol{x}$, we want to find the k training observations that are "closest" to $\boldsymbol{x}$, and then classify the new observation using majority vote among these k neighbors. We define "closeness" based on Eucledian distance in our model.

```
#### HOTEL 1 ####
# Fit knn model for H1
h1.knn <- train(form, data = h1.train, method = "knn",
                trControl=trctrl, preProcess = c("center", "scale"))

# Obtain confusion matrix and results for H1
h1.knn.pred <- predict(h1.knn, newdata = h1.test)
h1.knn.CM <- confusionMatrix(h1.knn.pred, h1.test$IsCanceled)
h1.knn.res <- c(h1.knn.CM$overall[1], h1.knn.CM$byClass[1:2])

#### HOTEL 2 ####
# Fit knn model for H2
h2.knn <- train(form, data = h2.train, method = "knn",
                trControl=trctrl, preProcess = c("center", "scale"))

# Obtain confusion matrix and results for H2
h2.knn.pred <- predict(h2.knn, newdata = h2.test)
h2.knn.CM <- confusionMatrix(h2.knn.pred, h2.test$IsCanceled)
h2.knn.res <- c(h2.knn.CM$overall[1], h2.knn.CM$byClass[1:2])
```

Logistic Regression

The logistic regression model uses the "logit" function $log(\frac{p}{1-p})$, which links the mean to the linear form of the regression model, $\boldsymbol{X\beta}$. Using it for binary classification, we round the fitted values either up or down to 1 or 0.

```
#### HOTEL 1 ####
# Fit logistic regression model for H1
h1.glm <- glm(form, data = h1.train,  family = binomial(link = "logit"))

# Obtain confusion matrix and results for H1
h1.glm.pred <- predict(h1.glm, newdata = h1.test, type = "response")
h1.glm.pred <- h1.glm.pred %>% round() %>% as.factor()
h1.glm.CM <- confusionMatrix(h1.glm.pred, h1.test$IsCanceled)
h1.glm.res <- c(h1.glm.CM$overall[1], h1.glm.CM$byClass[1:2])

#### HOTEL 2 ####
# Fit logistic regression model for H2
h2.glm <- glm(form, data = h2.train,  family = binomial(link = "logit"))

# Obtain confusion matrix and results for H2
```

```
h2.glm.pred <- predict(h2.glm, newdata = h2.test, type = "response")
h2.glm.pred <- h2.glm.pred %>% round() %>% as.factor()
h2.glm.CM <- confusionMatrix(h2.glm.pred, h2.test$IsCanceled)
h2.glm.res <- c(h2.glm.CM$overall[1], h2.glm.CM$byClass[1:2])
```

Model Comparison

Below are two tables of the accuracy, sensitivity, and specificity of each model fit.

Table 1: H1 Model Results

	Accuracy	Sensitivity	Specificity
Basic Tree	0.7270	0.7807	0.6638
Random Forest	0.8239	0.8370	0.8084
SVM	0.7670	0.7304	0.8101
KNN	0.7798	0.8030	0.7526
Logit. Reg.	0.7262	0.7319	0.7195

Table 2: H2 Model Results

	Accuracy	Sensitivity	Specificity
Basic Tree	0.6765	0.7022	0.6463
Random Forest	0.7926	0.8030	0.7805
SVM	0.7502	0.7289	0.7753
KNN	0.7422	0.7659	0.7143
Logit. Reg.	0.7014	0.7422	0.6533

Conclusion

Looking at the output of the machine learning models on the two data sets, we can see that the Random Forest model had the highest accuracy (about 80%), with the KNN and SVM models falling close behind it in the mid-70s. The worst performance in terms of prediction accuracy came from the logistic regression model, with an accuracy right around 70%. When considering the non-information rate of 54%, all of the models were effective in at least increasing the prediction accuracy by about 20%.

The fact that the random forest so drastically outperformed the basic classification tree did not come as a surprise, since as an average of many classification trees, ensemble methods generally improve prediction accuracy and reduce overall variance. We suspect that other ensemble methods, such as bagging or boosting, may also be able to get higher prediction accuracies. One of the main disadvantages to using ensemble techniques like the random forest is the fact that they tend to be very computationally intensive. The basic classification tree, logistic regression and KNN models took significantly less time to run than the random forest and the SVM. If we wanted to run an ensemble method on a very large data set, it would be a nightmare with the minuscule computational power of our personal computers.