**A close up of a logo

Description automatically generated**

**IST707**

**Final Project**

**Customer Churn Prediction**

Professor

Jesse L Cases

Submitted by

Manideep Kannaiah

426693015

**Introduction / Problem Statement**

Resorts/Hotels cannot expect all customers who book a room to visit. Few of them either cancel the booking or don't show up. Such a scenario not only blocks the revenue to hotels but hinders them from offering a booking to a customer who can visit. If we study the patterns of customer behavior in the past-bookings and predict the number of customers who would cancel the booking on a particular day, then we can oversell the rooms and make up for cancellations/No-show.

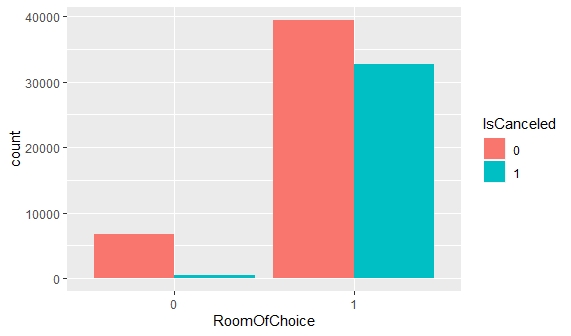
**Dataset**

The dataset is extracted from an [article](https://www.sciencedirect.com/science/article/pii/S2352340918315191) written by Nuno Antonio, Ana Almeida, and Luis Nunes for Data in Brief, Volume 22, February 2019. The data set has 119,390 observations of two types of hotels – Resort hotels (H1) and City Hotel (H2). Each observation has 31 attributes. All data pertaining to customer or hotel identification are removed from this data set due to privacy concerns.

Building a machine learning model on 119,390 observations would create huge computational overhead especially while tuning the models. To avoid this, we considered only City Hotels(H2) data set that has 79,326 observations for analysis. All data pertaining to Resort hotels are excluded from the analysis.

**Data cleaning**

1. The column IsCanceled is our target Variable. It is of type numeric and therefore converted to factor. The same operation is done on the column IsRepeated - indicates if the booking name was from a repeated guest (1) or not (0) .
2. The columns assignedRoomType and reservedRoomType contain 7 factor levels each. Most of times the customers got the room of their choice. Also, the adjacent plot suggests that assigning the customer with a different room type doesn’t have any effect on cancellation. (Cancellation rate in this scenario is almost negligible.) These two columns are decomposed into a single of RoomOfChoice which has value 1 if the above two columns contain same value and zero otherwise.



1. The column ReservationStatus provides similar information that the column IsCanceled provide and so removed from the analysis.
2. The other time series columns like ArrivalDateYear, ArrivalDateMonth, ArrivalDateDayOfMonth, ArrivalDateWeekNumber, ReservationStatusDate are removed because these attributes are captured by the column Leadtime.
3. The Country (178 levels), Agent (224 levels), Company (208 levels) columns are removed because of high impurity and including these columns might overfit the model. (Model were also built using these attributes. It took a long time even for the simpler model to run and accuracy was in the pretty similar range.)
4. Using dataframes for this data set can potentially increase the run time for model building owing to the number of observations and attributes the dataset has.
5. To avoid this, data tables(data.table) are used. A huge jump is observed in the time taken for the models to run. For example, a 3-fold cross-validation of the c5.0 algorithm with a param grid of length 5 had taken 30 min to execute. When data table is used a param grid of length 14 took 3 to 4 min to complete the model building.

**Model Building**

Our target variable ‘IsCanceled’ is a binary column. Therefore, we use algorithms that support binary classification. E.g. Decision Trees, Naive Bayes, Support vector machines (SVM), K-Nearest Neighbors (KNN). In R, different packages are available that implement these algorithms. A huge drawback with most of those packages is manual tuning. We must run multiple models and compare their performance manually to arrive at an optimal set of hyperparameters. Caret package is the solution to this tedious process.

Caret (short for **C**lassification **A**nd **RE**gression **T**raining) library is an exceptional environment for automatic parameter tuning and training the classifiers. It also provides great functions to sample the data (for training and testing), preprocessing, evaluating the model, etc. Given a grid of hyperparameters, it builds a model for each set, compare their performance, and selects the best model.

A K-fold cross-validation technique is employed to evaluate the performance of the models. K-fold cross-validation performs model selection by splitting the dataset into a set of non-overlapping randomly partitioned folds which are used as separate training and test datasets e.g., with k=3 folds, K-fold cross-validation will generate 3 (training, test) dataset pairs, each of which uses 2/3 of the data for training and 1/3 for testing. Each fold is used as the test set exactly once. The other evaluation techniques are Holdout method and bootstrap sampling

**Holdout method**: The data is split into training and testing sets. The model is built on a training dataset while evaluated on the testing dataset.

**Bootstrap Sampling**: the bootstrap allows examples to be selected multiple times through a process of sampling with replacement. This means that from the original dataset of n examples, the bootstrap procedure will create one or more new training datasets that will also contain n examples, some of which are repeated. The corresponding test datasets are then constructed from the set of examples that were not selected for the respective training datasets.

Sometimes the model associated with the best performance may tend to overfit. To avoid overfitting, the best model is selected using oneSE function which selects the simpler model within one standard error of the empirically optimal model. The other options available are ‘best’ and ‘tolerance’ functions.

**best** function simply chooses the hyperparameters associated with the best model

**tolerance** function selects the simpler model that is within a percent tolerance of the empirically optimal model

**Naive Bayes:**

* **Overview of model**

The Naive Bayes classifier is a simple probabilistic classifier which is based on Bayes theorem but with strong assumptions regarding independence. With Naive Bayes, we assume that the predictor variables are conditionally independent of one another given the response value.

* **Data Preparation specific to the model**

Coming to data preparation for the Naïve Bayes model, it works only with categorical values as it uses the frequency tables to learn the data. Since numeric features do not have categories of values, the preceding algorithm does not work directly with numeric data. However, this can be addressed by discretizing the data using the following techniques

1. Create bins of equal interval or Sizes
2. Use an appropriate probability density estimation

Probability density estimation cannot be used on this dataset due to multiple reasons. Therefore, bins of equal intervals are created for continuous variables.

* **Hyperparameters available for tuning**

We can tune following hyperparameters for a Naive Bayes model.

**usekernel** parameter allows us to use a kernel density estimate for continuous variables versus a gaussian density estimate,

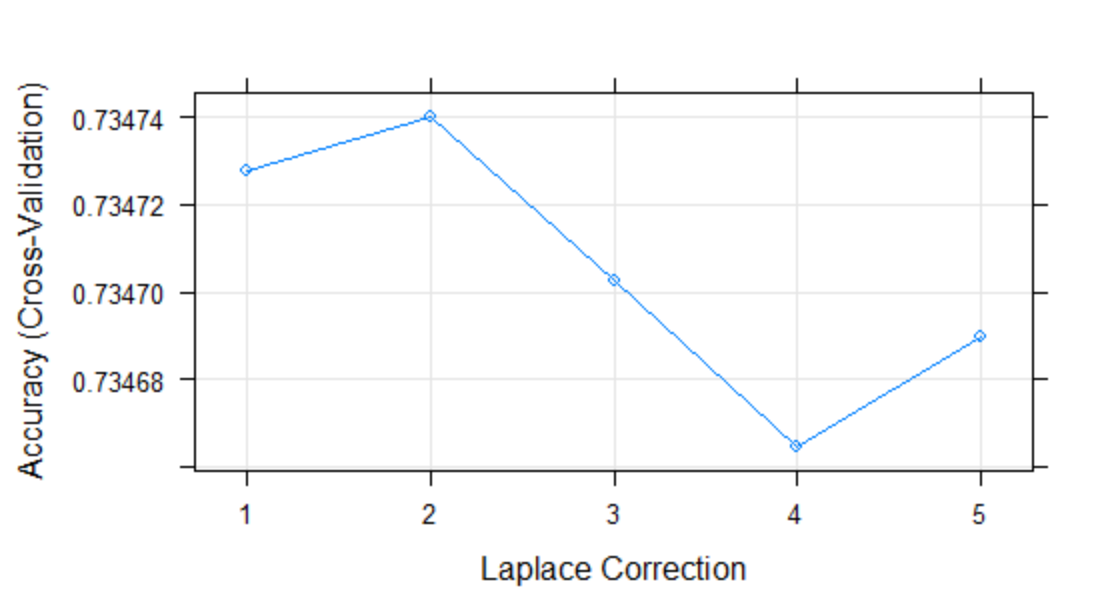
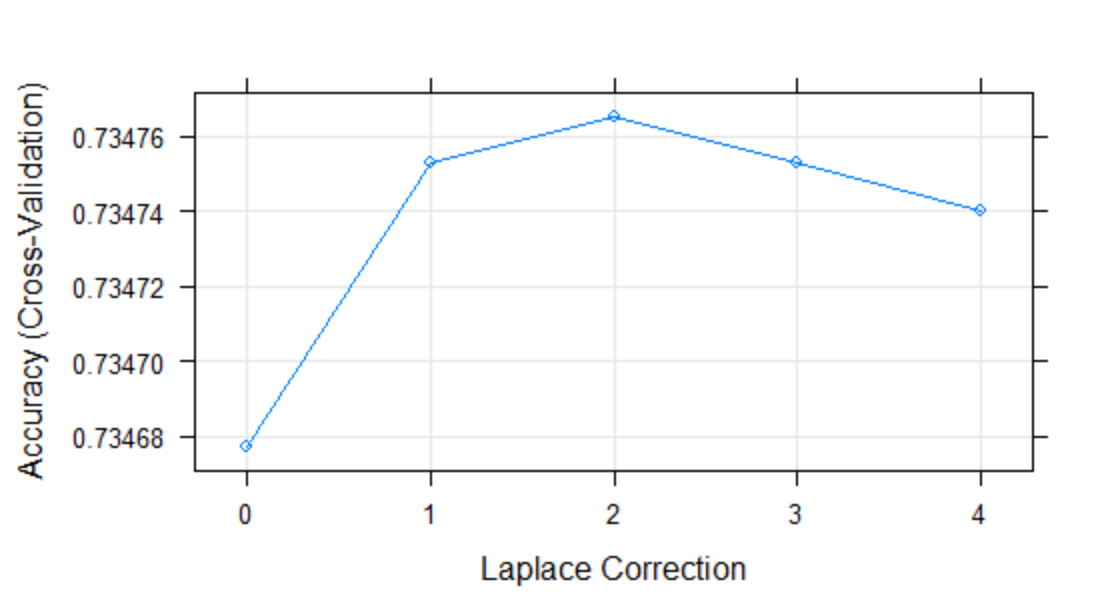
**adjust** allows us to adjust the bandwidth of the kernel density (larger numbers mean more flexible density estimate),

**fL** allows us to incorporate the Laplace smoother.

Usekernel and adjust parameters are used for tuning if our data has continuous attributes. Since continuous variables are transformed into categorical using the binning technique, we are left with just one parameter i.e., fL for tuning.

* **Performance of the Model**

A 3-fold cross-validation of the base model (fL=0) has an accuracy of 73.47%. After tuning the only available hyperparameter fL, the final optimal model has fL=0 which means that the base model is the best performing model. From the figure, we can observe that accuracy for fL = 2, is the highest. But the final model turns out to have an fL value of zero because the selection function ‘OneSE’ selects the simpler model within one standard error of the empirically best model.



Final parameters: fL=0

Metrics

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | Run time |
| 73.47% | 79.46% | 49.04% | 400 sec(approx.) |

**Decision Trees**

* **Overview of model**

Decision tree is a [flowchart](https://en.wikipedia.org/wiki/Flowchart)-like structure in which each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

* **Data preparation specific to the model**

Decision can work with both categorical and numerical attributes. It is robust to outliers and therefore we can proceed with building models without scaling the numerical attributes.

* **Algorithm**

Caret library supports different algorithms that implement decision trees. But we would be looking into Rpart and C5.0 algorithms.

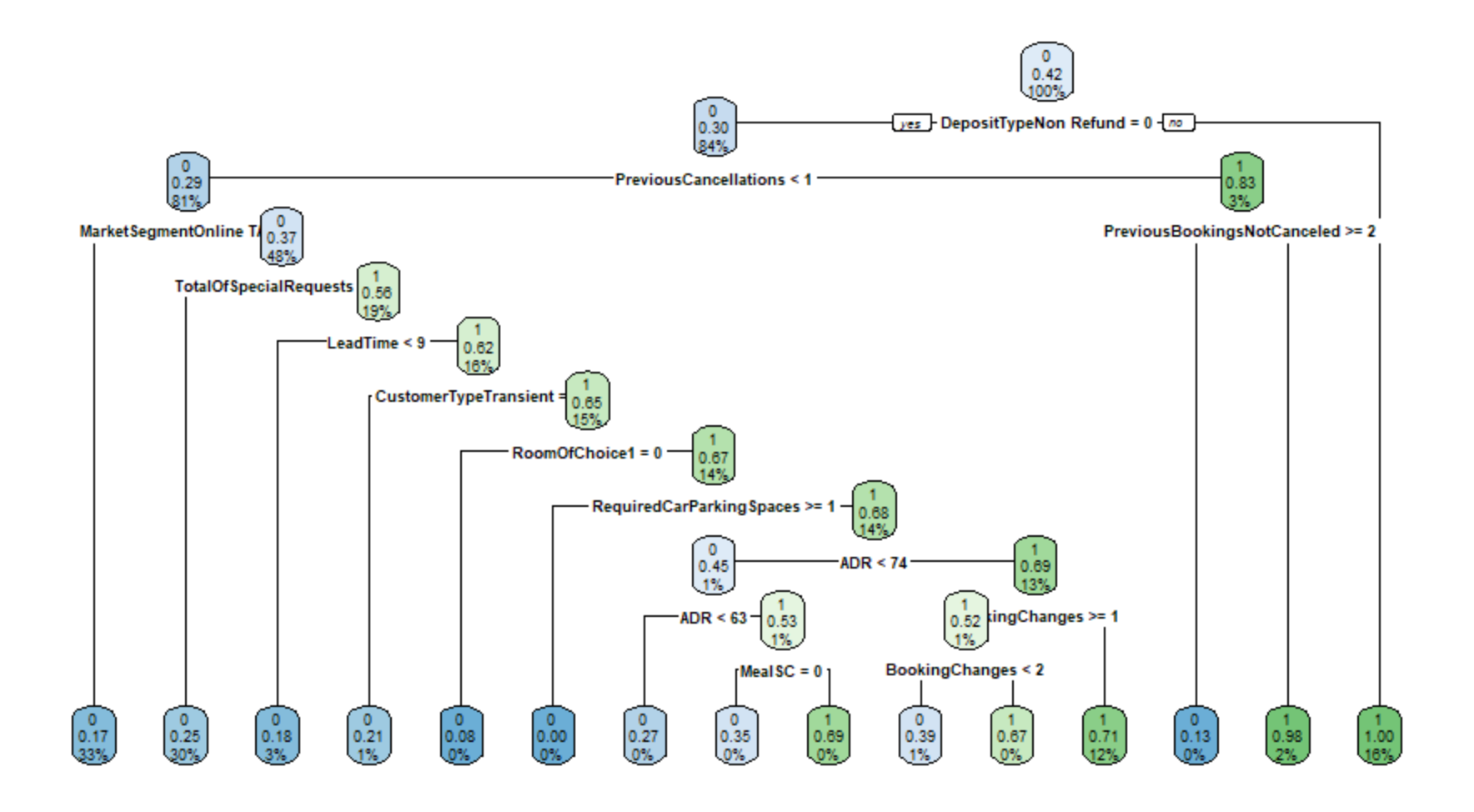
**RPart (Recursive Partitioning And Regression Trees)**

* **Hyperparameter available for tuning**

**CP (Complexity parameter):** It is the minimum improvement in the model needed at each node. It’s based on the cost complexity of the model. Any split that does not decrease the overall lack of fit by a factor of CP is not attempted.

* **Performance of the model**

3-fold cross-validation of the base model that uses all default parameters has an accuracy of 80.79%. A grid of cp values (0.001,0.01,0.1,1) are used for parameter tuning. The optimal model has an accuracy of 81.94% and a cp value of 0.001.

By reducing cp value, the accuracy keeps improving but at a very low rate indicating that the model classification ability reached saturation. At this point, we might run into overfitting issues. Also, the size of the tree also increases making it difficult for human interpretation. So, the tuning is stopped at this level (cp=0.01).

Metrics:

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | Run time |
| 82.15% | 86.54% | 67.70% | 8 sec(approx.) |

**C5.0**

* **Hyperparameters available to tune**

**trials** an integer specifying the number of boosting iterations. A value of one indicates that a single model is used.

**model** specifies whether to generate rules or trees

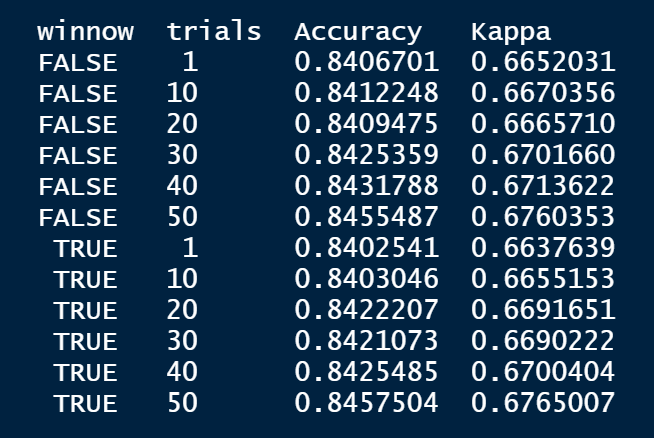
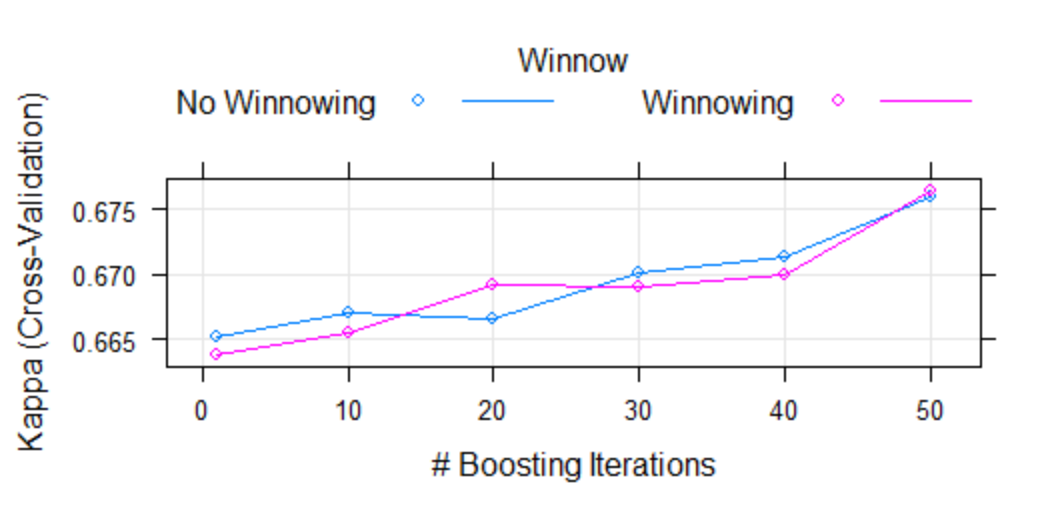
**winnow -** A logical: should predictor winnowing (i.e. feature selection) be used?

**How boosting works?**

This classifier will usually make mistakes on some cases in the data; the first decision tree, for instance, gives the wrong class for 8 cases When the second classifier is constructed, more attention is paid to these cases in an attempt to get them right. As a consequence, the second classifier will generally be different from the first. It also will make errors on some cases, and these become more important during the construction of the third classifier. This process continues for a pre-determined number of iterations or ***trials*** but stops if the most recent classifier is either extremely accurate or too inaccurate.

* **Performance of model**

The model with default parameters (trials = 1, Winnow = False, model =tree) has an accuracy of 84.15. By tuning the hyperparameters, the accuracy increased marginally to 84.58. The final values used for the model were trials = 50, model = tree and winnow = TRUE. The optimal model differs from the base model, but their accuracies are very similar.



Even though the 50-trial model offers the optimal performance, the 1-trial model offers nearly the same performance with a much simpler form. Not only are simple models more computationally efficient, but they also reduce the chance of overfitting the training data. So, we resort to the base model in this case.

Metrics:

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | Run time |
| 84.15% | 86.63% | 74.58% | 30 sec(approx.) |

**KNN (K – Nearest Neighbors):**

* **Overview of model**

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity by calculating the distance between points on a graph.

* **Data Preparation Specific to the model**

To capture the similarity between the data points, KNN calculates the distances and therefore all features must be numerical. Also, they must be in a uniform scale to make the model robust to outliers.

So, all categorical variables are converted to numerical using one-hot encoding. One hot encoding is the technique of generating dummy variables where a value of 1 indicates one category, and 0, the other. For instance, dummy coding for a depositType variable which has three levels could be constructed as:

Deposit\_No\_Deposit = 1 if (deposit type = No deposit), 0 otherwise

Deposit\_Non\_Refund = 1 if (deposit type = Non-Refund), 0 otherwise

Deposit\_Refundable = 1 if (deposit type = Refundable), 0 otherwise

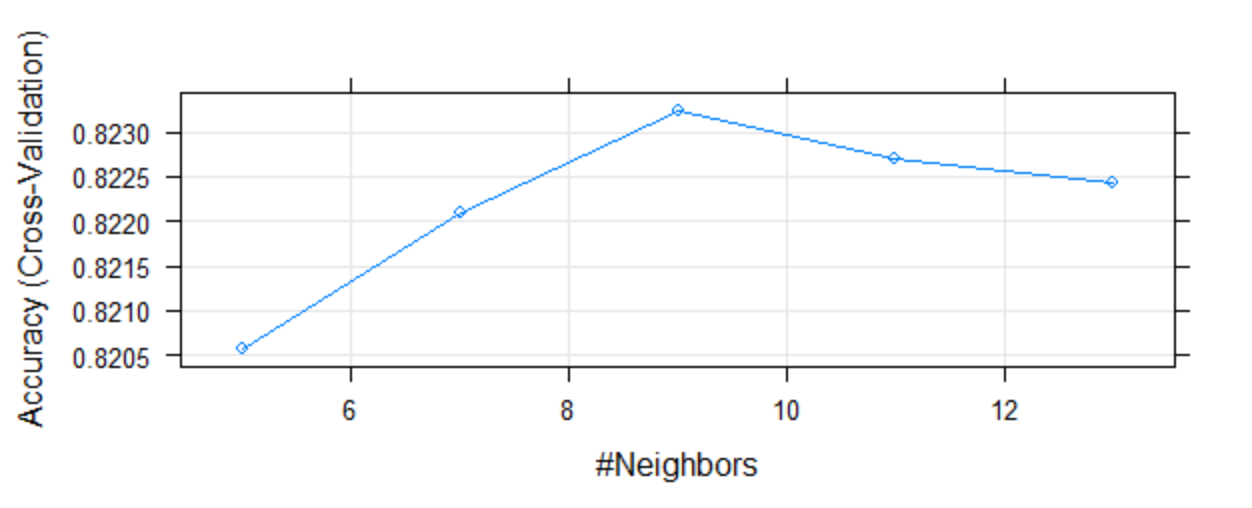
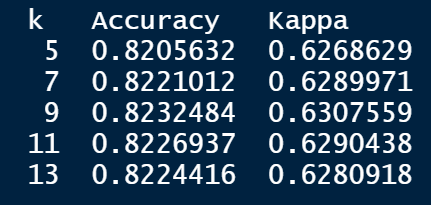
In this way, the categorical columns are decomposed into multiple columns depending upon the number of levels they have. One hot encoding is done using one\_hot function from mltools library. Caret’s train function allows us to perform scaling before building the model which can be specified using the preprocess attribute of the function.

* **HyperParameters available to tune:**

**K** – specifies number neighbors to consider in the process of classifying an observation/record.

* **Performance of model**

5 different model (K= 5,7,9,11,13) built and evaluated using 3- fold cross-validation technique. The final value used for the model was k = 13. We can observe that accuracy keeps improving until k=9 and follows a reverse direction for k>9. This model took a very long time to run. It is not ideal to induce a big parameter grid on this dataset for this model.



Metrics:

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | Run time |
| 82.24% | 82.96% | 72.24% | 1800 sec(approx.) |

**SVM (Support Vector Machines):**

* **Overview of model**

A Support Vector Machine (SVM) can be imagined as a surface that creates a boundary between points of data plotted in multidimensional that represent examples and their feature values. The goal of SVM is to create a flat boundary called a hyperplane, which divides the space to create fairly homogeneous partitions on either side.

* **Data Preparation Specific to the model**

SVM also needs data in numerical format. The preprocessing steps are similar to those performed for KNN.

* **Performance of model**

Unlike other models, SVM is built using kernlab library as it would take a long time to run this model caret library and evaluated using holdout method.

A couple model are built using rbfdot (Radial kernel) and vanilladot (Linear kernal).

rbfdot kernel has an accuracy of 81.34 while linear kernel- vanilladot has an accuracy of 80.18%.

we choose rbfdot kernel for the final model as it has better accuracy

Metrics:

|  |  |  |  |
| --- | --- | --- | --- |
| Accuracy | Precision | Recall | Run time |
| 81.34% | 84.75% | 68.56% | 2400 sec(approx.) |

**Comparison of Models**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Accuracy | Precision | Recall | Runtime in Sec(approx.) |
| Naive Bayes | 73.47% | 79.46% | 49.04% | 400 |
| Rpart | 82.15% | 86.54% | 67.70% | 8 |
| C5.0 | 84.15% | 86.63% | 74.58% | 30 |
| KNN | 82.24% | 82.96% | 72.24% | 1800 |
| SVM | **81.34%** | 84.75% | 68.56% | 2400 |

We cannot compare the of SVM with other models, as we evaluated other using 3-fold cross-validation technique and SVM is evaluated using holdout method.

Among the other models, C5.0 implementation of decision trees has highest accuracy taking relatively very time to build the tree.

**Interpretation**

The metric that makes the most sense in this scenario is Recall, which is the measure of our model ability to identify the booking that may get canceled. In more detail, if a booking is predicted to get canceled, how often it is correct? Based on this measure, we can go ahead and make overbookings to make up for the loss incurred by the cancellations.

Let’s consider a scenario of 200 bookings where the model predicted 100 bookings to get cancel and the other 100 customers show-up. Consider the recall and precision of the C5.0 algorithm. Recall of 74.58% means we are sure that 75 out of 100 bookings definitely get canceled and therefore we can make 75 overbookings. we must compensate the customer if we are unable to provide him a room because of the overbookings made. So, making 75 overbooking, in this case, is risky and therefore only a fraction of suggested overbookings are made.

**Conclusion:**

Decision trees performed the best on this dataset. Using Random forest and bagging models would improve the model further. The overbooking recommendations are made using the recall of the models.

**References:**

1. <http://topepo.github.io/caret/index.html>
2. <http://uc-r.github.io/predictive>
3. Machine Learning with R by Brett Lantz