Is Yelp Helpful to a Restaurant's Sustainability?

Team XKL

Contents

1 Abstract

2 Introduction

2.1 Project Goal and Background

2.2 Literature Review

2.3 Data Description

2.4 Relevant Variables

2.5 Purpose and Result

3 Methods

3.1 Pre-processing

3.2 Classification Tree

We used non-parametric method decision trees for modeling. It was convenient and could deal with categorical variables directly. A decision tree was a flowchart-like structure in which each internal node represents a "test" on an attribute. The paths from the root to leaf represented classification rules. In decision analysis, a decision tree and the closely related influence diagram were used as a visual and analytic decision support tool, where the expected values (or expected utility) of competing alternatives were calculated.

In a classification tree, each node denotes a test, and each branch represents an outcome of that test. In general, we apply a CT to data when our response variable is qualitative or quantitative discrete. CT classify observations based on all available all explanatory variables and is supervised by the response variable. CT is able to classify observations using simple rules regardless of non-linearity and interactivity of explanatory variables.

However, one drawback of the classification tree is that it is largely dependent on the data and thus making the tree over-fit the data. In other words, a tree will have low bias but very high variance. One small change in observed data might completely change the tree. Therefore, it is hard to generalize a single tree for other circumstances. To avoid the problem, we would cross-validated the data so as to prune it.

In this project, we use the tree to determine if the label can be correctly identified. We first use the corresponding training dataset to tune the trained the tree and found out that the best cp. Then we continue to train the tree model using the same parameter and recorded their points and error rates. Furthermore, to test if the single tree tends to over-fit the data, we cross-validated over the same set of training sets and calculated the mean error rate derived from each fit. Based on the table shown below, we can see that the mean error does not vary much from the error rate of the original tree model. Therefore, we can say that the overfitting of a single tree is not the factor influencing the error rate for this model.

3.3 Random Forest

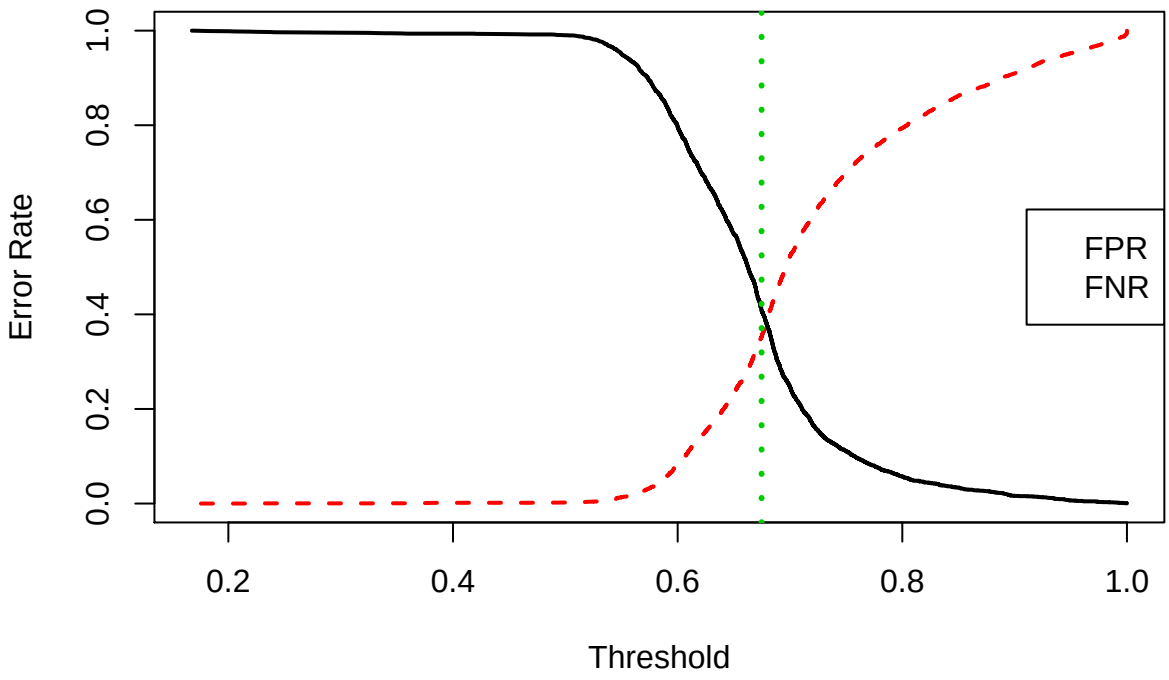
In short, the random forest is conducted by constructing a multitude of decision trees using different parts of the same training set and averaging trees to output the mode of the classes (classification) or mean prediction (regression) of the individual trees (Yeh & Lien). As compare to decision trees, random forest avoid the problem of overfitting the training set, but in exchange, there will be some increase in its bias.

Furthermore, it might not be as interpretable as the decision tree is. But overall its model should have a better performance in predicting and classifying than decision or classification trees. Overall, we can see that the random forest did perform better regarding its misclassification rate.

Random forest is the ensemble of decision trees which is fast to train but inefficient to predict. And it is a corollary that the more accuracy we would like to achieve, more trees are needed, which means the model is more time-consuming. In most practical situations this approach is fast enough, but there can certainly be situations where run-time performance is important and therefore other approaches would be preferred.

3.4 Logistic Regression

Logistic regression is usually considered as a special case of linear regression models. Nevertheless, the binary response variable does not hold a normal distribution as required by other linear regression models. A logistic regression model assumes that the probability of the event, which in our case is the probability of default, is a linear combination of explanatory variables. The advantage of this approach is that the probability of classification can be easily obtained. The weaknesses is that interactions between explanatory variables will greatly influence the performance of the model. Also, these variables need to be linear to use this approach. The plot below shows the threshold which we used to make decision and the method implemented to derive it is through calculating the euclidean distance.



3.5 Support Vector Machine

SVM primarily intends for binary classification and it is based on the idea of a separating hyperplane where hyperplane is a "flat affine subspace of dimension $p-1$". Only observations that lie on the margin or violate the margin affect the learned hyperplane boundary and these observations are called the "support vectors". Observations that lie on the correct side of the margin have no effect. In SVM, cost function C is a tuning parameter which controls bias-variance. When C is large, the margin is wide, and many observations will affect classification. Choice of kernel and kernel parameters γ also affect bias-variance.

For this project, we love the accuracy that SVM provides us, but the algorithm itself largely depends on its regularization parameter, say kernel. Therefore, the parameters works for one dataset might perform poorly in another. Thus, we tuned the parameter for each dataset we sampled and making it extremely time-consuming. Even its error rate is pleasing, its time-consuming property need to be taken into consideration when choosing the final model.

3.6 KNN

KNN classify observations using the majority rules. "When given an unknown sample, a KNN classifier searches the pattern space for the KNN that are closest to the unknown sample" (Yeh & Lien). Closeness is defined in terms of distance and an unknown sample is assigned to its closest neighbor by distance.

Overall speaking, the KNN is relatively precise in predicting the labels. We love KNN because we do not need to build a predictive model before classification. Instead, it directly calculates the distance between the training set and K nearest neighbors. Then the label will be predicted as the most frequentist label appeared among the neighbors. Meanwhile, this is also a drawback, for that as the training sample size increases, its processing time also increases dramatically. In addition, the curse of dimensionality in classification is not negligible. We often find that the closest neighbor is usually "far away" because the density of the training samples decreased exponentially when we increased the dimensionality.

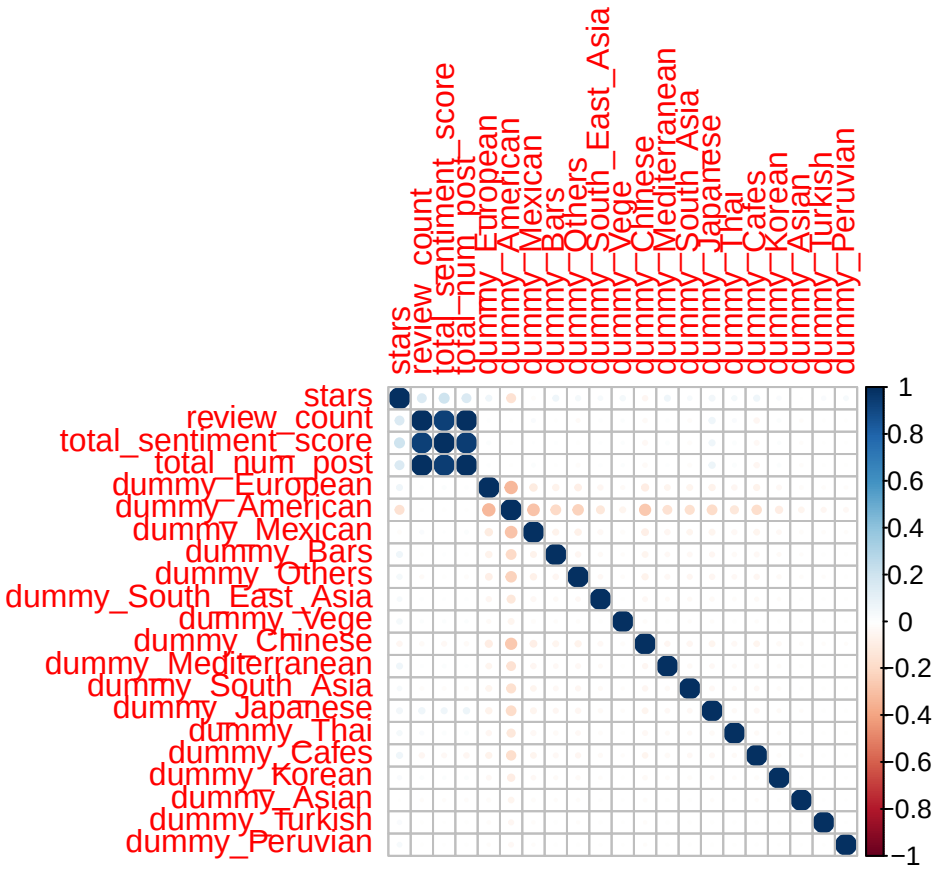
For this project, to avoid the curse and the overfitting problem coming along with it, we first cross-validated over the current training model and find the best number of neighbors to apply to the formal training model. We also added in the scale parameter so as to expedite the processing time. Also, for efficiency purpose, we trained the model with 200 trees at last.

3.7 Naive Bayes

Naive Bayes is a probabilistic classifier that makes classifications using posterior decision rule in a Bayesian setting. Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred and teh theory is mathematically stated below:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In addition, for precessing efficiency, we scaled and center the data.


stars
review_count
total_sentiment_score
total_num_post
dummy_European
dummy_American
dummy_Mexican
dummy_Bars
dummy_Others
dummy_South_East_Asia
dummy_Vege
dummy_Chinese
dummy_Mediterranean
dummy_South_Asia
dummy_Japanese
dummy_Thai
dummy_Cafes
dummy_Korean
dummy_Asian
dummy_Turkish
dummy_Peruvian
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1

4 Results

4.1 Error Table

Table 1: Errors

	Train Error	Test Error
Classification Tree	0.296	0.289
Random Forest	0.016	0.294
Logistic Regression	0.369	0.378
Support Vector Machine	0.294	0.281
KNN	0.260	0.296
Naive Bayes	0.297	0.280

4.2 ROC Curve

In addition to compare performance of different models with test and training errors, we will use ROC curves and area under the curve (AUC) to compare performances of different classifiers in `R`. Based on errors and curves, the "best" model we selected from six methods analyzed in this project is Naive Bayes and we will validate our conclusion using results obtained before.

Table 2: Area Under the Curve

	Classification Tree	Random Forest	Logistic Regression	Support Vector Machine	KNN	Naive Bayes
auc	0.5	0.65	0.67	0.59	0.61	0.59

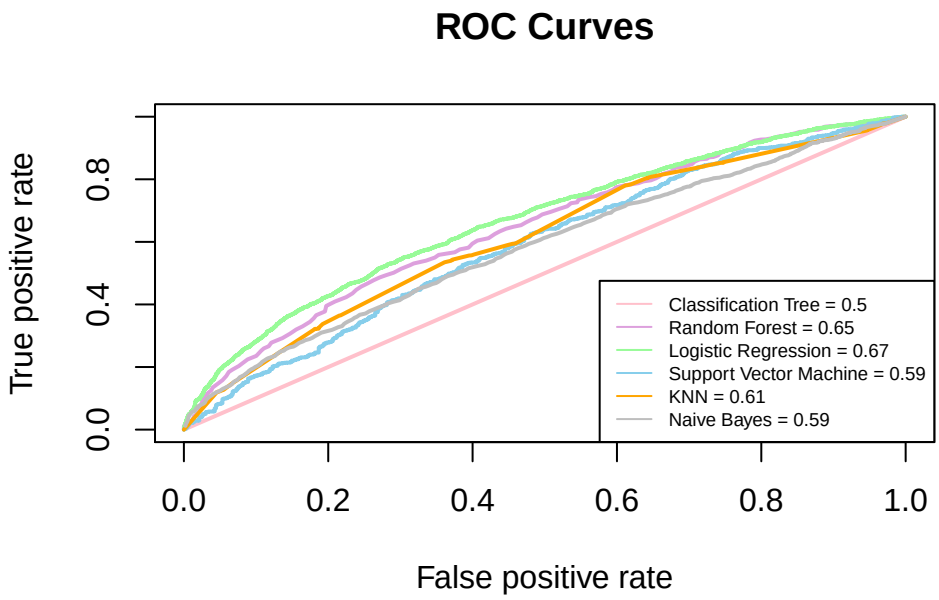


Figure 1: ROC Curves

5 Discussion

We will first focus on comparing methods of random forest and classification trees. As comparing the train errors of classification tree and random forest, We find out that the training error is way much more smaller than that of the tree model, and apparentlyl there is a overfitting problem with the random forest classifier. The test error rate of random forest is higher than that of classification tree. Literally, random forest should be a better method than classification tree because it constructs many classification trees and the randomness within random forest makes it a more robust classifier than a single tree. However, it is not the case for our project.

We then continue to compare other models. As shown in Figure 1, the ROC curves of logistic Regression Model has the margest area under the curve and yet its predicting accuracy is the worst. As for Support Vector Machine, KNN and Naive Bayes, KNN has comparatively higher test error rate and a lower training error. In Table 1 we see that all models have similar values in both training and test errors except that random forest has an extremely small train error as compare to those of other models. Naive Bayes and Support Vector Machine was quite similar in both error rates. If taking time consumption into consideration, then naive bayes out performed the svm. Therefore, we would say that naive bayes is the best method out of five classifiers we implemented in this project.

5.1 Future Work

As discussed above, we still do not understand why random forest does not perform well in this project. We will further work to try to figure out the problem.

6 Miscellaneous

6.1 Reference

Software: R studio

Yeh, I. C., & Lien, C. H. (2009). *The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients.* Expert Systems with Applications, 36(2), 2473-2480

UCI Machine Learning Repository https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

Background information: Seven Pillars Institue http://sevenpillarsinstitute.org/case-studies/taiwans-credit-card-crisis

The reference of Random Forest can be found at
http://statweb.stanford.edu/~jtaylo/courses/stats202/ensemble.html

Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani An Introduction to Statistical Learning, Springer, 2014 http://www-bcf.usc.edu/~gareth/ISL/ISLR%20Seventh%20Printing.pdf

Garrett Grolemund and Hadley Wickham R for Data Science http://r4ds.had.co.nz/

Principal component analysis: https://en.wikipedia.org/wiki/Principal_component_analysis

Cross Validation: https://www.cs.cmu.edu/~schneide/tut5/node42.html

7 Appendix

```
knitr::opts_chunk$set(echo = TRUE)

library(flexdashboard)
library(shiny)
library(rmarkdown)
library(knitr)
library(Hmisc)
library(DT)
library(data.table)
library(RColorBrewer)
library(scales)
library(lattice)
library(tidyverse)
# library(dplyr)
library(magrittr)
library(plotly)
library(leaflet,quietly=TRUE)
library(maps,quietly=TRUE)

library(ggplot2)
library(ROCR)
library(tree)
library(randomForest)
library(maptree)
library(class)
library(gbm)
library(e1071)
library(imager)
library(reshape2)
library(rpart)
library(caret)
library(purrr)
library(adabag)

knitr::opts_chunk$set(echo = FALSE, cache = TRUE, fig.pos = "H")
indent1 = '    '
fill_in_NA <- function(cuisine_ls){
  for(i in cuisine_ls){
    DT[, "idx" := str_detect(DT$categories, i)]
    DT[get("idx") == 1, eval(cuisine.name) := cuisine_ls[1]]
    # DT[get("idx") == 1 & is.na(get(cuisine.name))==TRUE, eval(cuisine.name) := cuisine_ls[1]]
    DT[,"idx":=NULL]
  }
}

erate <- function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
}

#Set formula
create.formula <- function(outcome.name, input.names, input.patterns = NA,
```

```
                            all.data.names = NA, return.as = "character") {
  variable.names.from.patterns <- c()
  if (!is.na(input.patterns[1]) & !is.na(all.data.names[1])) {
    pattern <- paste(input.patterns, collapse = "|")
    variable.names.from.patterns <- all.data.names[grep(pattern = pattern, x = all.data.names)]
  }
  all.input.names <- unique(c(input.names, variable.names.from.patterns))
  all.input.names <- all.input.names[all.input.names != outcome.name]
  if (!is.na(all.data.names[1])) {
    all.input.names <- all.input.names[all.input.names %in%
    all.data.names]
  }
  input.names.delineated <- sprintf("`%s`", all.input.names)
  the.formula <- sprintf("`%s` ~ %s", outcome.name, paste(input.names.delineated, collapse = " + "))
  if (return.as == "formula") {
    return(as.formula(the.formula))
  }
  if (return.as != "formula") {
    return(the.formula)
  }
}

reduce.formula <- function(dat,the.initial.formula, max.categories= NA){
  require(data.table)
  dat <- setDT(dat)

  the.sides <- strsplit(x= the.initial.formula, split="~")[[1]]
  lhs <- trimws(x=the.sides[1], which="both")
  lhs.original <- gsub(pattern = "`",replacement="",x=lhs)
  if (!(lhs.original %in% names(dat))){
    return("Error: Outcome variable is not in names(dat).")
  }
  the.pieces.untrimmed <- strsplit(x= the.sides[2], split= "+", fixed =T)[[1]]
  the.pieces.untrimmed.2 <- gsub(pattern = "`",replacement = "",x= the.pieces.untrimmed, fixed = T)
  the.piece.in.names <- trimws(x= the.pieces.untrimmed.2,which = "both")

  the.pieces <- the.piece.in.names[the.piece.in.names %in% names(dat)]
  num.variables <- length(the.pieces)
  include.pieces <- logical(num.variables)

  for (i in 1:num.variables){
    unique.values <- dat[,unique(get(the.pieces[i]))]
    num.unique.values <- length(unique.values)
    if (num.unique.values >= 2){
      include.pieces[i] <- TRUE
    }
    if (!is.na(max.categories)){
      if (dat[, is.character(get(the.pieces[i]))|is.factor(get(the.pieces[i]))]==TRUE) {
        if (num.unique.values > max.categories){
          include.pieces[i] <- FALSE
        }
      }
    }
```

```
  }
  pieces.rhs <- sprintf("'%s'", the.pieces[include.pieces == TRUE])
  rhs <- paste(pieces.rhs,collapse = "+")
  the.formula <- sprintf("%s ~%s",lhs,rhs)
  return(the.formula)
}

eval.dat <- function(x){
  return(eval(as.name(x)))
}

### Business
fill_in_NA <- function(cuisine_ls){
  cat = cuisine_ls[1]
  # print(cat)
  for(i in cuisine_ls){
    DT[, "idx" := str_detect(DT$categories, i)]
    # DT[get("idx") == 1, eval(cuisine.name) := cuisine_ls[1]]
    DT[get("idx") == 1 & is.na(get(cuisine.name))==TRUE, eval(cuisine.name) := cat]
    DT[,"idx":=NULL]
  }
}

combine_regions <- function(region_ls){
  cat = region_ls[1]
  for(i in region_ls){
    DT[, "idx" := str_detect(DT$cuisine.info, i)]
    DT[get("idx") == 1, eval(cuisine.name) := cat]
    DT[,"idx":=NULL]
  }
}

fill_results <- function(result, model_name){
  record[model_name, ] <<- result[[1]]
  auc[, model_name] <<- result[[2]]
}
cities.name <- c("All", "Phoenix", "Las Vegas")
business.id.name = "business_id"
business.name.name = "name"
address.name= "address"
city.name = "city"
state.name = "state"
zipcode.name = "postal_code"
latitude.name = "latitude"
longitude.name = "longitude"
stars.name = "stars"
review.coutn.name = "review_count"
category.name = "categories"
is.open.name = "is_open"
cuisine.name = "cuisine.info"
pattern.attributes <- "Attributes_"

cuisine_american <- c('American', 'American (New)', 'Breakfast & Brunch', 'Burgers', 'Fast Food','Ameri
```

```
cuisine_bars <- c("Bars","Beer Bar","Breweries","Dive Bars","Cocktail Bars","Music Venues","Nightlife",
cuisine_cafe <- c("Cafes","Bubble Tea","Bakeries","Breakfast & Brunch","Bagels","Juice Bars & Smoothies
cuisine_asian <-c("Asian","Asian Fusion")
cuisine_Vietnamese <- c("Vietnamese", "Pho")
cuisine_vege <-c("Vege", "Gluten-Free","Vegetarian","Vegan")

cuisine_south_asia <- c("South_Asia", "Bangladeshi", "Indian","Pakistani", "Sri Lankan")
# cuisine_east_asia <- c("East_Asia", "Chinese", "Japanese","Korean")
cuisine_se_asia <- c("South_East_Asia", "Filipino", "Indonesian", "Laotian","Malaysian", "Singaporean",
cuisine_europe <- c("European", "French", "Greek","Italian","Portuguese","Polish","Russian","Ukrainian"

cuisine_ls <- c(cuisine_american, cuisine_bars, cuisine_cafe, cuisine_asian, cuisine_Vietnamese, cuisin
# levels(as.factor(DT$cuisine.info))

N <- 10000
c("Classification Tree", "Random Forest", "Logistic Regression", "Support Vector Machine", "KNN", "Naiv

count.fig = 1; count.table = 1
### user and restaurant
# user <- fread(input = "../Data/yelp_user.csv", verbose = FALSE)
# restaurant <- fread(input = "../Data/restaurant.csv")

# read.csv("../Data/yelp_business.csv") -> business
# head(business)
# # business[complete.cases(business), ] -> business.comp
# business %>% select(business_id, Attributes_GoodForKids, Attributes_RestaurantsReservations,
#                     Attributes_GoodForMeal, Attributes_Caters, Attributes_NoiseLevel,
#                     Attributes_RestaurantsTableService, Attributes_RestaurantsTakeOut,
#                     Attributes_RestaurantsPriceRange2, Attributes_OutdoorSeating,
#                     Attributes_BikeParking, Attributes_HasTV,Attributes_WiFi,
#                     Attributes_RestaurantsAttire, Attributes_RestaurantsGoodForGroups,
#                     Attributes_RestaurantsDelivery, Attributes_BusinessAcceptsCreditCards) -> busines
# business_selcted_features[complete.cases(business_selcted_features), ] -> business.comp
set.seed(621)
##############################################################################
# read in saved data
##############################################################################
DT <- readRDS("../Data/full.restaurant.w.cuisine.rds", refhook = NULL)
# all(str_detect(DT$categories, "American") == grepl("American", DT$categories))

##############################################################################
# Fill in NA
##############################################################################
fill_in_NA(cuisine_american); fill_in_NA(cuisine_bars)
fill_in_NA(cuisine_cafe); fill_in_NA(cuisine_asian)
fill_in_NA(cuisine_Vietnamese); fill_in_NA(cuisine_vege)
combine_regions(cuisine_south_asia); combine_regions(cuisine_se_asia)
combine_regions(cuisine_europe)
DT[get(cuisine.name)=="Taiwanese", eval(cuisine.name) := "Chinese"]
DT[get(cuisine.name)=="Armenian", eval(cuisine.name) := "Others"]
DT[is.na(get(cuisine.name))==TRUE, eval(cuisine.name) := "Others"]

##############################################################################
```

```r
# Sample N Observations and Create Dummies
################################################################################
## For every unique value in the string column, create a new 1/0 column

DT %>% select(is_open, stars, review_count, cuisine.info, total.sentiment.score, total.num.post) %>%
  sample_n(., N) %>%
  mutate(is_open = ifelse(is_open==1, "Yes", "No")) %>%
  # mutate_if(is.character, as.factor)
  mutate(is_open = as.factor(is_open)) %>%
  mutate(cuisine.info = ifelse(cuisine.info=="Sri Lankan", "Sri_Lankan", cuisine.info))%>%
  mutate(total_sentiment_score=total.sentiment.score,
         total_num_post=total.num.post,
         cuisine_info = as.factor(cuisine.info)) %>%
  select(-c(cuisine.info, total.sentiment.score, total.num.post)) -> dat

DT %>% select(is_open, stars, review_count, cuisine.info, total.sentiment.score, total.num.post) %>%
  sample_n(., N) %>%
  mutate(is_open = ifelse(is_open==1, "Yes", "No")) %>%
  mutate(is_open = as.factor(is_open)) %>%
  mutate(cuisine.info = ifelse(cuisine.info=="Sri Lankan", "Sri_Lankan", cuisine.info))%>%
  mutate(total_sentiment_score=total.sentiment.score) %>%
  mutate(total_num_post=total.num.post) -> dat.dummies

for(level in unique(dat.dummies$cuisine.info)){
  dat.dummies[paste("dummy", level, sep = "_")] <- ifelse(dat.dummies$cuisine.info == level, 1, 0)
}
dat.dummies %>% select(-c(cuisine.info, total.sentiment.score, total.num.post)) -> dat.dummies
# business <- DT
# attributes.list <- names(business)[grep(pattern = pattern.attributes, x = names(business))]
#
# unique.id <- business[, unique(get(business.id.name))]
# unique.name <- business[, unique(get(business.name.name))]
# unique.address <- business[, unique(get(address.name))]
# unique.city <- business[, unique(get(city.name))]
# unique.state <- business[, unique(get(state.name))]
# unique.zipcod <- business[, unique(get(zipcode.name))]
# unique.cuisine <- restaurant[, unique(get(cuisine.name))]
# unique.restaurant <- restaurant[, unique(get(business.id.name))]
#
# num.business <- length(unique.id)
# num.restaurant <- length(unique.restaurant)
# num.cuisine <- length(unique.cuisine)
#
# respondent.variables1 <- c(city.name, state.name, cuisine.name, review.coutn.name, stars.name)
# respondent.variables2 <- c(stars.name)
# dependent.variables <- c(is.open.name)
set.seed(621)
# is.factor(dat$is_open)

samp <- sample(1:nrow(dat), 0.8*nrow(dat))
trn <- dat[samp,]; test <- dat[-samp,]
trn.dummies <- dat.dummies[samp,]; test.dummies <- dat.dummies[-samp,]
set.seed(621)
```

```
nfold <- 10
folds = seq.int(nrow(trn)) %>%
    cut(breaks = nfold, labels=FALSE) %>%
    sample()
## error matrix
record <- matrix(NA, nrow = length(ls.models), ncol = 2)
colnames(record) <- c("Train Error", "Test Error")
rownames(record) <- ls.models

## auc matrix
auc <- matrix(NA, nrow = 1, ncol = length(ls.models))
colnames(auc) <- ls.models
rownames(auc) <-  c("auc")
names <- names(dat)
formula.fit <- as.formula(create.formula(outcome.name = names[1],
                                         input.names = names[2:length(names)]))
model.tree <- function(trn, tst = test, formula = formula.fit, dependent = is.open.name){

  features <- setdiff(names(trn), dependent)
  YTrain <- trn[, dependent]; XTrain <- trn[, features]
  YTest <- tst[, dependent]; XTest <- tst[, features]

  control <- trainControl(method = "cv", number = 10)
  tunegrid <- expand.grid(.cp = seq(0.01, 0.5, 0.01))
  garbage <- capture.output(tune.ct <- train(formula, data = trn, method = "rpart", trControl = control

  rpart.model <- rpart(formula, data=trn, method="class", control = rpart.control(cp = tune.ct$bestTune

  pred.tree.trn <- predict(rpart.model, trn, type="class")
  tree.trn.err <- erate(pred.tree.trn, YTrain)
  pred.tree.val <- predict(rpart.model, tst, type="class")
  tree.val.err <- erate(pred.tree.val, YTest)

  prob.tree <- predict(rpart.model, XTest)[,2]
  pred.tree <- prediction(prob.tree, YTest)
  perf.tree <- performance(pred.tree, measure = "tpr", x.measure = "fpr")
  auc.tree <- as.numeric(performance(pred.tree, "auc")@y.values)

  return(list(c(tree.trn.err, tree.val.err), auc.tree, perf.tree))
}
result.tree <- model.tree(trn = trn)
fill_results(result.tree, "Classification Tree")
model.rf <- function(trn, tst = test, numTrees = 200, formula = formula.fit, dependent = is.open.name){

  features <- setdiff(names(trn), dependent)
  YTrain <- trn[, dependent]; XTrain <- trn[, features]
  YTest <- tst[, dependent]; XTest <- tst[, features]

  mod.rf <- randomForest(formula = formula,
                         data = trn,
                         ntree = numTrees,
                         importance = TRUE
                         )
```

```
  pred.tf.trn <- predict(mod.rf, trn, type="class")
  rf.trn.err <- erate(pred.tf.trn, YTrain)
  pred.rf.val <- predict(mod.rf, test, type="class")
  rf.val.err <- erate(pred.rf.val,YTest)

  prob.rf <- predict(mod.rf, XTest, type="prob")[,2]
  pred.rf <- prediction(prob.rf, YTest)
  perf.rf <- performance(pred.rf, measure = "tpr", x.measure = "fpr")
  auc.rf <-  as.numeric(performance(pred.rf, "auc")@y.values)

  return(list(c(rf.trn.err, rf.val.err), auc.rf, perf.rf))
}
result.rf <- model.rf(trn = trn)
fill_results(result.rf, "Random Forest")
# record; auc
model.logit <- function(trn, tst=test, formula = formula.fit, dependent = is.open.name){

  features <- setdiff(names(trn), dependent)
  YTrain <- trn[, dependent]; XTrain <- trn[, features]
  YTest <- tst[, dependent]; XTest <- tst[, features]

  glm.fit <- suppressWarnings(glm(formula, data=trn, family=binomial()))
  prob.training <- predict(glm.fit, type="response")

  pred.logit <- prediction(prob.training, YTrain)
  perf.logit <- performance(pred.logit, measure="tpr", x.measure="fpr")
  auc.logit <- as.numeric(performance(pred.logit, "auc")@y.values)

  fpr <- performance(pred.logit, "fpr")@y.values[[1]]
  cutoff <- performance(pred.logit, "fpr")@x.values[[1]]
  fnr <- performance(pred.logit,"fnr")@y.values[[1]]

  rate <- as.data.frame(cbind(Cutoff=cutoff, FPR=fpr, FNR=fnr))
  rate$distance <- sqrt((rate[,2])^2+(rate[,3])^2)
  index <- which.min(rate$distance)
  best.threshold <- rate$Cutoff[index]

  matplot(cutoff, cbind(fpr,fnr), type="l",lwd=2, xlab="Threshold",ylab="Error Rate")
  legend("right", legend=c("FPR","FNR"), col=c(1,2), cex=1)
  abline(v=best.threshold, col=3, lty=3, lwd=3)

  pred.logit.trn <- predict(glm.fit,type="response")
  pred.Ytr <- ifelse(pred.logit.trn > best.threshold, "Yes", "No")
  logit.trn.err <- erate(pred.Ytr,YTrain)

  pred.logit.test <- predict(glm.fit, test[,-1], type="response")
  pred.Yvl <- ifelse(pred.logit.test > best.threshold, "Yes", "No")
  logit.test.err <- erate(pred.Yvl, YTest)

  return(list(c(logit.trn.err, logit.test.err), auc.logit, perf.logit))
}
result.logit <- model.logit(trn = trn)
fill_results(result.logit, "Logistic Regression")
```

```r
# record; auc
model.svm <- function(trn, tst= test, formula = formula.fit, dependent = is.open.name){

  ## standardize continuous variables
  trn %>% mutate_at(vars("stars", "review_count", "total_sentiment_score", "total_num_post"),
                    scale) -> trn
  tst %>% mutate_at(vars("stars", "review_count", "total_sentiment_score", "total_num_post"),
                    scale) -> tst

  features <- setdiff(names(trn), dependent)
  YTrain <- trn[, dependent]; XTrain <- trn[, features]
  YTest <- tst[, dependent]; XTest <- tst[, features]

  ## creating new formula
  formula <- as.formula(create.formula(outcome.name = dependent,
                                       input.names = features))

  tune.out <- tune(svm, formula, data = trn,
                   scale = FALSE,
                   kernel = "radial",
                   ranges = list(
                     cost=c(0.01, 0.1, 1, 10)
                     # gamma = 2^c(-8,-4,0,4)
                     )
                   )  #train to find the best cost

  tune.best <- tune.out$best.model
  svm.trn.err <- erate(predict(tune.best, XTrain), YTrain)
  svm.test.err <- erate(predict(tune.best, XTest), YTest)

  plot(tune.out, transform.x = log2, transform.y = log2)

  predict.svm <- predict(tune.best, XTest, decision.values=TRUE)
  prob.svm <- attr(predict.svm,"decision.values")
  pred.svm <- prediction(prob.svm, YTest)
  perf.svm <- performance(pred.svm,"tpr","fpr")
  auc.svm <- performance(pred.svm,"auc")@y.values[[1]]
  # plot(tune.out, type = "perspective", theta = 120, phi = 45)

  return(list(c(svm.trn.err, svm.test.err), auc.svm, perf.svm))
}
# result.svm <- model.svm(trn = trn.dummies, tst = test.dummies) # 0.2890 0.2835
# save(result.svm, file = "result_svm.rda")
# ls()
load(file = "result_svm.rda")
fill_results(result.svm, "Support Vector Machine")
# record; auc
model.knn <- function(trn, tst = test, dependent = is.open.name){

  features <- setdiff(names(trn), dependent)
  YTrain <- trn[, dependent]; XTrain <- trn[, features]
  YTest <- tst[, dependent]; XTest <- tst[, features]
```

```
  ## Creating new formula
  names <- names(trn)
  formula <- as.formula(create.formula(outcome.name = dependent,
                                       input.names = features))

  control <- trainControl(method  = "cv", number  = 5)
  garbage <- capture.output(fit <- train(formula,
                                 method     = "knn",
                                 tuneGrid   = expand.grid(k = 1:10),
                                 trControl  = control,
                                 preProcess = "scale",
                                 metric     = "Accuracy",
                                 data       = trn))

  pred.YTrn <- knn(train=XTrain, test=XTrain, cl=YTrain, k=fit$bestTune$k)
  pred.YTest <- knn(train=XTrain, test=XTest, cl=YTrain, k=fit$bestTune$k)
  knn.trn.err <- erate(pred.YTrn, YTrain)
  knn.test.err <- erate(pred.YTest, YTest)

  fit.knn <- knn(train=XTrain, test=XTest, cl=YTrain, k=fit$bestTune$k, prob = TRUE)
  prob <- attr(fit.knn, "prob")
  prob.knn <- 2*ifelse(fit.knn == "-1", 1-prob, prob) - 1
  pred.knn <- prediction(prob.knn, YTest)
  perf.knn <- performance(pred.knn, "tpr", "fpr")
  auc.knn <- performance(pred.knn,"auc")@y.values[[1]]

  return(list(c(knn.trn.err, knn.test.err), auc.knn, perf.knn))
}
result.knn <- model.knn(trn = trn.dummies, tst = test.dummies)
fill_results(result.knn, "KNN")
# record; auc
# features <- setdiff(names(trn.dummies), is.open.name)
#   YTrain <- trn.dummies[, is.open.name]; XTrain <- trn.dummies[, features]
#   YTest <- test.dummies[, is.open.name]; XTest <- test.dummies[, features]
#
#   formula <- as.formula(create.formula(outcome.name = is.open.name,
#                                        input.names = features))
#
# NBclassfier = naiveBayes(formula, data=trn.dummies)
#
#   trainPred=predict(NBclassfier, newdata = trn.dummies, type = "class")
#   trainTable=table(YTrain, trainPred)
#   testPred=predict(NBclassfier, newdata=test.dummies, type="class")
#   testTable=table(YTest, testPred)
#   trainAcc=(trainTable[1,1]+trainTable[2,2])/sum(trainTable)
#   testAcc=(testTable[1,1]+testTable[2,2])/sum(testTable)
#   # message("Accuracy")
model.nb.h2o <- function(trn, tst = test, dependent = is.open.name){

  trn %>%
  filter(!!as.name(dependent) == "Yes") %>%
  select_if(is.numeric) %>%
  cor() %>%
```

```r
corrplot::corrplot()

sink("trash.txt")
library(h2o)
h2o.no_progress()
h2o.init()
sink()

# do a little preprocessing
preprocess <- preProcess(trn, method = c("BoxCox", "center", "scale", "pca"))
train_pp   <- predict(preprocess, trn)
test_pp    <- predict(preprocess, tst)

# convert to h2o objects
train_pp.h2o <- train_pp %>%
  mutate_if(is.factor, factor, ordered = FALSE) %>%
  as.h2o()
test_pp.h2o <- test_pp %>%
  mutate_if(is.factor, factor, ordered = FALSE) %>%
  as.h2o()

# get new feature names --> PCA preprocessing reduced and changed some features
y <- dependent
x <- setdiff(names(train_pp.h2o), y)

# create tuning grid
hyper_params <- list(
  laplace = seq(0, 5, by = 0.5)
  )

# build grid search
grid <- h2o.grid(
  algorithm = "naivebayes",
  grid_id = "nb_grid",
  x = x,
  y = y,
  training_frame = train_pp.h2o,
  nfolds = 10,
  hyper_params = hyper_params
  )

# Sort the grid models by mse
sorted_grid <- h2o.getGrid("nb_grid", sort_by = "accuracy", decreasing = TRUE)
# sorted_grid
best_h2o_model <- sorted_grid@model_ids[[1]]
best_model <- h2o.getModel(best_h2o_model)

# confusion matrix of best model
# h2o.confusionMatrix(best_model)

auc <- h2o.auc(best_model, xval = TRUE)
fpr <- h2o.performance(best_model, xval = TRUE) %>% h2o.fpr() %>% .[['fpr']]
tpr <- h2o.performance(best_model, xval = TRUE) %>% h2o.tpr() %>% .[['tpr']]
```

```
  data.frame(fpr = fpr, tpr = tpr) -> perf.nb
  #   ggplot(aes(fpr, tpr) ) +
  #   geom_line() +
  #   ggtitle( sprintf('AUC: %f', auc) )

  # evaluate on test set
  # h2o.performance(best_model, newdata = test_pp.h2o)

  # predict new data
  # train model
  h2o.predict(best_model, newdata = train_pp.h2o) ->pred.h2o.nb.trn
  erate(as.vector(pred.h2o.nb.trn$predict), trn[, is.open.name]) -> nb.trn.err
  h2o.predict(best_model, newdata = test_pp.h2o) ->pred.h2o.nb.test
  erate(as.vector(pred.h2o.nb.test$predict), tst[, is.open.name]) -> nb.test.err

  # shut down h2o
  # h2o.removeAll()
  h2o.shutdown(prompt = FALSE)
  return(list(c(nb.trn.err, nb.test.err), auc, perf.nb))
}
# trn test 0.2925 0.2885
# dummies 0.296625 0.280000
result.nb <- model.nb.h2o(trn = trn.dummies, tst = test.dummies)
fill_results(result.nb, "Naive Bayes")
# record; auc
knitr::kable(record, digits = 3, caption = "Errors")
apply(auc, 1, round, 2) -> auc.rounded
sapply(ls.models, function(model.name){paste0(model.name, " = ",
                                              auc.rounded[model.name, ])}) -> ls.models.auc
knitr::kable(auc, digits = 2, caption = "Area Under the Curve")
# Classification tree
plot(result.tree[[3]], col='pink',lwd = 2, main="ROC Curves")
legend('bottomright', ls.models.auc,
       lwd=c(1, 1),
       col=c("pink","plum",'palegreen', 'skyblue', 'orange', 'grey'), cex = 0.6)

# random forest
plot(result.rf[[3]], col='plum',lwd = 2, add=TRUE)

# logistic regression
plot(result.logit[[3]], col='palegreen', add=TRUE, lwd = 2)

# svm
plot(result.svm[[3]], col='skyblue', add=TRUE, lwd = 2)

# knn
plot(result.knn[[3]], col='orange', add=TRUE, lwd = 2)

# naive bayes
lines(result.nb[[3]], col = 'grey', lty=1, lwd = 2)
```