1. Data Description

➢ Restaurant Id: Unique id of every restaurant across various cities of the world

➢ Restaurant Name: Name of the restaurant

➢ Country Code: Country in which restaurant is located

➢ City: City in which restaurant is located

➢ Address: Address of the restaurant

➢ Locality: Location in the city

➢ Locality Verbose: Detailed description of the locality

➢ Longitude: Longitude coordinate of the restaurant's location

➢ Latitude: Latitude coordinate of the restaurant's location

➢ Cuisines: Cuisines offered by the restaurant

➢ Average Cost for two: Cost for two people in different currencies

➢ Currency: Currency of the country

➢ Has Table booking: yes/no

➢ Has Online delivery: yes/ no

➢ Is delivering: yes/ no

➢ Switch to order menu: yes/no

➢ Price range: range of price of food

➢ Aggregate Rating: Average rating out of 5

➢ Rating color: depending upon the average rating color

➢ Rating text: text based on rating

➢ Votes: Number of ratings given by people

2. Executive Summary:

The data we are to use here is the Zomato restaurants data. Zomato analysis is one of the most useful analyses for food connoisseurs. This analysis helps to determine restaurants that fetch maximum value for money in various parts of a country. Moreover, this analysis caters to various categories of people such as locals and tourists in order to discover the best restaurants in a neighborhood or city in terms of food, service, location and price. Zomato data consisted of 9551 rows each of which had 21 features.

Zomato dataset had features which had to be corrected like number of Votes which we converted into standardized votes by multiplying votes with aggregate rating. The currency variable was standardized to one single currency (USD). There were a few features such as Restaurant Id, Restaurant Name, Country Code, City, Address, Locality Verbose, Longitude, Latitude, Switch to order menu which were not used in performing analysis.

After data cleaning we did predictive modeling to predict both numeric and categorical variables for different problem statements. We did Linear Regression, Step AIC, Decision Trees (CART), Logistic Regression to predict aggregate rating. Linear Discriminant Analysis, Random Forest to predict rating text.

We used resampling techniques such as Cross Validation, K-Fold, Leave one out cross validation (LOOCV) and selected the model which gave us the least error.

3. Project Motivation/Background:

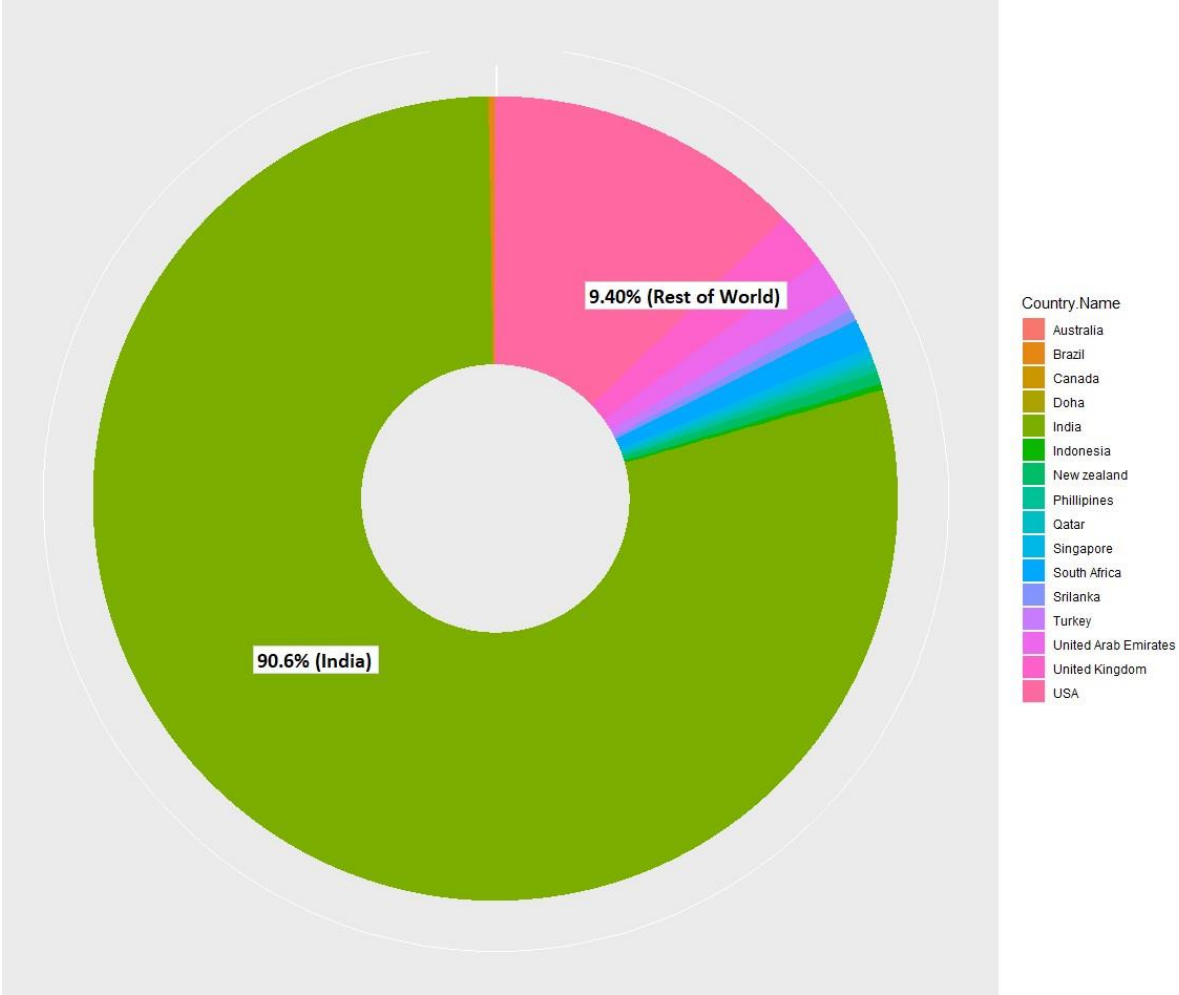
Before trying a new restaurant, we frequently consult with review platforms, such as Yelp, Zomato, or Google, where we can read comments from previous diners. Reading those reviews helps to make more informed decisions and can lead to a better dining experience with friends and family. Reviews are influential with diners, but how powerful can a single review be?

If a business receives one more star in overall rating, it can generate a 5 to 9% increase in revenue according to Michael Luca's research, "Reviews, Reputation, and Revenue. On the flip side, a single negative review may cost a business 30 future customers, which implies that an unfavorable comment can significantly damage reputation, profitability, and trustworthiness of a business.

Given that a single bad review can harm a business, how can the business owner mitigate the negative impact? One simple solution is to improve the business's customer relationship management system, by raising review complaints to the business's attention and developing an automated real-time complaint response engine.

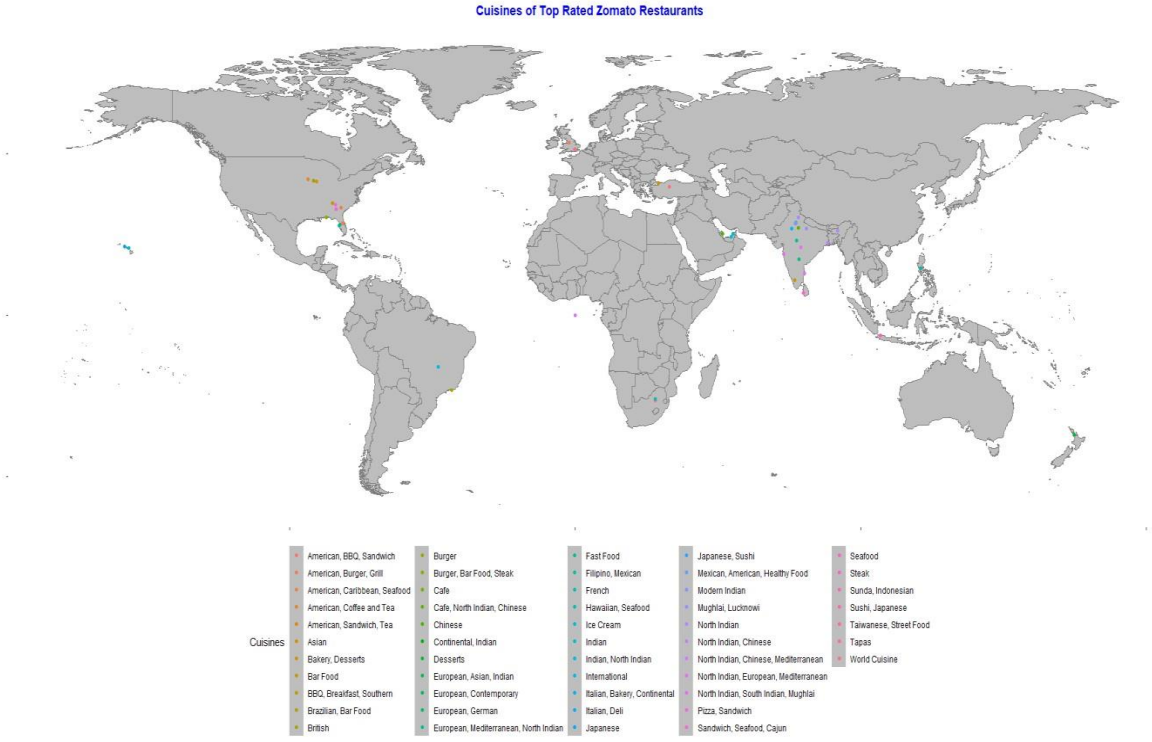
The motivation behind this analysis is to predict aggregate rating and what category do the top restaurants fall under.

4. Exploratory Data Analysis:



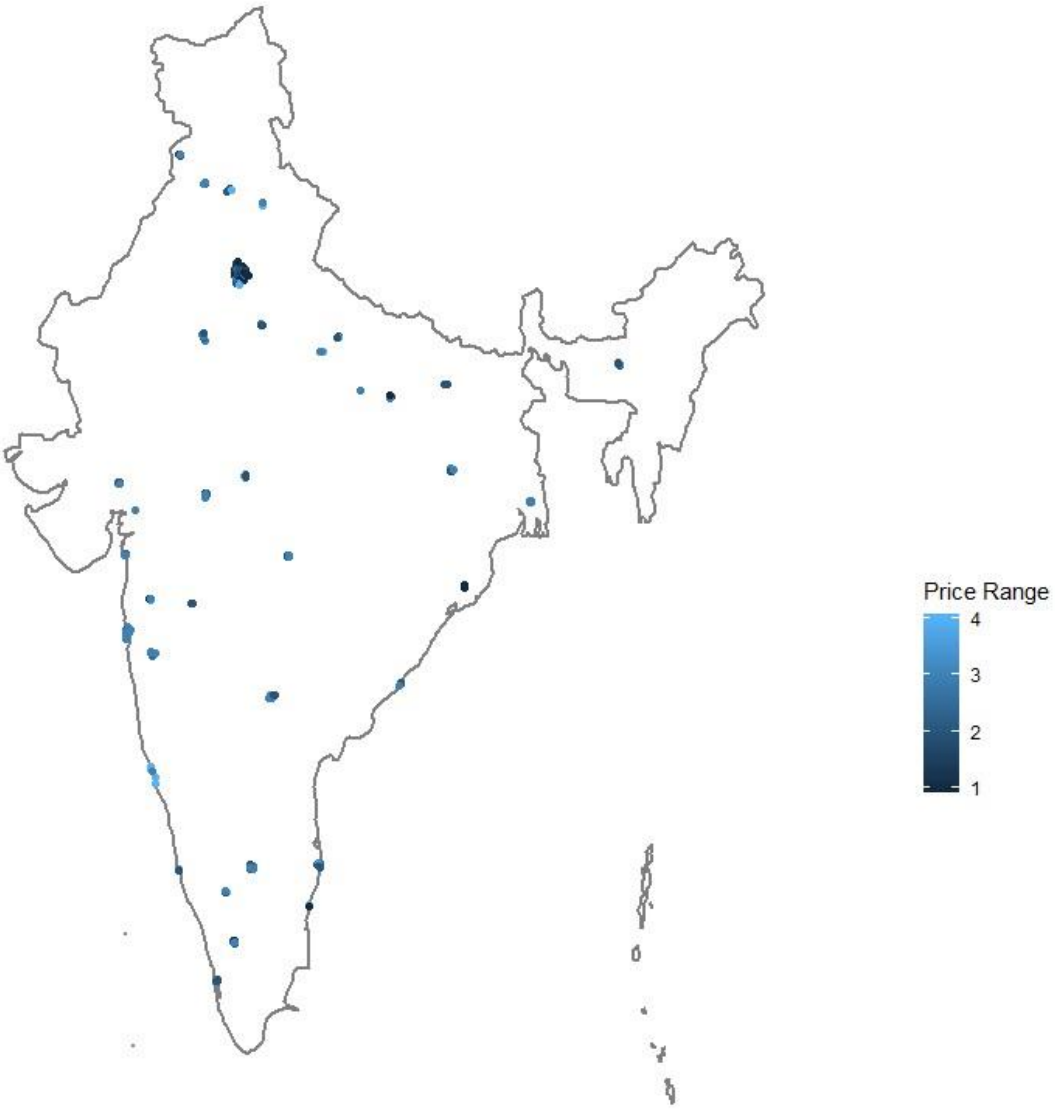
This Pie chart shows us the distribution of restaurants worldwide. Majority of the data is coming from India and rest of the world contributes to approximately 10%

4.1 Cuisines of Top Rated Zomato Restaurants:



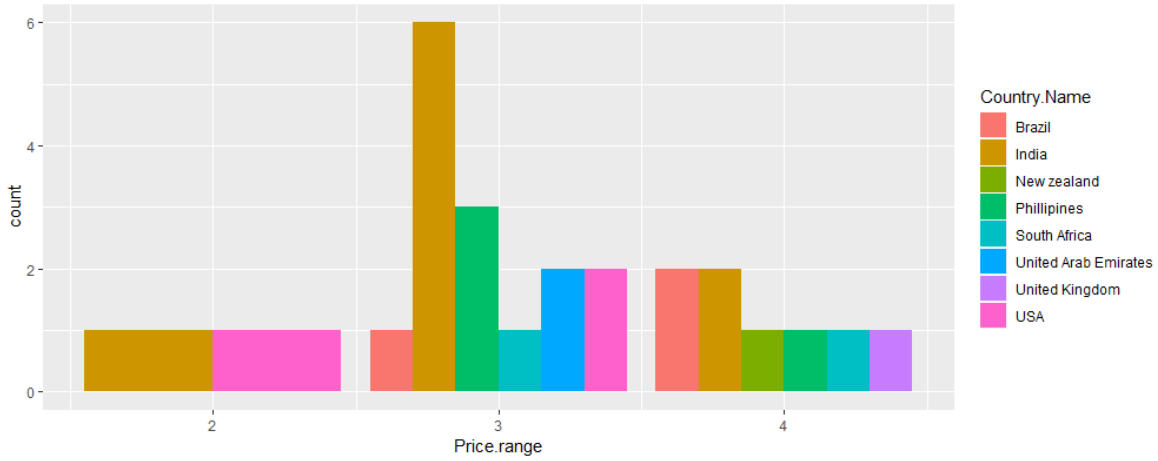
The above visualization shows a comparative study of restaurants covered by Zomato worldwide and in India. It is pretty evident that Zomato has its major concentration in India followed by the United States of America. Moreover, in India, Zomato has majorly focused on Restaurants in the northern part with its prime location being National Capital Region of India (aka NCR).

4.2 Price Range in India:

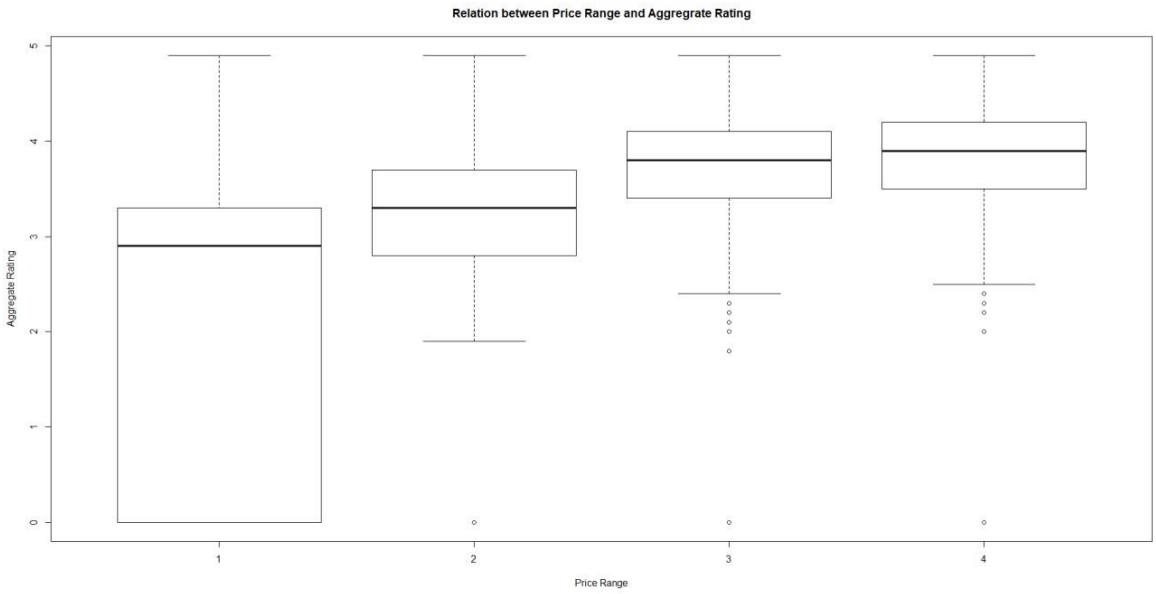


This plot shows the distribution of data in India based on the Price range, where 4 depicts the highest priced restaurant and 1 being the least.

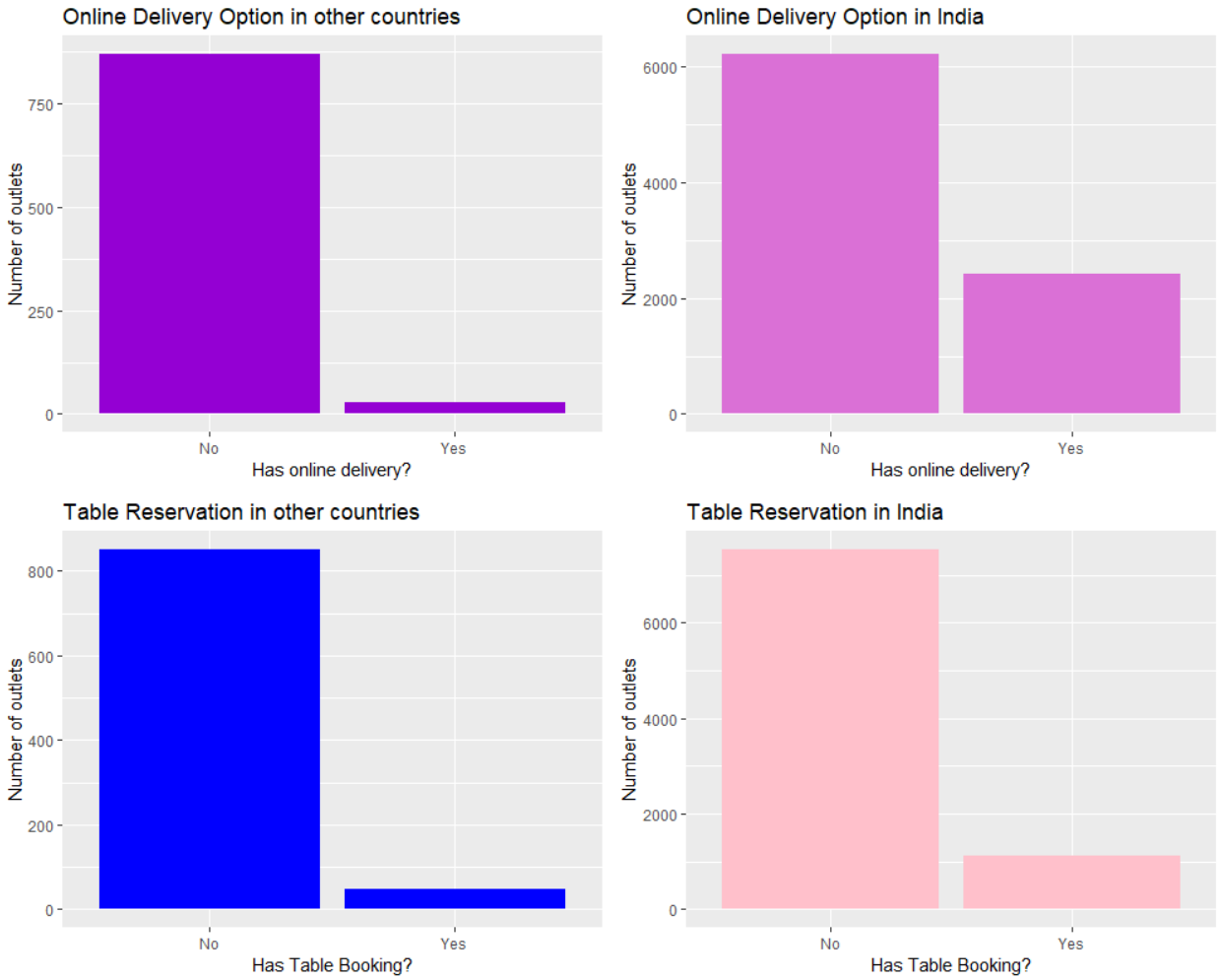
4.3 Price Range distribution and it's relation with aggregate rating:



This bar chart shows the restaurants which have aggregate rating of 4.9 (these fall under the top-rated restaurants) worldwide for each price range.



This boxplot gives an understanding of how the aggregate rating is distributed for each price range



A visualization of how many restaurants provide a home delivery and reservation options to its diners.

Which one is preferred more in India as well as outside India. Comparatively Indians like the food delivered at the door step whereas diners from the rest of the world like a table booked to avoid no last-minute hassles.

Introduction

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

There are two main types of Decision Trees:

1. Classification trees (Yes/No types)
   Here the decision is a categorical, e.g. a High/Low, Good/Bad
2. Regression trees (Continuous data types)
   Here the decision or the outcome variable is Continuous, e.g. a number like 123.

Basic definitions for formulas used in decision trees

Entropy: Entropy, also called as Shannon Entropy is denoted by H(S) for a finite set S, is the measure of the amount of uncertainty or randomness in data.

$$H(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

Intuitively, it tells us about the predictability of a certain event. Example consider a coin toss whose probability of heads is 0.5 and probability of tails is 0.5. Here the entropy is the highest possible, since there's no way of determining what the outcome might be. Alternatively, consider a coin which has heads on both the sides, the entropy of such an event can be predicted perfectly since we know beforehand that it'll always be heads. In other words, this event has no randomness hence its entropy is zero. Lower values imply less uncertainty while higher values imply high uncertainty.

Information Gain: Information gain is also called as Kullback-Leibler divergence denoted by IG (S, A) for a set S is the effective change in entropy after deciding on an attribute A. It measures the relative change in entropy with respect to the independent variables.

$$IG(S, A) = H(S) - H(S, A)$$

Alternatively,

$$IG(S,A) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$

where IG (S, A) is the information gain by applying feature A. H(S) is the Entropy of the entire set, while the second term calculates the Entropy after applying the feature A, where P(x) is the probability of event x.

Advantages of Decision Trees

- Simple to understand, interpret, visualize.
- Decision trees implicitly perform variable screening or feature selection.
- Can handle both numerical and categorical data. Can also handle multi-output problems.
- Decision trees require relatively little effort from users for data preparation.
- Nonlinear relationships between parameters do not affect tree performance.

Code Explanation

- Here we are predicting aggregate rating of the restaurant using CART (Classification and Regression Trees) decision tree model.

The following code randomly splits our train data into training data (75%) and test data (25%).

Code:

```
train_rows = sample(x = 1:nrow(z2), 0.75*nrow(z2), replace = FALSE)
train_cus = z2[train_rows,]
test_cus = z2[-train_rows,]
```

Code:

```
cart_model <- rpart(train_cus$Aggregate.rating~., data = train_cus)
rpart.plot::prp(cart_model)
plot(cart_model)
text(cart_model)

cart_predictions <- predict(cart_model, newdata = test_cus)
regr.eval(trues = test_cus,"Aggregate.rating"], preds
= cart_predictions)
```

The above code, fits a rpart(Decision tree model), plots the model and uses the learned model to make predictions on the test data.

```
> regr.eval(trues = test_cus,"Aggregate.rating"], preds = cart_predictions)
       mae        mse       rmse       mape
0.15624430 0.04873307 0.22075567        NaN
```

Code:

```
train.control <- trainControl(method = "CV", number = 10)
model <- train(Aggregate.rating~., data = z2, method = "rpart",
          trControl = train.control)
cv_predictions <- predict(model, newdata = test_cus)
regr.eval(trues = test_cus[,"Aggregate.rating"], preds = cv_predictions)
```

- The above code uses cross-validation method to do resampling and fits a rpart model on the training data.
- Predictions are done on the test data using the cross-validation model.

```
> regr.eval(trues = test_cus[,"Aggregate.rating"], preds = cv_predictions)
      mae       mse      rmse      mape
0.2843685 0.1675726 0.4093563       NaN
```

Code:

```
train.control <- trainControl(method = "CV", number = 10, repeats = 3)
kfold_model <- train(Aggregate.rating~., data = z2, method = "rpart",
               trControl = train.control)
results_kfold <- print(kfold_model)

kfold_predictions <- predict(model, newdata = test_cus)
regr.eval(trues     =     test_cus[,"Aggregate.rating"],     preds
=
kfold_predictions)
```

- The above code uses kfold cross-validation method to do resampling and fits a rpart model on the training data.
- Predictions are done on the test data using the k-fold cross-validation model.

```
> regr.eval(trues = test_cus[,"Aggregate.rating"], preds = kfold_predictions)
      mae       mse      rmse      mape
0.2843685 0.1675726 0.4093563       NaN
```

Code:

```
train.control <- trainControl(method = "LOOCV")
loocv_model <- train(Aggregate.rating~., data = z2, method = "rpart",
               trControl = train.control)
results_loocv <- print(model)
loocv_predictions <- predict(model, newdata = test_cus)
regr.eval(trues     =     test_cus[,"Aggregate.rating"],     preds
=
loocv_predictions)
```

```
> regr.eval(trues = test_cus[,"Aggregate.rating"], preds = loocv_predictions)
      mae       mse      rmse      mape
0.2843685 0.1675726 0.4093563       NaN
```

Logistic Regression

Introduction

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

Comparison to linear regression

Given data on time spent studying and exam scores. Linear Regression and logistic regression can predict different things:

- **Linear Regression** could help us predict the student's test score on a scale of 0 - 100. Linear regression predictions are continuous (numbers in a range).
- **Logistic Regression** could help use predict whether the student passed or failed. Logistic regression predictions are discrete (only specific values or categories are allowed). We can also view probability scores underlying the model's classifications.

Types of logistic regression

- Binary (Pass/Fail)
- Multi (Cats, Dogs, Sheep)
- Ordinal (Low, Medium, High)

Logistic Function

Logistic regression is named for the function used at the core of the method, the logistic function. The logistic function, also called the sigmoid function was developed by statisticians to describe
properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

1 / (1 + e^-value)

Where e is the base of the natural logarithms (Euler's number or the EXP() function in your spreadsheet) and value is the actual numerical value that you want to transform. Below is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function.

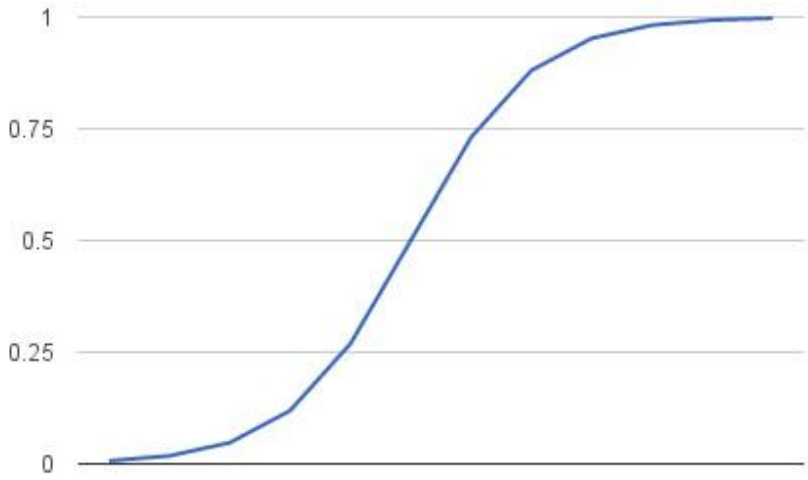


Fig: Logistic Function

Representation Used for Logistic Regression

Logistic regression uses an equation as the representation, very much like linear regression.

Input values (x) are combined linearly using weights or coefficient values (referred to as the

Greek
capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value.

Below is an example logistic regression equation:

y = e^(b0 + b1*x) / (1 + e^(b0 + b1*x))

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

The actual representation of the model that you would store in memory or in a file are the coefficients in the equation (the beta value or b's).

Logistic Regression Predicts Probabilities

Logistic regression models the probability of the default class (e.g. the first class).

For example, if we are modeling people's sex as male or female from their height, then the first class could be male, and the logistic regression model could be written as the probability of male given a person's height, or more formally:

P(sex=male|height)

Written another way, we are modeling the probability that an input (X) belongs to the default class (Y=1), we can write this formally as:

P(X) = P(Y=1|X)

We're predicting probabilities? I thought logistic regression was a classification algorithm?

Note that the probability prediction must be transformed into a binary values (0 or 1) in order to actually make a probability prediction. More on this later when we talk about making predictions.

Logistic regression is a linear method, but the predictions are transformed using the logistic function. The impact of this is that we can no longer understand the predictions as a linear combination of the inputs as we can with linear regression, for example, continuing on from above, the model can be stated as:

p(X) = e^(b0 + b1*X) / (1 + e^(b0 + b1*X))

we can turn around the above equation as follows

ln(p(X) / 1 – p(X)) = b0 + b1 * X

This is useful because we can see that the calculation of the output on the right is linear again (just like linear regression), and the input on the left is a log of the probability of the default class.

This ratio on the left is called the odds of the default class (it's historical that we use odds, for example, odds are used in horse racing rather than probabilities). Odds are calculated as a ratio of the probability of the event divided by the probability of not the event, e.g. 0.8/(1-0.8) which has the odds of 4. So, we could instead write:

ln(odds) = b0 + b1 * X

Because the odds are log transformed, we call this left-hand side the log-odds or the probit. It is possible to use other types of functions for the transform (which is out of scope_, but as such it is common to refer to the transform that relates the linear regression equation to the probabilities as the link function, e.g. the probit link function.

We can move the exponent back to the right and write it as:

odds = e^(b0 + b1 * X)

All of this helps us understand that indeed the model is still a linear combination of the inputs, but that this linear combination relates to the log-odds of the default class.

Making Predictions with Logistic Regression

Making predictions with a logistic regression model is as simple as plugging in numbers into the logistic regression equation and calculating a result.

An example.

Let's say we have a model that can predict whether a person is male, or female based on their height. Given a height of 150cm is the person male or female.

We have learned the coefficients of b0 = -100 and b1 = 0.6. Using the equation above we can calculate the probability of male given a height of 150cm or more formally P(male|height=150). We will use EXP () for e, because that is what you can use if you type this example into your spreadsheet:

y = e^(b0 + b1*X) / (1 + e^(b0 + b1*X))

y = exp(-100 + 0.6*150) / (1 + EXP(-100 + 0.6*X))

y = 0.0000453978687

Or a probability of near zero that the person is a male.

In practice we can use the probabilities directly. Because this is classification and we want a crisp answer, we can snap the probabilities to a binary class value, for example:

0 if p(male) < 0.5

1 if p(male) >= 0.5

Advantages / Disadvantages

It is a widely used technique because it is very efficient, does not require too many computational resources, it's highly interpretable, it doesn't require input features to be scaled, it doesn't require any tuning, it's easy to regularize, and it outputs well-calibrated predicted probabilities.

Like linear regression, logistic regression does work better when you remove attributes that are unrelated to the output variable as well as attributes that are very similar (correlated) to each other. Therefore, Feature Engineering plays an important role in regard to the performance of Logistic and also Linear Regression. Another advantage of Logistic Regression is that it is incredibly easy to implement and very efficient to train. I typically start with a Logistic Regression model as a benchmark and try using more complex algorithms from there on.

Because of its simplicity and the fact that it can be implemented relatively easy and quick, Logistic Regression is also a good baseline that you can use to measure the performance of other more complex Algorithms.

A disadvantage of it is that we can't solve non-linear problems with logistic regression since its decision surface is linear.

Logistic Regression is also not one of the most powerful algorithms out there and can be easily outperformed by more complex ones. Another disadvantage is its high reliance on a proper presentation of your data. This means that logistic regression is not a useful tool unless you have already identified all the important independent variables. Since its outcome is discrete, Logistic Regression can only predict a categorical outcome. It is also an Algorithm that is known for its vulnerability to overfitting.

Code

```
zo1<-zo[, -c(1,2,3:7,8,9,10,12,13:16,19)]
View(zo1)
```

The above code removes the columns which are not required for the model.

```
zo1["cat.agg"]<-NA
View(zo1$Aggregate.rating)
median = median(zo1$Aggregate.rating, na.rm = FALSE)
zo1$cat.agg<-ifelse(zo1$Aggregate.rating >median, c(1), c(0))
```

The above code adds a new column cat.agg which is categorical aggregate based on the median values.

If Aggregate value is greater than the median, then cat.agg = 1 else cat.agg = 0. Doing so converts the scattered values down to just two.

```
ex1<- sample(2, nrow(zo1),
             replace = TRUE,
             prob = c(0.7,0.3))
training<- zo1[ex1==1,]
testing<- zo1[ex1==2,]
```

The above code divides the data into two parts. Training and testing data which are 70% and 30%
of the data respectively.

```
##logistic regression with training data.
log.reg <- glm(cat.agg~., data = training)
log.reg
```

```
Call:
glm(formula = cat.agg ~ ., data = training)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.46340  -0.11494   0.00360   0.09871   0.54378

Coefficients: (5 not defined because of singularities)
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)            -3.510e+00  6.953e-02 -50.478  < 2e-16 ***
Latitude                6.916e-04  2.407e-04   2.873  0.00408 **
Has.Online.deliveryYes  5.769e-02  6.337e-03   9.104  < 2e-16 ***
Is.delivering.nowYes   -9.934e-03  4.268e-02  -0.233  0.81595
Price.range             8.852e-03  4.433e-03   1.997  0.04589 *
Aggregate.rating        9.673e-01  1.478e-02  65.433  < 2e-16 ***
Rating.colorGreen       4.408e-01  1.887e-02  23.363  < 2e-16 ***
Rating.colorOrange      7.642e-01  2.877e-02  26.559  < 2e-16 ***
Rating.colorRed         1.205e+00  4.214e-02  28.595  < 2e-16 ***
Rating.colorWhite       3.465e+00  6.924e-02  50.043  < 2e-16 ***
Rating.colorYellow      8.923e-01  2.233e-02  39.959  < 2e-16 ***
Rating.textExcellent           NA         NA      NA       NA
Rating.textGood                NA         NA      NA       NA
Rating.textNot rated           NA         NA      NA       NA
Rating.textPoor                NA         NA      NA       NA
Rating.textVery Good           NA         NA      NA       NA
Votes                  -1.832e-05  7.004e-06  -2.616  0.00890 **
standardized_price      3.105e-03  1.076e-03   2.887  0.00390 **
Standardized votes     -8.087e-04  2.468e-04  -3.277  0.00106 **

Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.04481949)

    Null deviance: 1672.20  on 6726  degrees of freedom
Residual deviance:  300.87  on 6713  degrees of freedom
AIC: -1781.7

Number of Fisher Scoring iterations: 2
```

```
## logistic regression prediction with training data.

log.reg.pred <- round(predict(log.reg, training, type = "response"),0)

log.reg.pred
summary(log.reg.pred)

table(log.reg.pred)
```

The above code builds a logistic regression model by taking in the training data. The second block does logistic regression prediction with the training data. This step is not a compulsion though.

```
## logistic regression prediction with testing data.

log.reg.pred2 <- round(predict(log.reg, testing, type = "response"),0)

log.reg.pred2

summary(log.reg.pred2)

table(log.reg.pred2)
```

```
log.reg.pred2
   0    1
1568 1247
```

This block of code does prediction with the testing data which 30% proportion of the dataset. Rounding of values is used for better interpretability. We can infer that about 1247 restaurants are rated above the median.

```
##accuracy of model

mean(log.reg.pred2==testing$cat.agg)*100
```

The above block of code tests the accuracy of the model. Running this gives accuracy about
96.6%.

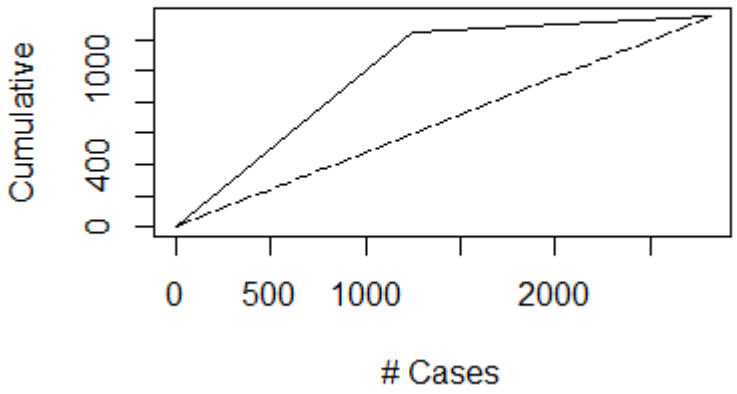
```
#Lift curve plotting

install.packages('gains')

library(gains)

curve<-gains(testing$cat.agg, log.reg.pred2, groups = 10)
```

```
plot(c(0,curve$cume.pct.of.total*sum(testing$cat.agg))~c(0,curve$cume.
obs),
     xlab = "# Cases", ylab = "Cumulative", main = "", type = "l")
lines(c(0,sum(testing$cat.agg))~c(0, dim(testing)[1]), lty =5)
```



The above block gives us the lift curve. A lift curve is a way of visualizing the performance of a classification model. The area between the random guess and model cumulative curve are compared. The greater the area between the two curves greater the effectiveness of our model.

Linear Discriminant Analysis

Introduction

Discriminant Analysis

Discriminant analysis is used to predict the probability of belonging to a given class (or category) based on one or multiple predictor variables. It works with continuous and/or categorical predictor variables.
The logistic regression for two-class classification problems, is when the outcome variable has two possible values (0/1, no/yes, negative/positive). Both logistic regression and discriminant analysis can be used for binary classification tasks.

Compared to logistic regression, the discriminant analysis is more suitable for predicting the category of an observation in the situation where the outcome variable contains more than two classes. Additionally, it's more stable than the logistic regression for multi-class classification problems.

Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a well-established machine learning technique and classification method for predicting categories. Its main advantages, compared to other classification algorithms such as neural networks and random forests, are that the model is interpretable, and that prediction is easy. Linear Discriminant Analysis is frequently used as a dimensionality reduction technique for pattern recognition or classification and machine learning. LDA looks for linear combinations of the independent variables to best explain the data and predict the different classes. Discriminant scores are calculated for each observation for each class based on these linear combinations. The scores are calculated using the below equation:

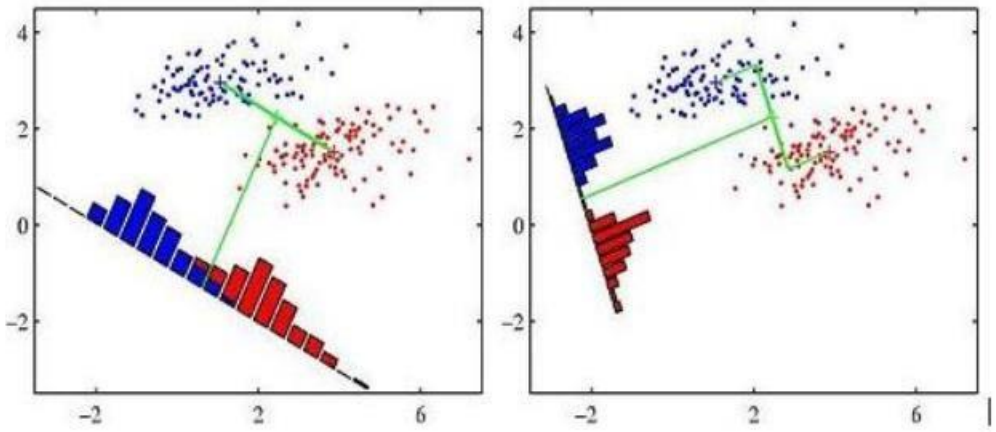
$$\delta_i(X) = -\frac{1}{2}\mu_i^T \Sigma^{-1} \mu_i + \mu_i^T \Sigma^{-1} X + \ln(\pi_i)$$

where

- δ_i is the discriminant score for class i
- X is the matrix of independent variables
- $\mu_i$is a vector containing the means of each variable for class i
- Σ is the covariance matrix of the variables (assumed to be the same for all classes)
- π_i is the prior probability that an observation belongs to class i

How does LDA work?

Each observation will have a score for each class. The class with the largest score will be the classification prediction for that observation. Below is a visual of what LDA is doing. Think of the data as if it were projected onto a linear combination of the variables (or linear discriminant) in one-dimensional space. LDA finds the linear discriminants where the greatest distinction between classes can be observed. In this picture the green line divides the space between the two classes. Observations on one side of the line will be predicted to be from the blue class while observations on the other side will be predicted to be from the red class. The more overlap between the classes, the greater the error rate will be.



Summarizing the LDA approach in 5 steps:

1. Compute the d-dimensional mean vectors for the different classes from the dataset.
2. Compute the scatter matrices (in-between-class and within-class scatter matrix).
3. Compute the eigenvectors (e_1,e_2,....,e_d) and corresponding eigenvalues (λ_1,λ_2,....,λ_d) for the scatter matrices.
4. Sort the eigenvectors by decreasing eigenvalues and choose kk eigenvectors with the largest eigenvalues to form a d x k dimensional matrix W (where every column represents an eigenvector).
5. Use this d x k eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the matrix multiplication: Y = X × W (where X is a n × d-dimensional matrix representing the n samples, and y are the transformed n × k-dimensional samples in the new subspace).

CODE

```
set.seed(100)
split_data<- sample(2,
            nrow(zo), replace
            = TRUE,
            prob =
c(0.7,0.3)) training<-
zo[split_data==1,] testing<-
zo[split_data==2,]
```

Code Explanation:

- The above code is used to divide the available dataset into two parts of 70% and 30%.
- The data with 70% of the data is used for training while the data with the remaining 30% is used for validation.
- Firstly, random sampling is done on the dataset "zo" and is assigned to a variable "split_data".
- Now, the 70% of the data of "zo" is assigned to a variable "training". Similarly, the remaining 30% data from "zo" is assigned to "testing".

Code:

```
lda1                                                                    <-
lda(Rating.text~Aggregate.rating+Country.Code+Has.Online.delivery+Has.
Table.booking
+Price.range+ standardized_price+ Standardized_votes, data = training)

summary(lda1)
```

Code Explanation

- An LDA model has been created and was assigned to a variable "lda1".
- Model was created using a column from the training data "Rating.Text" which is a dependent variable and the rest of the columns used for the model in the training table are the independent variables.
- The model is analyzed using the summary and then we further do the analysis by comparing the testing data and the training data and check the model performance.

Code

```
pred_test <- predict (lda1, testing)
pred_test
pred_train <- predict (lda1, training) pred_train
confusionMatrix(testing$Rating.text,pred_test$class)
confusionMatrix(training$Rating.text,pred_train$class)
```

Code Explanation

- In the above code, a prediction is made between the model that was created and training
  (which is the 70% of the data), and then assigned to a variable “pred_train”
- A data frame is made between pred_train and Rating.text from the table training.
- Similarly, a prediction is made between the LDA model and testing (which is the 30% of the data), and then assigned to a variable “pred_test”.
- A confusion matrix is made between pred_test and Rating.text from the table of testing.

OUTPUT:

Model Output

```
Call:
lda(Rating.text ~ Aggregate.rating + Country.Code + Has.Online.delivery +
    Has.Table.booking + Price.range + standardized_price + Standardized_votes,
    data = training)

Prior probabilities of groups:
   Average  Excellent       Good  Not rated       Poor  Very Good
0.39020045 0.03073497 0.21855976 0.22821084 0.01915367 0.11314031

Group means:
          Aggregate.rating Country.Code Has.Online.deliveryYes
Average           3.047945     3.274734              0.29528158
Excellent         4.663285   117.241546              0.12560386
Good              3.681250    23.512228              0.39062500
Not rated         0.000000     1.513988              0.04033832
Poor              2.298450     8.697674              0.62015504
Very Good         4.169685    68.322835              0.27559055
          Has.Table.bookingYes Price.range standardized_price
Average             0.10350076    1.624810           8.085594
Excellent           0.13526570    2.859903          40.806329
Good                0.18885870    2.129076          14.416568
Not rated           0.02081978    1.233572           4.849005
Poor                0.11627907    1.875969          10.019845
Very Good           0.23622047    2.594488          22.891706
          Standardized_votes
Average             24.84428
Excellent          191.29405
Good                53.41290
Not rated            0.00000
Poor                23.06623
Very Good           95.60625
```

```
Coefficients of linear discriminants:
                                 LD1         LD2          LD3         LD4
Aggregate.rating        5.6621971545 -0.66588216 -0.111109113  0.17628068
Country.Code           -0.0005714608  0.01291278  0.007725101  0.00470659
Has.Online.deliveryYes  0.1709529040 -0.20754407  1.224563269 -1.83358602
Has.Table.bookingYes   -0.0126597992 -0.04276806  0.426534795  1.50221857
Price.range            -0.0925268934  0.96073879  0.685395117  0.15797895
standardized_price      0.0293044790 -0.28128429  0.137688706  0.01308722
Standardized_votes     -0.0068207852  0.06658491 -0.037883279 -0.01275662
                                LD5
Aggregate.rating        0.16180291
Country.Code            0.01503201
Has.Online.deliveryYes  0.43205758
Has.Table.bookingYes   -0.68998020
Price.range            -0.65450903
standardized_price      0.07772589
Standardized_votes     -0.02188963

Proportion of trace:
   LD1    LD2    LD3    LD4    LD5
0.9916 0.0075 0.0007 0.0002 0.0000
```

Prediction between model and testing set:

```
> confusionMatrix(testing$Rating.text,pred_test$class)
Confusion Matrix and Statistics

           Reference
Prediction  Average Excellent Good Not rated Poor Very Good
  Average       992         0   87         0   30         0
  Excellent       0        57    0         0    0        37
  Good            0         0  608         0    0        20
  Not rated       0         0    0       611    0         0
  Poor            0         0    0         0   57         0
  Very Good       0        13   24         0    0       280

Overall Statistics

               Accuracy : 0.9251
                 95% CI : (0.9147, 0.9345)
    No Information Rate : 0.3523
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8991
```

Random Forest

Introduction

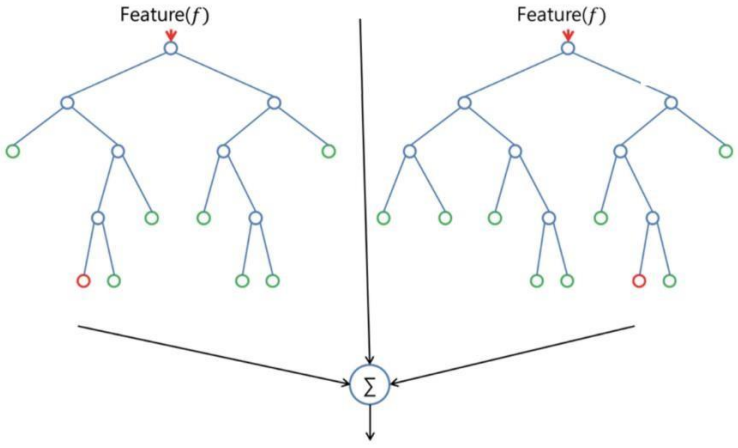
Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Decision trees are a popular method for various machine learning tasks. Trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

How Random Forests work?

Random Forest is a supervised learning algorithm. Like you can already see from its name, it creates a forest and makes it somehow random. The „forest“ it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

One big advantage of random forest is, that it can be used for both classification and regression problems, which form most current machine learning systems. I will talk about random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with two trees:



Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a

better model. Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

Unsupervised learning with Random Forests:

As part of their construction, random forest predictors naturally lead to a dissimilarity measure among the observations. One can also define a random forest dissimilarity measure between unlabeled data: the idea is to construct a random forest predictor that distinguishes the "observed" data from suitably generated synthetic data. The observed data are the original unlabeled data and the synthetic data are drawn from a reference distribution. A random forest dissimilarity can be attractive because it handles mixed variable types very well, is invariant to monotonic transformations of the input variables, and is robust to outlying observations. The random forest dissimilarity easily deals with many semi-continuous variables due to its intrinsic variable selection; for example, the "Addcl 1" random forest dissimilarity weighs the contribution of each variable according to how dependent it is on other variables. The random forest dissimilarity has been used in a variety of applications, e.g. to find clusters of patients based on tissue marker data.

Features and Advantages:

- It is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier.
- It runs efficiently on large databases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.
- Generated forests can be saved for future use on other data.
- Prototypes are computed that give information about the relation between the variables and the classification.
- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.

Disadvantages

- Random forests have been observed to overfit for some datasets with noisy classification/regression tasks.
- For data including categorical variables with different number of levels, random forests are biased in favor of those attributes with more levels. Therefore, the variable importance scores from random forest are not reliable for this type of data.

CODE

```
set.seed(100)
train <- sample(nrow(main1), 0.75*nrow(main1), replace = FALSE)
trainset <- main1[train,]
validset <- main1[-train,]
summary(trainset)
summary(validset);
```

Code Explanation

- The above code is used to divide the available dataset into two parts of 75% and 25%.
- The data with 75% of the data is used for training while the data with the remaining 25% is used for validation.
- Firstly, random sampling is done on the dataset “main1”. The 75% of the data is assigned to a variable “train”.
- Now, the 75% of the data of “main1” is assigned to a variable “trainset”. Similarly, the remaining data from “main1” (which is the remaining 25%) is assigned to “validset”.

Code

```
model <- randomForest(formula = trainset$Rating.text ~ ., data = trainset, importance = TRUE)
summary(model);
```

Code Explanation:

- A random forest model has been created and has been assigned to a variable “model”.
- Model is created using a column from the trainset “Rating.Text” which is a dependent variable and the rest of the columns in the trainset table are the independent variables.
- “importance = TRUE” is a parameter which will provide all the important variables that are available in the trainset table.

Code

```
predTr <- predict(model, trainset)

table(predTr, trainset$Rating.text)

predvalid <- predict(model, validset)

table(predvalid, validset$Rating.text);
```

Code Explanation:

- In the above code, a prediction is made between the model that was created and trainset
  (which is the 75% of the data), and then assigned to a variable “predTr”
- A data frame is made between predTr and Rating.text from the table trainset.
- Similarly, a prediction is made between the random forest model and validset (which is the 25% of the data), and then assigned to a variable “predvalid”.
- A data frame is made between predvalid and Rating.text from the table validset.

OUTPUT:

Model Output:

```
Call:
 randomForest(formula = trainset$Rating.text ~ ., data = trainset,      importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 0.24%
Confusion matrix:
          Average Excellent Good Not rated Poor Very Good class.error
Average      2786         0    0         0    0         0  0.00000000
Excellent       0       228    1         0    0         7  0.03389831
Good            0         0 1599         0    0         0  0.00000000
Not rated       0         0    0      1598    0         0  0.00000000
Poor            7         0    0         0  134         0  0.04964539
Very Good       0         0    2         0    0       801  0.00249066
```

Prediction between model and training set:

```
predTr      Average Excellent Good Not rated Poor Very Good
  Average      2786         0    0         0    0         0
  Excellent       0       235    0         0    0         0
  Good            0         1 1599         0    0         2
  Not rated       0         0    0      1598    0         0
  Poor            0         0    0         0  141         0
  Very Good       0         0    0         0    0       801
```

Prediction between model and validation set:

predvalid	Average	Excellent	Good	Not rated	Poor	Very Good
Average	951	0	0	0	1	0
Excellent	0	65	0	0	0	0
Good	0	0	501	0	0	1
Not rated	0	0	0	550	0	0
Poor	0	0	0	0	44	0
Very Good	0	0	0	0	0	275

Confusion Matrix Statistics for Validation Set:

```
Confusion Matrix and Statistics

          Reference
Prediction  Average Excellent Good Not rated Poor Very Good
  Average       951         0    0         0    1         0
  Excellent       0        65    0         0    0         0
  Good            0         0  501         0    0         1
  Not rated       0         0    0       550    0         0
  Poor            0         0    0         0   44         0
  Very Good       0         0    0         0    0       275

Overall Statistics

               Accuracy : 0.9992
                 95% CI : (0.997, 0.9999)
    No Information Rate : 0.3982
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9989

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

Conclusions :

From all the different models, Random forest has given the best accuracy of 99% and Rpart(Decision tree) has given the least RMSE of 0.22.

Indians like the food delivered at the doorstep whereas diners from the rest of the world like a table booked to avoid no last-minute hassles.

References:

1). https://www.kaggle.com/shrutimehta/zomato-restaurants-data