Project Report

Predictive Analysis of Cuisines from Ingredients

Rajesh Reddy Vanga

(800936087)

Samarth Belachikatte Shamanna

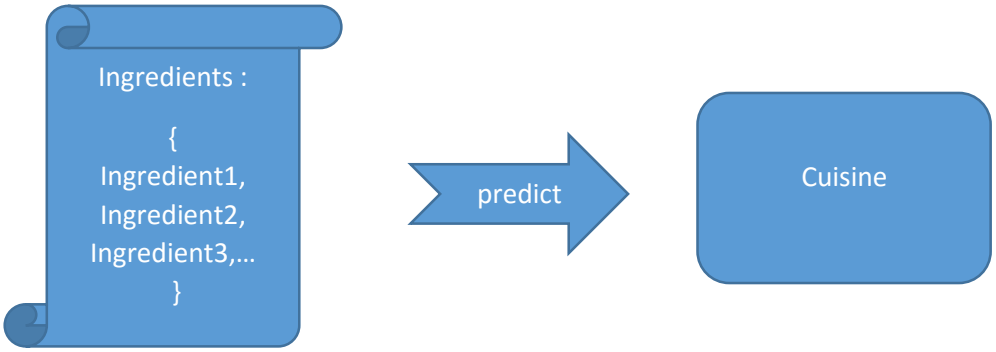
(800937728)

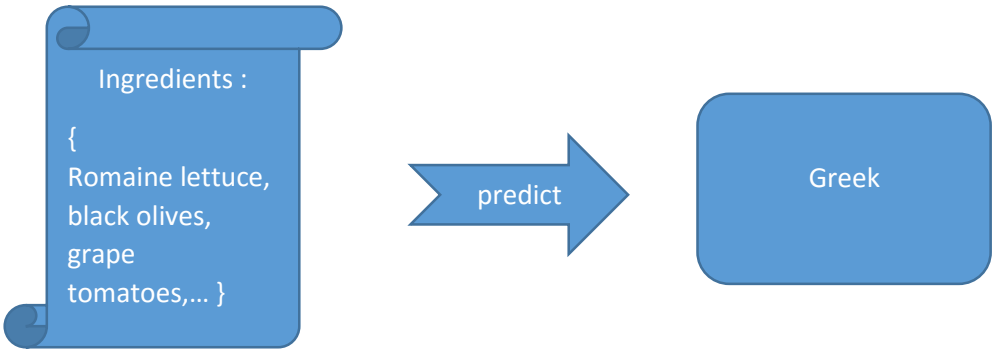
PROJECT DESCRIPTION

For an input set of ingredients in a recipe, predict the type of cuisine that the recipe belongs to. A list of recipes with ingredients are provided as the input and the type of cuisine (Italian, Chinese, Indian etc.) is predicted for the given input recipe(s).

Introduction:

Food has always been a major factor in every culture, but in the last few decades, cuisines of the world have migrated from their original homes to become known all over. 1000s of such cuisines have originated from all around the world. And each cuisine has a set of ingredients that make the cuisine unique. Like the Tortilla in Mexican cuisine, the rich spices in the Indian cuisine and many more. The type of the cuisine can be guessed from the ingredients alone and the goal of the project is to use recipe ingredients to categorize the recipe into its respective cuisine.





Dataset:

The dataset that was used for the project was obtained from Kaggle which was provided by Yummly, a huge database of various recipes from around the world. The dataset was initially available in JSON format with the training dataset consisting of around 39774 records and the test dataset consisting of 9944 records.

Training Data : 39774 records in JSON format

Test Data : 9944 records in JSON format


```
[
  {
    "id": 10259,
    "cuisine": "greek",
    "ingredients": [
      "romaine lettuce",
      "black olives",
      "grape tomatoes",
      "garlic",
      "pepper",
      "purple onion",
      "seasoning",
      "garbanzo beans",
      "feta cheese crumbles"
    ]
  },
  {
    "id": 25693,
    "cuisine": "southern_us",
    "ingredients": [
      "plain flour",
      "ground pepper",
      "salt",
      "tomatoes",
      "ground black pepper",
      "thyme",
      "eggs",
      "green tomatoes",
      "yellow corn meal",
      "milk",
      "vegetable oil"
    ]
```



```
[
  {
    "id": 18009,
    "ingredients": [
      "baking powder",
      "eggs",
      "all-purpose flour",
      "raisins",
      "milk",
      "white sugar"
    ]
  },
  {
    "id": 28583,
    "ingredients": [
      "sugar",
      "egg yolks",
      "corn starch",
      "cream of tartar",
      "bananas",
      "vanilla wafers",
      "milk",
      "vanilla extract",
      "toasted pecans",
      "egg whites",
      "light rum"
    ]
  },
```


Training Record Attributes:

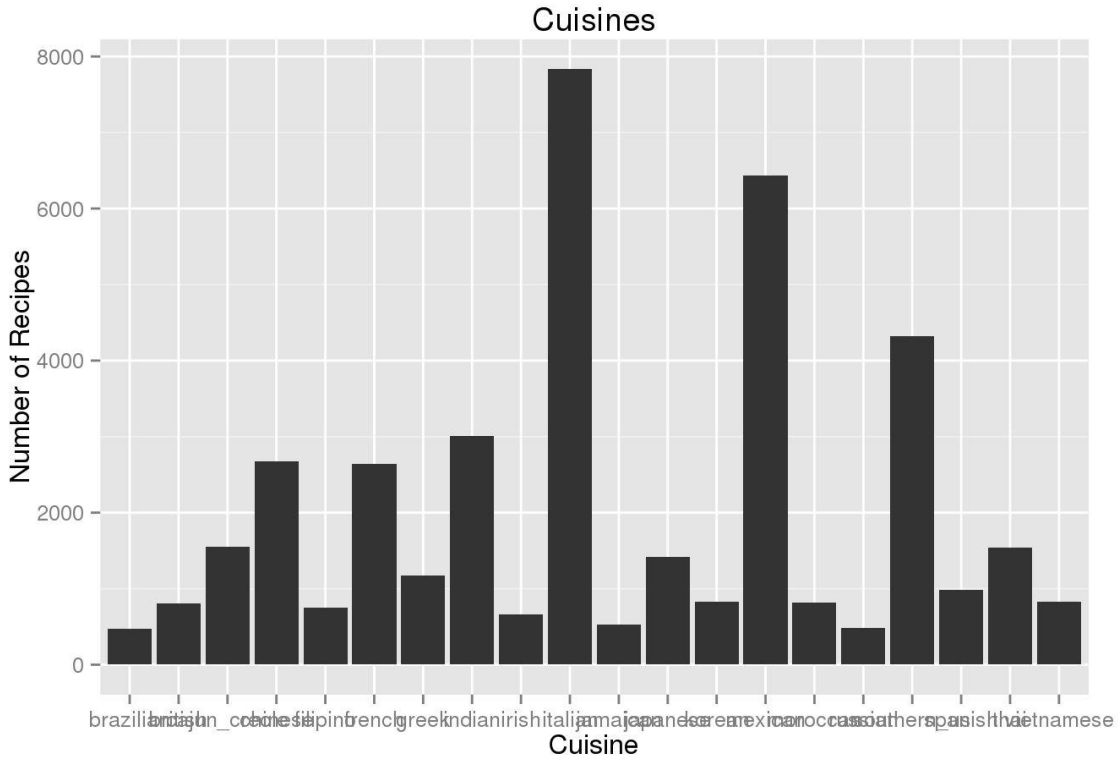
Attributes ->	**id**	**cuisine**	**ingredients**
Description	The id number of the record	The type of cuisine that the record belongs to	The list of ingredients
Example	22213	Indian	["water", "vegetable oil", "wheat", "salt"]

Test Record Attributes:

Attributes ->	**id**	**ingredients**
Description	The id number of the record	The list of ingredients
Example	22213	["water", "vegetable oil", "wheat", "salt"]

Dataset Cuisine Distribution:

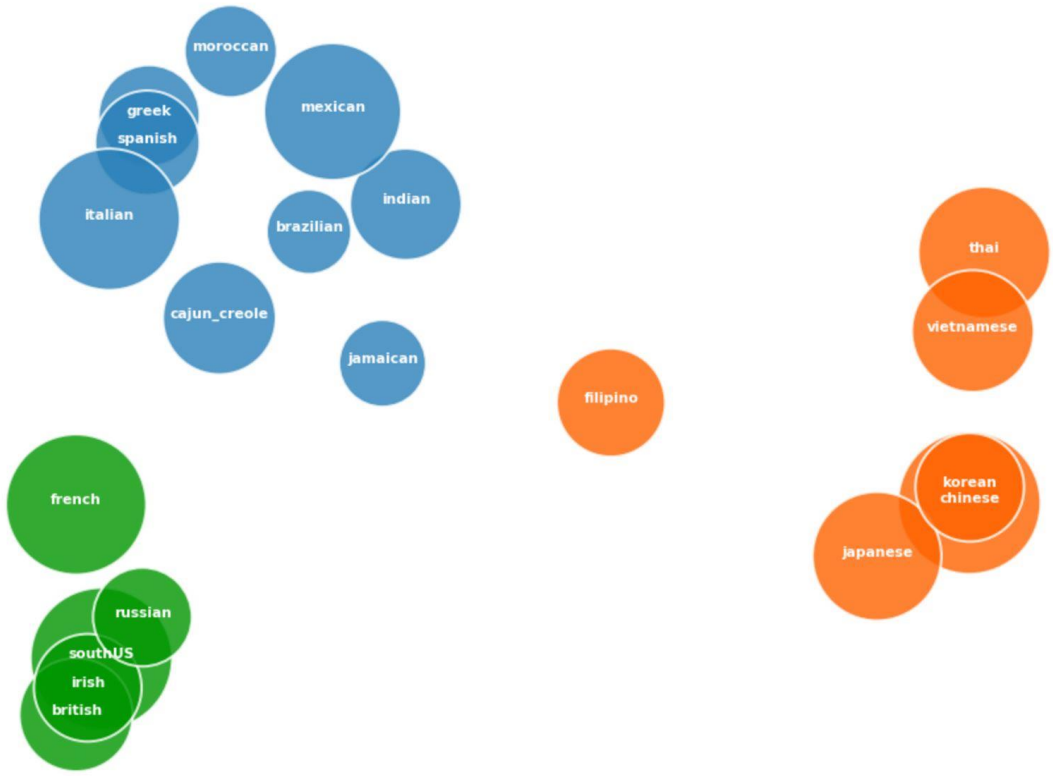
The Dataset consists of recipes belonging to 20 different cuisines like Chinese, Italian, Indian, Filipino, Mexican, Brazilian etc. The Distribution of cuisines can be analyzed from the below bar graph. The bar graph is plotted between the number of recipes to that of the number of Cuisines. And it can be inferred from the graph that the number of recipes belonging to Italian are the highest with more than 7000 recipes with Mexican being a close 2nd with more than 6000 recipes.



Cultural Diffusion by Recipes:

To analyze the cultural diffusion by Recipes. Each cuisine is converted to a **TF-IDF** vector (based on its ingredients), using **PCA** to obtain two vectors for visualization purposes, and clustering them by **K-Means**. – It can be inferred from the result that the Asian cuisines Korean, Chinese and Japanese appear together, and that of Italian,

Spanish and Greek appear together. Filipino is separated from the rest since it is known to be different from other types of cuisines.



Data Analysis :

R programming is used in the project to perform various tasks. Since the dataset is in JSON format, the data is loaded into R as a list to proceed with the various data mining tasks.

```
train_raw  <- fromJSON("../input/train.json", flatten = TRUE)
submit_raw <- fromJSON("../input/test.json", flatten = TRUE)
```

train_raw

Filter

	id	cuisine	ingredients
1	10259	greek	c("romaine lettuce", "black olives", "grape tomatoes..
2	25693	southern_us	c("plain flour", "ground pepper", "salt", "tomatoes",..
3	20130	filipino	c("eggs", "pepper", "salt", "mayonaise", "cooking oi...
4	22213	indian	c("water", "vegetable oil", "wheat", "salt")
5	13162	indian	c("black pepper", "shallots", "cornflour", "cayenne p..
6	6602	jamaican	c("plain flour", "sugar", "butter", "eggs", "fresh ging..

Showing 1 to 8 of 39,774 entries

Data Preprocessing:

Since the ingredients were already in the right format, with no stop words and other unformatted data, Basic set of pre-processing steps have been used on the dataset like

- Lower Case Conversion
- Remove Punctuations (But retain Hyphens and Underscores e.g low-fat)
- Stemming (Snowball)

Step1: Lower case Conversion:

All the ingredients are converted to lowercase before proceeding. Converting the ingredients column into lowercase makes it easier for the sake of comparison.

```
train$ingredients <- lapply(train$ingredients, FUN=tolower)
```

Step2 : Remove Punctuations:

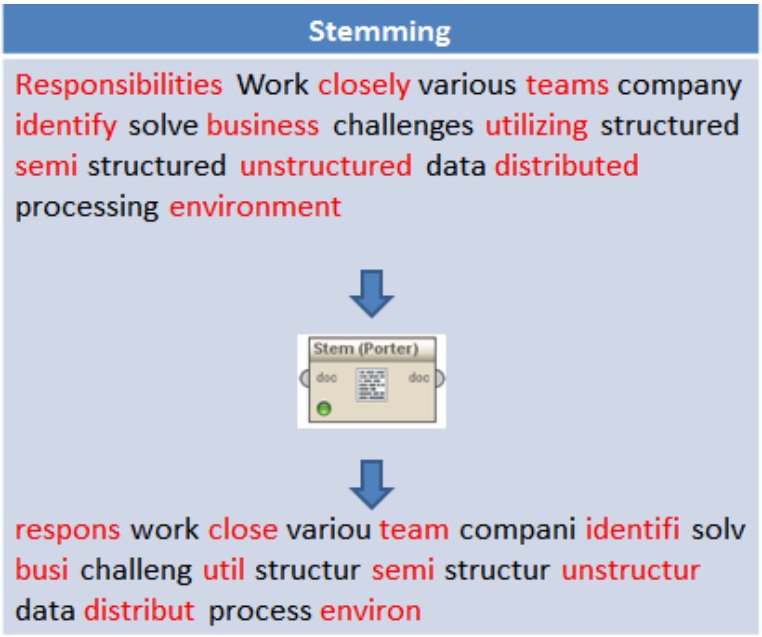
Punctuations have been removed from the ingredients column. Few punctuations like Hyphens and Underscores have been retained since most of the ingredients have hyphens eg: low-fat. If low and fat are separated, the whole meaning of the ingredient changes and might result in wrongful classification. Also, The spaces in the ingredients have been retained since most ingredients are a combination of more than one word eg: cheddar cheese, ground pepper etc.

Step3: Stemming:

Stemming techniques are used to find out the root/stem of a word. Stemming converts words to their stems, which incorporates a great deal of language-dependent linguistic knowledge. Behind stemming, the hypothesis is that words with the same stem or word root mostly describe same or relatively close concepts in text and so words can be conflated by using stems. For example, the words, user, users, used, using all can be stemmed to the word 'USE'.In linguistic morphology and information retrieval, stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since 1968. Many search engines treat words with the same stem as synonyms as a kind of query broadening, a process called conflation. Stemming programs are commonly referred to as stemming algorithms or stemmers.

A stemmer for English, for example, should identify the string "cats" (and possibly "catlike", "catty" etc.) as based on the root "cat", and "stemmer", "stemming", "stemmed" as based on "stem". A stemming algorithm reduces the words "fishing", "fished", "fish", and "fisher" to the root word, "fish". On the other hand, "argue", "argued", "argues", "arguing", and "argus" reduce to the stem "argu" (illustrating the case where the stem is not itself a word or root) but "argument" and "arguments" reduce to the stem "argument".

There are several types of stemming algorithms which differ in respect to performance and accuracy and how certain stemming obstacles are overcome. Since R has an implementation of SnowballC stemmer included in the tm package. We have used it for stemming the ingredients.



Ingredients Document Term Matrix (Bag of Ingredients):

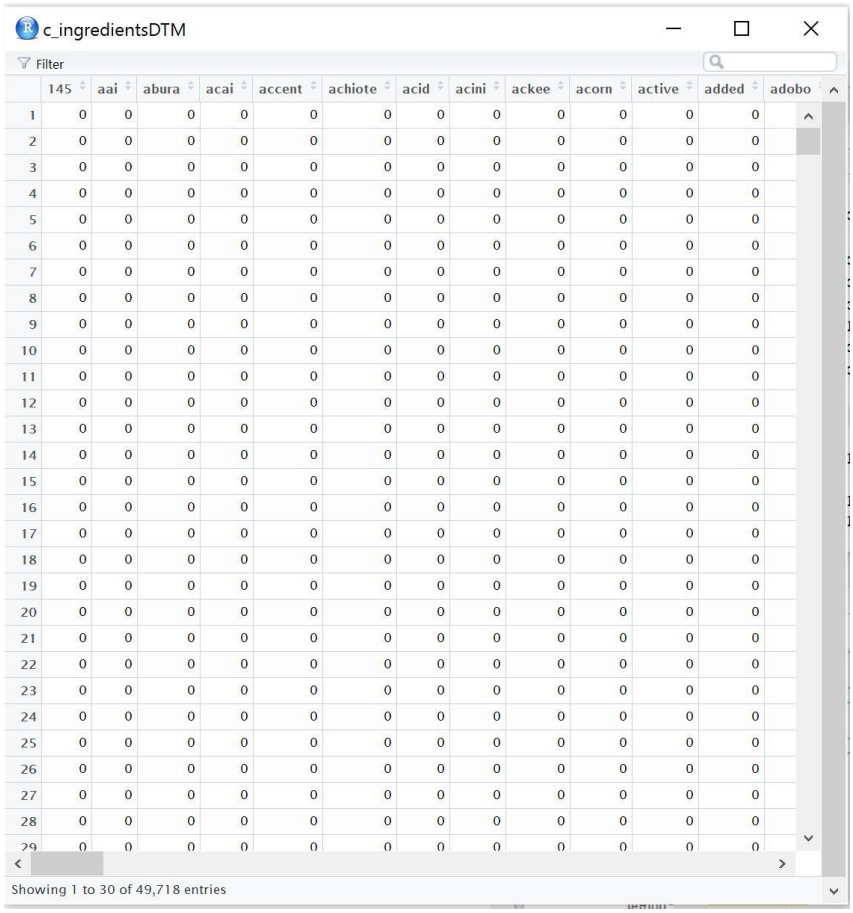
A **document-term matrix** or **term-document matrix** is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a document-term matrix, rows correspond to documents in the collection and columns correspond to terms. There are various schemes for determining the value that each entry in the matrix should take. One such scheme is tf-idf. They are useful in the field of natural language processing.

The preprocessed data is converted into a vector source, A vector source interprets each element of the vector x as a document. In this case, each recipe is regarded as a document and the ingredients forming the bag of words of each recipe. A document term matrix is generated from the corpus of the vector source. A simple representation of the DTM can be inferred from the table below. 1 denotes the presence of an ingredient in a recipe. 0 denotes the absence of an ingredient in a recipe.

Recipe / Ingredients	Ingredient1	Ingredient2	…
Recipe1	0	1	…
Recipe2	1	0	…
. . .			

Both the training and the test dataset are combined into a single DTM for ease of computation and further processing. A DTM with 49,718 entries consisting of recipes from both training and test dataset can be observed from the below image.

```
#- create a matrix of ingredients in both the TRAIN and SUBMIT set
c_ingredients <- c(Corpus(VectorSource(train_raw$ingredients)), Corpus(VectorSource(submit_raw$ingredients)));
```



c_ingredientsDTM

Filter

	145	aai	abura	acai	accent	achiote	acid	acini	ackee	acorn	active	added	adobo
1	0	0	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	0	0	
13	0	0	0	0	0	0	0	0	0	0	0	0	
14	0	0	0	0	0	0	0	0	0	0	0	0	
15	0	0	0	0	0	0	0	0	0	0	0	0	
16	0	0	0	0	0	0	0	0	0	0	0	0	
17	0	0	0	0	0	0	0	0	0	0	0	0	
18	0	0	0	0	0	0	0	0	0	0	0	0	
19	0	0	0	0	0	0	0	0	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	0	0	
21	0	0	0	0	0	0	0	0	0	0	0	0	
22	0	0	0	0	0	0	0	0	0	0	0	0	
23	0	0	0	0	0	0	0	0	0	0	0	0	
24	0	0	0	0	0	0	0	0	0	0	0	0	
25	0	0	0	0	0	0	0	0	0	0	0	0	
26	0	0	0	0	0	0	0	0	0	0	0	0	
27	0	0	0	0	0	0	0	0	0	0	0	0	
28	0	0	0	0	0	0	0	0	0	0	0	0	
29	0	0	0	0	0	0	0	0	0	0	0	0	

Showing 1 to 30 of 49,718 entries

Remove Sparse Terms:

Since the above Document Term Matrix is huge and the computational complexity increases by a huge margin, a sparse matrix of the DTM is created by eliminating the less occurring ingredients from the DTM. The end goal of creating a sparse matrix was to reduce the size of the document term matrix to reduce the computational complexity and also to disregard the less occurring features(ingredients) that are of not much significance for building a classifier.

Two different approaches have been followed to reduce the dimension of the matrix. One is to consider ingredients that occur in more than 1% of the dataset and discard the ingredients that occur less than 1% in the entire dataset.

```
datasetMAIN<- removeSparseTerms(datasetMAIN, 0.99)
```

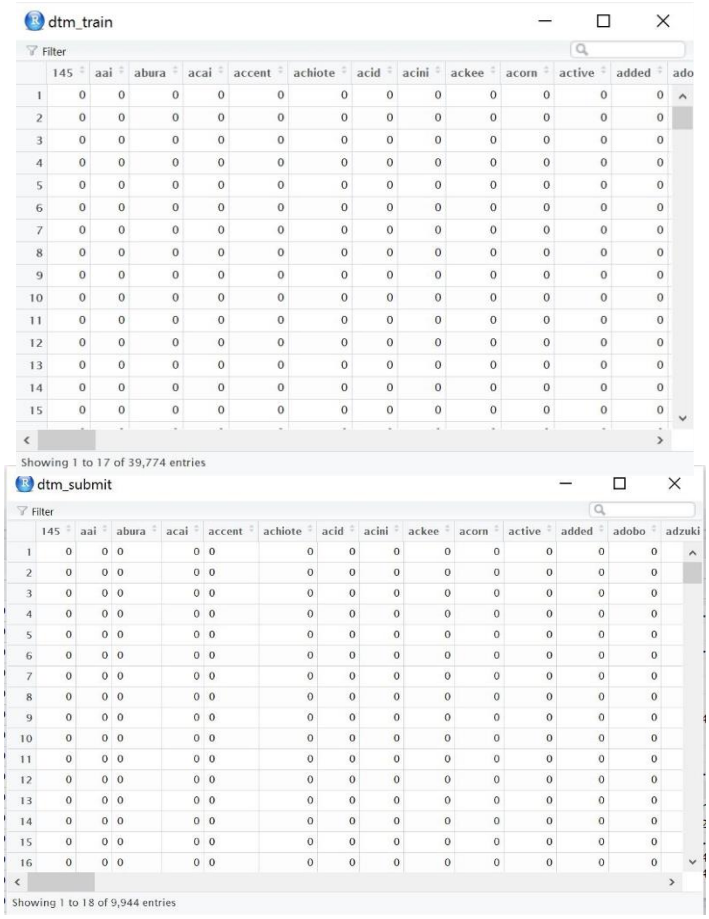
The other workaround that was followed was to eliminate all the ingredients that occur less than 3 times.

```
c_ingredientsDTM <- removeSparseTerms(c_ingredientsDTM, 1-3/nrow(c_ingredientsDTM));
```

Training DTM and Testing DTM :

Now that we have a pre processed DTM consisting of both training and test record sets ready for data analysis, the DTM is split to obtain the individual Training and Test Document Term Matrices.

```
#- split the DTM into TRAIN and SUBMIT sets
dtm_train  <- c_ingredientsDTM[1:nrow(train_raw), ];
dtm_submit <- c_ingredientsDTM[-(1:nrow(train_raw)), ];
```

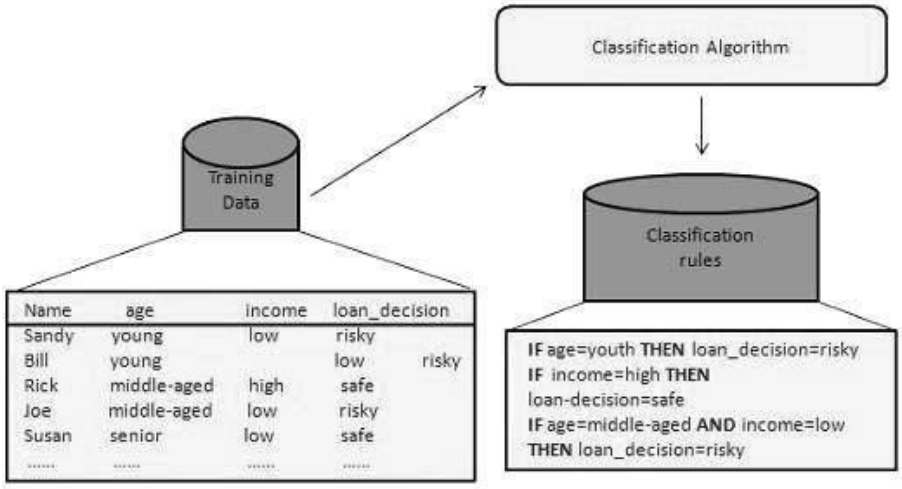


Classification:

Classification is a data mining function that assigns items in a collection to target categories or classes. The goal of classification is to accurately predict the target class for each case in the data. For example, a classification model could be used to identify loan applicants as low, medium, or high credit risks.

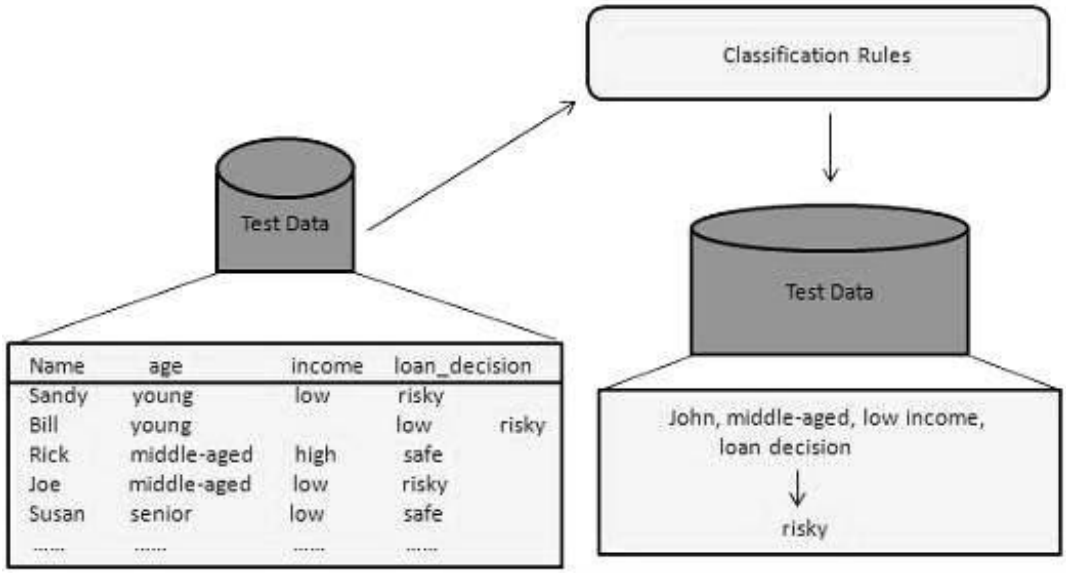
Building the Classifier or Model :

- This step is the learning step or the learning phase.
- In this step the classification algorithms build the classifier.
- The classifier is built from the training set made up of database tuples and their associated class labels.
- Each tuple that constitutes the training set is referred to as a category or class. These tuples can also be referred to as sample, object or data points.



Using Classifier for Classification

In this step, the classifier is used for classification. Here the test data is used to estimate the accuracy of classification rules. The classification rules can be applied to the new data tuples if the accuracy is considered acceptable.



In this project we have used various Classifier algorithms to build classification models to predict the class label, in our case the type of cuisines for all the recipes present in the test data.

Algorithm 1: CART (RPART):

Tree models are computationally intensive techniques for recursively partitioning response variables into subsets based on their relationship to one or more (usually many) predictor variables. Classification trees involve a categorical response variable and regression trees a continuous response variable; the approaches are similar enough to be referred to together as CART (Classification and Regression Trees). Classification and regression trees (as described by Brieman, Freidman, Olshen, and Stone) can be generated through the **rpart** package in R.

To grow a tree, use
rpart(*formula*, **data=**, **method=,control=)** where

formula	is in the format *outcome ~ predictor1+predictor2+predictor3*+ect.
data=	specifies the data frame
method=	**"class"** for a classification tree **"anova"** for a regression tree
control=	optional parameters for controlling tree growth. For example, control=rpart.control(minsplit=30, cp=0.001) requires that the minimum number of observations in a node be 30 before attempting a split and that a split must decrease the overall lack of fit by a factor of 0.001 (cost complexity factor) before being attempted.

```
fit<-rpart(cuisine~., data=trainDataset, method="class") #method-> classification
```

Rpart function in R is used to create a classification model on the training Dataset using the cuisine column as the class label. “class” method parameter is passed for a classification tree instead of “anova” which is for a regression tree.

A classification tree is built with the following 6 ingredients used in tree construction:

Cilantro ,Masala ,Oliv (Olive stemmed) ,

Parmesan , soy , tortilla

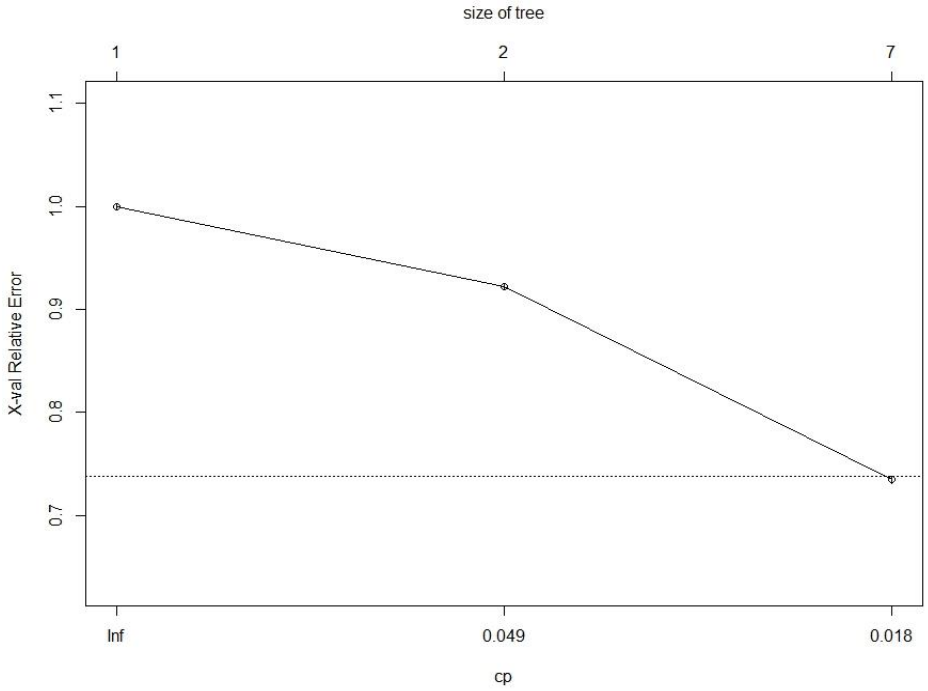
```
Classification tree:
rpart(formula = cuisine ~ ., data = trainDataset, method = "class")

Variables actually used in tree construction:
[1] cilantro masala   oliv     parmesan soy      tortilla

Root node error: 31936/39774 = 0.80294

n= 39774

        CP nsplit rel error  xerror      xstd
1 0.077812      0   1.00000 1.00000 0.0024841
2 0.031438      1   0.92219 0.92219 0.0027376
3 0.010000      6   0.73622 0.73268 0.0030733
```




```
Call:
rpart(formula = cuisine ~ ., data = trainDataset, method = "class")
  n= 39774

          CP nsplit rel error    xerror        xstd
1 0.07781187      0 1.0000000 1.0000000 0.002484064
2 0.03143788      1 0.9221881 0.9221881 0.002737618
3 0.01000000      6 0.7362224 0.7323084 0.003073667

Variable importance
    tortilla     parmesan          soy       masala        garam         oliv     cilantro        grate        sesam
          19           13           12            8            7            7            7            4            4
         oil         sauc       chees extra_virgin         chip         jack        salsa    enchilada     monterey
           3            2            2            2            2            2            2            1            1
       mirin       chines         wine      parsley
           1            1            1            1
```

A decision tree with low cp and optimal depth are chosen. High cp value would result in a sparse tree that would result in an inaccurate prediction. From the results above, the lowest cp value 0.010000 has number of splits to be 6. This is optimal and since there is

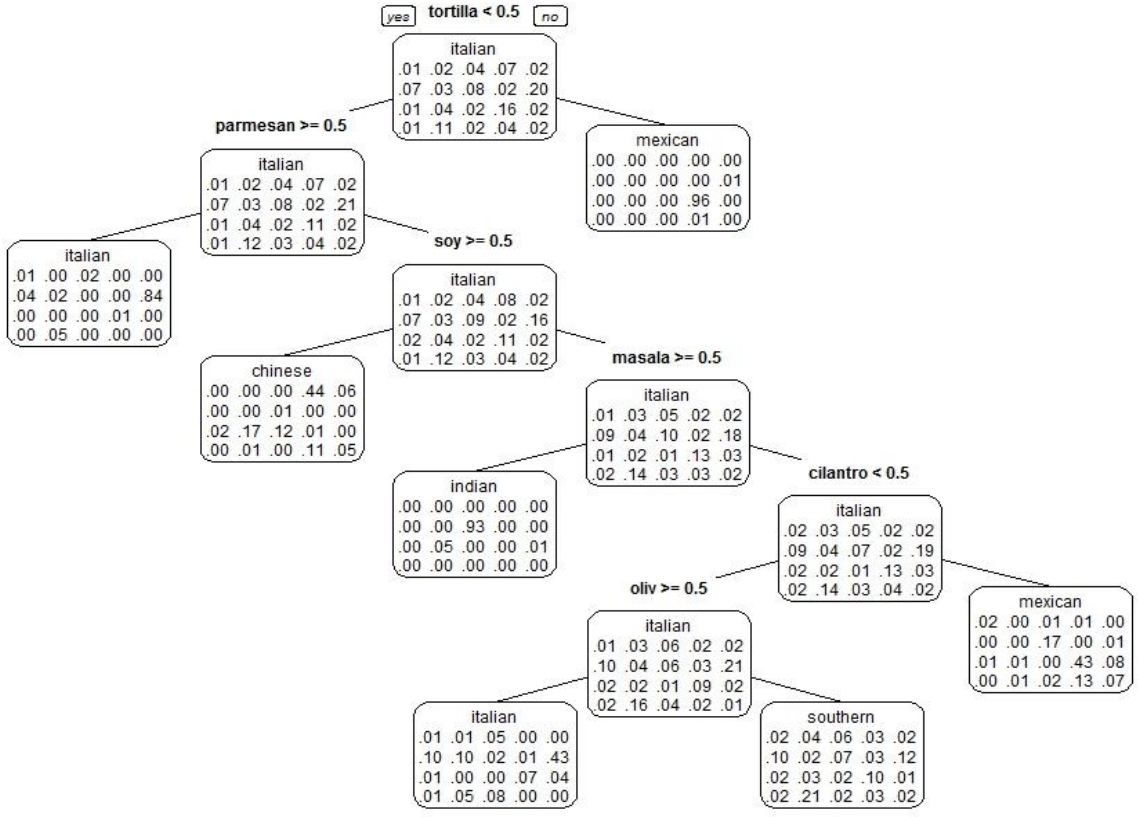
no other cp value lower than this, we choose cp 0.01 with nsplit = 6 for building our classification tree model.

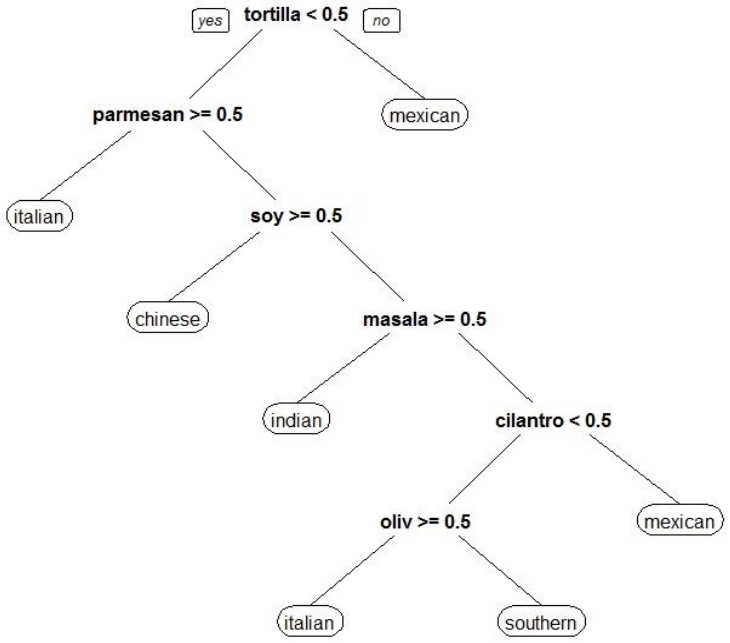
From the variable importance, 6 variables are chosen in the order of the variable importance for building the decision tree.

The following 6 variables are taken based on their variable importance

Variable (Ingredient)	Variable Importance
tortilla	19
Parmesan	13
Soy	12
Masala	8
Garam	7
Oliv	7

The following decision tree model is obtained:





The root of the tree model is split on the variable with the highest feature importance i-e tortilla and further split based on the feature importance.

The model that is obtained can classify the given recipes into only 5 cuisines respectively namely Mexican, Italian, Chinese, Indian, Southern.

Output: A csv file with the recipe Ids mapped to the predicted cuisines is obtained.

1: id	2: cuisine
5	mexican
7	indian
11	mexican
12	southern_us
13	southern_us
17	italian
18	italian
23	italian
26	southern_us
28	southern_us
35	mexican
36	italian
42	southern_us
47	indian
51	italian
52	chinese

Algorithm 2: Softmax (Xgboost):

Xgboost is short for e**X**treme **G**radient **Boost**ing package.

XGBoost is a library designed and optimized for boosting trees algorithms. Gradient boosting trees model is originally proposed by Friedman et al. The underlying algorithm of XGBoost is similar, specifically it is an extension of the classic gbm algorithm. By employing multi-threads and imposing regularization, XGBoost is able to utilize more computational power and get more accurate prediction.

It is an efficient and scalable implementation of gradient boosting framework. Two solvers are included:

- *linear* model ;
- *tree learning* algorithm.

It supports various objective functions, including *regression*, *classification* and *ranking*. The package is made to be extendible, so that users are also allowed to define their own objective functions easily.

One evidence of its accuracy is that XGBoost is used in more than half of the winning solutions in machine learning challenges hosted at Kaggle.

Softmax:

It is a regression technique that uses an iterative Algorithm to estimate the parameters of the model. This is similar to Multiclass SVM but with a different Loss function. It is Competitive in terms of CPU and memory consumption. A softmax regression has two steps: first we add up the evidence of our input being in certain classes, and then we convert that evidence into probabilities.

In our case, a recipe is taken as an input, probability of the recipe in a particular class(cuisine) is calculated and the highest probability class is returned.

Suppose if the probability of our recipe belonging to cusine Italian is highest at 0.7 compared to all other probabilities which are less than 0.7, then Italian is assigned as the class label to the recipe.

Xgboost() function is used in R to build the model.

```
#- prepare the spare matrix (note: feature index in xgboost starts from 0)
xgbmat     <- xgb.DMatrix(Matrix(data.matrix(dtm_train[, !colnames(dtm_train) %in% c("cuisine")])), label=as.numeric(dtm_train$cuisine)-1);

#- train our multiclass classification model using softmax
xgb        <- xgboost(xgbmat, max.depth = 25, eta = 0.3, nround = 200, objective = "multi:softmax", num_class = 20);
```


```
xgb        <- xgboost(xgbmat, max.depth = 25, eta = 0.3, nround = 200, objective = "multi:softmax", num_class = 20);
```

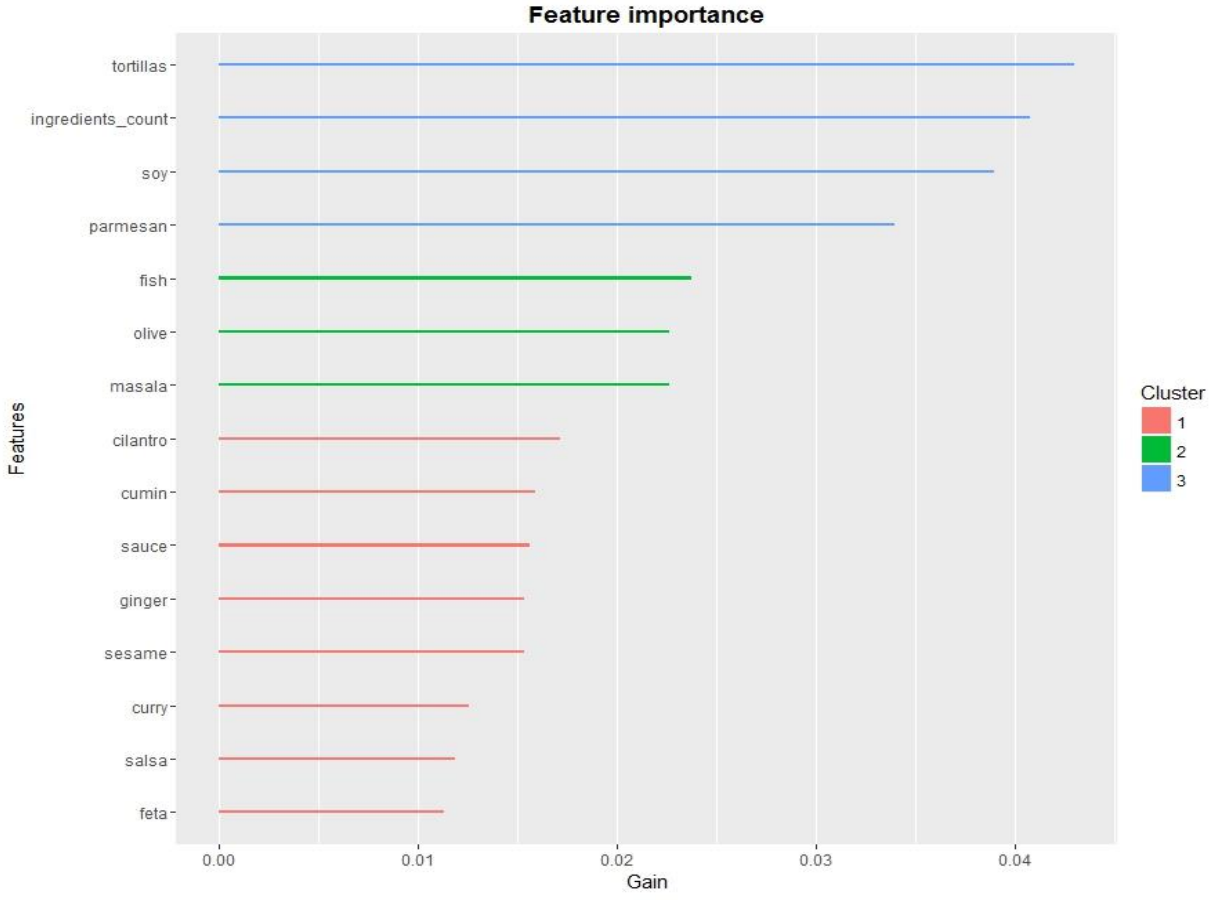

- Max.depth : maximum depth of a tree, increase this value will make model more complex / likely to be overfitting.

- Eta: step size shrinkage used in update to prevents overfitting. After each boosting step, we can directly get the weights of new features. and eta actually shrinks the feature weights to make the boosting process more conservative.
- Nround : The number of rounds for boosting
- Objective: "multi:softmax" --set XGBoost to do multiclass classification using the softmax objective, you also need to set num_class(number of classes)
- Num_class: Number of classes

Feature Importance:

In xgboost, each split tries to find the best feature and splitting point to optimize the objective. We can calculate the gain on each node, and it is the contribution from the selected feature. In the end we look into all the trees, and sum up all the contribution for each feature and treat it as the importance. If the number of features is large, we can also do a clustering on features before we make the plot. Here's an example of the feature importance plot from the function xgb.plot.importance:

A Model is built using the feature importance of the ingredients. The feature(Ingredient) having the highest gain is considered and the node is split on that particular attribute.



Output: A csv file with the recipe Ids mapped to the predicted cuisines is obtained.

1: id	2: cuisine
5	italian
7	spanish
11	southern_us
12	mexican
13	moroccan
17	italian
18	mexican
23	french
26	mexican
28	indian
35	italian
36	french
42	indian
47	italian
51	british

Output Evaluation:

CART vs xgboost:

Building a classifier using the rpart package yielded a tree with nsplit = 6 with 5 classes into which the recipes can be classified into. Since the total number of cuisines is 20 , there is bound to be a lot of misclassification in the prediction stage. To calculate the accuracy and to estimate the error of the model, the "train" dataset/bag of ingredients is separated into a training and validation set.

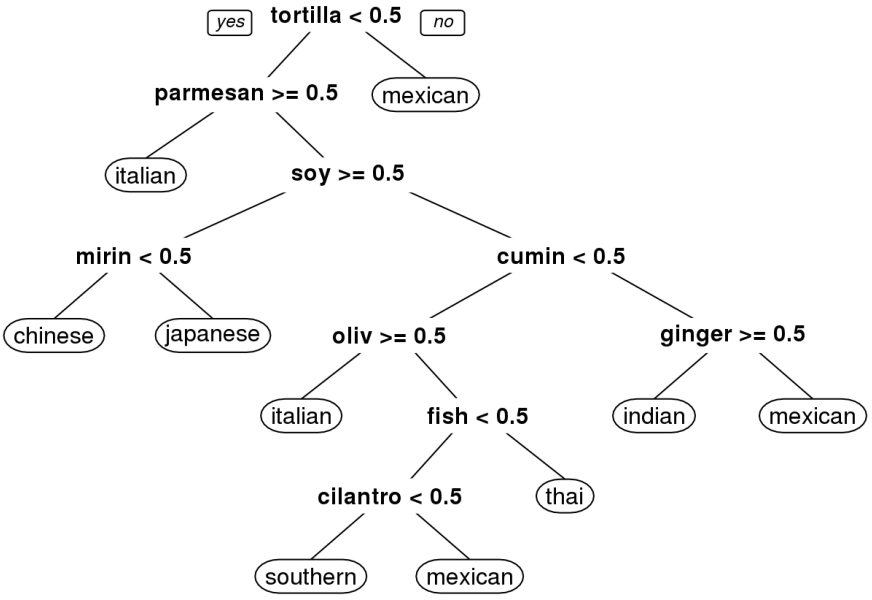
```
inTrain <- createDataPartition(y = ingredientsDTM$cuisine, p = 0.6, list = FALSE)
training <- ingredientsDTM[inTrain,]
testing <- ingredientsDTM[-inTrain,]
```

The training Dataset is partitioned into 60:40 training:testing datasets and a model is generated with the 60/100 training data partition. A confusion matrix was generated for this model and the accuracy was found to be quite poor.

```
Overall Statistics

               Accuracy : 0.4362
                 95% CI : (0.4285, 0.444)
    No Information Rate : 0.1971
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3502
 Mcnemar's Test P-Value : NA
```



The accuracy was only 43%. Clearly this was to be expected since the Predictions were almost exclusively Chinese, Indian, Italian, Mexican, Southern US and Thai. But the total number of cuisines is 20. Therefore, it might be useful to explore the ingredients unique to the other cuisines.

Whereas when we classify with softmax/-xgboost, the results are far better. The accuracy was almost doubled. This is expected and obvious since the number of class labels selected for the xgboost model is 20. The accuracy of the xgboost was similarly calculated and the error rate was found to be only 19%. The algorithm was able to classify 81% of the cuisines accurately when compared to rpart which was only 43%.

References:

1. Dataset: www.kaggle.com
2. An Introduction to XGBoost R package - http://www.r-bloggers.com/an-introduction-to-xgboost-r-package/
3. Practical Machine Learning Project - https://rpubs.com/flyingdisc/practical-machine-learning-xgboost
4. XGBoost R Tutorial - http://xgboost.readthedocs.io/en/latest/R-package/xgboostPresentation.html
5. An Introduction to Recursive Partitioning Using the RPART Routines - Terry M. Therneau Elizabeth J. Atkinson Mayo Foundation
6. R language for programmers – John D. Cook