Loading data

```
library(jsonlite) ## to load our dataset in json format
library(rpart)    ## to build basic CART model
library(Matrix)
library(tm)       ## basic text mining package for pre-processing data

## loading data
data1 <- fromJSON("train.json")
```

Pre-Processing Data

```
## pre-processing data
corpus_data1= Corpus(VectorSource(data1$ingredients))

## convert text to lowercase
corpus_data1= tm_map(corpus_data1, content_transformer(tolower))

## remove punctuation
corpus_data1= tm_map(corpus_data1, content_transformer(removePunctuation))

## remove whitespaces
corpus_data1= tm_map(corpus_data1, content_transformer(stripWhitespace))

## remove stop words
corpus_data1= tm_map(corpus_data1, removeWords, stopwords("english"))

## stemming
corpus_data1= tm_map(corpus_data1, content_transformer(stemDocument))
```

```
## document term matrix and remove sparse terms

maxdocfreq_data1= length(corpus_data1)


mindocfreq_data1= length(corpus_data1)*0.0001


## converting our data into document term matrix so as to get frequency of
each ingredient in each row. As there will be several ingredients whose fr
equency in the entire data set will be very low, they are not going to imp
act accuracy of our model, so we remove them by using control in the Docum
entTermMatrix and put a threshold on the sum of the frequencies of ingredi
ent in the entire dataset. As we can see below, we applied a threshold of
4 appearance in the dataset and this number was achieved with hit and tria
l method using the command above.

dtm_data1= DocumentTermMatrix(corpus_data1, control = list(bounds = list(g
```

```
lobal = c(mindocfreq_data1, maxdocfreq_data1), weighting= weightTfIdf)))


freq= sort(colSums(as.matrix(dtm_data1)), decreasing= TRUE)


tail(freq)

##     cherdez      cadobo      slider cwatercress     sashimi     starter
##           4           4           4           4           4           4
```

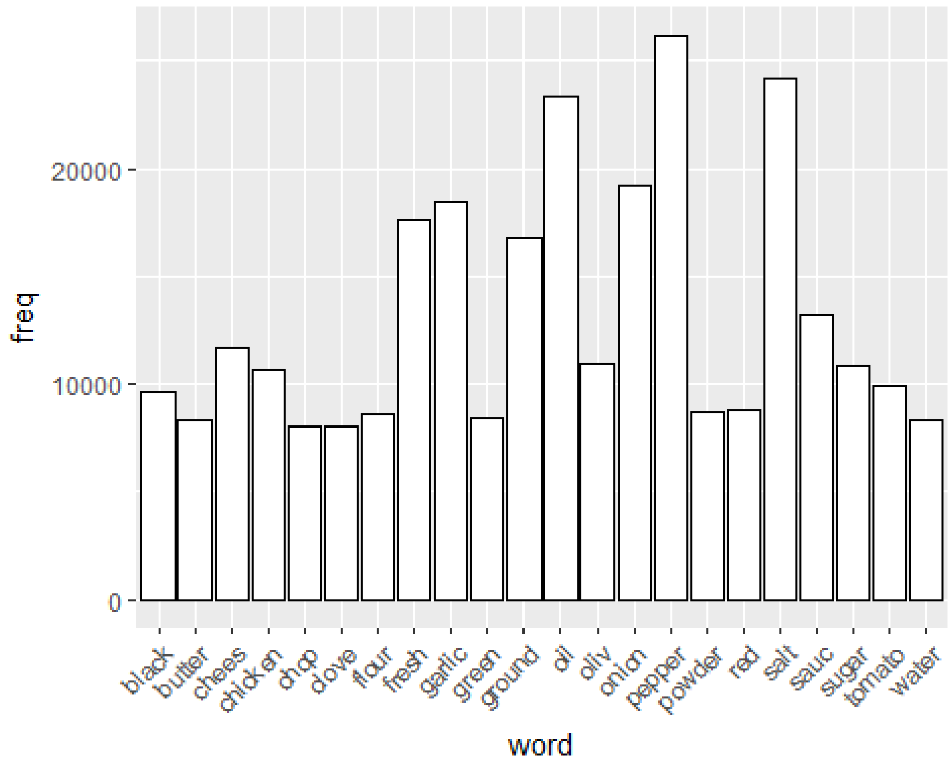
```
## converting into data frame
```

```
dtm2_data1= as.data.frame(as.matrix(dtm_data1))
```

```
dtm2_data1$cuisine= as.factor(data1$cuisine)
```

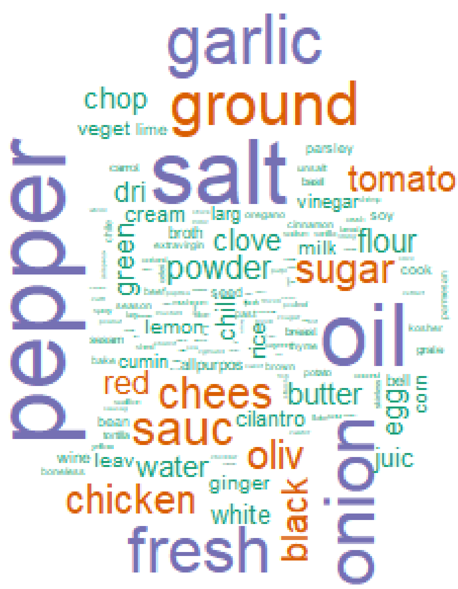
Exploratory Data Analysis

```
## data visualization
library(ggplot2)

## plotting ingredients with frequency greater than 8000

wf <- data.frame(word = names(freq), freq = freq)
head(wf)

##          word  freq
## pepper pepper 26189
## salt     salt 24177
## oil       oil 23303
## onion   onion 19208
## garlic garlic 18504
## fresh   fresh 17614

chart <- ggplot(subset(wf, freq >8000), aes(x = word, y = freq))
chart <- chart + geom_bar(stat = 'identity', color = 'black', fill = 'whit
e')
chart <- chart + theme(axis.text.x=element_text(angle=45, hjust=1))
chart
```



From the above graph we can see that, pepper, onion, ground, garlic, fresh, salt are some of the most frequent terms and we plot the frequencies of ingredients appearing more than 1000 in the form of a wordcloud below.

```
library(wordcloud)

wordcloud(names(freq), freq, min.freq = 1000, scale = c(3, 0.00005), color
s=brewer.pal(1, "Dark2"))
```



Sampling of Data and Splitting into Training & Testing Data Sets

While Cross-Validation is usually preferred over Validation Set Technique, due to the size of the dataset, the execution of cross-validation was found to be impractical. To circumvent this difficulty, we split the dataset into training data and test data in five different ways (sizes), then build the models on the training dataset and tested the model in the held-out data set.

The sizes of the datasets are as per table below.

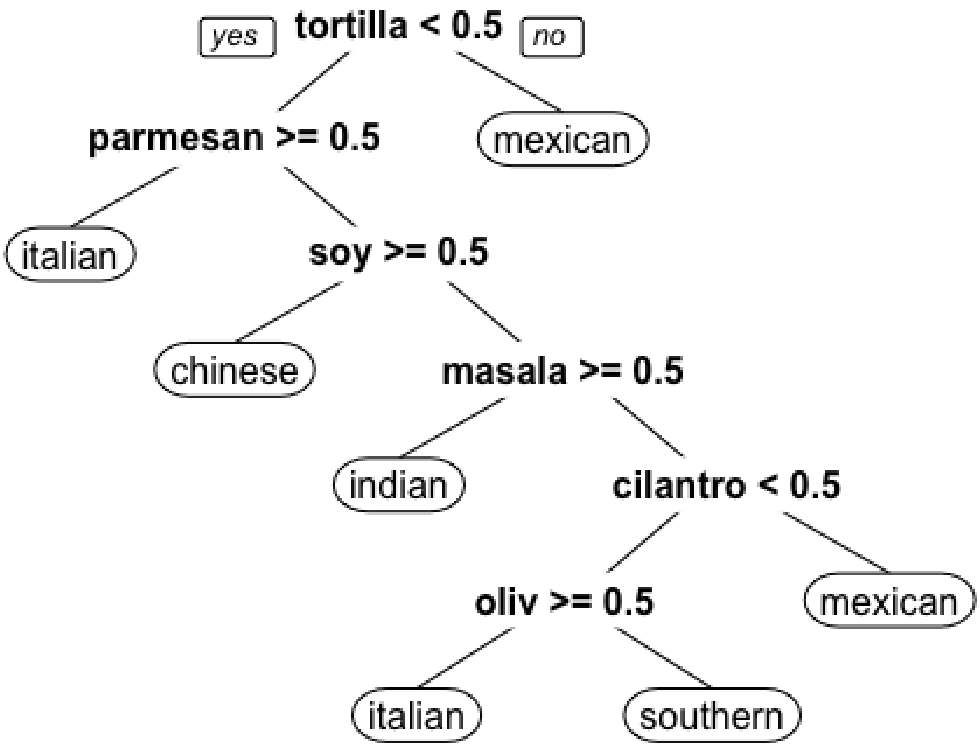
Training Dataset Size	Test Dataset Size
4000	35774
8000	31774
12000	27774
18000	21774
25000	14774

CART Model

We chose to first build a CART Model because of the ease of interpretability, and in order to better visualize the data. Classification Decision trees use a top-down greedy approach to predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs [5]. We also tested the CART model on the Test Dataset and obtained the resulting confusion matrix.

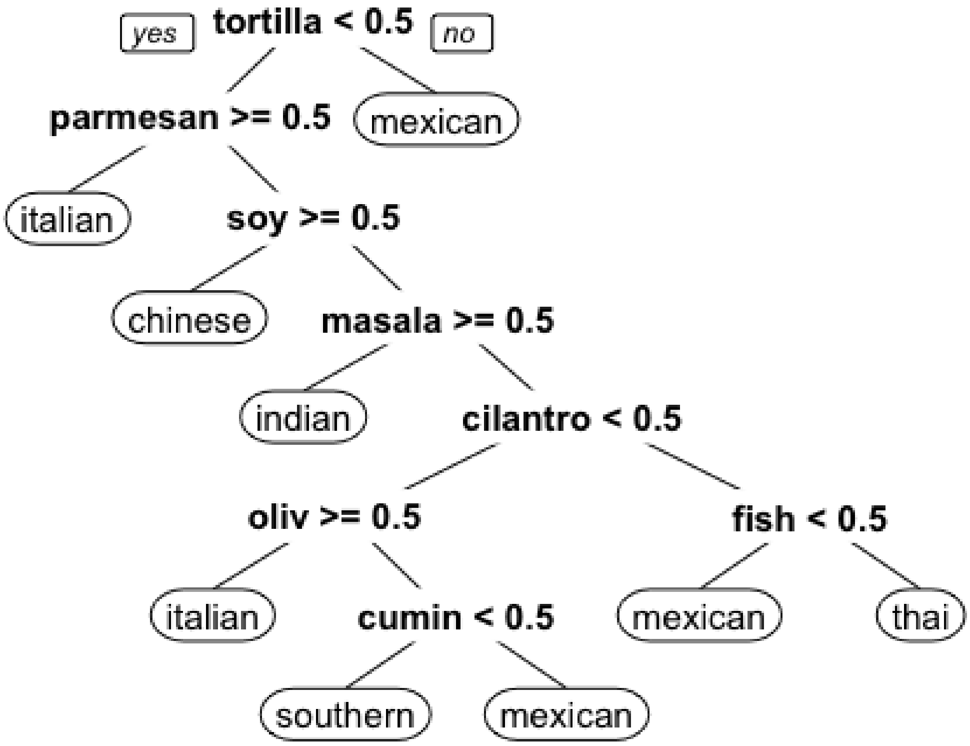
```r
# splitting data : 5 different sample sizes ~[4k,8k,12K,18K,25k] split int
o test and train data
set.seed(1)
train_index1=sample(1:nrow(dtm2_data1), 4000)
train1= dtm2_data1[train_index1,]
test1 = dtm2_data1[-train_index1,]

train_index2=sample(1:nrow(dtm2_data1), 8000)
train2= dtm2_data1[train_index2,]
test2 = dtm2_data1[-train_index2,]

train_index3=sample(1:nrow(dtm2_data1),12000)
train3= dtm2_data1[train_index3,]
test3 = dtm2_data1[-train_index3,]

train_index4=sample(1:nrow(dtm2_data1), 18000)
train4= dtm2_data1[train_index4,]
test4 = dtm2_data1[-train_index4,]

train_index5=sample(1:nrow(dtm2_data1),25000)
train5= dtm2_data1[train_index5,]
test5 = dtm2_data1[-train_index5,]
train_vec = c(train1,train2,train3,train4,train5)
test_vec = c(test1,test2,test3,test4,test5)
```

```
## CART model
#Assigning test and train data


library(rpart)
library(rpart.plot)

## model 1


train<-train1
test<-test1

treefit= rpart(cuisine~., data= train, method='class')
prp(treefit)
```

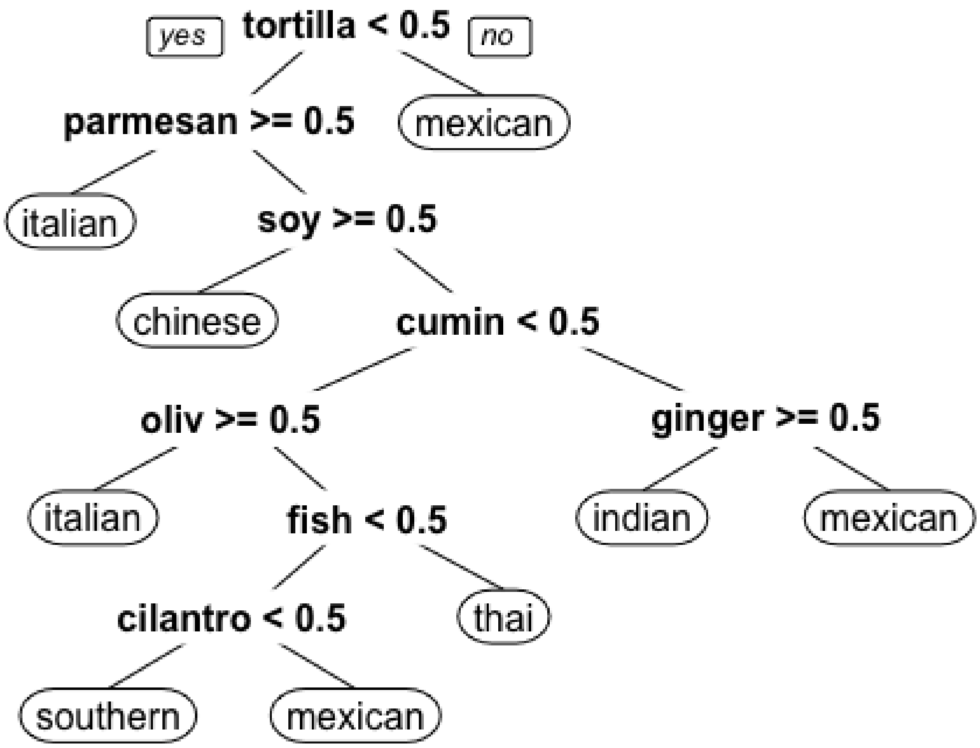



```
prob.tree= predict(treefit, newdata = test, method= 'class')
pred.tree1= colnames(prob.tree)[max.col(prob.tree)]
mean(pred.tree1==test$cuisine)

## [1] 0.3924918

## model 2

train<-train2
test<-test2
```

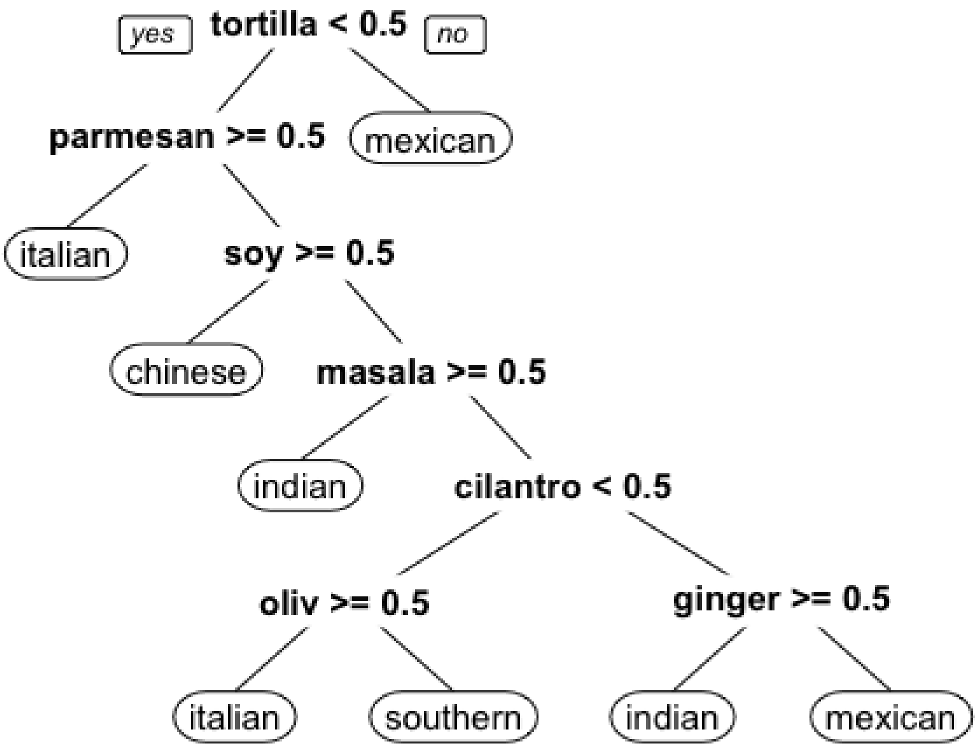
```
treefit= rpart(cuisine~., data= train, method='class')
prp(treefit)
```



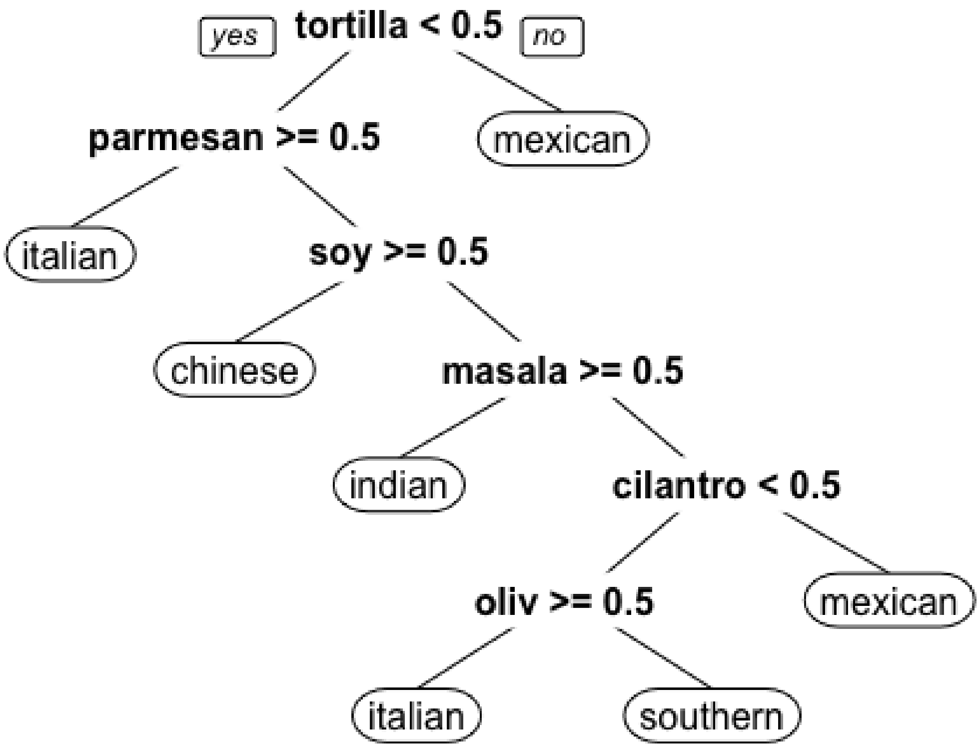

```
prob.tree= predict(treefit, newdata = test, method= 'class')
pred.tree1= colnames(prob.tree)[max.col(prob.tree)]
mean(pred.tree1==test$cuisine)

## [1] 0.4104299

## model 3

train<-train3
test<-test3

treefit= rpart(cuisine~., data= train, method='class')
prp(treefit)
```




```
prob.tree= predict(treefit, newdata = test, method= 'class')
pred.tree1= colnames(prob.tree)[max.col(prob.tree)]
mean(pred.tree1==test$cuisine)

## [1] 0.411572

## model 4

train<-train4
test<-test4

treefit= rpart(cuisine~., data= train, method='class')
prp(treefit)
```




```
prob.tree= predict(treefit, newdata = test, method= 'class')
pred.tree1= colnames(prob.tree)[max.col(prob.tree)]
mean(pred.tree1==test$cuisine)

## [1] 0.4005236

## model 5

train<-train5
test<-test5

treefit= rpart(cuisine~., data= train, method='class')
prp(treefit)
```




```
prob.tree= predict(treefit, newdata = test, method= 'class')
pred.tree1= colnames(prob.tree)[max.col(prob.tree)]
mean(pred.tree1==test$cuisine)

## [1] 0.3969135
```

Additional Interpretation and Explanation:

In this code, we test a simple CART model on our data using a vector of training sample sizes ranging from 1000 recipes to 25000 recipes. After analysing the results, it appears this model is not up to our expectations in terms of accuracy, but it is simple to interpret and explain. If looking at the models above, each one gives simple split specifications to determine which cuisine is most likely given whether or not it has a given ingredient. Based on the sample sizes used, we conclude that increasing the sample size past 12000 rows (recipes) does not increase our accuracy, which reaches a maximum of around 41 %.

k-Nearest Neighbours

```
##knn
train<-train1
test<-test1

train.x= train[,!colnames(train) %in% c('cuisine')]
test.x= test[,!colnames(test) %in% c('cuisine')]
knn.pred= knn(train.x, test.x, train$cuisine, k=100) ## try with different
k values
mean(knn.pred==test$cuisine)

## [1] 0.485

train<-train2
test<-test2

train.x= train[,!colnames(train) %in% c('cuisine')]
test.x= test[,!colnames(test) %in% c('cuisine')]
knn.pred= knn(train.x, test.x, train$cuisine, k=100) ## try with different
k values
mean(knn.pred==test$cuisine)

## [1] 0.4985

train<-train3
test<-test3

train.x= train[,!colnames(train) %in% c('cuisine')]
test.x= test[,!colnames(test) %in% c('cuisine')]
knn.pred= knn(train.x, test.x, train$cuisine, k=100) ## try with different
k values
mean(knn.pred==test$cuisine)

## [1] 0.542
```

Additional Interpretation and Explanation:

In this code, we use k nearest neighbours to classify the cuisines. We test KNN using three different training data sets and then three different K values for the most accurate training sample (which was the largest one). When trying larger data sets than those used above, running time became an issue. In this model, accuracy was once again below our expectations, although not entirely disappointing considering there are 20 different cuisines to choose from and the optimal model correctly classified over 50% of the time.

Support Vector Machines

```
set.seed(1)
train_index1=sample(1:nrow(dtm2_data1), 4000)
train1= dtm2_data1[train_index1,]
test1 = dtm2_data1[-train_index1,]

train_index2=sample(1:nrow(dtm2_data1), 8000)
train2= dtm2_data1[train_index2,]
test2 = dtm2_data1[-train_index2,]

train_index3=sample(1:nrow(dtm2_data1),12000)
train3= dtm2_data1[train_index3,]
test3 = dtm2_data1[-train_index3,]

train_index4=sample(1:nrow(dtm2_data1), 18000)
train4= dtm2_data1[train_index4,]
test4 = dtm2_data1[-train_index4,]

train_index5=sample(1:nrow(dtm2_data1),25000)
train5= dtm2_data1[train_index5,]
test5 = dtm2_data1[-train_index5,]


library(e1071)

## cross validation

svmfit= tune(svm,cuisine~., data= train2, kernel= 'radial', ranges=list(co
st=c(100,200,300,400,500)))

svm_best=svmfit$best.model
summary(svmfit)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   200
##
## - best performance: 0.28825
##
## - Detailed performance results:
##   cost    error  dispersion
## 1  100 0.292000 0.017482134
## 2  200 0.288250 0.009632122
## 3  300 0.289750 0.011752659
## 4  400 0.294750 0.013954290
## 5  500 0.298625 0.015925717
```

```r
pred.svm= predict(svm_best, test2)
mean(pred.svm==test2$cuisine)

## [1] 0.7156165

## from cross-validation, we got best cost as 200, so will be using

cost=200 in every svm model.


## model 1
train<-train1
test<-test1

svmfit= svm(cuisine~., data= train, kernel= 'radial', cost=200)

pred.svm= predict(svmfit, test)
mean(pred.svm==test$cuisine)

## [1] 0.6811371

## model 2

train<-train2
test<-test2

svmfit= svm(cuisine~., data= train, kernel= 'radial', cost=200)

pred.svm= predict(svmfit, test)
mean(pred.svm==test$cuisine)

## [1] 0.7153648

## model 3

train<-train3
test<-test3

svmfit= svm(cuisine~., data= train, kernel= 'radial', cost=200)

pred.svm= predict(svmfit, test)
mean(pred.svm==test$cuisine)

## [1] 0.7343559

## model 4

train<-train4
test<-test4

svmfit= svm(cuisine~., data= train, kernel= 'radial', cost=200)

pred.svm= predict(svmfit, test)
mean(pred.svm==test$cuisine)

## [1] 0.7493341
```

```r
## model 5

train<-train5
test<-test5

svmfit= svm(cuisine~., data= train, kernel= 'radial', cost=200)

pred.svm= predict(svmfit, test)
mean(pred.svm==test$cuisine)

## [1] 0.7647895

summary(pred.svm)

##    brazilian      british cajun_creole      chinese     filipino
##          126          179          495         1053          233
##       french        greek       indian        irish      italian
##         1035          392         1137          197         3204
##     jamaican     japanese       korean      mexican     moroccan
##          136          417          276         2536          242
##      russian  southern_us      spanish         thai   vietnamese
##          118         1988          250          548          212

## model 6

svmfit= svm(cuisine~., data= train, kernel= 'linear', cost=200)

pred.svm= predict(svmfit, test)
mean(pred.svm==test$cuisine)

## [1] 0.7109787

summary(pred.svm)

##    brazilian      british cajun_creole      chinese     filipino
##          166          263          517         1007          316
##       french        greek       indian        irish      italian
##          864          502         1107          323         2803
##     jamaican     japanese       korean      mexican     moroccan
##          189          504          295         2491          298
##      russian  southern_us      spanish         thai   vietnamese
##          191         1815          312          562          249

## model 7

svmfit= svm(cuisine~., data= train, kernel= 'radial', cost=100)

pred.svm= predict(svmfit, test)
mean(pred.svm==test$cuisine)

## [1] 0.7603222

summary(pred.svm)

##    brazilian      british cajun_creole      chinese     filipino
##          117          151          480         1067          203
##       french        greek       indian        irish      italian
```

```
##         1015         375        1127         180        3339
##     jamaican    japanese      korean     mexican    moroccan
##          134         401         266        2540         236
##      russian southern_us     spanish        thai  vietnamese
##           86        2083         216         553         205
```

```
## model 8
```

```
svmfit= svm(cuisine~., data= train, kernel= 'radial', cost=300)
```

```
pred.svm= predict(svmfit, test)
mean(pred.svm==test$cuisine)
```

```
## [1] 0.7628943
```

```
summary(pred.svm)
```

```
##    brazilian     british cajun_creole     chinese    filipino
##          122         194          506        1047         244
##       french       greek       indian       irish     italian
##         1010         400         1136         219        3158
##     jamaican    japanese      korean     mexican    moroccan
##          145         430         277        2523         240
##      russian southern_us     spanish        thai  vietnamese
##          125        1972         259         552         215
```

```
#Actual Distribution
```

```
summary(test$cuisine)
```

```
##    brazilian     british cajun_creole     chinese    filipino
##          177         292          564         972         266
##       french       greek       indian       irish     italian
##         1006         447         1113         243        2876
##     jamaican    japanese      korean     mexican    moroccan
##          190         539         299        2473         276
##      russian southern_us     spanish        thai  vietnamese
##          184        1645         367         541         304
```

Additional Interpretation and Explanation:

In this code, we use a support vector machine to classify cuisines with 5 different training data sets. Using Cross Validation, we found the best cost to be 200, which we show in the final steps of the code. SVM increased in accuracy as we increased the training data set size until we reached an optimal point of around 76.5% accuracy with a training sample of 25000 cuisines. We then tested the effect of using a linear vs. radial model to show the superior accuracy of the radial model. The final tables show the distributions of cuisine determination as predicted by the models, while the actual test distribution of cuisines is included at the end to compare. Overall SVM is solid on accuracy, but somewhat difficult to interpret or explain due to its more confusing mathematical nature.

Neural Networks

```
set.seed(1)
train_index1=sample(1:nrow(dtm2_data1), 5000)
train<-dtm2_data1[train_index1,]
train1= train[1:4000,]
test1 = train[4001:5000,]

train_index2=sample(1:nrow(dtm2_data1), 10000)
train<-dtm2_data1[train_index2,]
train2= train[1:8000,]
test2 = train[8001:10000,]

train_index3=sample(1:nrow(dtm2_data1),15000)
train<-dtm2_data1[train_index3,]
train3= train[1:12000,]
test3 = train[12001:15000,]

train_index4=sample(1:nrow(dtm2_data1), 22500)
train<-dtm2_data1[train_index4,]
train4= train[1:18000,]
test4 = train[18001:22500,]

train_index5=sample(1:nrow(dtm2_data1),31250)
train<-dtm2_data1[train_index5,]
train5= dtm2_data1[1:25000,]
test5 = dtm2_data1[25001:31250,]


## model 1

train<-train1
test<-test1
library(nnet)


nnetfit= nnet(cuisine~., data=train, size=3, MaxNWts=10000)

## # weights:  6932
## initial  value 13305.328559
## iter  10 value 8401.495473
## iter  20 value 7034.720829
## iter  30 value 6370.348944
## iter  40 value 5927.162420
## iter  50 value 5523.139001
## iter  60 value 5190.217243
## iter  70 value 4920.401165
## iter  80 value 4707.362297
## iter  90 value 4500.470740
## iter 100 value 4328.375879
## final  value 4328.375879
## stopped after 100 iterations
```

```
prob.nnet= predict(nnetfit, test)
pred.nnet= colnames(prob.nnet)[max.col(prob.nnet)]
mean(pred.nnet==test$cuisine)

## [1] 0.451

## model 2

train<-train2
test<-test2

nnetfit= nnet(cuisine~., data=train, size=3, MaxNWts=10000)

## # weights:  6932
## initial  value 25694.608645
## iter  10 value 19935.478162
## iter  20 value 17898.081813
## iter  30 value 15523.995675
## iter  40 value 14378.826765
## iter  50 value 13236.320727
## iter  60 value 12357.792414
## iter  70 value 11794.192854
## iter  80 value 11277.172636
## iter  90 value 10792.610040
## iter 100 value 10483.562641
## final  value 10483.562641
## stopped after 100 iterations

prob.nnet= predict(nnetfit, test)
pred.nnet= colnames(prob.nnet)[max.col(prob.nnet)]
mean(pred.nnet==test$cuisine)

## [1] 0.4775

## model 3

train<-train3
test<-test3

nnetfit= nnet(cuisine~., data=train, size=3, MaxNWts=10000)

## # weights:  6932
## initial  value 37449.241410
## iter  10 value 26904.282061
## iter  20 value 22812.223449
## iter  30 value 20515.830617
## iter  40 value 18785.400819
## iter  50 value 17919.664665
## iter  60 value 16783.257690
## iter  70 value 16060.097788
## iter  80 value 15564.689214
## iter  90 value 15191.736202
## iter 100 value 14924.131053
## final  value 14924.131053
## stopped after 100 iterations
```

```r
prob.nnet= predict(nnetfit, test)
pred.nnet= colnames(prob.nnet)[max.col(prob.nnet)]
mean(pred.nnet==test$cuisine)

## [1] 0.539

## model 4

train<-train4
test<-test4

nnetfit= nnet(cuisine~., data=train, size=3, MaxNWts=10000)

## # weights:  6932
## initial  value 57473.336712
## iter  10 value 39172.488185
## iter  20 value 31960.874470
## iter  30 value 29399.664435
## iter  40 value 26399.448044
## iter  50 value 24626.901401
## iter  60 value 23516.219730
## iter  70 value 22736.558964
## iter  80 value 22126.282690
## iter  90 value 21666.999903
## iter 100 value 21294.226789
## final  value 21294.226789
## stopped after 100 iterations

prob.nnet= predict(nnetfit, test)
pred.nnet= colnames(prob.nnet)[max.col(prob.nnet)]
mean(pred.nnet==test$cuisine)

## [1] 0.5673333

## model 5

train<-train5
test<-test5

nnetfit= nnet(cuisine~., data=train, size=3, MaxNWts=10000)

## # weights:  6932
## initial  value 84534.612544
## iter  10 value 53426.855948
## iter  20 value 45648.817131
## iter  30 value 42712.324028
## iter  40 value 40798.286541
## iter  50 value 38557.142441
## iter  60 value 37083.177626
## iter  70 value 35997.426188
## iter  80 value 34952.007354
## iter  90 value 34048.748111
## iter 100 value 33185.447935
## final  value 33185.447935
## stopped after 100 iterations
```

```r
prob.nnet= predict(nnetfit, test)
pred.nnet= colnames(prob.nnet)[max.col(prob.nnet)]
mean(pred.nnet==test$cuisine)

## [1] 0.55744
```

Additional Interpretation and Explanation:

In this code, we use an artificial neural network to classify the cuisines. First, we tested some of the inputs to test their effect on accuracy. We altered the number of iterations, number of hidden layers, and the maximum number of weights to test how it affected our accuracy. We determined that 100 iterations were satisfactory to get close enough to the maximum accuracy for a given data set, since the change in accuracy was insignificant when increasing the max number of iterations to 500. We also found that 3 hidden layers was the optimal value since anything less hurt accuracy for a given number of iterations, while anything more had little to no effect. The maximum accuracy result appeared with 18000 cuisines in the training set and amounted to about 56% accuracy. This result was better than CART and KNN, but still not quite up to our expectations.

Extreme Gradient Boosting

```
# splitting data : 5 different sample sizes ~[4k,8k,12K,18K,25k] split
into test and train data
set.seed(1)
train_index1=sample(1:nrow(dtm2_data1), 4000)
train1= dtm2_data1[train_index1,]
test1 = dtm2_data1[-train_index1,]

train_index2=sample(1:nrow(dtm2_data1), 8000)
train2= dtm2_data1[train_index2,]
test2 = dtm2_data1[-train_index2,]

train_index3=sample(1:nrow(dtm2_data1),12000)
train3= dtm2_data1[train_index3,]
test3 = dtm2_data1[-train_index3,]

train_index4=sample(1:nrow(dtm2_data1), 18000)
train4= dtm2_data1[train_index4,]
test4 = dtm2_data1[-train_index4,]

train_index5=sample(1:nrow(dtm2_data1),25000)
train5= dtm2_data1[train_index5,]
test5 = dtm2_data1[-train_index5,]

library(xgboost)
```

Data size- 4000 rows and learning rate=0.1

```
xgtrain_4000 <- xgb.DMatrix((data.matrix(train1[,!colnames(train1) %in% c(
'cuisine')])), label = as.numeric(train1$cuisine)-1)

xgbmodel <- xgboost(data = xgtrain_4000, max.depth = 25, eta = 0.1, nround
= 75, objective = "multi:softmax", num_class = 20, verbose = 1)

## [1]  train-merror:0.297500
## [2]  train-merror:0.263500
## [3]  train-merror:0.238750
## [4]  train-merror:0.224000
## [5]  train-merror:0.204250
## [6]  train-merror:0.193750
## [7]  train-merror:0.183250
## [8]  train-merror:0.169500
## [9]  train-merror:0.159500
## [10] train-merror:0.148750
## [11] train-merror:0.137000
## [12] train-merror:0.129750
## [13] train-merror:0.123000
```

```
## [14] train-merror:0.113500
## [15] train-merror:0.106500
## [16] train-merror:0.100000
## [17] train-merror:0.094250
## [18] train-merror:0.090000
## [19] train-merror:0.084250
## [20] train-merror:0.080000
## [21] train-merror:0.075250
## [22] train-merror:0.071000
## [23] train-merror:0.067750
## [24] train-merror:0.064000
## [25] train-merror:0.061250
## [26] train-merror:0.057500
## [27] train-merror:0.054000
## [28] train-merror:0.048750
## [29] train-merror:0.045750
## [30] train-merror:0.040500
## [31] train-merror:0.036750
## [32] train-merror:0.033750
## [33] train-merror:0.031000
## [34] train-merror:0.027750
## [35] train-merror:0.026000
## [36] train-merror:0.025000
## [37] train-merror:0.024000
## [38] train-merror:0.022500
## [39] train-merror:0.020750
## [40] train-merror:0.018500
## [41] train-merror:0.016750
## [42] train-merror:0.014500
## [43] train-merror:0.013500
## [44] train-merror:0.012500
## [45] train-merror:0.011750
## [46] train-merror:0.011500
## [47] train-merror:0.010500
## [48] train-merror:0.010250
## [49] train-merror:0.008750
## [50] train-merror:0.008250
## [51] train-merror:0.008250
## [52] train-merror:0.006750
## [53] train-merror:0.006000
## [54] train-merror:0.005500
## [55] train-merror:0.005500
## [56] train-merror:0.005000
## [57] train-merror:0.004250
## [58] train-merror:0.004250
## [59] train-merror:0.004000
## [60] train-merror:0.004000
## [61] train-merror:0.003750
## [62] train-merror:0.003750
## [63] train-merror:0.003750
## [64] train-merror:0.003000
## [65] train-merror:0.002750
## [66] train-merror:0.002500
## [67] train-merror:0.002500
```

```
## [68] train-merror:0.002250
## [69] train-merror:0.002250
## [70] train-merror:0.002000
## [71] train-merror:0.002000
## [72] train-merror:0.002000
## [73] train-merror:0.002000
## [74] train-merror:0.001750
## [75] train-merror:0.001750
```

```
xgbmodel.predict <- predict(xgbmodel, newdata = data.matrix(test1[, !colna
mes(test1) %in% c('cuisine')]))
xgbmodel.predict.text <- levels(train1$cuisine)[xgbmodel.predict + 1]
mean(xgbmodel.predict.text==test1$cuisine)
```

```
## [1] 0.684799
```

Data size= 8000 rows

```
xgtrain_8000 <- xgb.DMatrix((data.matrix(train2[,!colnames(train2) %in% c(
'cuisine')])), label = as.numeric(train2$cuisine)-1)

xgbmodel <- xgboost(data = xgtrain_8000, max.depth = 25, eta = 0.1, nround
= 75, objective = "multi:softmax", num_class = 20, verbose = 1)
```

```
## [1]  train-merror:0.277500
## [2]  train-merror:0.244250
```

..

..

```
## [74] train-merror:0.004250
## [75] train-merror:0.004000
```

```
xgbmodel.predict <- predict(xgbmodel, newdata = data.matrix(test2[, !colna
mes(test2) %in% c('cuisine')]))
xgbmodel.predict.text <- levels(train2$cuisine)[xgbmodel.predict + 1]
mean(xgbmodel.predict.text==test2$cuisine)
```

```
## [1] 0.7181029
```

Data size= 12000 rows

```
xgtrain_12000 <- xgb.DMatrix((data.matrix(train3[,!colnames(train3) %in% c
('cuisine')])), label = as.numeric(train3$cuisine)-1)
```

```
xgbmodel <- xgboost(data = xgtrain_12000, max.depth = 25, eta = 0.1, nround = 75, objective = "multi:softmax", num_class = 20, verbose = 1)

## [1]  train-merror:0.267583
## [2]  train-merror:0.240917

...

...
```

```
## [74] train-merror:0.008000
## [75] train-merror:0.007917

xgbmodel.predict <- predict(xgbmodel, newdata = data.matrix(test3[, !colnames(test3) %in% c('cuisine')]))
xgbmodel.predict.text <- levels(train3$cuisine)[xgbmodel.predict + 1]
mean(xgbmodel.predict.text==test3$cuisine)

## [1] 0.7404407
```

Data size= 18000 rows

```
xgtrain_18000 <- xgb.DMatrix((data.matrix(train4[,!colnames(train4) %in% c('cuisine')])), label = as.numeric(train4$cuisine)-1)

xgbmodel <- xgboost(data = xgtrain_18000, max.depth = 25, eta = 0.1, nround = 75, objective = "multi:softmax", num_class = 20, verbose = 1)
```

```
## [1]  train-merror:0.250833
## [2]  train-merror:0.227000
```

...

...

```
## [74] train-merror:0.012333
## [75] train-merror:0.011889

xgbmodel.predict <- predict(xgbmodel, newdata = data.matrix(test4[, !colnames(test4) %in% c('cuisine')]))
xgbmodel.predict.text <- levels(train4$cuisine)[xgbmodel.predict + 1]
mean(xgbmodel.predict.text==test4$cuisine)

## [1] 0.7553504
```

Data size= 25000 rows

```
xgtrain_25000 <- xgb.DMatrix((data.matrix(train5[,!colnames(train5) %in% c('cuisine')])), label = as.numeric(train5$cuisine)-1)
```

```
xgbmodel <- xgboost(data = xgtrain_25000, max.depth = 25, eta = 0.1, nroun
d = 75, objective = "multi:softmax", num_class = 20, verbose = 1)
```

```
## [1]  train-merror:0.248000
## [2]  train-merror:0.226760
```

…

…

```
## [74] train-merror:0.016760
## [75] train-merror:0.016400

xgbmodel.predict <- predict(xgbmodel, newdata = data.matrix(test5[, !colna
mes(test5) %in% c('cuisine')]))
xgbmodel.predict.text <- levels(train5$cuisine)[xgbmodel.predict + 1]
mean(xgbmodel.predict.text==test5$cuisine)

## [1] 0.7694599

## we got the best accuracy till now with data size= 25000. So, we will tr
y to change parameters in this model like learning rate and number of iter
ations, etc to see if we get better accuracy than now.
```

Data size= 25000 rows with learning rate=0.01

```
xgtrain_25000 <- xgb.DMatrix((data.matrix(train5[,!colnames(train5) %in%
c('cuisine')])), label = as.numeric(train5$cuisine)-1)
```

```
xgbmodel <- xgboost(data = xgtrain_25000, max.depth = 25, eta = 0.01,
nround = 75, objective = "multi:softmax", num_class = 20, verbose = 1)
```

```
## [1]  train-merror:0.248000
## [2]  train-merror:0.242400
```

…

…

```
## [74] train-merror:0.159960
## [75] train-merror:0.159520

xgbmodel.predict <- predict(xgbmodel, newdata = data.matrix(test5[, !colna
mes(test5) %in% c('cuisine')]))
xgbmodel.predict.text <- levels(train5$cuisine)[xgbmodel.predict + 1]
mean(xgbmodel.predict.text==test5$cuisine)

## [1] 0.725396 ## accuracy reduced
```

Data size= 25000 rows with learning rate=0.5

```
xgtrain_25000 <- xgb.DMatrix((data.matrix(train5[,!colnames(train5) %in% c
('cuisine')])), label = as.numeric(train5$cuisine)-1)
xgbmodel <- xgboost(data = xgtrain_25000, max.depth = 25, eta = 0.5, nroun
d = 100, objective = "multi:softmax", num_class = 20)

## [1]  train-merror:0.248000
## [2]  train-merror:0.197400

…
…

## [99] train-merror:0.000200
## [100]    train-merror:0.000200

xgbmodel.predict <- predict(xgbmodel, newdata = data.matrix(test5[, !colna
mes(test5) %in% c('cuisine')]))
xgbmodel.predict.text <- levels(train5$cuisine)[xgbmodel.predict + 1]
mean(xgbmodel.predict.text==test5$cuisine)

## [1] 0.7739949
```

Data size= 25000 rows with learning rate=0.3

```
xgtrain_25000 <- xgb.DMatrix((data.matrix(train5[,!colnames(train5) %in% c
('cuisine')])), label = as.numeric(train5$cuisine)-1)

xgbmodel <- xgboost(data = xgtrain_25000, max.depth = 25, eta = 0.3, nroun
d = 75, objective = "multi:softmax", num_class = 20, verbose = 1)

## [1]  train-merror:0.248000
## [2]  train-merror:0.203280
## [3]  train-merror:0.173960
## [4]  train-merror:0.150960
## [5]  train-merror:0.131400
## [6]  train-merror:0.114000
## [7]  train-merror:0.100040
## [8]  train-merror:0.087760
## [9]  train-merror:0.077200
## [10] train-merror:0.067600
## [11] train-merror:0.061280
## [12] train-merror:0.054360
## [13] train-merror:0.048480
## [14] train-merror:0.043640
## [15] train-merror:0.039400
## [16] train-merror:0.035600
## [17] train-merror:0.032240
## [18] train-merror:0.029400
## [19] train-merror:0.026120
## [20] train-merror:0.023560
## [21] train-merror:0.021320
```

```
## [22] train-merror:0.019640
## [23] train-merror:0.018280
## [24] train-merror:0.017080
## [25] train-merror:0.016000
## [26] train-merror:0.015000
## [27] train-merror:0.013800
## [28] train-merror:0.013200
## [29] train-merror:0.012400
## [30] train-merror:0.011880
## [31] train-merror:0.011120
## [32] train-merror:0.010440
## [33] train-merror:0.009400
## [34] train-merror:0.008960
## [35] train-merror:0.008520
## [36] train-merror:0.008200
## [37] train-merror:0.007640
## [38] train-merror:0.007000
## [39] train-merror:0.006480
## [40] train-merror:0.006120
## [41] train-merror:0.006040
## [42] train-merror:0.005880
## [43] train-merror:0.005760
## [44] train-merror:0.005520
## [45] train-merror:0.005160
## [46] train-merror:0.005120
## [47] train-merror:0.004720
## [48] train-merror:0.004520
## [49] train-merror:0.004240
## [50] train-merror:0.003920
## [51] train-merror:0.003840
## [52] train-merror:0.003640
## [53] train-merror:0.003440
## [54] train-merror:0.003280
## [55] train-merror:0.003200
## [56] train-merror:0.003120
## [57] train-merror:0.002840
## [58] train-merror:0.002680
## [59] train-merror:0.002520
## [60] train-merror:0.002200
## [61] train-merror:0.002080
## [62] train-merror:0.001960
## [63] train-merror:0.001800
## [64] train-merror:0.001840
## [65] train-merror:0.001760
## [66] train-merror:0.001600
## [67] train-merror:0.001600
## [68] train-merror:0.001400
## [69] train-merror:0.001320
## [70] train-merror:0.001320
## [71] train-merror:0.001280
## [72] train-merror:0.001160
## [73] train-merror:0.001160
## [74] train-merror:0.001120
## [75] train-merror:0.001080
```

```r
xgbmodel.predict <- predict(xgbmodel, newdata = data.matrix(test5[, !colna
mes(test5) %in% c('cuisine')]))
xgbmodel.predict.text <- levels(train5$cuisine)[xgbmodel.predict + 1]
mean(xgbmodel.predict.text==test5$cuisine)

## [1] 0.7733857
```

Data size= 25000 rows and # iterations= 150 (best model)

```r
xgtrain_25000 <- xgb.DMatrix((data.matrix(train5[,!colnames(train5) %in% c
('cuisine')])), label = as.numeric(train5$cuisine)-1)

xgbmodel <- xgboost(data = xgtrain_25000, max.depth = 25, eta = 0.3, nroun
d = 150, objective = "multi:softmax", num_class = 20, verbose = 1)

## [1]  train-merror:0.248000
## [2]  train-merror:0.203280

..
..

## [149]    train-merror:0.000240
## [150]    train-merror:0.000240

xgbmodel.predict <- predict(xgbmodel, newdata = data.matrix(test5[, !colna
mes(test5) %in% c('cuisine')]))
xgbmodel.predict.text <- levels(train5$cuisine)[xgbmodel.predict + 1]
mean(xgbmodel.predict.text==test5$cuisine)

## [1] 0.7767023 ## best accuracy when we increased the number of iteratio
ns.
```

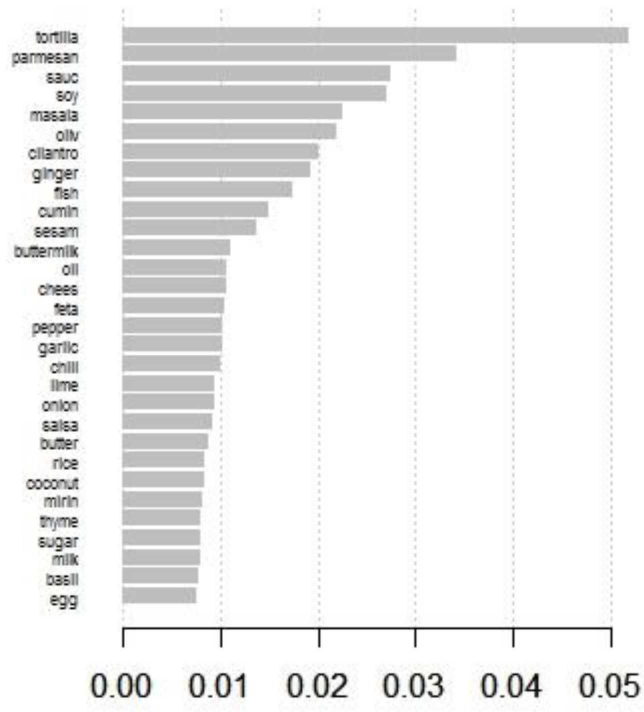
Data size= 25000 rows with different parameters

```r
xgtrain_25000 <- xgb.DMatrix((data.matrix(train5[,!colnames(train5) %in% c
('cuisine')])), label = as.numeric(train5$cuisine)-1)
xgbmodel <- xgboost(data = xgtrain_25000, max.depth = 7, gamma = 2, min_ch
ild_weight = 2, eta = 0.1, nround = 75, objective = "multi:softmax", num_c
lass = 20, verbose = 2)

xgbmodel.predict <- predict(xgbmodel, newdata = data.matrix(test5[, !colna
mes(test5) %in% c('cuisine')]))
xgbmodel.predict.text <- levels(train5$cuisine)[xgbmodel.predict + 1]
mean(xgbmodel.predict.text==test5$cuisine)

## [1] 0.7509138
```

Interpretation: From the above 10 xgboost models, we interpret that with less number of training dataset, the testing accuracy is low, and as we increased the training size from 4000 to 25000, accuracy increased from 68.4 % to 76.7 %. So xgboost need a bigger dataset to predict accurately. Also, when the datasize was 25000 rows, when we increased the number of iterations from 75 to 150, prediction accuracy increased. So we interpret that xgboost needs to be trained for a longer time to increase the accuracy. Due to computational limitations and long training time, we were not able to do more iterations.

Plotting important features in our best model

```
names <- colnames(train5[, !colnames(train5) %in% c("cuisine")])

importance_matrix= xgb.importance(names, model=xgbmodel)
```



5. Comparison

Method	Training Dataset Size	Test Data Set	Parameters	Prediction Accuracy
CART	4000	35774		0.3924918
	8000	31774		0.4104299
	12000	27774		0.411572
	18000	21774		0.4005236
	25000	14774		0.3969135
KNN	4000	1000	K = 100	0.485
	8000	2000	K = 100	0.4985
	12000	3000	K = 100	0.542
NNet	4000	1000	Size = 3	0.451
	8000	2000	Size = 3	0.4775
	12000	3000	Size = 3	0.539
	18000	4500	Size = 3	0.5673333
	25000	6250	Size = 3	0.55744
SVM	4000	35774	Kernel = radial,cost= 200	0.6811371
	8000	31774	Kernel = radial, cost = 200	0.7153648
	12000	27774	Kernel = radial, cost = 200	0.7343559
	18000	21774	Kernel = radial, cost = 200	0.7493341
	25000	14774	Kernel = radial, cost = 200	0.7647895
SVM	25000	14774	Kernel = linear Cost = 200	0.7109787
XGBoost	4000	35774	Max depth = 25 Learning Rate = 0.1 Iterations = 75	0.684799
	8000	31774		0.7181029
	12000	27774		0.7404407
	18000	21774		0.7553504
	25000	14774		0.7694599
	25000	14774	Max depth = 25 Learning Rate = 0.01 Iterations = 75	0.725396
	25000	14774	Max depth = 25 Learning Rate = 0.5 Iterations = 75	0.7739949
	25000	14774	Max depth = 25 Learning Rate = 0.3 Iterations = 75	0.7733857
	25000	**14774**	**Max depth = 25 Learning Rate = 0.3 Iterations = 150**	**0.7767023**
	25000	14774	Max depth = 7 Gamma = 2 Min Child wt = 2 Learning Rate = 0.1	0.7509138

Conclusion

After testing a number of models, a few themes began to emerge that may describe the nature of our data set. KNN and CART were somewhat lower level models that both performed with average accuracy: ~50%, while Artificial Neural Networks didn't do much better at around 57%. It seems that the two lower level models were able to form classifications for larger groups, but weren't able to distinguish between similar cuisines. This is a key insight into the nature of this data set, because it shows that there are main groups of cuisines that may also contain subgroups that are missed with lower level models or models that can't distinguish between similar recipes. SVM and XGBoost, however, seemed to be able to at least somewhat conquer the challenge of similar data, reaching accuracy marks of nearly 80%. These models seemed to have more power to distinguish the individual differences between the sub groups of cuisines, however, there remains somewhat of a trade-off in terms of interpretability. The best models in this case are also somewhat difficult to interpret and explain, while the simpler models can give insightful information in the form of pictures or models. The original CART model was helpful in its ability to inform us of some of the most distinguishing ingredients for given cuisines, information we would not be able to see in the more complex models. For instance, the original depiction of the tree formed from the CART model showed the tortilla to be the most distinguishing ingredient in cuisines, which is intuitively pleasing when you think about the use of tortillas and how they can indicate what type of food you are likely eating. Overall CART models and grouping models can give us valuable insight into the data set, but accuracy can be much improved using more complex models such as XGBoost and SVM. In conclusion, XGBoost resulted in the best accuracy out of all the models at around 77.7% and that is the model that we would prefer to use in order to predict the type of cuisine given a sample recipe.