CAN YOU TELL WHAT I AM?

JUNE 2, 2018

MARIJA STANOJCIC
NetID: gk3964

Contents

Introduction

Classifying products into different cataegories is something that is needed for every retail store, wether it's only online shoping, store shopping, or both. Classifying products can be useful to decide which strategies should be used for promoting a particular product.

In this project Random Forest and XGBoost models will be used to classify Otto Group products. The dataset is from Kaggle Competition - Otto group.

About The Otto Group

Description from Kaggle:

"The Otto Group is one of the world's biggest e-commerce companies, with subsidiaries in more than 20 countries, including Crate & Barrel (USA), Otto.de (Germany) and 3 Suisses (France). We are selling millions of products worldwide every day, with several thousand products being added to our product line.

A consistent analysis of the performance of our products is crucial. However, due to our diverse global infrastructure, many identical products get classified differently. Therefore, the quality of our product analysis depends heavily on the ability to accurately cluster similar products. The better the classification, the more insights we can generate about our product range."

About data

The data for this project is train data (train.csv) from Otto Kaggle Competition. There are 61878 products from 9 categories and 93 features for describing products. Each row represents a single product. 93 features represent count of different events. Unfortunately, all features are obfuscated, so we won't know which feature represents what. Categories are 9 most important categories for Otto group, for example fashion, electronics, etc. Categories all also obfuscated, so we won't know which category is what.

Step 1 - getting the data

Data can be found here: https://www.kaggle.com/c/otto-group-product-classification-challenge/data.

Step 2 – exploring and preparing the data

The preview of the first few rows and columns.

```
##   id feat_1 feat_2 feat_3 feat_4 feat_5 feat_6 feat_7 feat_8 feat_9
## 1  1      1      0      0      0      0      0      0      0      0
## 2  2      0      0      0      0      0      0      0      1      0
## 3  3      0      0      0      0      0      0      0      1      0
## 4  4      1      0      0      1      6      1      5      0      0
## 5  5      0      0      0      0      0      0      0      0      0
## 6  6      2      1      0      0      7      0      0      0      0
```

Data has 61878 observations and 95 variables. There are no missing values.

As the data has an ID variable which is the perfect predictor, before running any model, ID variable first need to be removed from the data.

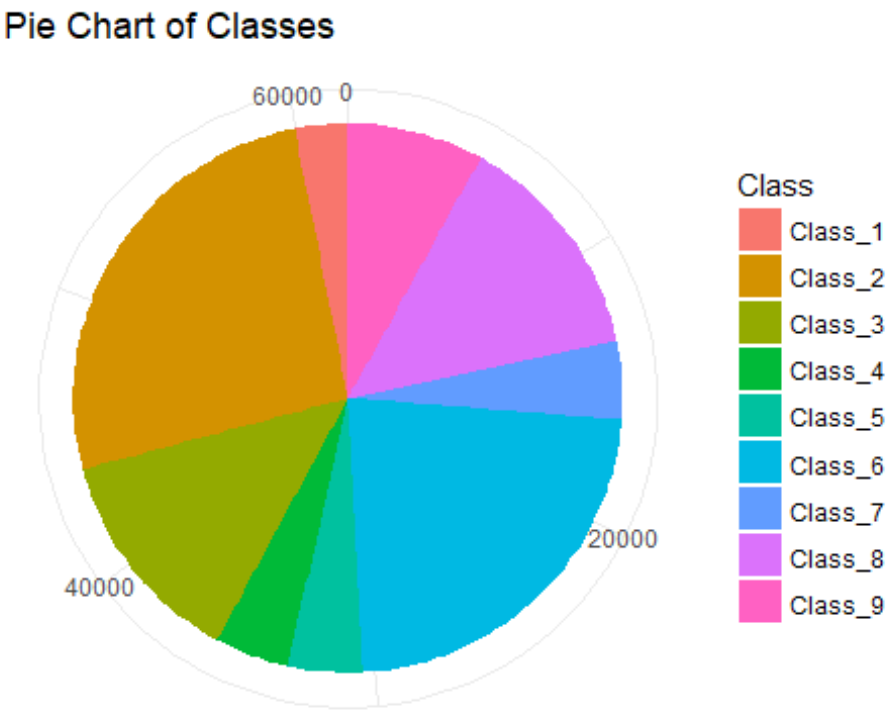
Target variable is variable that specifies from which category is the product. Categories are Class_1, Class_2, … , Class_9.

Closer look at the target variable.

```
## Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8 Class_9
##    1929   16122    8004    2691    2739   14135    2839    8464    4955
```

Proportion of classes in the target variable.

```
##         percentage
## Class_1       3.12
## Class_2      26.05
## Class_3      12.94
## Class_4       4.35
## Class_5       4.43
## Class_6      22.84
## Class_7       4.59
## Class_8      13.68
## Class_9       8.01
```



Otto Group has the most products in Class_2 and in Class_6, while the smallest number if products are for Class+1, Class_4, Class_5 and Class_7.

Data needs to be split into training and test set, test set will be used only for evaluating model performance, model will not see any information from test set. 85% of the data (52596 products) was used for training models and 15% (9282 products) for evaluating models.

For sample to be representative, both training and test set should have approximately the same proportion of classes in them as in the full data.

Checking if the sample is representative.

```
Train
```

```
##         percentage
## Class_1       3.12
## Class_2      26.06
## Class_3      12.93
## Class_4       4.35
## Class_5       4.43
## Class_6      22.84
## Class_7       4.59
## Class_8      13.68
## Class_9       8.01
```

```
Test
```

```
##          percentage
## Class_1       3.11
## Class_2      26.05
## Class_3      12.94
## Class_4       4.35
## Class_5       4.43
## Class_6      22.84
## Class_7       4.59
## Class_8      13.68
## Class_9       8.00
```

Percenteges are close, and sample is representative.

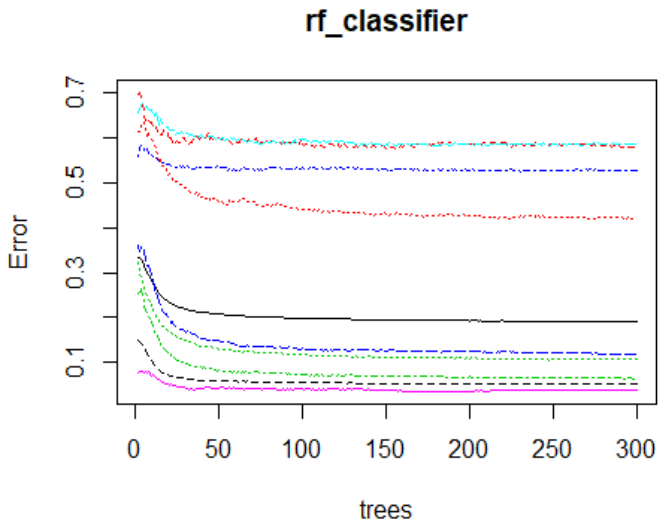
Step 3 - training the model

Random forest is ensemble of decision trees, which means it builds many trees and averages them to find the best model. There are different libraries in R that can implement random forest algorithm. First model was built using randomForest library and 300 trees.

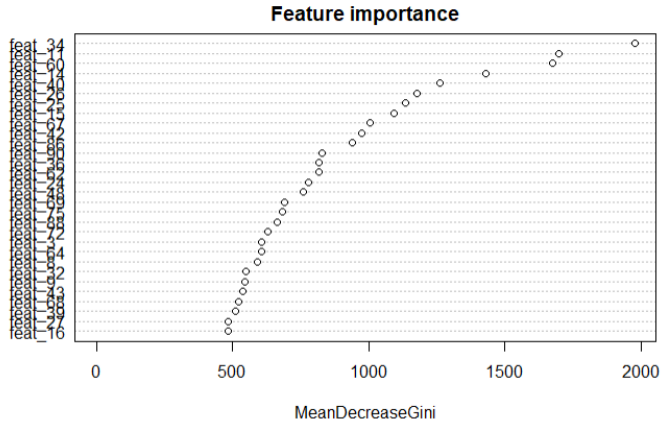
```
## Call:
##  randomForest(formula = target ~ ., data = train_products, ntree = 300)
##                Type of random forest: classification
##                      Number of trees: 300
## No. of variables tried at each split: 9
##
##         OOB estimate of  error rate: 19.15%
## Confusion matrix:
##         Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8
## Class_1     683      59       7       1       4     151      44     346
## Class_2       1   12255    1222      66      15      31      68      30
## Class_3       1    3369    3256      53       0      12      74      29
## Class_4       0     943     289     938      16      78      19       3
## Class_5       1      66       5       0    2240       5       5       1
## Class_6      32     113      17      11       3   11364     117     220
## Class_7      22     322     184      30      13     208    1379     227
## Class_8      52      90      12       0       4     201      25    6719
## Class_9      71      94       0       0       7     167      20     164
##         Class_9 class.error
## Class_1     345  0.58353659
## Class_2      16  0.10573555
## Class_3       9  0.52138762
## Class_4       1  0.58985571
## Class_5       5  0.03780069
## Class_6     138  0.05418227
## Class_7      28  0.42851223
## Class_8      91  0.06602724
## Class_9    3689  0.12416904
```

Random forest algorithms have an OOB (out-of-bag) error rate – it is unbiased estimator of the classification error. The OOB error here is 19.15%, for a default random forest model, which randomly selected 9 variables at each split.

Step 4 - evaluating model performance



The black line represents OOB error rate and the other 9 lines are error rates for 9 different categories. For every line we see that after 150 trees there isn't much of a change.



The five most important features to determine the class of the product are features 34, 11, 60, 14 and 40.

Predicting the classes for test set. Preview of first six predictions.

```
##        4        5        8       11       16       20
## Class_1 Class_1 Class_6 Class_9 Class_6 Class_1
## 9 Levels: Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 ... Class_9
```

Making confusion matrix – checking for accuracy and kappa statistics.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8
##    Class_1     120       0       0       0       0       7       4       4
##    Class_2      13    2173     600     160       9      23      51      16
##    Class_3       1     213     566      42       0       0      30       5
##    Class_4       0      13      12     189       0       2       3       0
##    Class_5       1       2       0       2     399       0       3       1
##    Class_6      29       3       1       9       1    2020      37      38
##    Class_7       4       8      15       1       1      24     254       1
##    Class_8      54       3       5       0       1      29      39    1188
##    Class_9      67       3       2       1       0      15       5      17
##           Reference
## Prediction Class_9
##    Class_1      12
##    Class_2      14
##    Class_3       3
##    Class_4       0
##    Class_5       2
##    Class_6      32
##    Class_7       5
##    Class_8      30
##    Class_9     645
##
## Overall Statistics
##
##                Accuracy : 0.8138
##                  95% CI : (0.8058, 0.8217)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7722
##  Mcnemar's Test P-Value : NA
##
```

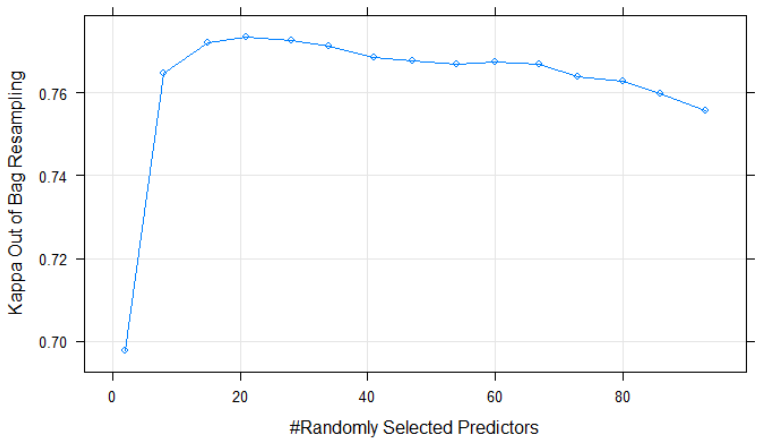
Accuracy is 81.38% and Kappa is 77.22%.

Step 5 – improving model performance

Tuning random forest parameters with caret package - searching for the best value of mtry, number of variables selected at random for building each tree in the forest.

```
## Random Forest
##
## 52596 samples
```

```
##    93 predictor
##     9 classes: 'Class_1', 'Class_2', 'Class_3', 'Class_4', 'Class_5',
## Class_6', 'Class_7', 'Class_8', 'Class_9'
##
## No pre-processing
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.7577382  0.6979493
##    8    0.8078371  0.7646588
##   15    0.8132938  0.7721413
##   21    0.8139212  0.7732476
##   28    0.8131987  0.7725787
##   34    0.8119629  0.7711941
##   41    0.8096243  0.7683777
##   47    0.8089589  0.7676182
##   54    0.8082174  0.7668310
##   60    0.8085976  0.7673094
##   67    0.8081983  0.7668914
##   73    0.8057077  0.7638657
##   80    0.8047950  0.7628016
##   86    0.8021713  0.7596440
##   93    0.7989581  0.7557563
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 21.
```



When 21 features were selected at random, kappa statistics and accuracy were the highest, for random forest with 300 trees.

```
## Call:
##  randomForest(formula = target ~ ., data = train_products, ntree = 300,
mtry = 21)
##                Type of random forest: classification
##                      Number of trees: 300
```

```
## No. of variables tried at each split: 21
##
##         OOB estimate of  error rate: 18.79%
## Confusion matrix:
##         Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8
## Class_1     728      57      13       1       5     134      51     309
## Class_2       1   11913    1513      92      14      30      87      31
## Class_3       1    3029    3552      81       1      12      88      29
## Class_4       1     808     331    1029      13      72      24       6
## Class_5       4      61       7       1    2239       6       3       1
## Class_6      40     100      21      17       3   11342     126     235
## Class_7      32     276     194      29      13     175    1469     198
## Class_8      68      77      17       2       6     175      39    6708
## Class_9      65      92       1       2       6     143      23     148
##         Class_9 class.error
## Class_1     342  0.55609756
## Class_2      23  0.13069177
## Class_3      10  0.47787741
## Class_4       3  0.55006559
## Class_5       6  0.03823024
## Class_6     131  0.05601332
## Class_7      27  0.39121426
## Class_8     102  0.06755630
## Class_9    3732  0.11396011
```

OOB error rate has improved, it's 18.79%.

Preview of first six predicted classes and confusion matrix with accuracy and kappa values.

```
##       4       5       8      11      16      20
## Class_1 Class_1 Class_7 Class_9 Class_6 Class_1
## 9 Levels: Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 ... Class_9
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8
##    Class_1     129       0       0       0       0       8       7       8
##    Class_2      12    2108     527     135       6      21      40      13
##    Class_3       2     268     635      51       0       1      34       6
##    Class_4       0      19      14     204       0       3       3       0
##    Class_5       1       2       0       2     401       0       3       1
##    Class_6      23       1       1       9       1    2021      33      34
##    Class_7       7      13      16       1       2      23     268       3
##    Class_8      49       4       4       1       1      27      33    1187
##    Class_9      66       3       4       1       0      16       5      18
##           Reference
## Prediction Class_9
##    Class_1      16
##    Class_2      11
##    Class_3       3
```

```
##    Class_4       0
##    Class_5       1
##    Class_6      29
##    Class_7       5
##    Class_8      27
##    Class_9     651
##
## Overall Statistics
##
##                Accuracy : 0.8192
##                  95% CI : (0.8112, 0.827)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7798
##  Mcnemar's Test P-Value : NA
```

Accuracy is 81.92% and kappa is 77.98%. The model has improved.

We saw before that after 150 trees there is no much of a, let's see performance of the model with 150 trees and 21 random variables at each split.

```
## Call:
##  randomForest(formula = target ~ ., data = train_products, ntree = 150,
mtry = 21)
##                Type of random forest: classification
##                      Number of trees: 150
## No. of variables tried at each split: 21
##
##         OOB estimate of  error rate: 18.99%
## Confusion matrix:
##         Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8
## Class_1     730      56      11       1       7     143      58     298
## Class_2       2   11861    1553     103      16      36      81      31
## Class_3       1    3044    3540      83       1      14      81      28
## Class_4       0     799     340    1029      14      75      22       6
## Class_5       3      66       3       1    2241       5       3       1
## Class_6      41     103      20      16       3   11337     123     242
## Class_7      37     259     214      39      12     171    1448     205
## Class_8      73      71      23       1       8     176      41    6697
## Class_9      67      93       6       1       7     144      21     147
##         Class_9 class.error
## Class_1     336  0.55487805
## Class_2      21  0.13448628
## Class_3      11  0.47964133
## Class_4       2  0.55006559
## Class_5       5  0.03737113
## Class_6     130  0.05642946
## Class_7      28  0.39991712
```

```
## Class_8     104  0.06908535
## Class_9    3726  0.11538462
```

OOB error rate is 18.99%, it's better than initial model, but worse than model with 300 trees and mtry = 21.

Preview of first six predicted classes and confusion matrix.

```
##       4       5       8      11      16      20
## Class_1 Class_1 Class_7 Class_9 Class_6 Class_1
## 9 Levels: Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 ... Class_9
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8
##    Class_1     134       0       0       0       0       9       9       8
##    Class_2      11    2108     525     127       7      19      38      13
##    Class_3       4     266     635      59       0       2      35       6
##    Class_4       0      20      15     204       0       4       4       0
##    Class_5       1       2       0       2     400       0       2       1
##    Class_6      24       1       1       9       1    2014      35      32
##    Class_7       7      14      18       1       2      27     263       3
##    Class_8      48       4       3       1       1      28      35    1186
##    Class_9      60       3       4       1       0      17       5      21
##           Reference
## Prediction Class_9
##    Class_1      18
##    Class_2      13
##    Class_3       2
##    Class_4       0
##    Class_5       1
##    Class_6      28
##    Class_7       6
##    Class_8      27
##    Class_9     648
##
## Overall Statistics
##
##                Accuracy : 0.8179
##                  95% CI : (0.8099, 0.8257)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7783
##  Mcnemar's Test P-Value : NA
```

Accuracy is 81.79% and kappa is 0.7783.

XGBoost model

After implementing different random forest models, accuracy didn't change much. Applying extreme gradient boost model (xgboost) may help to improve accuracy. For xgboost model both training and test set should be xgb matrices. Target variable in the data is a factor, as xgboost works only with numeric type, target variable for both train and test sets, needed to be converted.

Closer look into target variable for the training set – preview of first few elements and summary statistics.

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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   5.000   4.842   7.000   9.000
```

Setting parameters and finding number of rounds (similar to number of trees for random forest). Best number of rounds is 141.

```
## ##### xgb.Booster
## raw: 3.1 Mb
## call:
##   xgb.train(params = param, data = dtrain, nrounds = n, watchlist =
list(val = dtest,
##     train = dtrain), print_every_n = 21, early_stopping_rounds = 10,
##     maximize = F)
## params (as set within xgb.train):
##   booster = "gbtree", num_class = "10", objective = "multi:softmax",
eval_metric = "merror", nthread = "4", max_depth = "6", eta = "0.3", gamma =
"0", subsample = "1", colsample_bytree = "1", silent = "1"
## xgb.attributes:
##   best_iteration, best_msg, best_ntreelimit, best_score, niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
##   cb.evaluation.log()
##   cb.early.stop(stopping_rounds = early_stopping_rounds, maximize =
maximize,
##     verbose = verbose)
## niter: 141
## best_iteration: 141
## best_ntreelimit: 141
## best_score: 0.083504
## evaluation_log:
##     iter val_merror train_merror
##        1   0.272247     0.272017
##        2   0.254687     0.242395
## ---
##      140   0.177548     0.083752
##      141   0.177656     0.083504
```

Preview of first six predicted classes.

```
## [1] 1 1 6 1 1 1
```

Evaluating this model – getting confusion matrix, accuracy and kappa values.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2    3    4    5    6    7    8    9
##          1  163    0    0    0    0   16    6   18   27
##          2    8 2068  512  109    7   12   34    9   11
##          3    4  291  635   50    0    9   24    6    3
##          4    0   27   23  224    0    2   10    0    1
##          5    0    4    0    0  401    0    2    1    0
##          6   24    3    2    8    0 2014   28   31   24
##          7    8   13   25   10    1   29  295    3    1
##          8   33    7    2    2    2   23   24 1183   26
##          9   49    5    2    1    0   15    3   19  650
##
## Overall Statistics
##
##                Accuracy : 0.8223
##                  95% CI : (0.8144, 0.8301)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7844
##  Mcnemar's Test P-Value : NA
```

Accuracy increased, and it is 82.23% and kappa is 78.44%. This is the model with highest accuracy and kappa values.

Final model

The best model was xgboost model with 141 rounds, evaluation metric – merror, eta 0.3 and max depth of 6. It's interested to see which classes were misclassified for which the most.

This is the summary of misclassified categories.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    1.00    6.50   22.90   23.25  512.00
```

In the last quartile are pairs of classes that have been misclassified more than 23 times. Let's look at the first five misclassified pairs .

```
##    Prediction Reference Freq
## 20          2         3  512
## 12          3         2  291
## 29          2         4  109
## 30          3         4   50
## 9           9         1   49
```

Class 2 was predicted instead of Class 3 512 times, and Class 3 was predicted instead of Class 2 291 times. Class 2 and Class 3 must have similar features.

As default xgboost model gave better results than tuned random forest models, the next step would be to tune parameters for xgboost model to try to improve accuracy.

Appendix

More models

A different library for running random forest algorithm in R is ranger library. It runs faster than randomForest library.

- The first model with ranger is like initial model, with 300 trees.

```
## Ranger result
##
## Call:
##  ranger(train_products$target ~ ., train_products, num.trees = 300)
##
## Type:                             Classification
## Number of trees:                  300
## Sample size:                      52596
## Number of independent variables:  93
## Mtry:                             9
## Target node size:                 1
## Variable importance mode:         none
## OOB prediction error:             19.11 %
```

It is starting slightly better than randomForest, with OOB error rate of 19.11%.

Making prediction and evaluation of the model with confusion matrix.

```
## [1] Class_1 Class_1 Class_7 Class_1 Class_6 Class_1
## 9 Levels: Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 ... Class_9
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8
##    Class_1     116       0       0       0       0       7       5       3
##    Class_2      14    2160     587     156       7      21      49      12
##    Class_3       0     228     580      44       0       0      31       7
##    Class_4       0      11      10     191       0       2       3       0
##    Class_5       1       2       0       2     400       0       3       1
##    Class_6      31       1       2       8       1    2024      36      41
##    Class_7       6       9      15       1       2      24     256       1
##    Class_8      53       4       4       1       1      27      39    1190
##    Class_9      68       3       3       1       0      15       4      15
##           Reference
## Prediction Class_9
##    Class_1      11
```

```
##    Class_2      13
##    Class_3       5
##    Class_4       0
##    Class_5       1
##    Class_6      29
##    Class_7       5
##    Class_8      29
##    Class_9     650
##
## Overall Statistics
##
##                Accuracy : 0.8152
##                  95% CI : (0.8072, 0.8231)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7741
##  Mcnemar's Test P-Value : NA
```

Accuracy is 81.52% and kappa is 0.7741, which is better than initial model, but still worst than the other two models with randomForest function.

- Let's try ranger with 150 trees.

```
## Ranger result
##
## Call:
##  ranger(train_products$target ~ ., train_products, num.trees = 150)
##
## Type:                             Classification
## Number of trees:                  150
## Sample size:                      52596
## Number of independent variables:  93
## Mtry:                             9
## Target node size:                 1
## Variable importance mode:         none
## OOB prediction error:             19.38 %
```

OOB error is 19.38% and mtry is 9.

Preview of first six predictions and confusion matrix.

```
## [1] Class_1 Class_1 Class_7 Class_9 Class_6 Class_1
## 9 Levels: Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 ... Class_9
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8
##    Class_1     113       0       0       0       0       6       7       3
##    Class_2      14    2151     576     155       8      20      50      13
```

```
##    Class_3       0     236     593      44       0       2      31       7
##    Class_4       0       8      10     190       0       2       3       0
##    Class_5       1       2       0       2     399       0       3       1
##    Class_6      32       2       2       8       1    2025      38      38
##    Class_7       6      11      13       3       2      21     255       1
##    Class_8      53       5       5       1       1      29      36    1192
##    Class_9      70       3       2       1       0      15       3      15
##           Reference
## Prediction Class_9
##    Class_1      12
##    Class_2      17
##    Class_3       4
##    Class_4       0
##    Class_5       1
##    Class_6      31
##    Class_7       2
##    Class_8      29
##    Class_9     647
##
## Overall Statistics
##
##                Accuracy : 0.815
##                  95% CI : (0.807, 0.8229)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7738
##  Mcnemar's Test P-Value : NA
```

Accuracy is 81.5% and kappa is 0.7738

- Ranger with 150 trees and 21 random variables for each split.

```
## Ranger result
##
## Call:
##  ranger(train_products$target ~ ., train_products, num.trees = 150,
mtry = 21, min.node.size = 1)
##
## Type:                             Classification
## Number of trees:                  150
## Sample size:                      52596
## Number of independent variables:  93
## Mtry:                             21
## Target node size:                 1
## Variable importance mode:         none
## OOB prediction error:             19.05 %
```

```
## [1] Class_1 Class_1 Class_7 Class_9 Class_6 Class_1
## 9 Levels: Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 ... Class_9
```

Making Confusion Matrix.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8
##    Class_1     127       0       0       0       0      10       6      11
##    Class_2      11    2111     547     134       8      20      46      13
##    Class_3       3     260     615      48       0       2      30       5
##    Class_4       0      22      12     205       0       3       3       0
##    Class_5       1       2       0       2     401       0       2       1
##    Class_6      25       3       1      10       1    2019      36      33
##    Class_7       8      13      19       3       0      27     262       2
##    Class_8      51       4       3       1       1      24      34    1187
##    Class_9      63       3       4       1       0      15       7      18
##           Reference
## Prediction Class_9
##    Class_1      13
##    Class_2      15
##    Class_3       2
##    Class_4       0
##    Class_5       1
##    Class_6      26
##    Class_7       6
##    Class_8      27
##    Class_9     653
##
## Overall Statistics
##
##                Accuracy : 0.8166
##                  95% CI : (0.8086, 0.8245)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7765
##  Mcnemar's Test P-Value : NA
```

Accuracy is 81.66% and kappa is 0.7765.

```
## Ranger result
##
## Call:
##  ranger(train_products$target ~ ., train_products, num.trees = 300,
mtry = 21)
##
## Type:                             Classification
## Number of trees:                  300
```

```
## Sample size:                      52596
## Number of independent variables:  93
## Mtry:                             21
## Target node size:                 1
## Variable importance mode:         none
## OOB prediction error:             18.88 %
```

Predictions:

```
## [1] Class_1 Class_1 Class_6 Class_9 Class_6 Class_1
## 9 Levels: Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 ... Class_9
```

Making Confusion Matrix.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Class_1 Class_2 Class_3 Class_4 Class_5 Class_6 Class_7 Class_8
##    Class_1     127       0       0       0       0       7       5      10
##    Class_2      11    2116     541     134       8      20      42      12
##    Class_3       3     255     622      49       0       2      31       6
##    Class_4       0      24      15     203       0       2       3       0
##    Class_5       1       2       0       2     400       0       2       1
##    Class_6      26       1       1      10       1    2020      34      30
##    Class_7       6      12      15       4       1      24     267       2
##    Class_8      49       4       3       1       1      29      35    1194
##    Class_9      66       4       4       1       0      16       7      15
##           Reference
## Prediction Class_9
##    Class_1      12
##    Class_2      14
##    Class_3       2
##    Class_4       0
##    Class_5       1
##    Class_6      28
##    Class_7       6
##    Class_8      25
##    Class_9     655
##
## Overall Statistics
##
##                Accuracy : 0.8192
##                  95% CI : (0.8112, 0.827)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7797
##  Mcnemar's Test P-Value : NA
```

Comparing all models

Name of the model	OOB (%)	Accuracy (%)	Kappa (%)	Mtry	Number of trees/rounds
rf_classifier	19.15	81.38	77.22	9	300
rf_classifier2	18.79	81.92	77.98	21	300
rf_classifier2a	18.99	81.79	77.83	21	150
ranger_classifier	19.11	81.52	77.41	9	300
ranger_classifier1	19.38	81.5	77.38	9	150
ranger_classifier2	19.05	81.66	77.65	21	150
ranger_classifier2a	18.88	81.92	77.97	21	300
xgb	/	82.23	78.44	/	147

We see that two model, with the same properties (mtry = 21, ntrees = 300), performed almost the same with randomForest library and ranger library.

In addition, I have also tried using h20, it was faster, but it didn't have better result than this model.

However, the best model of those is the xgboost model.

R code

```
products <- read.csv("train.csv")
head(products)
nrow(products)
ncol(products)
products <- products[,-1]
anyNA(products)
(s <- summary(products$target))
X <- as.matrix(round(prop.table(table(products$target)) * 100, 2))
colnames(X) <- c("percentage")
X
s_names <- sprintf("%s (%s%%)", paste("Class ", c(1:9)), X)
names(s) <- s_names
s
library(ggplot2)
ggplot(products, aes(x = "", fill = factor(target))) +
  geom_bar(width = 1) +
  labs(fill = "Class",
       x = NULL, y = NULL,
       title = "Pie Chart of Classes") +
  coord_polar(theta = "y", start = 0) +
  theme_minimal()
library(caTools)
set.seed(123)
split = sample.split(products$target, SplitRatio = 0.85)
train_products = subset(products, split == TRUE)
test_products = subset(products, split == FALSE)
Train <- as.matrix(round(prop.table(table(train_products$target)) * 100, 2))
colnames(Train) <- c("percentage")
paste("Train: ")
Test <- as.matrix(round(prop.table(table(test_products$target)) * 100, 2))
colnames(Test) <- c("percentage")
paste("Test: ")
library(doParallel)
cl <- makePSOCKcluster(4)
registerDoParallel(cl, cores = 4)

library(e1071)
library(randomForest)
rf_classifier <- randomForest(target ~., train_products, ntree = 300)
rf_classifier
summary(rf_classifier)
plot(rf_classifier)
importance <- as.data.frame(rf_classifier$importance)
importance$features <- rownames(importance)
library(dplyr)
importance <- arrange(importance, desc(importance$MeanDecreaseGini))
importance %>% select(features, MeanDecreaseGini)
```

```
varImpPlot(rf_classifier, main = "Feature importance")
set.seed(123)
class_pred <- predict(rf_classifier, newdata = test_products[-94])
head(class_pred)
library(caret)
(cm <- confusionMatrix(class_pred, test_products$target))
set.seed(123)
trCtrl <- trainControl (method = "oob", allowParallel = TRUE)
tic()
(rf_random <- train (target ~ ., data = train_products, method = "rf",
metric = "Kappa", tuneLength = 15, trControl = trCtrl))
toc()
set.seed(123)
rf_classifier2 <- randomForest(target ~., train_products, ntree = 300, mtry =
21)
rf_classifier2
set.seed(123)
class_pred2 <- predict(rf_classifier2, newdata = test_products[-94])
head(class_pred2)
(cm2 <- confusionMatrix(class_pred2, test_products$target))
set.seed(123)
rf_classifier2a <- randomForest(target ~., train_products, ntree = 150, mtry
= 21)
rf_classifier2a
set.seed(123)
class_pred2a <- predict(rf_classifier2a, newdata = test_products[-94])
head(class_pred2a)
m <- as.data.frame(cm2$table)
m <- m[m$Prediction != m$Reference, ]
summary(m$Freq)
m <- as.data.frame(cm2$table)
m <- m[m$Prediction != m$Reference, ]
summary(m$Freq)
m1 <- m[m$Freq >= 18.5, ]
nrow(m1)
m1 <- m1[order(-m1$Freq), ]
m1[1:10, ]
#install.packages("ranger")
library(ranger)
set.seed(123)
ranger_classifier <- ranger(train_products$target ~ ., train_products,
num.trees = 300)
summary(ranger_classifier)
set.seed(123)
ranger1_pred <- predict(ranger_classifier, data = test_products[-94])
head(ranger1_pred$predictions)
set.seed(123)
ranger_classifier1 <- ranger(train_products$target ~ ., train_products,
num.trees = 150)
ranger_classifier1
```

```
set.seed(123)
ranger1a_pred <- predict(ranger_classifier1, data = test_products[-94])
head(ranger1a_pred$predictions)
(cm_ranger1a <- confusionMatrix(ranger1a_pred$predictions,
test_products$target))
set.seed(123)
ranger_classifier2 <- ranger(train_products$target ~ ., train_products,
num.trees = 150, mtry = 21, min.node.size = 1)
set.seed(123)
class_pred_ranger2 <- predict(ranger_classifier2, data = test_products[-94])
head(class_pred_ranger2$predictions)
(cm_ranger2 <- confusionMatrix(class_pred_ranger2$predictions,
test_products$target))
set.seed(123)
ranger_classifier2a <- ranger(train_products$target ~ ., train_products,
num.trees = 300, mtry = 21)
set.seed(123)
class_pred_ranger2a <- predict(ranger_classifier2a, data = test_products[-
94])
head(class_pred_ranger2a$predictions)
(cm_ranger2a <- confusionMatrix(class_pred_ranger2a$predictions,
test_products$target))
library(data.table)
train_xgb <- setDT(train_products)
test_xgb <- setDT(test_products)

train_label <- as.numeric(train_xgb$target)
test_label <- as.numeric(test_xgb$target)

head(train_label)

summary(train_label)

new_train <- train_xgb[,-94]
new_test <- test_xgb[,-94]

train.matrix <- as.matrix(train_xgb[,-94])
mode(train.matrix) = "numeric"
test.matrix <- as.matrix(test_xgb[,-94])
mode(test.matrix) = "numeric"

library(xgboost)
dtrain <- xgb.DMatrix(data = train.matrix, label = train_label)
dtest <- xgb.DMatrix(data = test.matrix, label = test_label)

(num.class = length(levels(train_products$target))+1)

param <- list(booster = "gbtree",
              num_class = num.class,
              objective = "multi:softmax",
              eval_metric = "merror",
```

```
              nthread = 4,
              max_depth = 6,
              eta = 0.3,
              gamma = 0,
              subsample = 1,
              colsample_bytree = 1)

library(tictoc)
tic()
set.seed(123)
xgbcv <- xgb.cv(params = param, data = dtrain, nrounds = 150, nfold = 5,
                          showsd = T., stratified = T, print_every_n = 10,
maximize = F)

toc()

xgbcv

n <- which.min(xgbcv$evaluation_log$test_merror_mean)
n

library(tictoc)
tic()
set.seed(123)
xgb <- xgb.train(params = param,data = dtrain, nrounds = n,
                  watchlist = list(val = dtest, train = dtrain),
                  early_stopping_rounds = 10, maximize = F,
                  print_every_n = 21)

toc()

set.seed(123)
xgbpred <- predict(xgb, dtest)
head(xgbpred)

is.factor(xgbpred)

is.factor(test_label)

head(test_xgb$target)

head(test_label)

head(xgbpred)

xgbpred_f <- factor(xgbpred)
head(xgbpred_f)

test_label_f <- factor(test_label)
head(test_label_f)

library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2

set.seed(123)
(cm_xgb <- confusionMatrix(xgbpred_f, test_label_f))

(mat <- xgb.importance(feature_names = colnames(new_train), model = xgb))
xgb.plot.importance(importance_matrix = mat[mat$Gain >= 0.01])

features <- mat$Feature[mat$Gain >= 0.005]

m <- as.data.frame(cm_xgb$table)
m <- m[m$Prediction != m$Reference, ]

summary(m$Freq)

m <- m[order(-m$Freq),]
m[1:5,]
```